Czech Technical University in Prague

Faculty of Electrical Engineering

Department of telecommunication engineering

Bachelor thesis

Mobile network optimization exploiting Multi-Access Edge Computing

Jakub Nový

Supervisor: Ing. Jan Plachý

May 2018

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Prague, 20.5.2018 …….

Acknowledgement

I would like to thank my supervisor, Ing. Jan Plachý, for all the advice and help he provided me during the work on this thesis. I am very grateful for his positive, supportive attitude he showed every time we met.

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nový Jakub** Personal ID number: **456919**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Telecommunications Engineering**

Study program: **Communications, Multimedia, Electronics**

Branch of study: **Network and Information Technology**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Mobile Network Optimization Exploiting Multi-Access Edge Computing

Bachelor's thesis title in Czech:

Optimalizace mobilní sítě pomocí Multi-Access Edge Computing

Guidelines:

Study the Multi-Access Edge Computing (MEC) concept, OpenAirInterface (OAI) platform and Self optimization of mobile networks. Analyze challenges and existing solutions of the MEC implementation into the OAI platform for control of mobile devices communication. Based on existing solutions implement a suitable solution. Exploit the MEC for processing of information collected from mobile devices to optimize the mobile network. Implement required functionalities in the OAI and demonstrate its functionality.

Bibliography / sources:

[1] P. Mach, Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," IEEE Communications Surveys & Tutorials 2017.
[2] C.C Wang, et al, "Mobile Edge Computing-enabled Channel-aware Video Streaming for 4G LTE," 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), 2017.
[3] N. Nikaein, et al., "OpenAirInterface: A Flexible Platform for 5G Research," SIGCOMM Comput. Commun. Rev. 44, 5 (October 2014), 33-38.

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Plachý, Department of Telecommunications En, FEL

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **11.01.2018** Deadline for bachelor thesis submission: _____

Assignment valid until: **30.09.2019**

Ing. Jan Plachý Supervisor's signature	Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Anotace

Každým rokem roste počet zařízení připojených do mobilních sítí. Zároveň se zvyšují požadavky na jejich výkon a funkcionalitu, které se v současnosti blíží k technickým limitům zařízení. Jedním z možných řešení je využít Multi-Access Edge Computing, který uživatelům poskytuje výpočetní prostředky na hraně mobilní sítě, například v základnové stanici. Dále poskytuje mobilním operátorům možnost optimalizovat mobilní síť na hraně sítě.

Tato práce se kromě seznámení s konceptem Multi-Access Edge Computing soustředí na jeho začlenění do emulační platformy mobilních sítí OpenAirInterface a představuje experimentální řešení implementace, včetně demonstrace funkcionality implementace.

Summary

The number of devices connected to the mobile networks increases every year. At the same time, the performance and functionality requirements of the devices grow and reach the technical limits. One of the possible solutions is to employ Multi-Access Edge Computing, which provides the mobile users Computing resources at the edge of the mobile network, for example at base stations. Furthermore, it provides the mobile operators with a possibility to do mobile network optimisation at the edge of the mobile network.

This thesis focuses on understanding the Multi-Access Edge Computing and its integration in the mobile networks emulation platform OpenAirInterface. The experimental implementation and demonstration of implementation functionality are presented as well.

Key words: 4G/5G mobile networks, LTE, Mobile-Edge Computing, Multi-Access Edge computing, OpenAirInterface

Contents

1 Introduction

Mobile telecommunications provide connectivity to millions of people all around the world. Many everyday tasks are almost unimaginable without telecommunication technics – it is not only calling and sending short messages (SMS), but also many other services which were introduced to mobile devices as an additional functions. Devices, communicating with the mobile networks are generally called User Equipments (UEs), as they cover a wide area of apparatuses, such as, smartphones, tablets, other specialised devices or devices from the category of the Internet of Things (IoT), etc. Nowadays the UEs can run variety of users' applications, from a simple application for alarm clock or calculator to the complex applications which used to run only on a desktop computer just a few years ago.

The evolution in the field of mobile telecommunications is ever ongoing, wich is shown in the provided division of mobile network generations. The first generation (1G) consisted only of analogue mobile radio systems in the 1980s capable of transferring voice signals, second generation (2G) brought digital mobile systems, and the third generation (3G) was able to handle broadband data. Present fourth generation (4G), often called Long-Term Evolution Advanced (LTE-A) and LTE-A Pro, was firstly introduced in 3rd Generation Partnership Project (3GPP) Release 10 (containing its specifications) [1]. Both LTE-A and LTE-A Pro are entirely packet-switched and offer a high-speed connection for the UEs. Nowadays, the main focus is on fifth generation (5G) of mobile networks, which should provide higher capacity, as well as providing connectivity to a large number of mobile devices, including so-called Machine Type Communication [2], [3], while providing lower latency than 4G.

The mobile communication is a worldwide field of research and standardisation management. To create a stable environment for cooperation between industry and research, several standardisation organisations were established. Important standardisation body in this area is 3GPP, which defined a standard for 3G and now continues with standardisation of succeeding generations of mobile networks. The second standardisation institution named International Telecommunication Union (ITU) is a specialised agency of United Nations (UN) and targets standardisation in the field of telecommunication in general. Especially for Europe, the European Telecommunications Standards Institute (ETSI) is important. Institute of Electrical and Electronics Engineers (IEEE) should be mentioned too because this institute plays an important role in telecommunication field research.

Alongside the development of mobile networks continues the development of the UEs, which connect mobile users to the Internet via mobile networks. Even in this area, the development is rapid, but the physical limitations and different speed of research in the development of the UEs (communication parts, processors, batteries) may not match the mobile users' requirements. The mobile users' requirements are increasing due to the development of novel applications with increasing processing complexity. Furthermore, energy consumption of the UEs is not matched by the development of batteries.

To prolong the battery lifetime of the UEs, computation demanding applications can be run in the cloud on remote computational resources, e.g., Mobile Cloud Computing (MCC) [4]. However, exploitation of remote computational resources is limited by the communication latency.

To reduce the communication latency, a concept formerly known as Mobile Edge Computing (MEC) was introduced. The MEC, nowadays known as Multi-Access Edge Computing, enables to offload computation processing of the UE's applications to the servers at the edge of a network called Multi-Access Edge Host (MEH) [5], located at the base stations, small cells and other network elements, to prolong the battery lifetime of the UE. Due to MEC proximity to the UEs, the communication latency is reduced. Furthermore, offloading computation of UEs' applications to the MEH not only leads to energy savings, but enables the users to enjoy faster computation, and increase in the overall Quality of Experience (QoE). The exploitation of the MEC also supports the development of new applications which were previously unimaginable., e.g. applications exploiting the data collected directly from the mobile network and providing the UEs with a brand-new type of content. It is not only the users who profit from the MEC but also the mobile operators and third parties, who can come up with new applications based on the abilities of the MEC.

Ongoing development of the MEC requires a suitable environment which enables fast and cost-efficient development. Several test-beds have been already introduced to demonstrate the possibilities of the MEC, but the implementation in the mobile operators' networks will take some time. At the present day, it is necessary to test newly proposed solutions in the environment closest to the real mobile network.

The process of verifying new solutions can be performed in different ways. One way of verifications is to do simulations – preparing the computer behaviour model of the studied object and then running the simulation program in the laboratory environment. The simulation does not reflect all the real-life issues due to a high level of abstraction. Because of this reason, the simulations cannot model the studied situation in the whole complexity.

The second way how to confirm expected outputs is via a testbed consisting of real hardware. It is usually a very expensive matter, but the results are the closest to the real system results. The time needed to prepare the test is much longer than in the case of simulation and reproducibility of the experiment is problematic.

The last option, combining benefits from both previously mentioned ways is an emulation. The emulator provides a virtualised base for real application and thus delivers a better view of the whole tested system. The results obtained with emulation can be considered close to results acquired while measured in the real system. On the other hand, the difficulty and the price are (slightly) higher than the simulation.

The most up to date mobile network emulator, which follows 3GPP standards is the OpenAirInterface (OAI) platform [6], [7]. This open-source project, jointly developed by

many companies, universities and individuals, is highly sophisticated and allows to simulate and emulate mobile network even with the connected real hardware devices. The OAI enables emulation of both Radio Access Network (RAN) and the Core Network (CN). The UE can be simulated, emulated, or real device can be used.

The goal of this thesis is to analyse possible implementation scenarios how to integrate the MEC into the OAI platform and demonstrate the implementation.

The rest of the thesis is organised as follows. Introduction of the MEC and its use cases are provided in Chapter 2. The OAI platform is introduced in Chapter 3. Chapter 4 provides an overall description of the proposed MEC implementation, while several key implementation challenges are closely examined in its sections. In chapter 5 the future challenges for the complete MEC integration are summarised. The following Chapter 6 presents the demonstration of current MEC implementation and measurement results. The last chapter concludes the thesis and outlines future work.

2 Multi-Access Edge Computing in mobile networks

Although the idea of the MEC is simple – to provide low-latency services, such as cloud storage or computing, to the UEs and exploit data collected from the radio network to control the network – the whole concept is much more complicated and still evolving.

The MEC concept brings well-known MCC services to the RAN and extends it with the access to radio information to offer context-related services. The fundamental element in the whole MEC structure is a computational and storage device called Multi-Access Edge Host (MEH) at the edge of the network. To enable lower latency, the host servers (MEHs) are located as close as possible to the UEs [8], [9]. By placing the MEH at the base stations (eNBs), small cells, remote radio heads, etc. the latency significantly decreases and, also, backhaul network load decreases [10].

On the contrary to the MCC, the MEC architecture is very flexible due to the deployment of MEHs at the base stations. The approach of distributed MEHs is highly effective because it brings the possibility to strengthen the computational power in the whole network while enabling load balancing by offloading computation load of the overloaded MEHs to less loaded MEHs. However, the decentralised nature of the MEC has a drawback in the slightly more complicated allocation of computational resources, as well as handling of the UEs' mobility, compared to the centralised MCC. Nevertheless, the benefits of much lower latency and a significant decrease in mobile operators backhaul network overcome this drawback. The concept of decentralised hosts also means that the computational power and storage in each MEH is limited, although the overall capacity can be comparable to MCC (via joining multiple MEHs into computing cluster). Comparison of the MCC and the MEC is shown in Table 2.1, partially based on [11].

Table 2.1 MCC and MEC comparison.

Aspect	MCC	MEC
Arrangement schema	Centralised	Distributed
Arrangement design	Simpler	More difficult
Arrangement flexibility	Lower	Higher
Latency	Higher	Lower
Distance from UE	Higher	Lower
Storage capacity	Higher	Lower
Computational Power	Higher	Lower

In the following Section 2.1, the description of possible use cases and service scenarios of the MEC are discussed. Evolution of the MEC concept is given in Section 2.2, followed by a description of the MEC system architecture and implementation in Section 2.3. The last Section 2.4 points out new services brought by the MEC.

2.1 Multi-Access Edge Computing use cases and service scenarios

The MEC ecosystem is full of opportunities for end-users, mobile operators, content providers, and many other parties. All of them should cooperate to achieve innovative services and applications. The main three use case categories, as stated in [11], are focused on users, mobile networks, and operators or third-party providers. A division of the use cases and the service scenarios based on this paper are shown in Figure 2.1. Use cases are also discussed in ETSI specification GR MEC 018 [12] chapter 4 - Mobility requirements and use cases. In the mentioned specification, the use cases are analysed mainly from the perspective of timing and relocation of computation resources when the UE is moving. However, this thesis focuses on data gathering and its processing at MEH, thus the MEH is considered static.

A closer description of each category with examples of service scenarios to illustrate the possibilities for each category, based on the [11], is provided in the next subsections.

