UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Implementation of a Follow-me Multi-access Edge Computing (MEC) scheme for 5G networks in Kubernetes

Diploma Thesis

Theodoros Tsourdinis

Supervisor: Athanasios Korakis

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Implementation of a Follow-me Multi-access Edge Computing (MEC) scheme for 5G networks in Kubernetes

Diploma Thesis

Theodoros Tsourdinis

Supervisor: Athanasios Korakis

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υλοποίηση μηχανισμών Follow-me Multi-access Edge Computing (MEC) σε δίκτυα 5ης γενιάς σε περιβάλλον Kubernetes

Διπλωματική Εργασία

Θεόδωρος Τσουρδίνης

Επιβλέπων/πουσα: Αθανάσιος Κοράκης

Βόλος 2021

Approved by the Examination Committee:

Supervisor **Athanasios Korakis**
Associate professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Antonios Argyriou**
Associate professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Bargiotas**
Associate professor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 15-7-2021

Acknowledgements

There are many who helped me along the way on this journey. I want to take a moment to thank them.

First, I would like to express my sincere gratitude to my supervisor, Mr. Athanasios Korakis for the trust that showed in me and his faith in my potential, by inspiring my interest in remarkable research topics. I would also like to extend my deepest gratitude and especially thank the postdoctoral Nikos Makris for his consistent support and guidance. His immense knowledge played a huge role in completing my dissertation.

I am extremely grateful to my family and my friends. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study. Last but not least, I would like to thank my life companion Olympia, for all her love and support.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Theodoros Tsourdinis
15-7-2021

Abstract

The advent of 5G in cellular networks has opened new horizons in technological advances such as Vehicle-to-Vehicle communication (V2V), Augmented Reality (AR), and Virtual Reality (VR). However, the above applications have several requirements in terms of latency (Ultra-Reliable Low Latency Communication - uRLLC) and throughput (Enhanced Mobile Broadband - eMBB). These requirements have brought new challenges in the design and implementation of network infrastructure schemes that are capable of managing the needs of such applications.

The integration of edge computing in the 5G network, seems to cover the needs of the above applications, since the services are closer to the end-user, offering high QoS / QoE perfomance. However, the challenges continue as the key feature of cellular networks is user mobility. Therefore live migrations of services are necessary, with the sole aim of low latency. Equally necessary is the management of edge services and consequently the management of network components of the 5G network.

The basis for managing the above network infrastructure schemes is to have a flexible core to edge cloud implementations and elastically scalable capacity for on-demand requirements. A container orchestrator like Kubernetes meets all of the above requirements since with Kubernetes we can rapidly deploy clusters and services in minutes, manage network functions running in containers, and scale-out capacity to meet peak demands. In this thesis, we manage to deploy an open-source implementation of a mobile network using the OpenAirInterface platform with the support of 3GPP (LTE) and non-3GPP (WiFi) technologies in the Kubernetes environment and implement a Follow-me Multi-access Edge Computing (MEC) scheme, all in the Kubernetes ecosystem.

All the experiments were conducted on real experimental infrastructure - testbed (Nitos Testbed) which consists of computers equipped with devices necessary for the operation of an LTE / 5G network down from the end user, up to the Core Network, with the use of Software

Defined Radios and commercial off-the-shelf equipment.

Περίληψη

Η έλευση του 5G σε κυψελοειδή δίκτυα άνοιξε νέους ορίζοντες στην τεχνολογική πρόοδο όπως η επικοινωνία μεταξύ οχημάτων (V2V), η επαυξημένη πραγματικότητα (AR) και η εικονική πραγματικότητα (VR). Ωστόσο, οι παραπάνω εφαρμογές έχουν αρκετές απαιτήσεις όσον αφορά τον χρόνο καθυστέρησης και την ταχύτητα. Αυτές οι απαιτήσεις έφεραν νέες προκλήσεις στο σχεδιασμό συστημάτων υποδομής δικτύου που είναι ικανός να διαχειριστεί τις ανάγκες αυτές.

Η ενσωμάτωση της τεχνολογίας Edge στο δίκτυο 5G, φαίνεται να καλύπτει τις ανάγκες των παραπάνω εφαρμογών, καθώς οι υπηρεσίες είναι πιο κοντά στον χρήστη, προσφέροντας υψηλά πρότυπα στην ποιότητα υπηρεσιών και εμπειρίας. Ωστόσο, οι προκλήσεις συνεχίζονται καθώς το βασικό χαρακτηριστικό των κυψελοειδών δικτύων είναι η κινητικότητα των χρηστών. Ως εκ τούτου, είναι απαραίτητες οι ζωντανές μετεγκαταστάσεις υπηρεσιών, με μοναδικό στόχο τη χαμηλή καθυστέρηση των δεδομένων. Εξίσου απαραίτητη είναι η διαχείριση των edge υπηρεσιών και κατά συνέπεια η διαχείριση των στοιχείων του δικτύου του δικτύου 5G.

Η βάση για τη διαχείριση των παραπάνω συστημάτων υποδομής δικτύου είναι να υπάρχει ένας ευέλικτος πυρήνας για την διαχείρηση των edge-cloud στοιχείων και μια ελαστική ικανότητα επέκτασης των στοιχείων αυτών. Ένας ενορχηστρωτής container όπως ο Kubernetes πληροί όλες τις παραπάνω προϋποθέσεις, καθώς με το Kubernetes μπορούμε να αναπτύξουμε γρήγορα συστοιχίες container και υπηρεσίες σε λίγα λεπτά, να διαχειριστούμε τις λειτουργίες δικτύου που εκτελούνται σε container και να επεκτείνουμε την ικανότητα να ανταποκριθούμε στις μέγιστες απαιτήσεις.

Σε αυτή τη διατριβή, καταφέρνουμε να υλοποιήσουμε ένα mobile network χρησιμοποιώντας την πλατφόρμα ανοιχτού κώδικα OpenAirInterface με την υποστήριξη τεχνολογιών 3GPP (LTE) και μη 3GPP (WiFi) στο περιβάλλον Kubernetes εφαρμόζοντας ένα Follow-me Multi-access Edge Computing σχήμα, όλα στο οικοσύστημα Kubernetes. Όλα τα πειράματα

διεξήχθησαν σε πραγματική πειραματική υποδομή - testbed (Nitos Testbed) που αποτελείται από υπολογιστές εξοπλισμένους με συσκευές απαραίτητες για τη λειτουργία ενός δικτύου LTE / 5G από τον τελικό χρήστη, μέχρι την ραχοκοκαλιά του δικτύου, με την χρήση των Software Defined Radios συσκευών.

Table of contents

List of figures

List of tables

Abbreviations

5G	5th Generation
AR	Augmented Reality
URLLC	Ultra-Reliable Low-Latency Communications
eMBB	Enhanced Mobile Broadband
QoS	Quality of Service
QoE	Quality of Experience
3GPP	3rd Generation Partnership Project
4G	4th Generation
LTE	Long-Term Evolution
MEC	Multiple-Access Edge Computing
NITOS	Network Implementation using Open-Source software
USRP	Universal Software-Defined Radio Peripherals
RAN	Radio Access Network
3GPP	3rd Generation Partnership Project
3G	3rd Generation
TV	Television
IP	Internet Protocol
2G	2nd Generation
WLAN	Wireless Locan Area Netw3th Generation
OFDM	Orthogonal Frequency Division Multiple Access
MIMO	Multiple Input Multiple Output
FEC	Forward Error Correction
ARQ	Automatic Repeat Request
HARQ	Hybrid Automatic Repeat Request
UE	User Equipment

EUTRAN	Evolved Universal Terrestrial Radio Access Network
EPC	Evolved Packet Core
MME	Mobility Management Entity
P-GW	Packet Data Network Gateway
S-GW	Serving Gateway
PCRF	Policy Control and Charging Rules Function
PDCP	Packet Data Convergence Protocol
RLC	Radio Link Control
PDU	Packet Data Unit
MAC	Media Access Control
PHY	Physical
RRC	Radio Resource Control
NAS	Network-Attached Storage
IoT	Internet of Things
ITU	International Telecommunication Union
mMTC	Massive Machine Type Communications
KPI	Key Performance Indicators
IMT	International Mobile Telecommunications
CSI	Channel State Information
NFV	Network Function Virtualization
CU	Central Unit
DU	Distributed Unit
WiFi	Distributed Unit
F1AP	F1 Application Protocol
V2X	Vehicle to Everything
FH	Fronthaul
BH	Backhaul
RTT	Round Trip Time
RNTI	Radio Network Temporary Identifier
UL	Uplink
DL	Downlink
VM	Virtual Machine