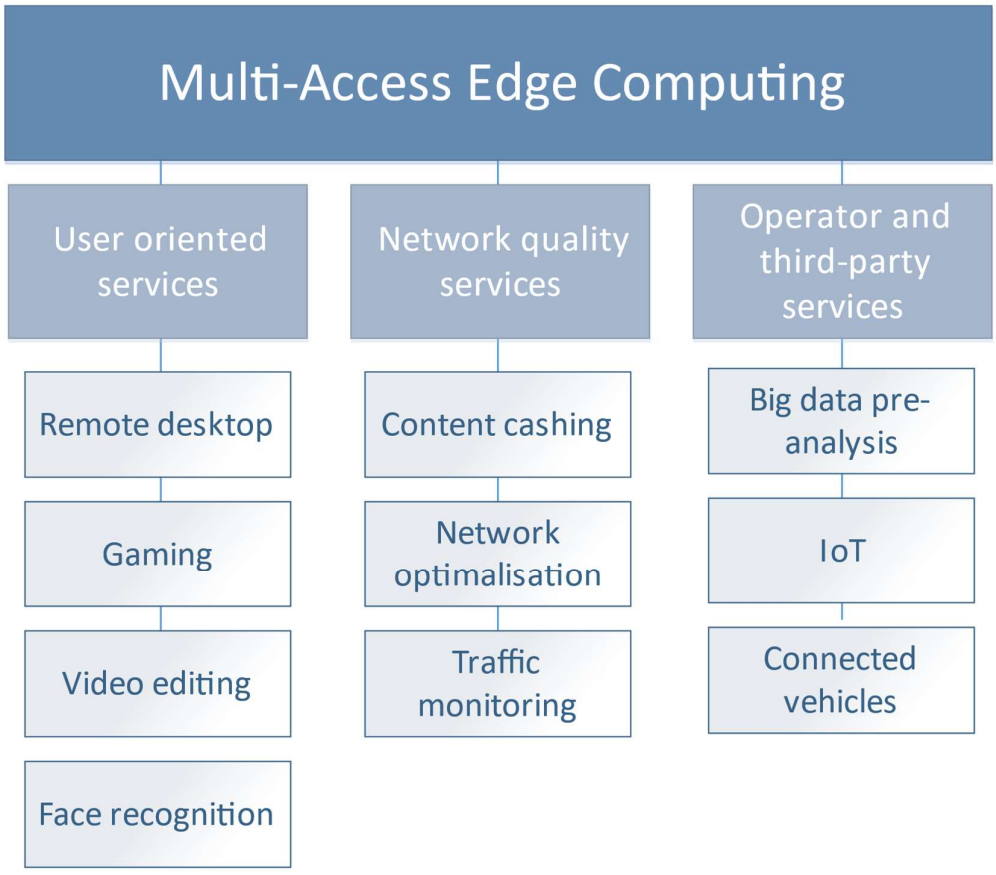

Figure 2.1 Division of the MEC use cases.

2.1.1 User-oriented services

The MEC is highly beneficial for users by providing the UEs with additional computational resources and storage capacity at the edge of the mobile networks. The additional computational resources can be exploited for example for application offloading [13]. To provide the UEs' with the best Quality of Service (QoS), several aspects must be considered.

In the offloading of UEs' applications, as introduced in [14], the UEs' application is presented as a set of tasks. The tasks are divided into two main groups: offloadable tasks, such as face recognition process or video editing, and non-offloadable tasks, which usually need hardware in the user device (e.g., sensors, user input, result output). The application itself usually comprises of a number of different tasks, where some of them are offloadable tasks whereas some are non-offloadable tasks. Naturally, only offloadable tasks can be offloaded to the MEC. Considering this fact, it is possible to offload just a part (one or more tasks) of the whole application, usually the one which is the most energy consuming.

The tasks can be related to each other, by requiring the output of one task as an input for the next task. When a group of tasks do not depend on each other, they can be processed in parallel at the same time. This fact can affect the resulting latency of the offloading and overall complexity. The execution can be done at the UE or at the assigned MEH. An example of parallel processing can be an analysis of the UE owner's face in several different ways. Note that the picture must be taken first through the phone camera. This simple example demonstrates task division – non-offloadable task (camera input) and offloadable task (face recognition) – as well as the dependencies between the tasks.

The question whether to offload or not has several aspects as shown in [15]. The offloading is only favourable when the communication conditions are appropriate. Thus, the accurate knowledge of current radio quality is crucial. Due to the UE's movement and varying environment (trees, vehicles, pedestrians, etc.), the quality of the connection between the UE and the base station changes in time. Without accurate information about the mobile network, more time can be spent by transmission of the offloaded task to the MEH and reception of the processed data than by the local (on-device) execution, leading to increased latency and energy consumption of the application [13].

It is clear that the offloading decision is a complex problem which requires multiple inputs for consideration. Algorithms, such as described in [15] and [16] have to decide whether to offload or not, based on the UE's current movement, amount of processed data, current MEC computational status compared to the UE's computational status, and the communication state.

2.1.2 Network quality services

The mobile network can significantly profit from the presence of the MEC. In this case, the MEC takes care of the network optimisation to improve the QoS. Due to the proximity

of the MEH and the RAN, it is easy to collect data from the connected devices, process them and optimise the network almost immediately. In the optimisation of the mobile network, several parameters related to the quality and resource allocation of the radio links can be adjusted. These parameters include a number of allocated resource blocks (RB) for uplink and downlink for each UE or Signal to Interference plus Noise Ratio (SINR) via a change of UE or base station transmission power. The mentioned parameters are part of Medium Access Control (MAC) layer and are obtained from the protocol stack [17]. Optimization via the MEC is not limited to the radio link parameters, as it is possible to change parameters of upper layers or other protocols such as Transmission Control Protocol (TCP), e.g. TCP windows scale option to increase the maximum value of bytes transferred in one TCP window [18]. Another way to improve QoS is to reroute the traffic via Software Defined Networking (SDN) [19]. The distributed nature of the MEC helps with the self-optimisation of the mobile network, which becomes necessary as the number of connected devices increases, making it impossible to optimise the whole mobile network manually. It is a matter of the software installed at the MEH which defines how significant is the influence on the UEs; theoretically, the possibilities are unlimited [20].

The MEC is beneficial together with new approaches in the field of mobile communications. A significant increase in the number of connected UEs and overall data transfers pushes the mobile operators to upgrade their infrastructure. To save expenditure, new approaches to the traditional mobile network structures are necessary to utilise all available resources efficiently. It can be the Cloud-RAN (C-RAN) architecture, where different base station resources are centralised to a single resource pool and then managed jointly [21]. The SDN architecture allows easy repolicing or reconfiguration via decoupling of the data and the control plane [22]. The MEC can be useful also in combination with Network Function Virtualization framework (NFV), which changes the architecture of core network (CN) to be more cost-efficient and enable dynamic changes depending on needs of the UEs [23].

The MEC can serve as local content storage for frequently accessed content in the MEC served area; this function of the MEC is denoted as content caching [24]. The amount of traffic flowing through the core network is then decreased by distributing the cached content at the MEH, instead of accessing the content beyond the CN.

It has to be noted that the MEC concept has still many open challenges to be solved, as, for example, shown in [11], [25]; the standardisation process is ongoing [26]. Nevertheless, ETSI defined several areas for a proof-of-concept, which are being completed to show the benefits of the MEC. As suggested by ETSI [20], and later presented in [27], the MEC can be used for channel-aware video streaming. The authors of [27] present the benefit of adaptive video transmission mechanism where the channel quality information is used to change the transmission bitrate to fit the estimated bandwidth dynamically. The demonstration was done via the emulated mobile network on the OpenAirInterface platform.

2.1.3 Operator and third-party services

Even in these days, the mobile operators collect a massive amount of data, known (due to their size) as Big Data about the network and connected devices. The Big Data is usually analysed at the operator's datacentre. However, it is very efficient to pre-process the data locally instead of transporting raw data to the core network. Thus, the MEC provides an option to pre-process the data to avoid transport of raw network data.

The MEC also plays a significant role in a boom of the IoT. These devices usually do not have enough memory, since they are usually single-purpose devices. In this case, the MEC can be used not only as a storage for the collected data, but also as a communication node for a group of neighbouring devices and serve as an IoT Gateway [20]. This type of processing is also known under name Fog computing [28]. Due to the high similarity between the Fog and MEC, the computational power of the MEC can be exploited for pre-processing and analysis of collected data. Then, only the pre-processed data are transferred to the core network or centralised cloud for further processing and analysis. This concept serves well for example for security purposes when a large number of video streams from the CCTVs are collected, but only times when there is an activity in the stream are transferred to the client [20]. As the MEH stores the whole stream, no data are lost and can be retrieved on demand.

Another service scenario presented by ETSI is a video stream analysis. MEC can analyse streams from the video cameras searching for a specific information, e.g. licence plates of vehicles entering an area and checking if these cars are allowed to enter the surveyed area.

Since the MEH is implemented directly at the base station or the at the edge of the CN, it is an operator's decision whether the infrastructure will be open for the third-parties, considering the potential economic profit. The third-parties and their systems and solutions can also have access to data from the mobile network (current load, QoS of devices) to enhance the performance of the provided service. The privacy of users has to be preserved, thus the given data must be anonymised or filtered before passing to the third party.

The MEC can provide infrastructure for connected car cloud, gathering various information from the vehicles and roadside sensors and then change the traffic structure.

The MEC gives ample possibilities for disruptive applications both providing users with new capabilities and supporting market growth. It is a new field full of challenges for many organisations and individuals, such as applications developers, mobile subscribers, content providers or software vendors [29].

2.2 Evolution and concepts of Multi-Access Edge Computing

Before the MEC has been standardised by ETSI, several concepts of edge computing were proposed. The idea of MEC origins from the MCC, which is still commonly used. During

the evolution, some parameters, such as MEC server location, changed. Short description of concepts is provided in following paragraphs.

The first idea to add computation and storage capabilities was introduced in 2012 [30]. The system named **Small Cell Cloud** (SCC) focused on enhanced small cells (SCeNBs). The SCeNBS are enhanced by additional storage and computational resources which can be exploited by the UEs for edge computing. The introduction of the SCeNBs requires a new entity called Small Cell Manager (SCM), which manages resource allocation. The SCC also considers an option to cluster multiple SCeNBs to provide more processing power for tasks which can be parallelised. The MEC resources are virtualised at the SCeNBs.

Another approach was introduced in **Mobile micro clouds** (MMC) [31]. No control entity is needed since the control is fully distributed.

Fast moving personal cloud (MobiScud) [32] architecture uses a software-defined network (SDN). The cloud resources are located at operator's cloud within RAN and not directly at the eNB. Control entity is called MobiScud control (MC).

The idea of **Follow me cloud** (FMC) [33] leaves the computational or storage power in CN network, more precisely to the distributed data (DC) centres which follow the UE when moving through the mobile network. Two additional entities are introduced: FMC controller and DC/GW mapping entity.

A concept named **CONCERT** [34] uses the SDN and Network function virtualisation (NFV). The resources are placed hierarchically within the network. In addition to local servers which are located at the base station (in case of LTE network eNB), the regional or central servers can be exploited as well. The control entity is called conductor.

Nowadays, the main focus is given to the MEC concept standardised by ETSI, which is supported by an increased number of researchers focusing on the MEC. The two following sections summarise the MEC as standardized by ETSI.

2.3 Multi-Access Edge Computing framework

The described MEC framework is based on the MEC standardisation by ETSI. Several standardisation aspects have been already examined, and other aspects are still being analysed. The primary focus of the standardisation process is on the Application Programming Interfaces (APIs) such as UE Identity API, Bandwidth Management API or General principles for Multi-Access Edge Service APIs, platform management and requirements.

Framework and reference architecture of the MEC is specified in ETSI GS MEC 003 [35]. The general entities are divided into three levels: network, host level, and system level, as illustrated in Figure 2.2.

The whole MEC system has two main parts covering several Multi-Access Edge (ME) elements – ME hosts (MEHs) and ME management. Short description of functional elements follows.

The essential device in the MEC ecosystem is **ME host** (MEH). This entity serves as an initial physical platform for all ME applications. The MEH offers storage, computational and network resources and performs virtualisation of computational resources for the ME applications. The virtualisation infrastructure of the MEH includes traffic routing between applications and other systems and services (DNS servers, networks, proxies).

The **ME platform** serves several purposes. It offers the environment for the ME application, instructs the data plane based on the traffic rules provided by the ME platform manager, configures a DNS server or proxy based on the inputs from the ME platform manager and hosts ME services.

The ME platform interacts with the **ME applications**. These applications running inside Virtual machines (VMs) benefit from the variety of additional inputs about network state compared to the applications running in the MCC. They advertise, consume and offer services. Each application is described by a number of requirements and rules such as maximal tolerable latency or required computational resources. The fulfilment of requirements is supervised by ME system level management.

ME system level management covers several functionalities to maintain the whole ME system. It consists of ME orchestrator (which controls core functions such as available resources, topology, integrity and authenticity for packages, selection of appropriate MEH, triggering application actions), Operations Support System and User application lifecycle management proxy.

The second part of ME management is the **ME host level management** which includes ME platform manager and the virtualisation infrastructure manager. ME host level management handes specific functionalities of given MEH.

On top of that, the whole MEC system contains **user equipment application** which has the capability to communicate with the ME system and **customer-facing service portal** for third-party customers.