RAT	Radio Access Technology
API	Application Programming Interface
QUEMU	Quick Emulator
WAN	Wide Area Network
VIM	Virtual Infrastructure Manager
VNFM	Virtual Network Function Manager
K8s	Kubernetes
RAM	Random Access Memory
CPU	Central Processing Unit
NFS	Network File System
SSH	Secure Shell
PV	Persistent Volume
PVC	Persistent Volume Claim
NAT	Network Adress Translation
CNI	Container Network Interface
CRD	Custom Resource Definition
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
SIM	Subscriber Identity Module
SDR	Software Defined Radios
OAI	Open Air Interface
RF	Radio Frequency
OAI	Open Air Interface
OAI	Open Air Interface
tc	Traffic Control
SGPGW-U	Serving Gateway and Packet Data Network Gateway User plane
SPGW-C	Serving Gateway and Packet Data Network Gateway Control plane
USB	Universal Serial Bus
I/O	Input/Output
SIP	Session Initiation Protocol
VoIP	Voice Over Internet Protocol
TCP	Transmission Control Protocol

SCTP	Stream Control Transmission Protocol
UDP	User Datagram Protocol
CQI	Channel Quality Indicator

Chapter 1

Introduction

As the fifth generation of mobile network access (5G) rolls out, new applications are coming to target an end-to-end converged ecosystem, hosting high bandwidth, low latency, and massively connected devices. These services come to serve, from the ordinary mobile user, up to companies with specialized manufacturing technologies such as Industry 4.0 [16], paving new avenues in communication between people, devices, and machines, taking factory automation to the next level. Also, based on the high throughput and the extremely low latency provided by the 5G networks, new technologies have been developed that utilize the autonomy of cars allowing communication between them (Vehicle to Vehicle Communication).

In such communications, where data must be sent and received quickly and correctly from one side to the other, reliable network infrastructures and schemes are a must to protect these time-sensitive data. These schemes should not be based on cloud-based solutions, as they contribute to latency due to the geographical distance between the end-user and the data processing centers. Also, another reason cloud computing is not suitable in such schemes is the constant maintenance costs and energy consumption plus the high risk of privacy leakage [17].

As 5G networks go beyond the monolithic architecture of LTE / 4G networks, bringing the architecture components closer to the edge, i.e. closer to the user, distributing the RANs (Radio Access Networks), allows us to shine more towards edge computing for the creation of the above-mentioned network infrastructures. However, with the presence of edge services, the components of the 5G network are increasing and their management requires a kind of orchestration for their proper operation, control, scale, and migration. Therefore, there is a

need for a centralized management platform on which a 5G network must be deployed.

1.1 The subject of Thesis

In this thesis, we will adequately cover all the above problems that mainly concern uRLLC - High Availability, Low Latency use cases in a 5G Disaggregated Architecture network which we deployed in the Kubernetes framework. This way, we take the advantage of the multiple benefits provided by an application container orchestrator like Kubernetes, such as management and monitoring of resources and dynamic scale of 5G network architecture components.

To the above Kubernetes deployment, we added an edge network infrastructure for realizing Multi-access Edge Computing (MEC) [18] that will support the needs of applications for low latency and real-time data management. MEC allows virtualization of services deployed in or close to the network edge. Then we implemented a Follow Me algorithm for live-triggered migration of edge services between different nodes for the services to be available regardless of the spatial mobility of the network end-users.

The containerized deployment relies on the open-source OpenAirInterface platform to which functionality for a Heterogeneous Disaggregated Setup has been added, providing the ability to a network end-user to connect via 3GPP (LTE) and non-3GPP (WiFi) technologies. The entire setup is based on a real-world testbed (NITOS Testbed) providing the appropriate hardware equipment for the operation of Radio Access Networks.

1.2 Thesis Organization

To complete the dissertation, several methodologies led to understand various concepts, identifying problems, and solving them. The contribution of this thesis can be summarized as follows:

In chapter 2 we study the key technologies used by modern cellular networks and the architecture of LTE / 4G & 5G networks. In chapter 3 we introduce Multi Edge Computing and the various related technologies. We also present in detail the scenarios for which the Live Migration of MEC services is necessary and we analyze the implementation of a Follow Me approach in the Kubernetes environment. In chapter 4 we analyze the Kubernetes framework as well as technologies offered by the Kubernetes ecosystem. In chapter 5 we present our experimental setup as well as the experimental results. In the last chapter, we summarize and conclude this thesis and suggest future work.

Chapter 2

Study of LTE/4G & 5G cellular networks

2.1 Introduction to 4G

The fourth generation of wireless standards for cellular systems is 4G. This is the later cellular mobile standard after 3G. The peak requirements for the 4G standard according to the ITU are 100 Mbps for a mobile connection, e.g. in a car, and 1Gbps for stationary connections, i.e. for the use of desktop devices. However, a typical data rate in a 4G system ranges from 20 to 100 Mbps. The 3rd Generation Partnership Project (3GPP) standards group has developed and maintains LTE technology and replaces the 3.75G (HSPA +) and UMTS standards. The reason for the replacement of the previous technologies used in the third generation of mobile network access (3G), was for the transition to a standard that will offer higher data rates and lower latency [19].

The 4G is developed to meet the quality requirements of service (Quality of service-QoS) and requirements set by future applications such as wireless broadband, multimedia services, video chat, high-definition TV, digital video streaming, and other services using high bandwidth. Users can use the above services regardless of time and location. In addition, 4G uses IPV6 technology to support a large number of wireless-enabled devices. It also offers the capability of a heterogeneous scheme as well as the connection to WLAN satellite and 3G systems is achievable. The 4G system, like the 3G, is IP-based (Internet protocol) but more specifically the 4G relies exclusively on all-Internet Protocol (all-IP) therefore unlike 3G, 4G uses IP even for voice data systems (IP telephony).

The band of frequencies used by the 4G ranges between 2-8 GHz while the bandwidth of the channels is 100MHz, taking better advantage of the spectrum compared to the 2G and

3G bands that were limited to the band of 1.8 and 2 GHz respectively.

2.1.1 4G Key Technologies

In order to meet the needs of applications and to offer unique standards in QoS, several technologies were developed in the 4G era. Below we will analyze some of them:

- **Orthogonal Frequency Division Multiple Access – OFDMA**: OFDMA was developed in 4G achieving high spectral performance, supported by a channel distribution scheme. This technique is used to break down the data to be transmitted along with the orthogonal carriers making it possible to reduce intersymbol interference (ISI) which comes from the delay spread of Multi-Path fading and to accomplish a diversity gain that leads to signal-to-interference-plus-noise ratio (SINR) improvement. As shown in figure 2.1 the OFDM symbols are grouped into resource blocks. The resource blocks have a total size of 180 kHz in the frequency domain and 0.5ms in the time domain.

- **Multiple Input Multiple Output – MIMO**: The use of multiple transmission and receiving antennas allows increasing capacity network due to higher data rates and higher number of users served. MIMO systems offer spatial multiplexing and diversity gain. Spatial multiplexing is mainly for achieving high data rate whereas diversity gain is mainly to increase the reliability of the system. As we see in figure 2.2 the reason for getting the improved error performance in MIMO system is just because of the benefit of both array gain and diversity gain [2].

- **Down-Link Adaptation**: Depending on the quality of the signal transmitted to and from a particular user, certain modifications are made to the signal to improve systems quality and coverage reliability. More specifically, if SINR is good then higher Modulation and Coding Schemes (MCS) are used. If the SINR is bad, then lower MCS are used.

- **Hybrid Automatic Repeat Request (hybrid ARQ or HARQ)**: This is a combination of high-rate forward error correction (FEC) and automatic repeat request (ARQ) error-control. Both are error-correcting codes and help reduce SINR.

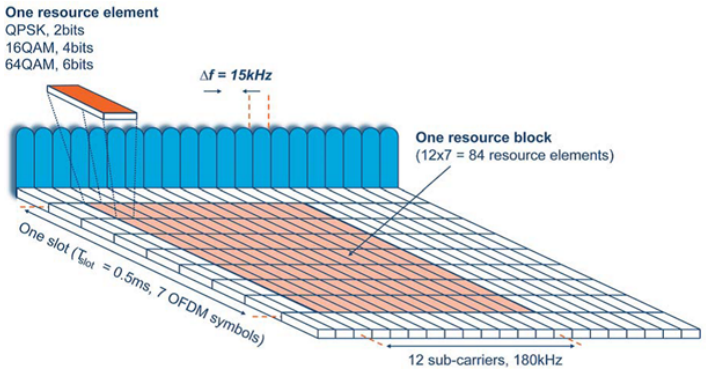

Figure 2.1: LTE downlink resource based on OFDM. [1]

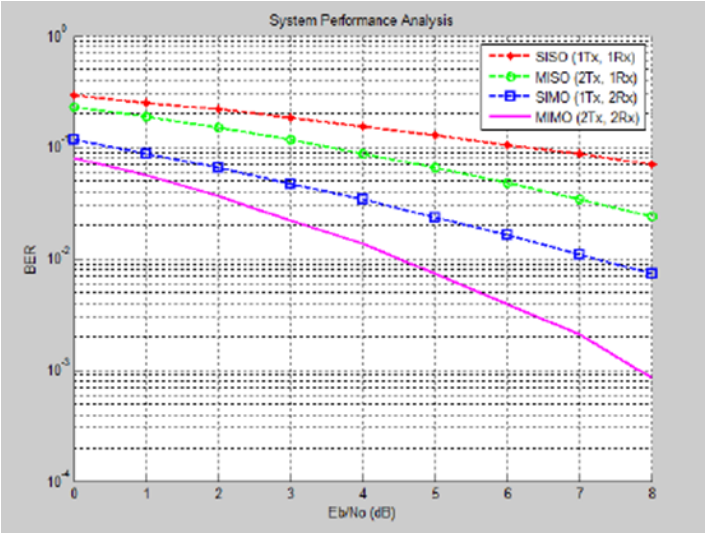

Figure 2.2: Performance (Bit Error Rate) Analysis on SISO , MISO (2x1), SIMO (1x2) and MIMO (2x2). [2]

2.1.2 LTE Architecture

The LTE architecture has three main components:

- **User Equipment (UE)**: This is mobile equipment. It is any device used directly by an end-user to communicate through an LTE network.