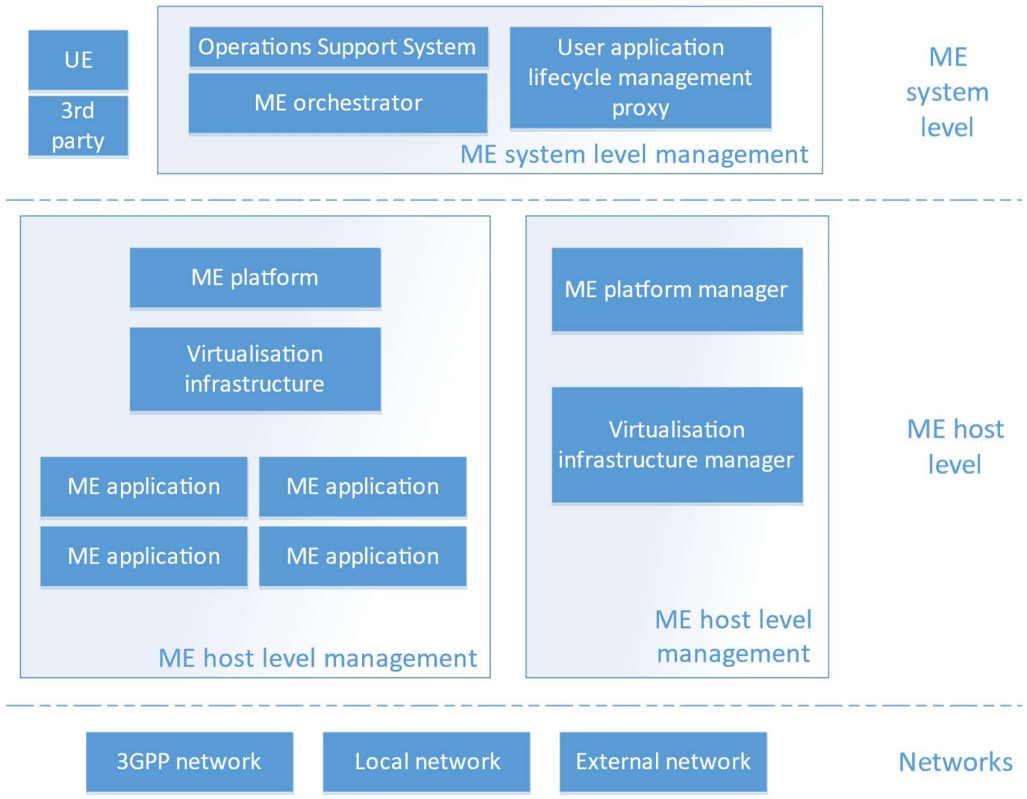

Figure 2.2 MEC framework with its parts.

2.4 Multi-Access edge services

It has been already mentioned several times that the MEC brings broader possibilities owing to the access to the additional data about the network and the UEs. This data can be divided into several groups called ME services [35].

The information related to the radio network is maintained by the **Radio Network information service**. It informs the application of actual radio network conditions, statistics and measurements related to the user plane, information about served UEs in the cell and change this information.

The **Location service** offers location-related data, such as the location of one, all or group of specified UEs served by the cell, list of UEs in a specific location and information about the location of radio nodes which are associated with the MEH. The format of location can vary: it can be Cell ID, geolocation, etc.

The **Bandwidth Manager** service can yield bandwidth to certain traffic and thus prioritise transfer of specific data from MEH.

3 OpenAirInterface

In 2004, Eurecom, French graduate school and research centre in communication technology, established OpenAirInterface™ (OAI) Software Alliance (OSA) [36]. The OSA consists of universities (including CTU in Prague) and research institutions as well as companies such as Orange, Alcatel-Lucent or Samsung. In this project, the academia meets the industry; the main purpose of this non-profit organisation is to provide a complex ecosystem for development of cellular systems which would speed up development and implementation of new functionalities and technologies.

The primary tool developed by OSA is OpenAirInterface platform – an open-source system consisting of Core Network (CN), in the 4G network called evolved packet core (EPC) and Radio Access Network (RAN), in 4G networks named evolved universal terrestrial Radio Access Network (E-UTRAN). The software implementation respects the division into two parts, openairCN for the CN functions and openair5G covering the RAN part of the whole cellular system. Both parts are in line with 3GPP, following architecture and protocols defined in 3GPP specifications.

The OAI is run on a generic PCs with Linux based operating system (preferably Ubuntu). It implements 3GPP standards for LTE (Release 8) and a subset of function from the Release 10 necessary for LTE-Advanced. Some other functionalities from further releases are continuously added.

In addition to the OAI, OSA develops special hardware devices for running of the RAN part via Software Defined Radio (SDR). Based on the software and hardware, the testbeds are being created for performance evaluation, demos and other research activities requiring close to the real network evaluation.

3.1 Software and system structure

The OAI project consists of two main parts – openairCN and openair5G, which are distributed separately due to use of some functionalities under different licences. Each part has specific system requirements on Operating System (OS) kernel version and type.

The OAI requires being run on a computer with OS Linux. The main part of both openair5G and openairCN is written in C-language. Other programming tools are used for special functions, such as ciphering.

The openair5G includes UE and base station E-UTRAN Node B (eNB). The openairCN covers the whole core network. It consists of the traditional nodes in LTE [37]:

- Home Subscriber Server (HSS) – a database with user's information,
- Mobility Management Entity (MME) – a control plane node which manages the bearers for the UEs and handover support,
- Packet Data Network Gateway (P-GW) – node which connects the CN to the internet, assigns the IP addresses to the UEs and controls QoS,

- Serving Gateway (S-GW) – user plane node between RAN and CN that works as a mobility anchor and collects data for charging.

The general relations between all nodes in LTE and LTE-Advanced networks are presented in Figure 3.1.

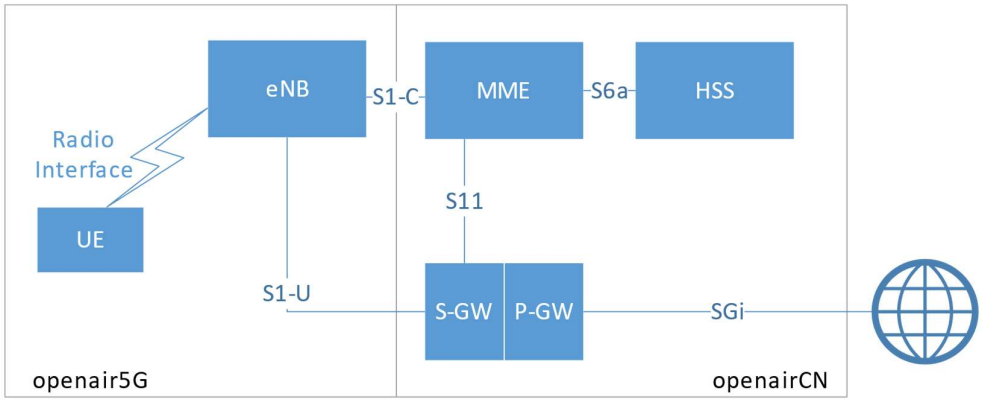

Figure 3.1 OAI structure with mobile network elements and their interconnection.

The UE and RAN are in the openair5G run as a single process. A wide spectrum of OAI compilations is available to the user, enabling him to choose the environment which suits his needs and interest the most. Once the chosen executable file is run from terminal, preliminary tasks take place to prepare the set-up. A new virtual network interface between the UE and eNB is created in the system, which can be checked or logged by several tools. One of these tools is a specialised ITTI analyser to analyse the internal communication between layers (RRC and S1AP, RRC and PDCP, PDCP and S1).

On the contrary, each CN component uses its own process and can be managed separately. Due to this structure, the communication channels are established and can be examined via many tools ranging from third party-software like Wireshark, Linux built-in utilities to special OAI tools (e.g. OAI timing analyser, OAI performance profiler). The main communication events are also displayed directly at the terminal with the process. To handle the amount of displayed information, log verbosity can be selected to omit unnecessary messages.

The RAN may be used in two modes, depending on the presence of the CN. The RAN can be run without the CN, denoted as an emulation without the S1 interface, or with CN, denoted as an emulation with the S1 interface. The S1 interface consists of the virtual interface over a physical one, providing connectivity between RAN and CN, as well as inter CN connection. When using emulation with the S1 interface, the base station interacts with CN.

3.2 OpenAirInterface platform

The OAI is an open-source code platform for mobile network simulations and emulations with a broad base of active developers. The source codes are freely available at the project website [38]. For the development and version control, git is used to ease the management enabling parallel development of multiple functionalities.

The OAI platform offers a wide spectrum of usage. The OAI can be exploited to emulate devices in a real-time (testing) or in non-real time (debugging). Communication between the eNB and the CN is achieved via IP, which encapsulates either message-oriented protocol with reliable transport Stream Control Transmission Protocol (SCTP) for control plane or message-oriented User Datagram Protocol (UDP), which does not guarantee the correct transfer of messages, for user data, as illustrated on Figure 3.2. If two or more nodes are emulated at one computer, direct memory transfer (via localhost connection) is used instead of Ethernet.

The OAI platform enables exploitation of both time division duplex (TDD) and frequency division duplex (FDD) communication for the division of uplink (UL) and downlink (DL) communication, as well as Hybrid automatic repeat request (HARQ) support and other functionalities described in Releases 8, 10 and partially 14. During the OAI emulation, the whole 3GPP protocol stack can be executed. Owing to this, the variety of indoor or outdoor experimentation can be carried out with the result as precise as possible.

The platform enables abstraction of Physical Layer (PHY) to speed up the simulation. The PHY abstraction works by exploitation of empirical models of the PHY layer. Without PHY abstraction, all functionalities of PHY layer are emulated, which make a more precise model of the real communication. However, the inaccuracy in PHY abstraction is very low as shown in [39].

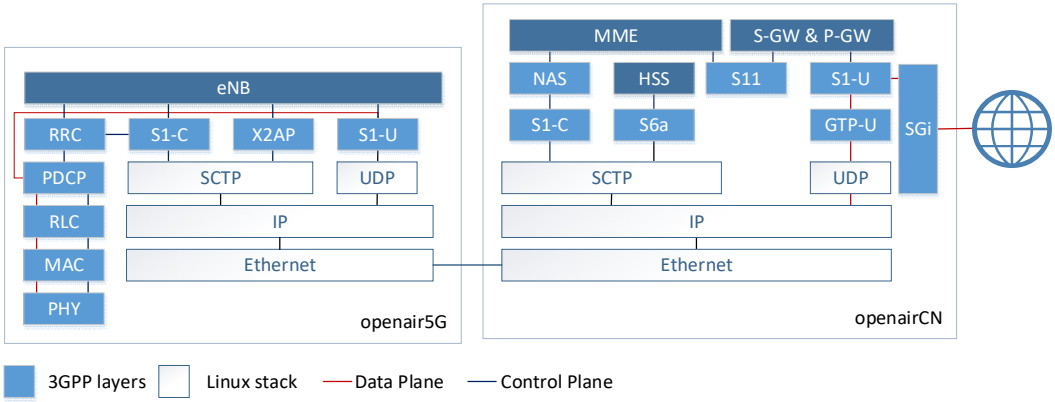

Figure 3.2 OAI protocol stack of openair5g and openairCN.

The OAI platform can be deployed in the combinations, based on [40], as follows:

- Commercial UE <-> OAI eNB + Commercial CN
- Commercial UE <-> OAI eNB + OAI CN

- Commercial UE <-> Commercial eNB + OAI CN
- OAI UE <-> Commercial eNB + OAI CN (experimental)
- OAI UE <-> Commercial eNB + Commercial CN (experimental)
- OAI UE <-> OAI eNB + Commercial CN (experimental)
- OAI UE <-> OAI eNB + OAI CN
- OAI UE <-> OAI eNB

It is obvious that OAI can work just in emulation mode as well as a part of a bigger set-up including commercial devices or devices developed directly by OSA for OAI.

3.3 OpenAirInterface structure

The RAN part of OAI platform is divided into several directories, each containing a specific part of the system. The core network (openair-cn) is also divided, but for purposes of the MEC implementation only RAN structure is needed. The simplified description of the RAN directory structure is given below:

- **Openair1** contains source files for physical layer together with the scheduling procedures for this layer.
- **Openair2** covers several layers and protocols, namely Radio link control (RLC) layer, Medium Access Control (MAC) layer, Packet Data Convergence Protocol (PDCP), The Radio Resource Control (RRC) and X2 Application Protocol (X2AP). It could be said that in this directory the main part of eNB app is located, as it contains the important schedulers. Also, source files for FlexRAN implementation, a software-defined network concept, are here.
- **Openair3** provides files for S1 Application Protocol (S1AP), Non-Access Stratum (NAS) and GPRS Tunnelling Protocol (GTPV1-U). Beside mentioned, the files ensuring the communication over the Stream Control Transmission Protocol (SCTP) and User Datagram Protocol (UDP) are stored here.
- The **common** directory offers OAI utilities and tools used over the whole platform.
- **cmake_targets** directory deals with Linux compilation requirements, such as build files and proper dependencies and included paths.

3.4 Internal messaging in OpenAirInterface

A Key utility for inter process/task/interface communication is a tool for internal messaging called ITTI. ITTI source files are located in `common/utils/itti` which indicate that it is a sub-system used in all levels of OAI.

ITTI plays two roles. When the whole system starts, it creates the threads for tasks. During the platform run, it ensures the communication between tasks and interfaces. Configuration files and files specifying the tasks are located in individual directories.