- **EUTRAN (Evolved Universal Terrestrial Radio Access Network)**: The E-UTRAN handles the radio communications between the mobile and the EPC. The EUTRAN consists of the evolved base stations, called eNodeB. An eNodeB is a part of the E-UTRAN radio access network and is the component that allows UEs to connect to the LTE network. An eNodeB typically communicates with the UE's through Uu interface, with other eNodeBs through X2 interface, and with the EPC through S1 interface. An

eNodeB performs Radio Control Management where it essentially handles the radio resource management for UEs in both idle and connected modes. It is also responsible for the setup, maintenance, and release of radio bearers and their resource configuration (radio bearer control). In addition, it routes the packets from the user plane towards the S-GW. It is responsible for selecting the MME, as it allows the UE to be served by a different MME while being in a network or while the UE is in the ”attach” procedure.

- **Evolved Packet Core (EPC)**: This is the LTE core network. It is comprised of components that have the following functions: mobility management, authentication, quality of service, routing upload and download IP packets, IP address allocation, and more. The EPC consists of the following:

    1. **Mobility Management Entity (MME)**: The MME is responsible for processing the signals between the UE and the EPC. It also manages the connections between the UE and the core network initially by exchanging authentication information between the UE and the HSS and then by allowing the subscriber's mobility within the network or across networks and keeps track of the subscriber's location updates. It also establishes bearers by deciding on a gateway router to the Internet if there are more gateways available. The session management is provided through S1-MME interface to eNodeB. The mobility management function is provided through S10 interface. The user data plane is controlled by the MME through the S11 interface. The MME is linked through the S6a interface to the HSS which contains all the user subscription information.

    2. **Home Subscriber Server (HSS)**: HSS maintains a central database that contains information about all network operators' subscribers. It is responsible for many functions of many of them include call and session establishment support and user authentication and access authorization. It also maintains profile information that describes service subscription states and user-subscribed Quality of Service information.

    3. **Packet Data Network Gateway (P-GW)**: P-GW is the termination point of the packet data interface towards the Packet Data Network. It is responsible for IP allocation for UEs and filtering of user Down Link IP packets in different QoS bearers. P-GW also handles policy enforcement, charging support, and lawful

interception. P-GW communicates with the Data Packet Networks (PDN) using the SGi interface.

4. **Serving Gateway (S-GW)**: S-GW routes all the user data packets and forwards them between the eNodeB and the PDN gateway. The S-GW also handles mobility and handover between 3GPP networks.

5. **Policy Control and Charging Rules Function (PCRF)**: PCRF is responsible for policy control decision making and charging rules functionality in the Policy Control Enforcement Function (PCEF), which is located within P-GW.

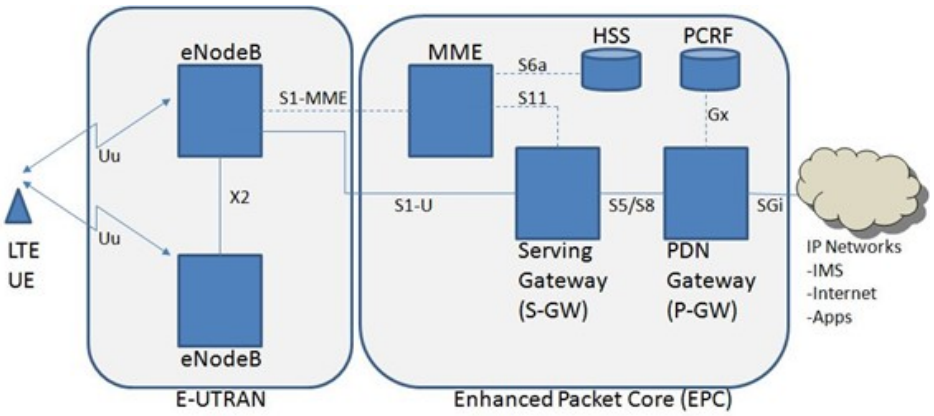

Figure 2.3: LTE architecture [3]

2.1.3 LTE Protocol Stack

The protocol stack has two planes:

- **The User Plane**: Includes network user traffic.
- **The Control Plane**: Includes signaling traffic.

As we see in figure 2.4 the protocol stack for each interface changes depending on the plane we are referring to. Let's see in detail the protocol stack for each of the user and the control plane.

- **PDCP**: The main function of Packet Data Convergence Protocol (PDCP) is the header compression of IP packets. It is also responsible for functions such as:

1. Transfer of C-Plane and U-Plane data between RLC and Higher U-Plane interface
2. Maintenance of PDCP Sequence Number.
3. Transfer of Sequence Number Status - ROHC header compression.
4. In-Sequence delivery of Upper Layer PDUs at re-establishment of lower layer.
5. Duplicate detection, elimination and retransmission of its own SDUs during handover at re-establishment of lower layer for RLC AM.

- **RLC**: The main function of Radio Link Control (RLC) is the transfer of upper layer PDUs. It is also responsible for error correction via ARQ and for concatenation, segmentation, and reassembly of RLC SDUs (only for UM and AM data transfer). Also, it provides functions such as duplicate detection (only for UM and AM data transfer), protocol error detection and recovery, RLC SDU discard (only for UM and AM data transfer), and RLC re-establishment.

- **MAC**: The main function of the MAC layer is the mapping between logical channels and transport channels. MAC is also responsible for the Multiplexing / demultiplexing of MAC SDUs belonging to one or different logical channels into/from transport blocks delivered from the physical layer on transport channels. In addition, it supports scheduling information, dynamic scheduling, and error correction through HARQ.

- **PHY**: Physical Layer includes all information from the MAC transport channels over the air interface. It is responsible for power control, link adaptation, cell search, and other measurements for the RRC layer.

- **RRC**: Radio Resource Control is responsible for managing the broadcast system information associated with Access Stratum and Non-Access Stratum (NAS). It is also responsible for the RRC connection between the UE and eNodeB and additionally manages UE measurements related to inter-system (inter-RAT) mobility.

- **NAS**: This is the communication protocol between the UE and MME. It is responsible for the mobility and session management of the UE and has functions for authentication and security control.

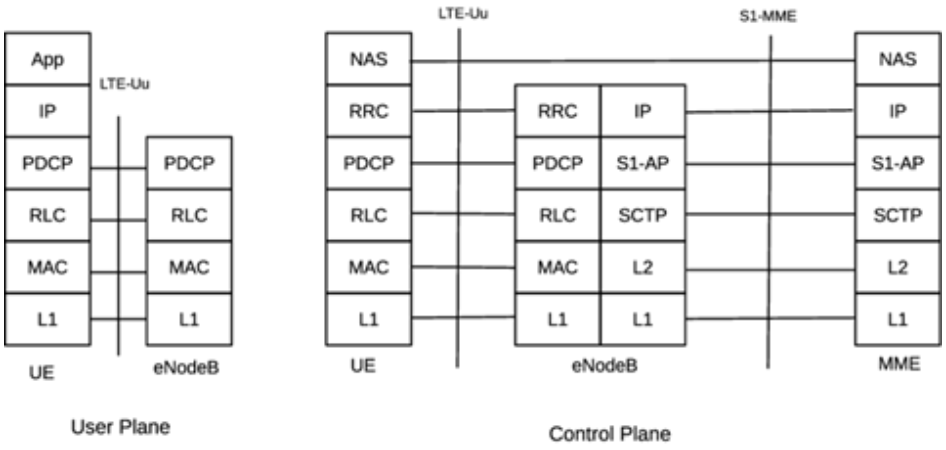

Figure 2.4: LTE protocol stack. [4]

The above protocols can be divided based on the Layers:

- Layer1: Consists of Physical layer.
- Layer2: Consists of PDCP, MAC, and RLC protocols.
- Layer3: Consists of RRC, NAS, and IP protocols.

2.2 Introduction to 5G

The fifth-generation wireless system (5G) is the next major phase of mobile telecommunications standards beyond the 4G. The 5G infrastructures are flexible and easily meet the growing demand and provide connectivity for multiple technologies such as Cloud Technology and the Internet of Things (IoT). The 5G network offers a heterogeneous solution to the network structure that integrates all previous generation networks, facilitating global mobility and service portability. According to International Telecommunication Union (ITU), the 5G networks are expected to offer unprecedented speeds and extremely low latency to the applications that will be integrated into the network. Depending on the features and requirements of the applications, ITU has defined three main application areas for the enhanced capabilities of 5G:

- **Enhanced Mobile Broadband (eMBB)**: This category includes cases that will improve the user experience such as access to multimedia content that requires high definition, data, and services like virtual and augmented reality.

- **Ultra-Reliable and Low-Latency Communications (URLLC)**: This category describes use cases of critical applications that have low latency and high reliability such as autonomous driving. In these applications, the requirements are increased in terms of performance, delay, and availability.
- **Massive Machine-Type Communications (mMTC)**: This category provides connectivity for many battery-powered, low-cost, and low data rate devices and sensors.

To meet the needs in the applications of the above categories, 5G Key Performance Indicators (KPIs) have been defined by IMT-2020 and are described in detail in the figure 2.5.

MINIMUM TECHNICAL PERFORMANCE REQUIREMENTS OF IMT 2020

KPI	Key Use Case	Values
Peak Data Rate	eMBB	DL: 20 Gbps, UL: 10 Gbps
Peak Spectral Efficiency	eMBB	DL: 30 bps/Hz, UL: 15 bps/Hz
User Experienced Data Rate	eMBB	DL: 100 Mbps, UL: 50 Mbps (Dense Urban)
5% User Spectral Efficiency	eMBB	DL: 0.3 bps/Hz, UL: 0.21 bps/Hz (Indoor Hotspot); DL: 0.225 bps/Hz, UL: 0.15 bps/Hz (Dense Urban); DL: 0.12 bps/Hz, UL: 0.045 bps/Hz (Rural)
Average Spectral Efficiency	eMBB	DL: 9 bps/Hz/TRxP, UL: 6.75 bps/Hz/TRxP (Indoor Hotspot); DL: 7.8 bps/Hz/TRxP, UL: 5.4 bps/Hz/TRxP (Dense Urban); DL: 3.3 bps/Hz/TRxP, UL: 1.6 bps/Hz/TRxP (Rural)
Area Traffic Capacity	eMBB	DL: 10 Mbps/m^2 (Indoor Hotspot)
User Plane Latency	eMBB, URLLC	4 ms for eMBB and 1 ms for URLLC
Control Plane Latency	eMBB, URLLC	20 ms for eMBB and URLLC
Connection Density	mMTC	1,000,000 devices/km^2
Energy Efficiency	eMBB	Capability to support high sleep ratio and long sleep duration to enable low energy consumption when there is no data
Reliability	URLLC	$1–10^{-5}$ success probability of transmitting a layer 2 protocol data unit of 32 bytes within 1 ms in channel quality of coverage edge
Mobility	eMBB	Up to 500 km/h
Mobility Interruption Time	eMBB, URLLC	0 ms
Bandwidth	eMBB	At least 100 MHz; Up to 1 GHz for operation in higher frequency bands (e.g., above 6 GHz)

Figure 2.5: Minimum Technical Performance Requirments of IMT 2020. [5]

2.2.1 5G Key Technologies

To cover the design and functional aspects of 5G, it was necessary to develop certain technologies. Some of them are extensions of 4G technologies, while others were developed exclusively for 5G. In this subsection we will analyze some of them.

- **Massive MIMO & Beamforming**: We have seen MIMO extensively in 4G / LTE technologies. It is a technology that uses multiple antennas which are configured as a

multi-dimensional phased array. In the case of 5G, Massive MIMO systems of the order of hundreds of antennas are used. By taking advantage of either spatial diversity (where the same data is transmitted by different paths and received by multiple antennas), or spatial multiplexing (where the data is divided into smaller parts and transmitted in multiple paths) we gain in reliability and data rate respectively. Despite the multiple uses of hundreds of antennas, another advantage of a massive MIMO system is energy efficiency. A single-antenna user in a massive MIMO system can reduce down its transmit power proportional to the number of antennas at the base station with perfect channel state information (CSI) [20]. At 5G, massive MIMO technology combines with Beamforming technology to deliver spectral and bandwidth efficiency to the network. The Beamforming technique refers to a technique of finding the most effective route from the base station to a specific one dissemination environment. More specifically the Beamforming steers the signals produced from an array of transmit antennas to an intended angular direction. In this way, the purpose is to minimize interference to and from other antenna systems and maximizing the amount of information transmitted.

- **Network Function Virtualization (NFV)**: NFV is a network architecture that uses virtualization technologies to simulate functions that allow multiple virtual networks to be created atop a shared physical infrastructure. With the use of NFV a 5G network can be fully virtualized. This can offer many benefits in terms of deployment, scaling and management of a 5G network. A research paper [21] related to the deployment of a Heterogeneous 5G Cloud-RAN set up in NFV on top of real-world testbed (NITOS-Testbed) showcases the above benefits as multiple virtual networks supporting different radio access networks (RANs). For thesis purposes, we rely on NFV technology using virtualized environments which we will analyze in detail in chapters 3 4 . These environments can fully visualize the network functions for scalable deployment and management of a 5G network.

- **Multi-access Edge Computing (MEC)**: MEC systems bring services near the edge of the network and therefore close to the end-user. This entity contains applications and a virtualization infrastructure that provides computers, storage, and network resources, as well as the functions required for applications. The MEC helps meet the 5G era requirements for expected performance, latency, and automation. The MEC allows extremely low latency and high bandwidth while at the same time it can provide

access to information in real-time for the network and the environment. This technology is the main object of study of this dissertation and will be adequately covered in the next chapter.

2.2.2 Functional Split Architecture

Taking the LTE architecture as a key reference, we can observe that the monolithic RAN (eNodeb) has some significant drawbacks regarding high-latency and lack of adequate coverage.

Therefore, the 3GPP proposed an improvement in the management of architecture components. More specifically, they proposed a distributed architecture based on the functional split of the base station (eNodeb) which brings significant improvements in network efficiency. As shown in figure 2.6, 3GPP proposed eight functional split options including several sub-options. Some of them are RRC-PDCP, PDCP-RLC, RLC-MAC, MAC-PHY. Although there are many options, most do not meet valid implementations and do not offer real benefits (e.g RLC-MAC) [6]. This is why most research focuses mainly on PDCP / RLC and MAC / PHY splits. Such research [22] compares these two splits in real-world testbed (Nitos Testbed) and through real experimentation and simulation it turns out that PDCP/RLC split has the least overhead and is compatible with various technologies enabling higher network capacity. Split architecture led to the creation of two new components in our Access Network topology:

- **Central Units (CUs)**: Provides support for the higher layers of the protocol stack such as SDAP, PDCP and RRC.
- **Distributed Units (DUs)**: Provides support for the lower layers of the protocol stack such as RLC, MAC and Physical layer.

The Core network up to the CU is called Backhaul, while the Fronthaul consists of the CU and DU components. The relationship between CU and DU is 1 to N, as a CU communicates with multiple DU's which may include 3GPP or non-3GPP (WiFi) technologies. On the other hand, each DU is associated with only a single CU instance. The communication between CU and DU is based on F1 Application Protocol (F1AP) via a newly introduced interface called F1. The F1 interface is divided into the following interfaces:

- **F1-U**: Responsible for User Plane communication. Uses the GPRS Tunneling Protocol (GTP).

- **F1-C**: Responsible for Control Plane communication. Uses the SCTP / IP protocol.

Through the above interfaces, the DU sends an F1 Setup Request, and then an RRC configuration setup is achieved between the CU and the DUs.

Other non-3GPP technologies have been integrated into the above architecture providing a Heterogeneous network. An example of such non-3GPP technology is WiFi which can be integrated as a DU [7].

For the packets to be sent from the WiFi DU side to the CU side (UL), some processes were made that encapsulate these packets with the corresponding PDCP headers. On the contrary, for DL communication, the CU after accepting the Data Requests decapsulates the PDCP headers of the packets before sending the payload to the WiFi DU. The above implementation constitutes the basis of our experimental setup.