ITTI sub-system is enabled for most of the OAI use-cases, only during special occasions (such as cooperation with some commercial hardware devices), ITTI is disabled. The decision to disable ITTI must be marked when preparing the compilation of the platform by a parameter or rewriting the template of CMakeLists file.

3.4.1 Tasks and threads

The list of tasks is specified in `targets\COMMON\create_tasks.c` and depending on compilation configuration covers the following:

- task for L2L1,
- task for eNB APP,
- task for SCTP,
- task for S1AP,
- task for UDP,
- task for GTPV1U,
- task for NAS UE,
- task for RRC eNB,
- task for RAL eNB,
- task for RRC UE,
- task for RAL UE.

The list of tasks above is not complete and shows only some of the most important tasks. It should also be noted that tasks in the EPC are different than in the RAN. ITTI offers a possibility to create sub-tasks as well.

As said earlier, each task has its own thread created. A loop usually runs in the main function to check the incoming messages. In the main loop, specific functions are called based on the internal state, which decides which part of the code should be run next. Before the start of the loop, each task must be marked by ITTI as ready in order to launch the platform parts in the right order.

3.4.2 Messaging

When all tasks are set and marked as ready, ITTI's focus heads toward ensuring the inter-task communication, even though it still tracks the running threads. To maintain the overall communication in OAI, a set of function is defined including work with tasks, events and messages.

Each task has the use of a list of messages. This list of messages is defined for each task in two header files, and the structure is easy to understand since the file name starts with the task name and continues with a key string messages_def and messages_types, e.g. `udp_messages_def.h` and `udp_messages_types.h`. The first mentioned file defines the

messages itself (name, priority, structure), while the second file contains the definition of the structure of each message using a C language struct.

Commonly used ITTI function is `itti_alloc_new_message` which is called at the beginning of procedure or function and contains the origin (currently running) task and type of message (e.g. `message_p=itti_alloc_new_message(TASK_MME_APP, NAS_IMPLICIT_DETACH_UE_IND)`) followed by the main function/procedure code which inserts the data to the message. Assigned data depends on the type of message and its structure defined in the header file. When all required information is saved in the message, the whole ITTI message can be sent with procedure `itti_send_msg_to_task`, where the target, instance and message are specified (e.g. `itti_send_msg_to_task(TASK_NAS_MME, INSTANCE_DEFAULT, message_p)`).

For the connection with nodes outside the local machine, communication over the Ethernet is exploited. The function `itti_subscribe_event_fd` watches sockets and thus allows the communication using SCTP or UDP. When a new packet is received, it is assigned to the task based on the file descriptor and the received data is transferred to the final task via ITTI internal messaging.

A number of ITTI functions and procedures is wider. For the sake of simplicity, other functions and procedures are not described here in detail as it is not a crucial part of the suggested proposal.

4 Proposed Multi-Access Edge computing implementation design

The main goal of this work is to explore possible ways for implementation of MEC in the OAI platform and implement the best alternative. The implementation including the challenges is described in this chapter. The general idea is illustrated in Figure 4.1, where the MEH is connected directly to the eNB to achieve the lowest possible latency.

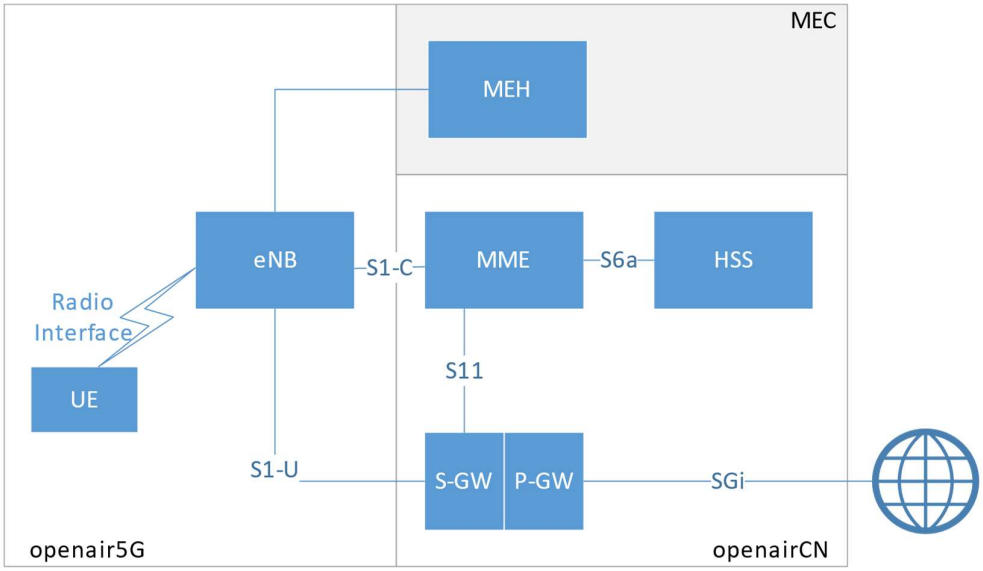

Figure 4.1 OAI with MEC structure.

Important prerequisites and tool for successful MEC implementation into the OAI platform are presented in Section 4.1.

The proposed architecture should generally aim to create a new connection between the eNB and the external MEH. This proposed connection will provide MEC with information from the eNB in real-time. Proposed implementation together with the description of possible protocols is shown in Section 4.2.

To make the MEC useful, it needs inputs from OAI. The parameters to be used by the MEH must be defined. Also, the locations, where to process the commands and requests from MEH needs to be in line with the current OAI code. The options in this area are introduced in Section 4.3.

To verify the ability and functionalities of enhanced OAI platform, MEH is necessary. Also, the current OAI code has to be changed to be capable of receiving, sending, and processing the data to the newly created MEH. The capability of the MEC depends on the code executed at the MEH - this simplified MEH application and code added to the OAI platform are introduced in section 4.4. The requirements on MEH in the sense of algorithm complexity and functionality were already discussed in the Subsection 2.1.1; the development of this algorithm is a difficult matter, as several teams work on this topic [8].

In the proposed system design the MEH acts only as an aggregation point with basic decision-making ability to demonstrate the possibilities of suggested design. Algorithms proposed by other work-groups can be then easily implemented later, as the communication between MEH and eNB remains the same.

4.1 Challenge 1: prerequisites for Multi-Access Edge Computing implementation

In the scope of ITTI, the exact location for the MEC source files is not crucial, as long as the code is capable of accessing all necessary functions (which means the required header files are included) and cooperates with ITTI.

4.1.1 Code location and integration

As the most suitable place for the implementation of the MEC code without extending the current file structure turned up the C file named `s1ap_eNB.c` located in `\openair3\S1AP`. The cause for selecting this exact file is simple: the functions located here are fundamentally able to communicate with the interfaces as well as with the main eNB part ensuring the scheduling, etc. Although the file is originally dedicated to the communication between the eNB and MME, adding extra code does not cause any harm to the present function. The next step is to put the MEC source code aside into its own file - this will make the platform well arranged. However, for testing purposes, the current solution is sufficient and easier too, as the dependencies in build structure do not have to be considered.

Depending on the demanded capabilities and level of integration into the OAI platform, several subsequent files must be changed besides the `s1ap_eNB.c`. Mainly header files so they include newly created MEC function and allow cooperation of the existing code with the newly added MEC.

At this moment, the MEC implementation is as a matter of fact only a separated system inserted into the OAI, allowing further integration in the future. This makes it a clean testing solution with a minimal impact on the rest of the system.

4.1.2 Thread allocation

To keep the MEC implementation clean and structured, the implementation follows OAI current code manner to divide task functionality into functions. As in any other program, these functions have clearly specified threshold event followed by the processing of the code. The OAI is optimised to achieve the highest performance. However, the optimisation makes almost impossible to test the proposed MEC solution within the task (or thread intended for the task) originally dedicated to another task. To ensure the uninterrupted MEC operation in the OAI, a new thread dedicated just to MEC must be created – this is the only way how to have separated space on the platform for MEC.

The tasks, which are set up, are (mostly) listed in file `create_tasks.c` in `\targets\COMMON`. Additional conditions, depending on the compiled version affect the number of started tasks. To allocate a thread for MEC, a new entry is added:

```
if (itti_create_task (TASK_MEC, mec_eNB_task, NULL) < 0) {
  LOG_E(UDP_, "Create task for MEC failed\n");
  return -1;
}
```

The function `itti_create_task(…)` starts a thread associated with the task. The parameters are a task to start and an entry point for this task; the optional argument to the start routine can be passed in the last argument. If the creation failed, the function return -1, otherwise 0.

When the task is created, the actual code has to be put to the function marked as a start point. In the proposed design, the procedure looks like this:

```
void *mec_eNB_task(void *arg)
{
  ...
  itti_mark_task_ready(TASK_MEC);
  ...
}
```

Procedure `void *mec_eNB_task(…)` is located in the file `s1ap_eNB.c`, which is the same file for starting S1AP, as mentioned previously.

It should also be noted that a procedure to mark the task to be in the ready state (`itti_mark_task_ready(…)`) is called at the beginning. Only the variables necessary for the proper MEC operation are defined before calling this procedure. The main loop starts after this procedure.

Creating a new task also brings another advantage: the ITTI tool knows that there is a part for MEC and internal messages can be addressed then to this task directly.

4.2 Challenge 2: connection between Multi-Access Edge Host and eNodeB

The MEC server usually does not need to exploit the information stored in CN, so the OAI openair-cn part is not considered in the proposed architecture. In the rare occasion when MEC needs a special network-state input from CN, it can be transmitted through the mediation of eNB. These data do not change frequently, so the incurred latency due to the mediator does not affect the MEC performance.

4.2.1 Connection options

OAI openair5G is currently endowed with two interfaces to communicate with other nodes over Ethernet: S1-C for MME and S1-U for S-GW. Several possible scenarios, how to

connect the MEH and OAI (eNB respectively) are discussed, as each of them has its advantages and disadvantages.

The first possible approach is to exploit the existing interface. The suitable existing interface for this purpose is an S1-C as it does not use any additional encapsulation protocol.

To separate the communication, packet-filtering based on a key attribute (keyword, flag) is necessary. The function of branching off the packet destined to the MEH and routing the traffic must be performed by an additional node between the eNB and the MME (e.g. Open vSwitch [41]).

This approach requires fewer code changes in OAI structure, as the already made procedures and function would be exploited. Contrarily, the additional software, e.g. Open vSwitch, developed outside the OSA is necessary and complicates the whole system architecture.

In the opposite to the previously mentioned approach is a creation of a new, third, interface designed only for the MEC communication. In this case, bigger changes in OAI would be necessary as the new interface has to be set up and maintain.

The third option is to use FlexRAN, a platform for SDR RAN developed by the University of Edinburgh, Eurecom and NCSR "Demokritos" [42]. Deploying of FlexRAN requires separation of data and control plane and involves new entity denoted as Master Controller. The Master Controller manages the SDN applications and a group of so-called FlexRAN Agents which are implemented in each eNB.

In 2016, FlexRAN was implemented to the OAI. The code had to be changed dramatically, as the data and control plane had not been separated entirely before [43]. Nowadays, even though the FlexRAN implementation remains in OAI, it is disabled by default due to higher delay in the MEC scheduling and thus overall throughput rate decrease.

However, for the MEC implementation, it is highly effective to exploit FlexRAN possibilities, even with the incurred delay. It is already implemented in OAI and provides many functions gathering the RAN statistics. However, due to mentioned lag in MEC scheduling, the direct engagement is not acceptable. Nevertheless, some functions processing RAN state information may be still implemented into proposed MEC without switching from internal eNB's scheduler to FlexRAN scheduler.

Creation of new interface only for MEC was decided to be the most suitable solution. It clearly separates the existing and newly created data traffic. The transition is also faster, as no additional software is needed and the overall system structure clearer. This arrangement also gives higher flexibility, e.g. in protocol choice.

4.2.2 Protocol choice

OAI platform currently uses two transferring protocols, as shown in Figure 3.2. First of them is User Datagram Protocol [44] which serves to transport datagrams (data units) between the computers. Just as a Transmission Control Protocol (TCP), UDP needs an Internet Protocol (IP) as the underlying layer. In the OSI model, UDP represents the transport layer.

The UDP is a lightweight protocol since it does not support any acknowledgements. Because of this fact, UDP is not suitable for the transmissions where the duplicate and delivery protection is needed. If the application needs a guaranteed transmission, other protocol should be used (such as STCP). The received data are checked by the upper layer. Because of this, minimum of protocol mechanism is necessary. UDP does not establish a connection prior the data transmission (it is connectionless).

The second protocol used in OAI is Stream Control Transmission Protocol [45] – an internet protocol originally designated for communication using Public Switched Telephone Network (PSTN) signalling messages. The SCTP operates on top of the IP networks and brings several useful features which make it useful not only for the MEC implementation but also other network applications:

- when data is transmitted using SCTP, the receiving side acknowledge non-duplicated and error-free reception of data to the sender,
- SCTP allows data fragmentation to fit maximum transmission unit (MTU) as well as bundling of multiple data messages into one SCTP packet,
- SCTP network-level is fault tolerable,
- SCTP supports multi-streaming for user messages and sequenced delivery of messages,
- SCTP supports multi-homing, which means that all network interfaces are accessible from all other network interfaces. This fact reduces internal routing of incoming IP datagram.