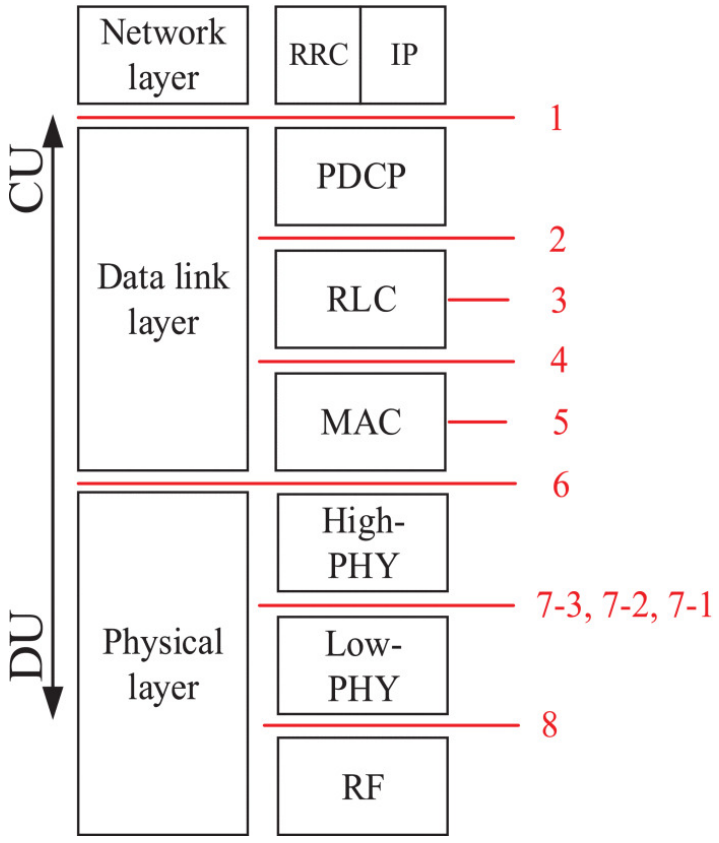

Figure 2.6: Eight functional splits proposed by 3GPP. [6]

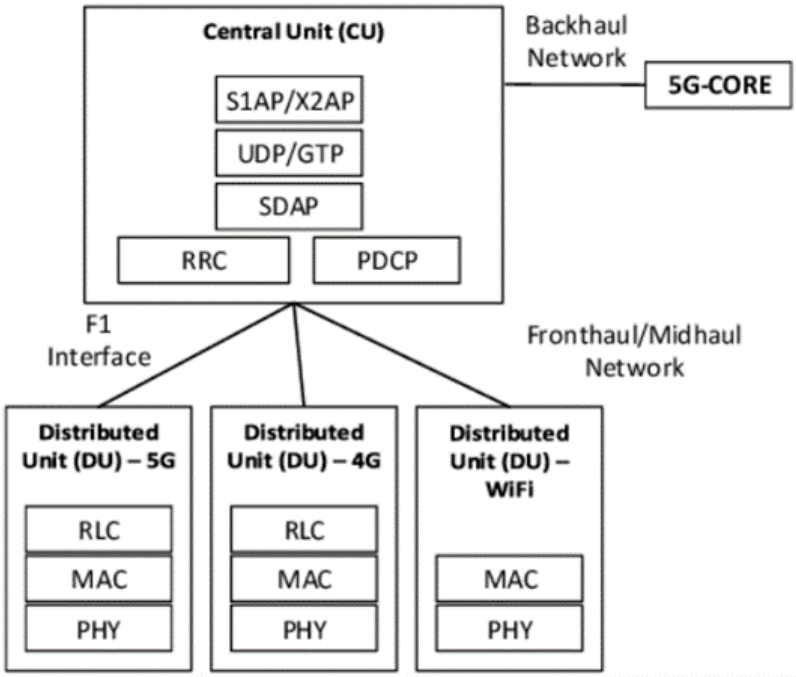

Figure 2.7: PDCP/RLC functional split [7].

Chapter 3

Study of Multi-Access Edge Computing

3.1 The defects of Cloud Computing

We can observe that centralized RAN has some significant drawbacks. Some of these are high-latency and high-capacity. High-latency due to the geographical distance between the Central Unit and the end-users. High-Capacity as the more functions are centralized, the more capacity is needed on the FH (fronthaul) for the user traffic, due to 1 to many relationship between CU and DU's [7, 23].

Therefore, the network bandwidth of cloud computing has been unable to meet the needs of time-sensitive applications and real-time performance. Such applications are Ultra-Reliable and Low-Latency Communications (URLLC), which require extremely low latency.

3.2 Introduction to Multiple-Access Edge Computing

The observations of section 3.1 led us to install computing infrastructures that are closer to the user. This logic is synonymous with the distributed computing paradigm called Multi-Access Edge Computing (MEC). Specifically, MEC systems bring services near the edge of the network and therefore close to the end-user. A MEC system contains applications and a virtualization infrastructure that provides computers, storage, and network resources, as well as the functions required for applications. By shifting the load of cloud computing to individual local servers, MEC can be considered as a perfect key enabler for various real-time applications as it helps reduce congestion on mobile networks and decrease latency, enhancing the quality of experience (QoE) for end users [24]. In detail, the MEC environment

is characterized by the following advantages:

- **Low latency**: Mobile Edge services can operate near end-user devices to provide the lowest possible delay.
- **Proximity**: Close to the source of information, Mobile Edge Computing is especially useful for getting basic information on analysis and big data.
- **High bandwidth**: The position of the Mobile Edge that is on the edge of the network in conjunction with the use of information from the cellular Real-time network can be used for bandwidth optimization for applications.
- **Location Awareness**: Mobile Edge can utilize low-level signaling information to determine the location of each connected device.
- **Real-time integration of information and Content**: Real-time network data can be used by applications and services to offer content-related services.

It is worth noting that MEC does not eliminate the role of cloud computing, but instead complements it. Nodes with MEC capabilities can directly serve the applications for which they have the necessary resources while assigning to a cloud server the service of those who are tolerant of delay. Both developments bring stability to connected devices in the Internet of Things network. The working method of the two can be that cloud computing is based on big data analysis and output, passed to the edge side, and then processed and executed by edge computing [25].

3.3 MEC System Implementations and Placements

The implementations of a MEC platform had already started from the LTE / 4G networks. The European Telecommunications Standards Institute (ETSI) had made the following proposals for the placement of MEC in LTE architecture [26]:

- **MEC deployed over SGi interface**: As we see in figure 3.1 the MEC host is located on the Backhaul, before SGW/PGW on the SGi interface. The closer the MEC host is to the core, the more accessible it is to Network Users. But the latency - RTT (Round Trip Time) is higher due to the geographical location of the core network. Therefore,

the placement of the MEC near the host is not so good practice, as we approach the latency of the services that are in the cloud and so we have no improvement in the management of time-sensitive data. The UL (uplink) flow of the packets, in this case, is as follows: UE-eNodeB-CORE-MEC.

- **MEC deployed over S1 interface**: The MEC host is located between the eNodeB and Centralized site (Backhaul). This type of MEC placement (closer to the Fronthaul) is preferable as the RTT (Round Trip Time) between the end-user and the MEC application is much lower than it would be if the MEC host were located close to the core, due to physical location. Therefore, with this placement, we achieve low latency. The UL (uplink) flow of the packets, in this case, is as follows: UE-eNodeB-MEC-CORE.

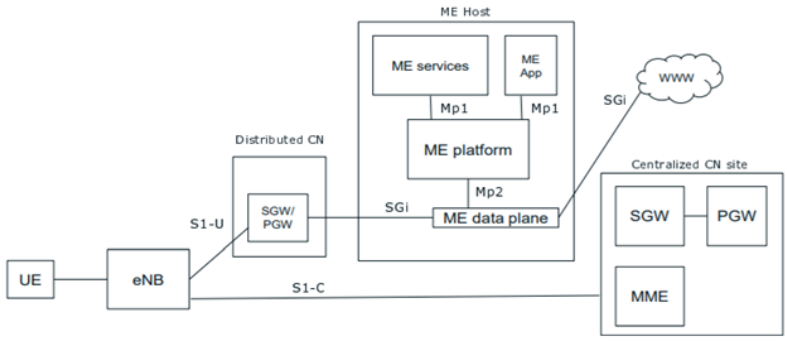

Figure 3.1: MEC Host is placed on the SGi interface.

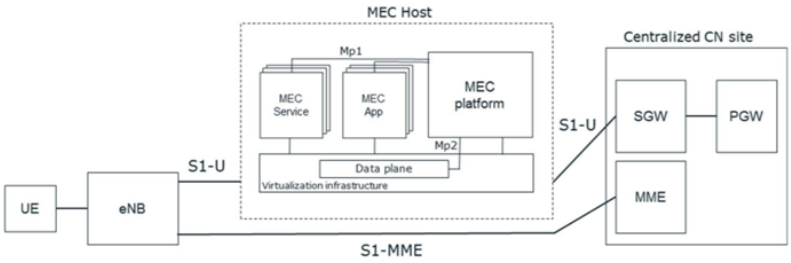

Figure 3.2: MEC Host is placed on the S1 interface.

3.3.1 Employing MEC on the Fronthaul of Heterogeneous 5G Architecture

In a similar philosophy to the second scheme, the placement of MEC is followed in the architectures of 5G networks. More specifically, research has been done on Employing MEC on the Fronthaul of Heterogeneous 5G Architecture [27], placing the MEC host next to the DUs. So, the UL is configured as follows: UE-DU-MEC-CU enabling the lowest latency between the end-user and the services. The above implementation is based on the F1 Application Protocol (F1AP) that we analyzed in chapter 2.2.2. This protocol is a key enabler for communication between CU-DUs. By the same token, the MEC Host, through the MEC agent who manages the packets going to and from the MEC services, communicates with the DUs. The agent holds a book-keeping process for mapping each RNTI value of each UE. Based on this RNTI information, the appropriate requests are made between DU-MEC and vice versa. More specifically when a DU has data to transmit to the MEC service creates a MEC data request message. This message is then handled by the MEC agent and its payload (user data packets) is delivered to the service. Similarly, for the reverse path, the MEC agent generates a MEC data indication for the DU that the client is registered with. The above implementation and the placement of the MEC-Agent that takes place in figure 3.3 are the basis for setting up our experimental setup.

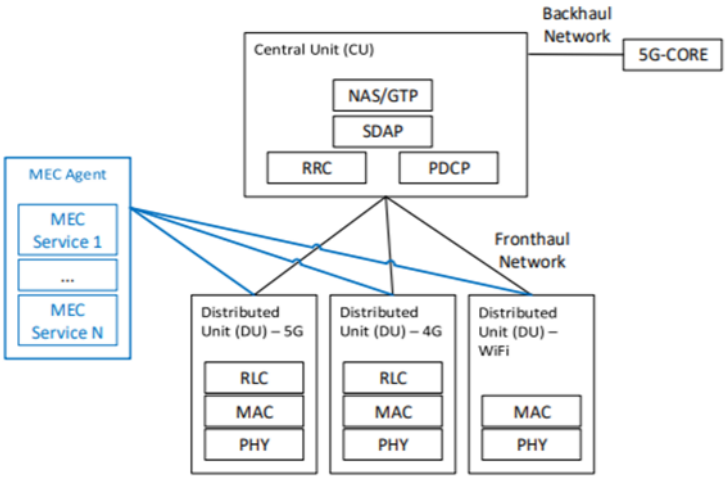

Figure 3.3: Employing MEC on top of DUs. [8]

3.4 MEC Services Type Deployment

To deploy MEC-Services, there are two available options: Virtual Machines and Containers. Both are capable of running MEC services, however, there is a dilemma in choosing the most suitable one as both technologies have pros and cons, which we will analyze below:

- **Virtual Machines (VMs)**: A VM is an abstraction of physical hardware. In short, VMs make a virtual copy of all the hardware that the operating system needs to run in order to be functional (Hypervisor-based virtualization). Therefore, they are heavy and require a slow startup to boot as they can take up a lot of system resources because of booting a complete OS. [28]. The isolation property of VMs provides users an independent system, irrespective of the underlying hardware offering more security [29].

- **Containers**: Unlike VMs, containers visualize only OS. This offers multiple benefits such as fast scalable provisioning and low resource consumption. In practice, a container can be instantiated in the scale of milliseconds but has process-level isolation, therefore it is possibly less secure. There are two types of containers: system containers (e.g. LXC, LXD) which containerize a complete operating system, and application containers (e.g. Docker) which provide a lightweight virtualization solution to run processes in isolation. MEC services can benefit from containers because they offer mechanisms for fast packaging and deployment to a large number of interconnected MEC platforms [30].

For our experimental setup, we chose to use both technologies to host MEC services. In the end, however, we ended up with Virtual Machines due to their developed ability to be Live Migrated, which brings many benefits in terms of managing MEC services as we will see in subsection 3.5.2. We also explored hybrid solutions that include nested containers inside VMs.

3.5 Live Migration of MEC Services

3.5.1 Purpose of Live Migration

The mobility of users plays a huge role in the QoS / QoE provided by the MEC services, since the farther the user is from the DU, and consequently from the corresponding MEC

service which is attached to this DU, the higher is the delay and therefore the worse the user experience will be. In particular, if MEC services are related to V2V (Vehicle to Vehicle) or V2X (Vehicle-to-everything) communications, wherein these cases the delay should be extremely low due to the time-sensitivity of the data, then the services should be constantly close to the user with the permanent purpose of low delay.

So in cases where the user moves away from the base station where he is served and approximates the next base station, in the meantime or before the handover between the base stations, there should be a live migration of services from the source (current) edge server to the destination edge server near the mobile user.

An interesting - intermediate solution that we implemented in our experimental setup and that is supported only in heterogeneous 5G networks, is the dynamic transition to non-3GPP technology (such as WiFi) depending on the quality of the link offered by 3GPP technology. In short, the MEC host through the MEC agent-controller constantly listens to the quality of the link between the user and the MEC application. Depending on the quality, it passes the traffic through the most suitable DU. For example, if the latency is low enough on 3GPP DU (eg LTE DU) side due to the mobility of the user from the DU, then the MEC Agent will switch the traffic to pass exclusively through non-3GPP technology (WiFi DU). In figures 3.4 and 3.5 we see in detail the process of the Radio Access Technology switch.

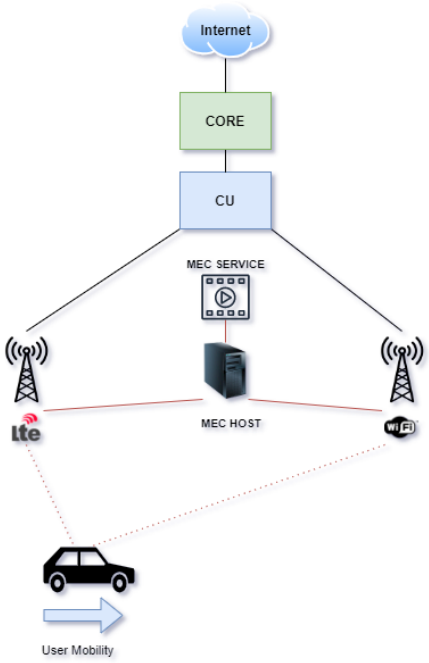

Figure 3.4: MEC traffic passed on dual technology DU's.

However, RAT switching does not completely solve the problem. As user mobility increases, the signal between the user and non-3GPP technology weakens. Therefore the need for live migration of services is imperative, to maintain the connection between the user and the service, as the user connects to the next RAN. As shown in figure 3.6 during the inter-eNB Handover, live migration of the MEC service takes place from Host MEC to the destination Host MEC of the next RAN, providing seamless service support. The above approach is also known as the “Follow Me” approach and it requires continuous monitoring of user mobility and a dynamic way of managing resources for live migration of services, to maintain the QoS / QoE at the highest levels.

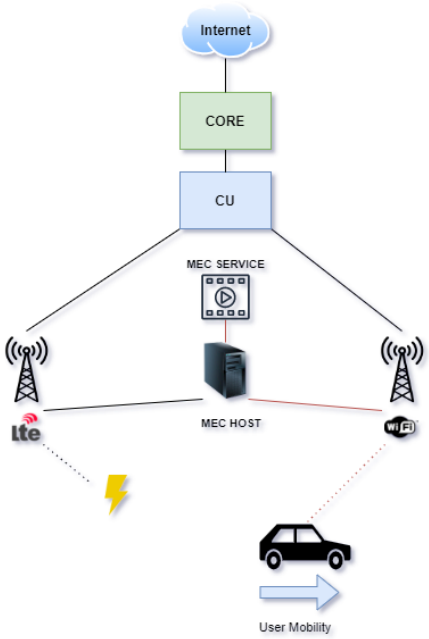

Figure 3.5: Radio Access Technology switch.

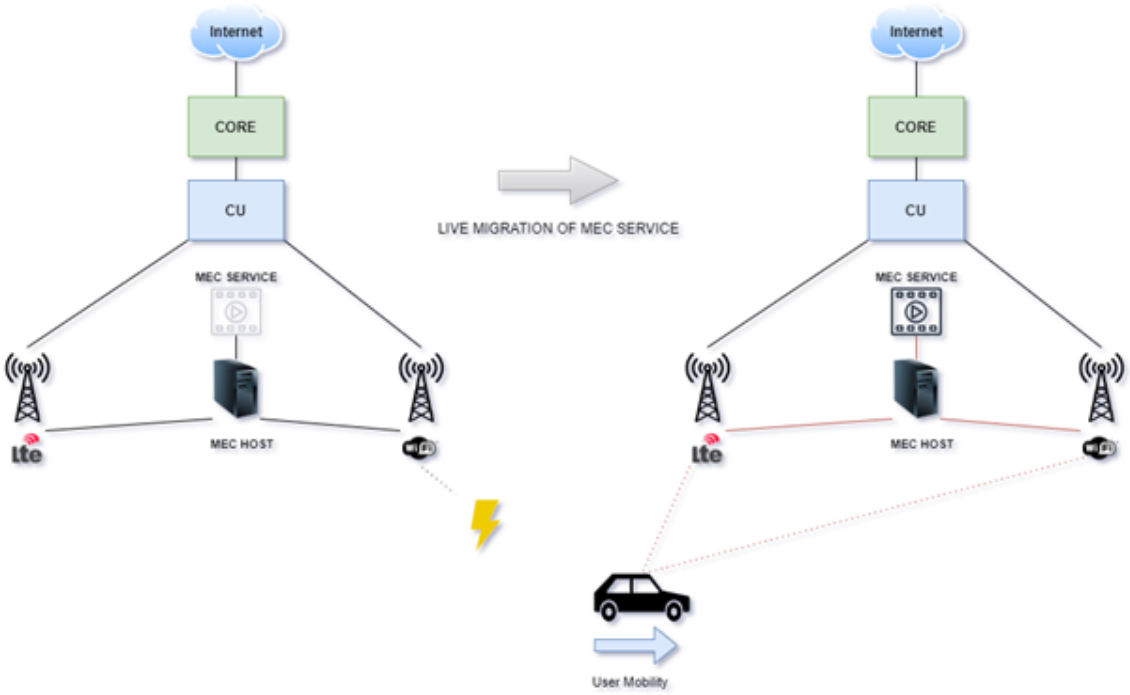

Figure 3.6: Live Migration of MEC service.