SCTP was developed to overcome some limitation of stream-oriented Transmission control protocol (TCP) such as the strict sequence order of delivered data or limited capacities of TCP sockets. SCTP deals with mentioned issues and brings a packet-oriented standard suitable for telecommunication applications.

The reliable transfer of data is provided by connection-oriented nature of SCTP between two endpoints. To be able to transfer user data, the connection must be established first. The initialisation is done by packet handshake, as illustrated in Figure 4.2. To establish a connection, a 4-way handshake is used: the connection is initiated by an SCTP client, the SCTP server answers with a cookie sent back to the client and waits for acknowledging to allocate the necessary resources. After the transmission of all data, the established connection should be torn down properly utilising so-called graceful SHUTDOWN chunk

as demonstrated in Figure 4.2. The connection can be instantaneously closed with ABORT chunk when unrecoverable errors appear.

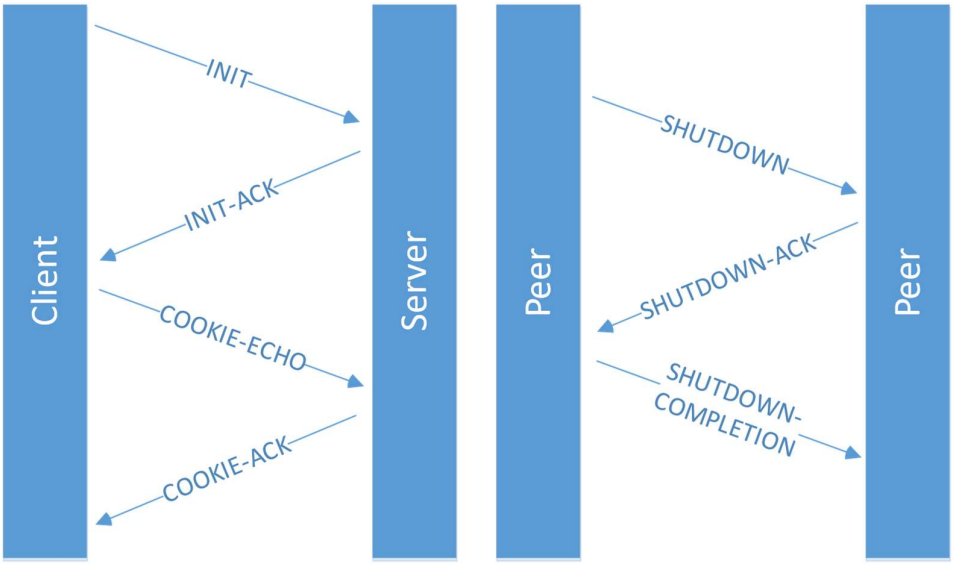

Figure 4.2 SCTP establishment and termination.

Considering the nature of the MEC data, the right choice for the communication protocol with MEH is SCTP due to its use for control plane transmission. The decision concurs with the overall OAI structure, where UDP is employed for data transfer between UEs and the Internet and the SCTP is used for communication between eNB and MME and between several eNBs.

4.2.3 Employment of ITTI

OAI is already endowed with a task specialised for communication over SCTP. This task creates, watches and closes the connection. Located at the bottom of OAI stack, SCTP task provides upper layers with pre-processed data through ITTI.

The basic scenario of SCTP layer interaction starts with a request from the upper layer with message `SCTP_NEW_ASSOCIATION_REQ` containing the name of the task, which require the connection establishment and the local and remote address, port number, SCTP Payload Protocol Identifier (PPID) together with the number of input and output streams. Functions of the SCTP task then process provided data and if no error occurs, respond to the requestor with `SCTP_NEW_ASSOCIATION_RESP`. Inside the SCTP task, the file descriptor (FD) of each connection is linked with the requestor and when new data arrives, the message is handed over to the upper task. The workflow in the opposite way is the same.

The MEC runs as a separated task which means that it can send and receive ITTI messages. However, it should be remembered that the utilisation of ITTI internal messaging function requires the allowed-message list.

A deeper analysis of required connections between task is necessary before employing the internal messaging functionality. Then the allowed-message list must be created and newly written function implemented to the remaining tasks as a new option of the incoming message. This follows the evolution of the simple test MEH server introduced in this thesis into the solution which is exploiting the whole functionality of the OAI.

However, the engagement of ITTI to serve the SCTP was already tested as follows. The procedure `static void s1ap_eNB_register_mme(…)` is modified to suit the MEC requirements, especially the port number and PPID, which are not given to the function as an argument. A new structure describing MEC is created, and after receiving the response, modified procedure `void s1ap_eNB_handle_sctp_association_resp(…)` stores the connection parameters into the newly created structure. The data to be sent to the SCTP are then transported using the procedure `void s1ap_eNB_itti_send_sctp_data_req(…)` to the SCTP layer.

The connection with this approach is always established correctly. The crucial issue, in this case, is the ciphering – procedures and functions in SCTP task expect that the provided data are already encrypted using an external library libasn1c. Owing to this fact, the transfer of plaintext data is unfeasible.

4.3 Challenge 3: eNodeB data harvesting and Multi-Access Edge Computing control

To develop a working MEC system, the data from the eNB have to be collected and after the processing in the MEC must control the system. The suitable functions and procedures in current source code must be found in order to collect and change the internal information.

4.3.1 Data harvesting

The variety of possible sources for data harvesting within OAI platform brings the question from where to collect the data for network optimisation. Two simple options are to either access the required data directly from the PHY layer or use the pre-processed data from RRC or MAC layer.

In the case of collecting data from the MAC layer, descriptor `MAC_xface`, which serves as an interface between MAC and PHY layer, is exploited. Its structure is defined in `\openair2\PHY_INTERFACE\defs.h` and contains many functions which are useful for further implementation.

Another option is to exploit the presence of FlexRAN source codes. The FlexRAN has its own internal processes, protocols, and functions. However, many of them can be easily rewritten to work alone without the rest of the FlexRAN software. Sometimes passing a different parameter than originally expected (a FlexRAN internal variable) is sufficient.

To demonstrate both possible approaches, three functions were selected (two from OAI, one from FlexRAN), implemented to the proposed MEC design, and tested. Short introduction of these functions is below:

- Function `uint32_t get_rx_total_gain_dB(…)` is one of the functions dealing with measurements of one specific UE based on its reference signal strength and transmission power of the eNB. The parameters are the module ID for the UE and Component carrier ID. The function is located in `openair1\PHY\LTE_ESTIMATION\defs.h` and processes elementary variables used in PHY layer.
- The second function also focuses on the connection status between the UE and eNB: `int16_t get_PL(…)`. The function to obtain the path-loss in dB is defined at the same place as the previously mentioned function. It has one extra parameter compared to the previous function, the index of the eNB on which to act.
- The last function originates from FlexRAN function `int flexran_get_tpc(…)` and returns the current TPC command. To suit the MEC purposes, the function was updated, so it does not use FlexRAN variables anymore and rely only on internal eNB variables. The successor function is now defined directly in `s1ap_eNB.c` file and under the name `mec_get_tpc()`.

4.3.2 eNodeB control

A proof of concept of OAI MEC implementation includes a part which utilises the power of MEC and puts received commands into its place. In the proposed system, this feature is represented by modifying uplink (UL) transmit power.

The UL power control is part of the uplink scheduler and can be found in `\openair2\LAYER2\MAC\eNB_scheduler_ulsch.c`. The power control algorithm takes into account several aspects, such as Path Loss and signalling from a higher layer, as further explained in [37]. The commands are sent to the UE through the Transmit Power Control (TPC) commands in Downlink Control Indicator (DCI) format. These commands are sent frequently, up to 1000 times per second. TPC command field in DCI format can have following values: 0 indicates a decrease of UE transmit power, 1 means no change, 2 indicates an increase of UE transmit power by 1 dB and 3 increases transmit power by 3 dB. However, in the OAI scheduler, the TPC value 3 is not utilised.

The decision which command to send to the UE is done based on a comparison between Normalized Received (NRX) power and Target Received (TRX) power. The NRX power is obtained from the Received Signal Strength Indicator (RSSI), the TRX power on is constant value loaded from the configuration file. The TRX power is also the parameter, which is modified based on the input from the MEC.

To do so, the global variable `control_tpc_mec` in OAI is added. In the future, this should be rewritten to use the internal messaging tool. The demonstration with a global variable,

accessible from both MEC and scheduler, is good for test purposes as this very straightforward solution is easy to control and debug. On the other hand, it does not separate these two parts of the platform, which is not desirable.

Three options of power control are integrated into the MEC – Received (RX) power increase, RX power decrease and reset to the original value. The step is set to 5 dBm in function `void schedule_ulsch_rnti(…)`:

```
target_rx_power = mac_xface->get_target_pusch_rx_power(module_idP, CC_id) +
5 * control_tpc_mec;
```

The higher difference was selected to the clearly demonstrate the behaviour of OAI once the command is received, and to observe the process of adjusting the RX power in the system. The MEC command modifies the RX power for all UEs and Component Carries (CC), which represent the Resource Blocks (basic scheduling unit in LTE-A). More complicated structure including UE id and CC id would be necessary to change only one specific device. Currently, the MEH expects only one device connected to the eNB. This simplification is also reflected in data harvesting functions, where only the UE with id 0 is considered.

4.4 Challenge 4: Multi-Access Edge Computing source code

To have a working test set-up, a new MEH must be written and additional functions and procedures added to the OAI platform. The MEH development is easier because it can be designed freely. On the contrary, the intervention into the OAI requires paying extra care to suit existing code and extend it appropriately.

4.4.1 Common characteristics

Proposed system design assumes that the MEH may be located on a different machine than OAI, connected over Ethernet network. Both nodes are accessible through the IP address of its interface. Neither OAI nor the MEH needs to have special interface dedicated only to the MEC, as any current existing interface (its IP address) can be exploited if the selected port is free.

The connection is done via SCTP. The SCTP special mark called Payload Protocol Identifier (PPID) indicates the type of message and is received from the upper layer. For the S1AP, the PPID 18 is assigned; for X2AP the PPID is 27. These values are generally accepted and external tools, for example, packet analysers, can benefit from their presence. For the proposed MEC connection, PPID 100 was chosen, as this number is not assigned to any special service [46].

In the proposed setup, the MEH acts as an SCTP server. This means that it awaits incoming connection from the SCTP client (OAI platform) and then accepts the connection. The exact process of creation and termination of the connection between server

and client is shown in Figure 4.2. The process diagram of communication for both MEH part and the OAI client part shows the Figure 4.3.

The whole system is currently designed as a request-oriented: the MEH sends requests, and the OAI client responds. The response can be either provision of requested value or acknowledgement of updating internal parameters.

For the sake of simplicity, some code parts in the following sections were replaced with an ellipsis ("…"), or keyword "error" and only important components are shown.

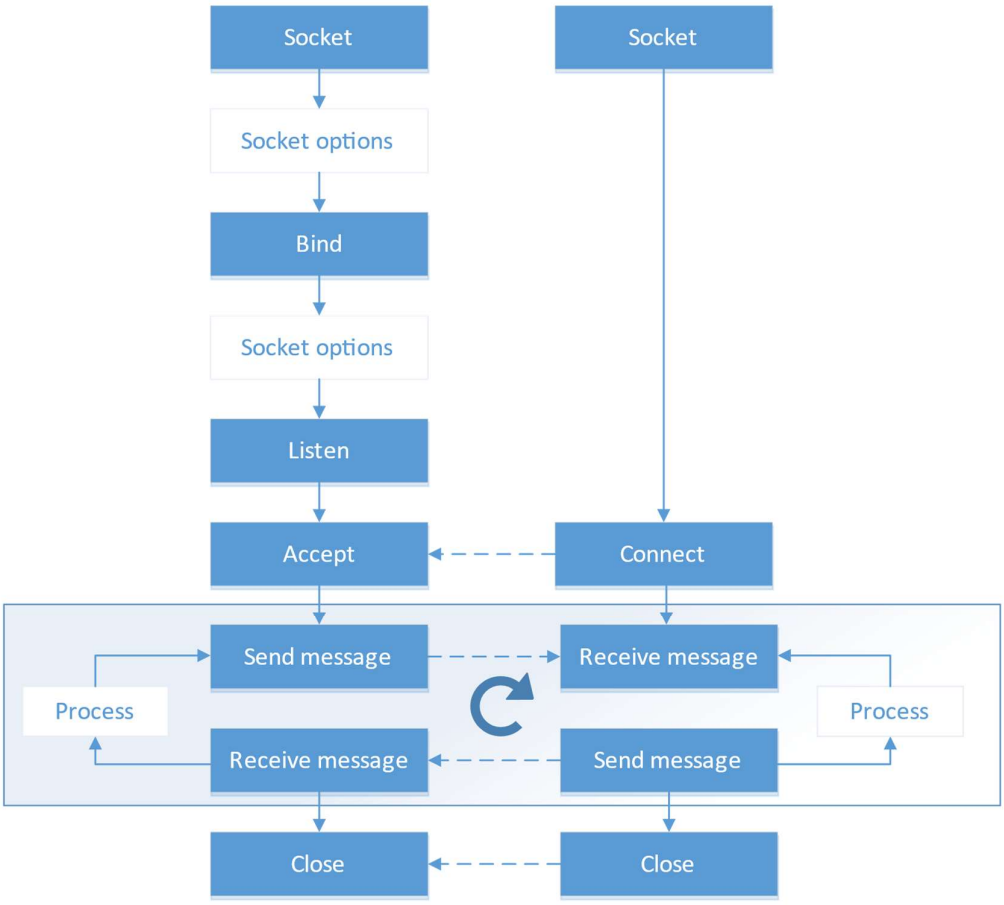

Figure 4.3 SCTP process flow diagram.