3.5.2 VM Live Migration

Migrating the virtual machine means that the in-memory state of VM can be transferred consistently and efficiently as transparently the VM moves from one host to another without perceived downtime. According to the following research [31] the aim of live VM migration consists of:

- Application performance optimization during VM migration.
- Efficient bandwidth utilization .
- Minimize high migration time and downtime during migration.

Ideally, during migrations, we want to maintain kernel state, active TCP / IP connections, application state, and sockets. However, many factors can be an obstacle to maintaining the above states. The long downtime due to the many memory pages that need to be copied is one such factor. Also, the network connectivity between the source host and the destination/target host plays an important role. Therefore, we are leading to either stateful or stateless/cold migrations. In the first category, the applications and the connection state are maintained. In the second category, the connections are lost but the memory pages and the disk are successfully copied.

Finally, an important factor is the different techniques/technologies that handle the VM memory through Live Migration. The most basics are Pre-copy VM Migration and Post-copy VM Migration.

- **Pre-copy VM Migration**: In this method during migration, memory pages are copied iteratively from source to destination, even as the virtual machine is running at the source. In the first phase, the modified memory pages of the source node are duplicated and copied to the destination node. Pages that are modified but not copied are used to estimate the downtime. When the modified pages (or dirty pages) are less than the re-copied pages, then the process of the instance on the source is stopped (causing a downtime), the remaining pages are transferred to the destination and the VM instance is resumed in the destination.

- **Post-copy VM Migration**: In contrast to the pre-copy migration, post-copy first suspends the migrating VM at the source node, copies minimal processor state to the target node, resumes the virtual machine on the target node, and begins fetching memory pages over the network from the source.

As shown in figure 3.7 more data is transmitted in the pre-copy than in the post-copy, but the pre-copy seems to have less downtime, as adaptive algorithm applications for managing dirty pages are possible [32].

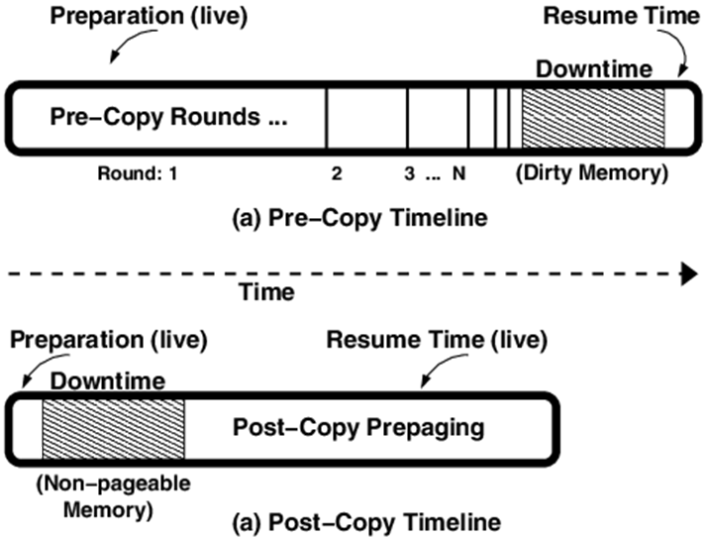

Figure 3.7: Pre-Copy Migration vs Post-Copy Migration [9].

3.5.3 Live Migration on KubeVirt

KubeVirt [13] is a Kubernetes add-on that extends Kubernetes capabilities by delivering virtual machines as container workloads. With KubeVirt we can take advantage of the benefits of VMs in a containerized environment and use hybrid solutions offered by the coexistence of VMs and Containers. The architecture of KubeVirt is presented in more detail in 4.4.

The significant addition of KubeVirt to the Kubernetes ecosystem brings an environment ideal for edge solutions in modern cellular networks. This is because we can manage edge services by defining the life cycle, scaling, and migrating them. Based on the above benefits, we decided to deploy the MEC host (MEC agent & MEC app) to a VM under the unique API of KubeVirt.

We can manage VMs as we could manage containers and take advantage of VMs LiveMigration by executing kubectl commands. More specifically, Live migration is initiated by posting an object VirtualMachineInstanceMigration to the cluster, indicating the VM name to migrate. Also through the ConfigMap object of Kubernetes, we can define the LiveMigration parameters such as the bandwidth that is reserved for each VM Live Migration and much more.

KubeVirt uses containerized libvirtd and QEMU technologies to deploy VMs. In addition, the pre-copy technique is used for the Live Migration of VM's, contributing to lower downtime. However, as of this writing, there is no capability to specify the target node, as the API selects the target node according to the lower resources utilization.

3.6 On Follow-Me Schemes

3.6.1 Related Work

In order to maintain the QoS / QoE at the highest levels, there should be a dynamic mechanism that, on the one hand, will monitor the change in user mobility, and on the other hand, will Live Migrate the MEC services depending on the nearest RAN to the mobile user. The above logic describes a Follow-Me scheme. The key requirement to implement such a scheme is the autonomy of MEC services in order to be able to move from one node to another without interrupting the connection with end-users. Equally important is the criterion for which Live Migrations will be triggered. In short, there should be a dynamic QoS-aware edge server

selection algorithm.

Some research papers suggest algorithms that taking consideration of the transmission power of the MEC server as a criterion for signaling the live migration of services [33], while some others focus on tracking user's position and predicting user movement [34]. However, the above research papers require perfect knowledge of user mobility within a given time and access to physical-layer. Therefore they are difficult to implement. In addition, most research work does not consider the heterogeneity of network (eg, 4G, WiFi, and 5G DUs), which leads to different network connections and transmission paths for data transferring between them. [32]. Also, a wide range of research has studied the options on the management and orchestration of the MEC applications, but most of them proposed MEC frameworks are unable to support service live migration [35].

3.6.2 Follow-Me MEC Implementation in Kubernetes Environment

Our Follow-me implementation scheme relies on the management of an autonomous Single MEC Host with the help of the Kubernetes framework. The MEC host is a VM instance delivered as container workload by the KubeVirt's component virt-handler. The placement of the MEC host is located in the fronthaul of a Heterogeneous 5G network, where the transmission path of the MEC service can be supported simultaneously through 3GPP and non-3GPP technologies. This is possible through the MEC agent running internally on the MEC Host and whose operation is described in detail in the subsection 3.3.1. We focus on MEC to MEC communication without the need to access the core network to apply Live Migration of services. MEC services are located either internally in the VM or nested Docker containers. The autonomy of the MEC Host is based on the fact that it has remote access to the Kubernetes cluster on which it has been deployed. Therefore it can Live Migrate itself depending on the circumstances.

The live migration of services is triggered by the MEC controller which also operates internally on the VM and can execute migrate commands as well as support RAT switch functions. In the figure 3.8 the complete architecture of the MEC host and how the MEC components are connected is presented. MEC service is a docker application that is attached by two macvlan interfaces M1 and M2. The M1 connects the MEC service with the MEC agent and through this interface passes all the traffic between the service and the end-user. On the other hand, the M2 interface connects the MEC service with the MEC controller.

Through this interface passes all the traffic related to the monitoring of the quality of the connection between the service and the end-user. More specifically, the connection between the MEC controller and the MEC application is based on server-client communication. The MEC service runs a server socket whose operation is described in the algorithm 1. In essence, it gathers information about the quality of the link it has with the end-user. This information is mainly related to RTT which is measured multiple times and after these measurements, the RTT average is calculated. Along with the RTT average, the number of packets and the packet loss rate are measured and all of these statistics are stored in a dictionary. This dictionary is then sent to the socket client which runs on the MEC controller and whose operation is described in the algorithm 2. The MEC controller after receiving the dictionary constantly monitors if the RTT average exceeds the RTT threshold which is defined depending on the type of application. In case the RTT average exceeds the threshold, then the controller makes a RAT switch by switching the transmission path from LTE DU to WiFi DU. If the delay is still high then the controller live migrates the MEC Host to another Kubernetes Node. After waiting for the average migration downtime to pass, which is updated at the end of each migration after being parsed by the log files, it switches to LTE DU. In the meantime, the user exchanges MEC data via WiFi DU.