4.4.2 Multi-Access Edge Host – server part

The server part of the MEC in proposed design controls the whole MEC system – it is capable of sending commands to the eNB or request values from the eNB once the connection is established.

Before accepting the connection request from the client, several operations have to be done. First of all, the socket has to be created. For this purpose serves the function `int socket(…)`, which creates the socket endpoint and returns a descriptor of the socket.

When the socket is created, an IP address must be assigned to it. The assignment is done through the function `int bind(…)`. The third step is to mark the socket as a passive socket: marked socket will accept incoming connection requests. A function which set the socket as a passive is `int listen(…)`. The last step is to establish the peer-to-peer connection by calling the function `int accept(…)`. The function creates a new connected socket from a request of the listening socket and returns a new file descriptor. At this point, the connection is established. However, in each mentioned step (as well as in the following steps) an error can occur – then the establishment is not successful and the program terminates.

The initial process of connection establishment is in the proposed system covered in the function `int connect_enb_SCTP(…)`. The input parameters are a port, on which the MEH is listening for incoming connections and address of the selected interface. The schematic structure of this function together with some important steps follows.

```
int connect_enb_SCTP(uint16_t port, char *address) {

...

listenSock = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
if (error) { ... }

...

servaddr.sin_addr.s_addr = inet_addr(address);

...

setsockopt(listenSock, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(int));
if (error) { ... }

bind(listenSock, ...);
if (error) { ... }

...

setsockopt(listenSock,      IPPROTO_SCTP,      SCTP_INITMSG,      &initmsg,
sizeof(initmsg));
if (error) { ... }

listen(listenSock, ...);
if (error) { ... }

...

connSock = accept(listenSock, (struct sockaddr *) NULL, (int *)NULL);
if (error) { ... }
```

```
else { return connSock; }
}
```

One attribute of the SCTP is the multi-homing ability. It supports multiple IP paths to the endpoint. If the machine has more active interfaces with assigned IP addresses, all of them can be used for the connection establishment. For testing purposes and analysis, only one address is used. This is the reason for passing one exact address to the function instead of utilising all available addresses through INADDR_ANY.

To avoid the error on binding, the socket is marked with flag SO_REUSEADDR allowing the socket to be bound to a socket address which is already used. Since the MEH is the only one who is using this current socket, no thread of data reception from other service comes.

The second `setsockopt(…)` function then specifies the initial message parameters: number of input and output streams and the maximum number of connection attempts. This setting is valid only for the SCTP protocol – IPPROTO_SCTP parameter is thus used. In the previous use of this function, the socket layer itself was referred by SOL_SOCKET.

Once the connection has been established, a loop for requests and commands takes place. Because during the simulation OAI does not run in real time, it is impossible to request the values and send commands periodically in desired time interval related to the eNB clock. The internal clock of the computer can be used, but this time reference will not match the time in OAI, where special functions are designed to provide a time references. The "speed" of time in OAI depends on the computational power of the hosting machine. The solution for this issue would be sending a message from OAI to the MEH in the exact time period. However, this requires studying of time estimation process in OAI. Until that, the system expects a user input through the terminal. It can also be easily modified to send commands in the selected time period if the test environment (computational power) will not vary during the time. This period must be set properly to correspond OAI internal time stamp.

The function `int listen_enb_sctp(…)` receives and prints the eNB response. Currently, it is outputted only to the terminal, but the string containing the desired data can be transformed into an integer and processed afterwards. The function overview is below:

```
int listen_enb_sctp(int connSock) {

...

sctp_recvmsg(connSock, buffer, sizeof(buffer), (struct sockaddr *) NULL, 0,
&sndrcvinfo, &flags);
if (error) { ... }
else
  {
  printf("%d bytes of data recieved:\n", ret);
  printf(" %s\n", buffer);

if (error) {
```

```
  close(connSock);
         return -1;
  }

...

}
```

The whole function is created to read the data from the provided socket as a parameter (`recvmsg()` function), put them to the buffer and print them if the reception passed off without any problems; otherwise, the function closes the socket and returns -1. The further reception is not possible, and the program terminates.

It is important to clear the buffer before its usage with the procedure `bzero()`. Even though the buffer is created within the function and thus its address is discarded once the function ends, the value stored at this address is not overwritten and causes trouble if the buffer is allocated again with the same address.

The procedure `command_enb_sctp(…)` sends the data passed in the buffer to the provided socket and tags the message with the assigned PPID. This procedure also handles errors which can occur during the process of message sending:

```
void command_enb_sctp(int connSock, int ppid, char *buffer) {

...

sctp_sendmsg(connSock, (void *)buffer, (size_t)strlen(buffer), NULL, 0,
htonl(ppid), 0, 0, 0, 0);
if (error) { ... }
else { ... }

}
```

The main function of MEH first calls the function `connect_enb_SCTP(…)` to prepare the connection and then awaits the eNB connection and initial message which is sent from the eNB and indicates that the connection has been established on both sides. Then continues with an infinite loop, where the user enters the command and – after the newline removal – the function passes the command to the function `command_enb_sctp(…)` and waits for the eNB's response though `listen_enb_sctp(…)`. If the process is successful, the loop starts again:

```
 main()
{
...

sd_enb = connect_enb_SCTP(...);
listen_enb_sctp(sd_enb);                    //initial message

...

while (1) {

 bzero(buffer, ...);
```

```
 printf("Enter command number: ");
 fgets(buffer, ..., stdin);
 buffer[strcspn(buffer, "\r\n")] = 0;
 command_enb_sctp(sd_enb, ..., buffer);

 listen_enb_sctp(sd_enb, ...);
 if (error)    {
               ...
               close(sd_enb);
               exit(1);
       }
 }
}
```

4.4.3 OpenAirInterface – client part

The client carries out the tasks received from the server. The task can be a request for a value or command to change a parameter.

The initial process of connection in OAI to the MEH is caused by function `int connect_mec_SCTP(…)` which returns the file descriptor for the MEC socket. Firstly, the socket is created; after that the connection to the remote peer (MEH) as specified in function call `connect(…)` follows:

```
int connect_mec_SCTP(uint16_t port, char *address) {

...

connSock = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP);
if (error) {...}

...

connect(connSock, (struct sockaddr *) &servaddr, sizeof(servaddr));
if (error) { ... }

return connSock;
}
```

The second function in the OAI MEC client serves as the initial data transmission. It indicates whether the transmission is successful. The function is implemented as follows:

```
int welcome_mec_sctp(int connSock, int ppid) {

...

sctp_sendmsg(connSock,   (void   *)buffer,   (size_t)datalen,   NULL,   0,
htonl(ppid), 0, 0, 0, 0);
bzero(buffer, ...);
if (error) { ... }

else
printf("Successfully sent %d bytes data to server\n", ...);
```

```
return 0;
}
```

The function `listen_mec_sctp(…)` watches the socket for any incoming messages. If no error occurs during the reception, the buffer is separated using the function `atoi(…)` and returned. This is due to the fact that the buffer is a string:

```
int listen_mec_sctp(int connSock) {

...

sctp_recvmsg(connSock, buffer, sizeof(buffer), (struct sockaddr *) NULL, 0,
&sndrcvinfo, &flags);

if (error) { ... }

else
 {
        printf("%d bytes of data recieved:\n", ret);
        command = atoi(buffer);
        printf(" Command : %s\n", buffer);
        bzero(buffer, ...);
        return command;
 }
}
```

The most important function on the client side is `int process_mec_sctp(…)` where the command code is evaluated and processed. First, the case structure is used to execute the proper function and put the response into the buffer. In the second part, the buffer is returned to the MEH. The response is a plain numeric value for data request or a word string for the commands:

```
int process_mec_sctp(int connSock, int ppid, int command) {

...

switch (command) {
 default:
        printf("detected unknown request!\n");
        strncpy(buffer, "error", 5);
        buffer[strlen(buffer)] = '\0';
        break;
 case 1:
        printf("detected PL request\n");
        sprintf(buffer,     "%d",     get_PL((*PHY_vars_UE_g[0])->Mod_id,
(*PHY_vars_UE_g[0])->CC_id, 0));
        break;
 case 2:
        printf("detected TPC request\n");
        sprintf(buffer,   "%d",   mec_get_tpc((*PHY_vars_UE_g[0])->Mod_id,
0));
        break;
 case 3:
        printf("detected GAIN request\n");
```

```
        sprintf(buffer,  "%d",  get_rx_total_gain_dB((*PHY_vars_UE_g[0])-
>Mod_id, 0));
        break;
 case 4:
        printf("detected TPC INCREASE request\n");
        control_tpc_mec++;
        strncpy(buffer, "TPC increased", 13);
        buffer[strlen(buffer)] = '\0';
        break;
 case 5:
        printf("detected TPC DECREASE request\n");
        control_tpc_mec--;
        strncpy(buffer, "TPC decreased", 13);
        buffer[strlen(buffer)] = '\0';
        break;
 case 6:
        printf("detected TPC RESTORATION request\n");
        control_tpc_mec = 0;
        strncpy(buffer, "TPC restored", 13);
        buffer[strlen(buffer)] = '\0';
        break;
 }

sctp_sendmsg(connSock,  (void  *)buffer,  (size_t)datalen,  NULL,  0,
htonl(ppid), 0, 0, 0, 0);

if (error) { ... }

...

}
```

5 Further and open research challenges

During the analysis of the possible MEC implementation and its later demonstration, several further challenges appeared. None of the mentioned open challenges poses an obstacle for the use of the proposed design. The reason for describing the open challenges is to define tasks to enable merge of the developed MEC implementation into the OAI. Due to the time required to merge proposed code into the OAI, it is not possible to start the merge process during the work on this thesis. Furthermore, the code for merge request must be revised by a group of OAI developers. Note that the open challenges do not impact the functionality of implemented MEC solution in any way.

5.1 Data routes

The provided MEC exploits global variables for communication, which should be replaced by the ITTI for the communication between the tasks. The MEC acts as an additional component and, thus, should use the messaging tool as its only communication option.

In order to do this, the exact functionalities of MEC need to be defined. After that, the message list is created and implemented into the all tasks, not only a MEC task. The commands are then sent through the ITTI messages, avoiding global variables as introduced in the proposed concept. The data to the MEH are passed using the ITTI and the SCTP task, not directly from the MEC task.

In the implemented MEC design, the UEs are served only by one MEH server, meaning that mobility of connected UEs is limited to one cell. To extended mobility, the communication model for MEC handover must be designed and implemented. The mobility aspects are discussed in [12].

5.2 Data transfer between the client and the server

The SCTP protocol is used to create a connection between the client (OAI) and the server (MEH). The protocol choice is valid; however, several further changes are not yet implemented.

- Data ciphering - In the present version of MEC, all information is transmitted in plain text. Additional protection to secure the connection is appreciated. OAI already uses a system to secure messages over the S1-C connection; the similar process should be done on the MEC connection. Another alternative is to create an encrypted tunnel connection.
- Streams settings - The SCTP protocol allows its users to use the number of streams, which are negotiated during the initial setup. The user messages are associated with streams numbers [45]. In OAI, stream 0 is used for non-UE associated signalling and streams with a higher number for specific UEs. There is no need to use streams in the current state since only one UE is expected to be

connected, but if more UEs use the MEC, the involvement of stream is highly beneficial.

- PPID - In the present day, there is no PPID associated with MEC. Once the PPID is assigned to the MEC by the assigning authority, it is necessary to change it also in MEC code. The proposed design uses the unassigned PPID 100.

5.3 OpenAirInterface client extension

The future versions of MEH implementation should be improved in several aspects, which are not yet solved:

- The MEC setting, especially IP address and port number, should be loaded from the configuration file in OAI,
- Proper termination of MEC task if an error occurred; sending a notification the peer in OAI,
- Extending the range of supported functions collecting and altering parameters,
- The source code separation into its files.

The work with MEC will be more comfortable if the open OAI extension challenges are solved. However, none of them is crucial.

5.4 Multi-Access Edge Host extension

Also, the server application of MEC system deserves to be more complex. Current MEH is a simple machine to demonstrate the main functionality of the MEC concept. It is possible to extend current code or rewrite brand new MEH exploiting the existing connection with OAI client with respect to the request oriented design. Another alternative is to redesign also the SCTP messages scheme to be able to transport more complicated structures (e.g. UE id, frame and subframe number together with the desired parameter) in one single message.

6 Multi-Access Edge Computing implementation demonstration

In this chapter the setup of the MEC testbed and evaluation is described. Both simulation and real hardware are used during the testing. The set-up and acquired results are presented in this chapter.

6.1 Environment setting

For both evaluation via simulation and testbed, two computers as described in Table 2.1 are exploited to run the RAN and the CN parts. On the computer with the CN, which is run as a virtual machine for the MEC is run. The computers are interconnected via 1 Gbit/s link, which makes the delay between the RAN and the MEC negligible. For the simulation, the PHY abstraction was turned off to receive as accurate results as possible.

For the emulation, the Universal Software Radio Peripheral (USRP) B210 board by Ettus Research is used as a Software Defined Radio in combination with Huawei P8 lite smartphone acting as a UE.

The eNB and CN parameters setting is the same for both experiments; however small changes had to be done due to the usage of the real HW. In all measurements, only one UE (simulated or real) is attached to the created LTE network. OAI v0.6.1 master branch was used for all measurements.

Table 6.1 Testbed specifications.

Specification	**PC1**	**PC2**
CPU	Intel® Core™ i5-4590 3.4 GHz, 4C/4T	Intel® Core™ i5-7400 3.0 GHz, 4C/8T
RAM	8 GB	32 GB
Connectivity	1 Gbit/s LAN	1 Gbit/s LAN
OS	Ubuntu 14.04 with 3.19 low latency kernel	Ubuntu 14.04 with 3.19 generic kernel

Specification	**EPC + MEC VM**	**RAN**
CPU	Dual-core 3.0 GHz	Intel® Core™ i5-4590 3.4 GHz, 4C/4T
RAM	4 GB	8 GB
Connectivity	1 Gbit/s LAN	1 Gbit/s LAN
OS	Ubuntu 14.04 with 4.7.7 generic kernel with GTP kernel support	Ubuntu 14.04 with 3.19 low latency kernel

6.2 Data harvesting

The MEH user interface is shown in Figure 6.1. When the request is detected in the eNB, command ID and its name is printed before sending the response back to the MEH, as shown in Figure 6.2. At the client side, the MEC log is printed directly to the terminal

together with other OAI logs. All three currently supported parameters, as described in Subsection 4.3.1, were tested.

The request-oriented system is able to obtain the desired parameters from the eNB with the virtualised UE by sending the pre-defined command's ID. When the SDR board is used together with the real UE, a segmentation fault appears once the function gathering the desired data is called. This issue is now being analysed with other OSA members.

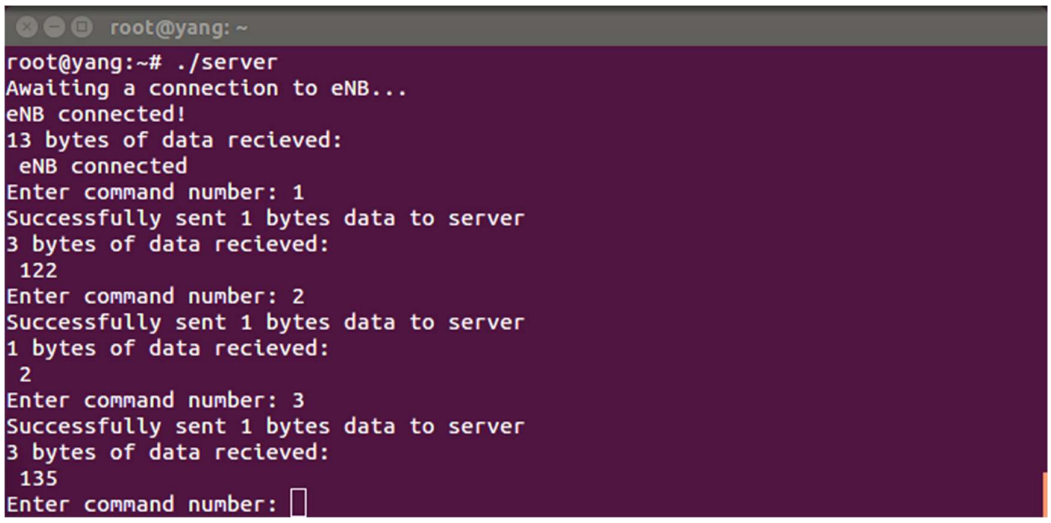

Figure 6.1 MEH user interface: data harvesting.

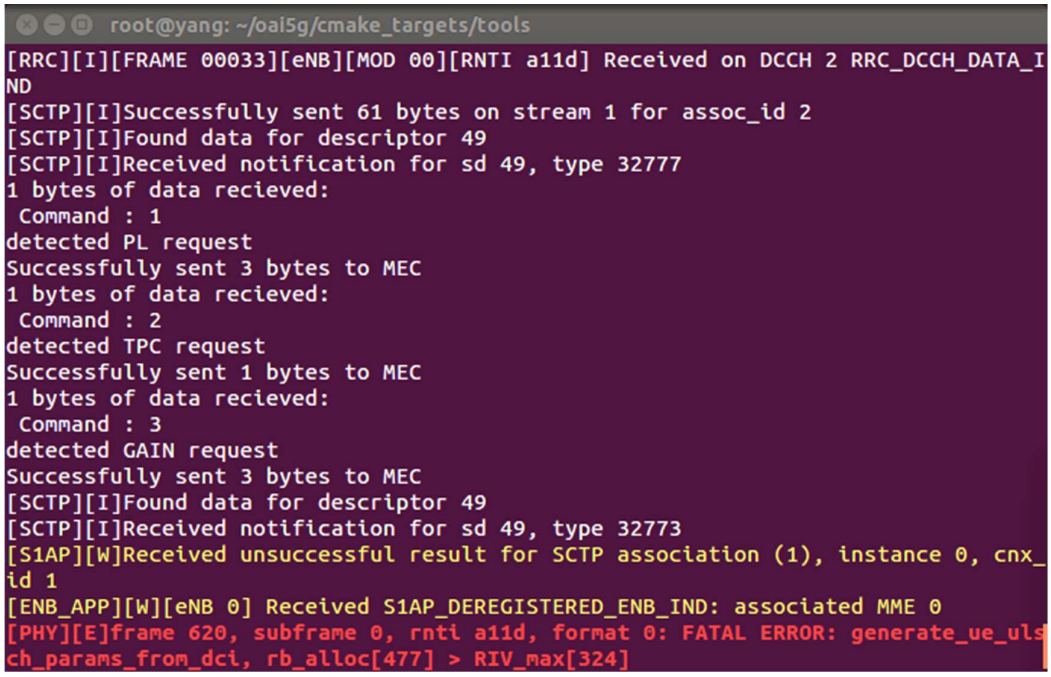

Figure 6.2 OAI user interface: data harvesting.

6.3 eNodeB control via Mobile Edge Host

The demonstration of the eNB control via the MEC is demonstrated on the P0 nominal Physical Uplink Shared Channel (PUSCH) parameter, which defines the target power level of a received resource block (RB) at the eNB. The request to increase or decrease the Target RX power is sent from the MEH to the eNB, where its internal scheduler decides how to modify the transmitting power of connected UE.

In the case of control commands, the response from the eNB is the confirmation whether the requested modification is processed successfully. The screenshot of MEH and OAI shows the Figure 6.3 and Figure 6.4.

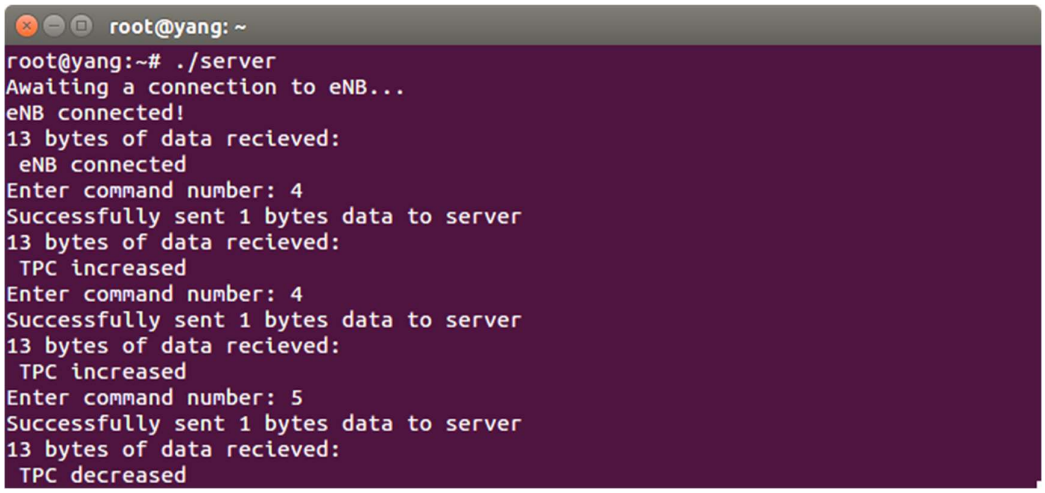

Figure 6.3 MEH user interface: eNB control.

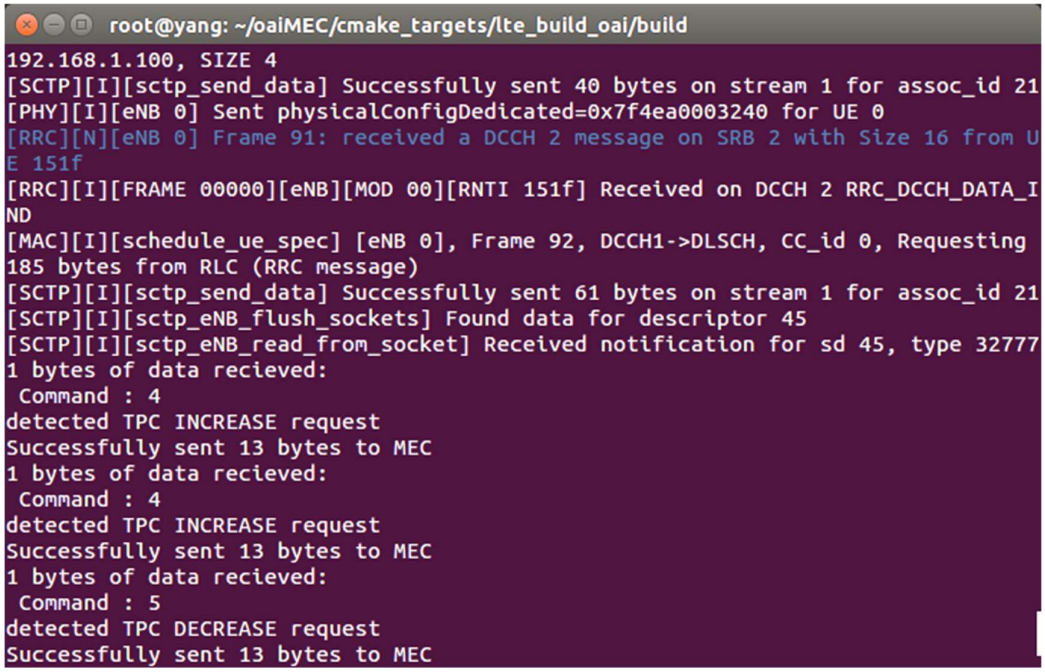

Figure 6.4 OAI user interface: eNB control.

The processes of the transmit power modification in dependency on time and TRX for simulated and real UE are shown in Figure 6.5 and in Figure 6.6, respectively. It is evident that the algorithm for PHY simulation works differently compared to the real testbed. Pleasingly the results gained from the real HW shows lower RSSI fluctuation.

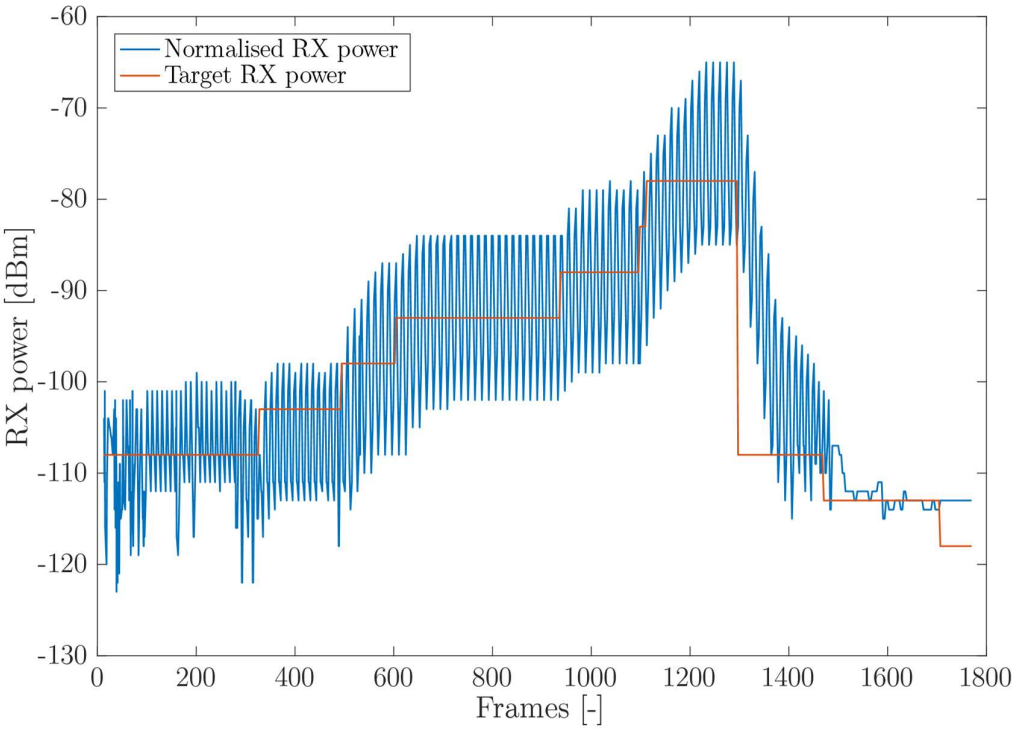

Figure 6.5 Comparison of Target and Normalised RX power: simulated UE.

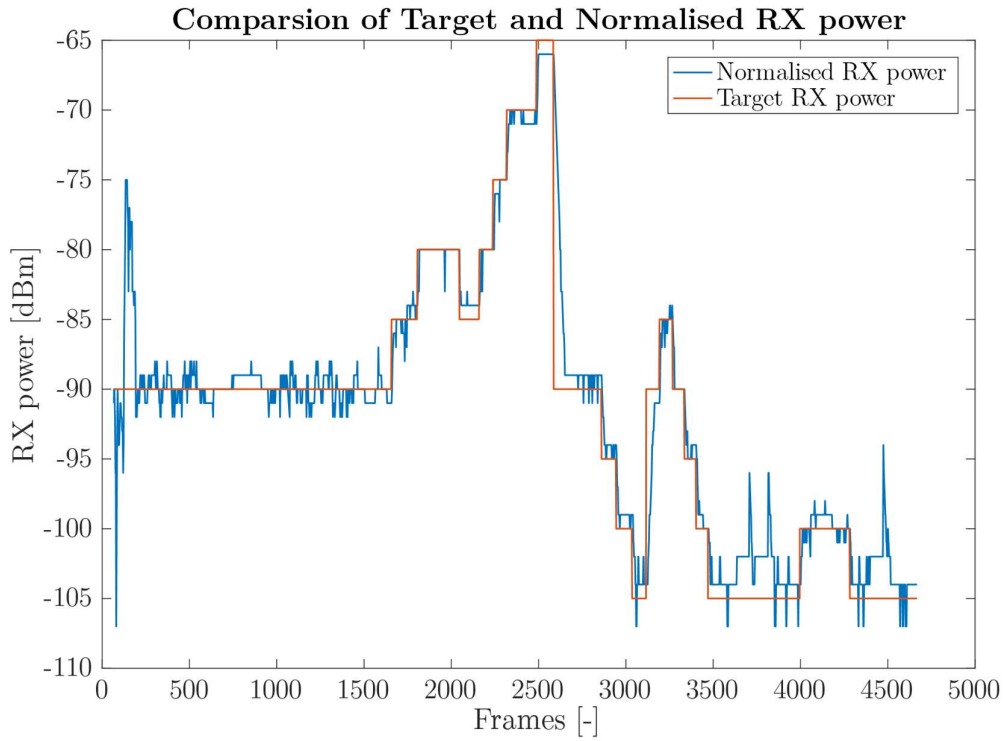

Figure 6.6 Comparison of Target and Normalised RX power: real UE.

7 Conclusion

Mobile network optimisation exploiting Multi-Access Edge Computing is a highly effective way how to satisfy the quickly developing mobile networks market. Even though the MEC concept is in its early era and it is still being developed, a number of papers in this area already exist.

To be able to capitalise the MEC, the MEC concept has been implemented into the OAI platform. During the consecutive testing, it has been proven that the MEC implementation is working and eNB follows the commands from the MEH. The provided implementation is to be submitted to the community of the OAI with a goal to merge it to the OAI code.

Current status of the MEC implementation gives a vast range of opportunities for further employment in various situations. Working groups can build their solutions on the presented implementation and extend the MEC exploitation to the areas of interest. All of this brings the MEC deployment to the real mobile networks closer.

Although focused on the MEC, this thesis also provides valuable information about implementation of a new task to the OAI platform. The process of creation and maintaining is the same for all threads in the OAI, not just limited to the MEC; relevant chapters can be thus helpful for the further OAI expansion.

The future work for this thesis consists in completing the outlined open challenges and extending the MEC implementation by the support of multiple MEHs communicating to optimise the mobile network on the larger area.

8 References

[1] ETSI MCC department, "Overview of 3GPP Release 10 V0.2.1", Sophia Antipolis Cedex, France, 2014.

[2] S. Mattisson, "Overview of 5G requirements and future wireless networks", *ESSCIRC 2017 - 43rd IEEE European Solid State Circuits Conference*, pp. 1-6, 2017.

[3] S. Lien, S. Hung, D. Deng and Y. Wang, "Efficient Ultra-Reliable and Low Latency Communications and Massive Machine-Type Communications in 5G New Radio", *IEEE Global Communications Conference*, pp. 1-7, 2017.

[4] S. Qureshi, T. Ahmad, K. Rafique and Shuja-ul-islam, "Mobile cloud computing as future for mobile applications - Implementation methods and challenging issues", *IEEE International Conference on Cloud Computing and Intelligence Systems*, pp. 467-471, 2011.

[5] ETSI Industry Specification Group, "ETSI GS MEC 001: Mobile Edge Computing (MEC); Terminology", Sophia Antipolis Cedex, France, 2016.

[6] N. Nikaein, M. Marina, S. Manickam, A. Dawson, R. Knopp and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33-38, 2014.

[7] B. Bilel, N. Navid, K. Raymond and B. Christian, "OpenAirInterface large-scale wireless emulation platform and methodology", *ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks - PM2HW2N '11*, pp. 109-112, 2011.

[8] K. Kumar, J. Liu, Y. Lu and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems", *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129-140, 2013.

[9] A. Huang, N. Nikaein, T. Stenbock, A. Ksentini and C. Bonnet, "Low latency MEC framework for SDN-based LTE/LTE-A networks", *IEEE International Conference on Communications (ICC)*, pp. 1-6, 2017.

[10] Y. Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, "Mobile Edge Computing: A key technology towards 5G", *ETSI White Paper No. 11*, 2015.

[11] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading", *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628-1656, 2017.

[12] ETSI Industry Specification Group, "ETSI GR MEC 018: Mobile Edge Computing (MEC); End to End Mobility Aspects", Sophia Antipolis Cedex, France, 2017.

[13] Y. Mao, C. You, J. Zhang, K. Huang and K. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective", *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, 2017.

[14] M. Deng, H. Tian and B. Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks", *IEEE International Conference on Communications Workshops (ICC)*, pp. 638-643, 2016.

[15] M. Barbera, S. Kosta, A. Mei and J. Stefa, "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing", *IEEE INFOCOM*, pp. 1285-1293, 2013.

[16] Y. Kao, B. Krishnamachari, M. Ra and F. Bai, "Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing", *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056-3069, 2017.

[17] 3rd Generation Partnership Project, "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification (Release 10)", Sophia Antipolis Cedex, France, 2010.

[18] V. Jacobson, R. Braden and D. Borman, "RFC 1323 - TCP Extensions for High Performance". 1992.

[19] J. Plachy, Z. Becvar and P. Mach, "Path selection enabling user mobility and efficient distribution of data for computation at the edge of mobile network", *Computer Networks*, vol. 108, pp. 357-370, 2016.

[20] ETSI Industry Specification Group, "ETSI GS MEC-IEG 004: Mobile - Edge Computing (MEC); Service Scenarios", Sophia Antipolis Cedex, France, 2015.

[21] C.-L. I, J. Huang, R. Duan, C. Cui, J. Jiang and L. Li, "Recent Progress on C-RAN Centralization and Cloudification", *IEEE Access*, vol. 2, pp. 1030-1039, 2014.

[22] F. Hu, Q. Hao and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation", *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 2181-2206, 2014.

[23] H. Hawilo, A. Shami, M. Mirahmadi and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)", *IEEE Network*, vol. 28, no. 6, pp. 18-26, 2014.

[24] X. Wang, M. Chen, T. Taleb, A. Ksentini and V. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5G systems", *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131-139, 2014.

[25] I. Farris, T. Taleb, M. Bagaa and H. Flick, "Optimizing service replication for mobile delay-sensitive applications in 5G edge network", *IEEE International Conference on Communications (ICC)*, pp. 1-6, 2017.

[26] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges", *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646, 2016.

[27] C. Wang, Z. Lin, S. Yang and P. Lin, "Mobile edge computing-enabled channel-aware video streaming for 4G LTE", *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 564-569, 2017.

[28] S. Yi, C. Li and Q. Li, "A Survey of Fog Computing", *Workshop on Mobile Big Data - Mobidata*, pp. 37-42, 2015.

[29] A. Ahmed and E. Ahmed, "A survey on mobile edge computing", *International Conference on Intelligent Systems and Control (ISCO)*, pp. 1-8, 2016.

[30] "FP7-ICT: Distributed computing, storage and radio resource allocation over cooperative femtocells", TROPIC project, 2012, [Online], Available: https://cordis.europa.eu/project/rcn/105469_en.html, [Accessed: 2018-02-12].

[31] S. Wang, G. Tu, R. Ganti, T. He, K. Leung, H. Tripp, K. Warr and M. Zafer, "Mobile Micro-Cloud: Application Classification, Mapping, and Deployment", *Annual Fall Meeting of ITA (AMITA)*, 2013.

[32] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe and K. Webb, "MobiScud", *Workshop on All Things Cellular: Operations, Applications and Challenges - AllThingsCellular*, pp. 19-24, 2015.

[33] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks", *IEEE Network*, vol. 27, no. 5, pp. 12-19, 2013.

[34] J. Liu, T. Zhao, S. Zhou, Y. Cheng and Z. Niu, "CONCERT: a cloud-based architecture for next-generation cellular systems", *IEEE Wireless Communications*, 2014.

[35] ETSI Industry Specification Group, "ETSI GS MEC 003: Mobile Edge Computing (MEC) ; Framework and Reference Architecture", Sophia Antipolis Cedex, France, 2016.

[36] "OpenAirInterface: 5G software alliance for democratising wireless innovation", 2018, [Online], Available: http://www.openairinterface.org/, [Accessed: 2018-02-12].

[37] E. Dahlman, S. Parkvall and J. Sköld, *4G: LTE/LTE-advanced for mobile broadband.* Amsterdam: Academic Press, 2011.

[38] "openairinterface5G: Openairinterface 5G Wireless Implementation", *GitLab.* 2018.

[39] I. Latif, F. Kaltenberger, R. Knopp and N. Nikaein, "Large scale system evaluations using PHY abstraction for LTE with OpenAirInterface", *International ICST Conference on Simulation Tools and Techniques*, pp. 1-7, 2013.

[40] N. Nikaein, R. Knopp, F. Kaltenberger, L. Gauthier, C. Bonnet, D. Nussbaum and R. Ghaddab, "OpenAirInterface: an open LTE network in a PC", *International conference on Mobile computing and networking - MobiCom '14*, pp. 305-308, 2014.

[41] "Open vSwitch", 2016. [Online]. Available: https://www.openvswitch.org/. [Accessed: 2018-04-17].

[42] "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks", *Edinburgh Networks Research Group.* Edinbourgh, United Kingdom, 2018.

[43] X. Foukas, N. Nikaein, M. Kassem and K. Kontovasilis, "FlexRAN: A flexible and programmable platform for software-defined radio access networks", *International on Conference on emerging Networking EXperiments and Technologies*, ACM, 2016.

[44] J. Postel, "RFC0768 - User Datagram Protocol". 1980.

[45] R. Stewart, "RFC4960 - Stream Control Transmission Protocol". 2007.

[46] "SCTP Payload Protocol Identifiers", *Stream Control Transmission Protocol (SCTP) Parameters*. 2018.