In this way, we keep the costs of migrations low, as service migration incurs additional operation costs such as usage of the expensive wide-area-network (WAN) bandwidth and system energy consumption [36]. And at the same time, we also take advantage of the benefits of a heterogeneous 5G network utilizing all the transmission paths that are available with the sole purpose of reducing latency.

However, in many cases changing the transmission path between different RATs can cause a connection reset between the end-user and the application. Therefore, in cases where the application does not support multihoming capabilities, the algorithm 3 is recommended, which follows an always migrate strategy depending on the latency values without a RAT switch.

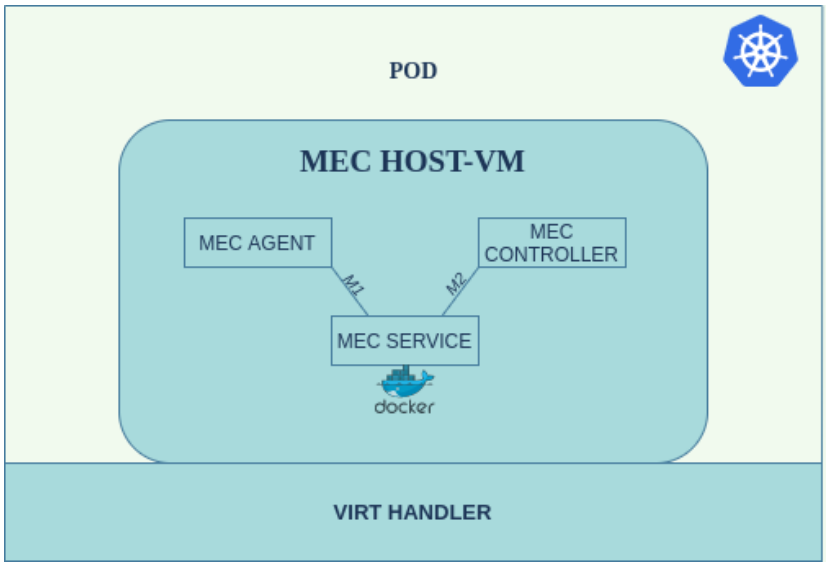

Figure 3.8: MEC Host Architecture.

Algorithm 1: Follow-Me procedure [Server]

Input: MEC-to-MEC server IP mec_ip, $port$, End User's IP ue_ip, #packets to send $num\_of\_packets$

```
1 Function follow_me_server(mec_ip, port, ue_ip, num_of_packets):
2     client_socket = init_server_socket(mec_ip, mec_port);
3     while True do
4         ping_results = get_ping_results(ue_ip, num_of_packets);
5         ping_statistics = ping_results.parse().as_dict();
6         msg = ping_statistics.serialize();
7         client_socket.send(msg)
8     end
9 End Function
```

Algorithm 2: Follow-Me procedure [Client]

Input: MEC-to-MEC server IP mec_ip, $port$, #packets to send $num\_of\_packets$, $rtt\_threshold$

```
1  Function follow_me_client(mec_ip, port, ue_ip, num_of_packets):
2      client_socket = connect_to_server_socket(mec_ip, mec_port);
3      avg_downtime = init_avg_downtime();
4      rat_switch = False;
5      while True do
6          msg = client_socket.recv();
7          ping_statistics = msg.deserialize();
8          if ping_statistics.rtt_avg > rtt_threshold then
9              if rat_switch == False then
10                 switch_to_wifi_du();
11                 rat_switch = True;
12             else
13                 kubectl.live_migrate(mec_host);
14                 sleep(avg_downtime);
15                 avg_downtime =
                       get_avg_downtime(log_of_previous_migrations);
16                 switch_to_lte_du();
17                 rat_switch = False;
18             end
19         end
20     end
21 End Function
```

Algorithm 3: Always-Follow-Me procedure [Client]

Input: MEC-to-MEC server IP mec_ip, $port$, #packets to send $num\_of\_packets$, $rtt\_threshold$

```
1  Function always_follow_me_client(mec_ip, port, ue_ip, num_of_packets):
2      client_socket = connect_to_server_socket(mec_ip, mec_port);
3      avg_downtime = init_avg_downtime();
4      while True do
5          msg = client_socket.recv();
6          ping_statistics = msg.deserialize();
7          if ping_statistics.rtt_avg > rtt_threshold then
8              kubectl.live_migrate(mec_host);
9              sleep(avg_downtime);
10             avg_downtime = get_avg_downtime(log_of_previous_migrations);
11         end
12     end
13 End Function
```

Chapter 4

Kubernetes Ecosystem

In recent years, microservices have become an attractive solution for applications that must be highly scalable and portable. The revolution of microservices is the technology of containers. In the field of telecommunications and computer networks, the arrival of the NFV technology is harmoniously combined with VMs since they satisfactorily visualize the Network Functions. The baton in this technological development is taken by the Kubernetes framework, as through its unique API it can orchestrate hundreds of containers and VM's through KubeVirt add-on that helps to take an existing virtual machine and deploy it inside a container. Thus, Kubernetes helps to reduce the costs of deploying and operating cloud-native network functions, playing a key role as a Virtual Infrastructure Manager (VIM) and VNF Manager (VNFM). A 5G network can be fully softwarized as all its functions and architecture components can be visualized as it promotes NFV technology. For this reason, the deployment of a 5G network in the Kubernetes environment is an ideal solution for the management of network resources, the monitoring of its status, its scale, and its easy installation. Therefore, it is a good time to analyze the Kubernetes ecosystem, mentioning the technologies it integrates and the various technologies we used to develop the dissertation.

4.1 Introduction to Docker

Docker [10] [https://www.docker.com/] is one of the most popular container technologies. Docker provides the ability to build - package isolated application configuration and execution environments. These environments are called containers. Docker has 2 basic objects:

- **Docker Containers**: A container is a standard unit of software that packages up code and all its dependencies. The containers are light and isolated from the environment in which they run. thus, the same container can run in a data center, one cloud computing platform, or even on a personal computer. A container contains only space operating system users, libraries, and services required for applications. Also, thanks to their isolation, it is possible to run them on the same machine multiple times simultaneously. For docker containers to run successfully, they need a Docker runtime, which sits on top of the host operating system. This runtime is called Docker Engine and it allows containerized applications to run on any infrastructure.

- **Docker Images**: A docker image is a file that contains all the necessary elements that an application needs to run in a container, such as libraries, configuration files, and system tools. Docker images become containers under docker engine.

4.1.1 Docker architecture

The docker architecture is based on client-server communication. The server is called Docker Daemon and is responsible for build, run and distribute Docker Containers. Before each action, Docker Daemon communicates through a REST API through UNIX sockets with the client called Docker Client. For the storage and distribution of Docker Containers, there are storage spaces that store Docker Images. These spaces are called Docker Registries. Docker Daemon retrieves and stores images from and in Docker Registries.

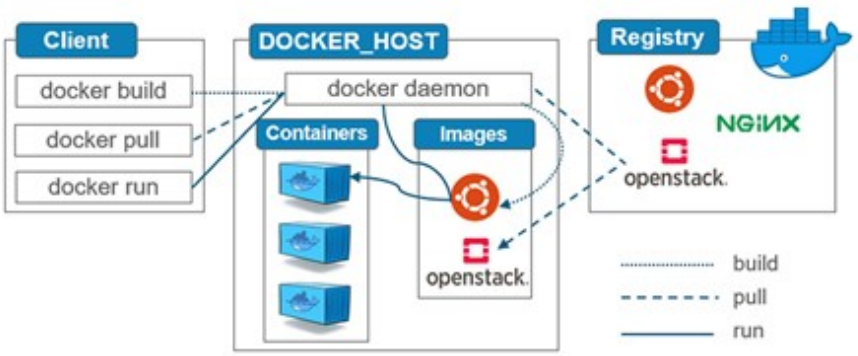

Figure 4.1: Docker Architecture. [10]

4.2 Introduction to Kubernetes

Kubernetes [11], also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes is therefore a container orchestrator which, through its framework, provides multiple benefits to distributed systems that use containerized applications. Some of the features it offers are:

1. **Load balancing**: Kubernetes has a convenient function to expose containerized applications. This function is called Service. By this, the applications are exposed on the internet (outside the cluster) via the DNS name or the IP address. Kubernetes within the service can guarantee that the load assigned to each container is evenly distributed.

2. **Storage Management**: Kubernetes allows volume mounting from various storage systems, such as local storage, public cloud providers, NFS, and more.

3. **Continuous Control of the Desired State of Deployment**: The description of the state of deployed containers can be easily done through YAML or JSON files. Kubernetes is then responsible for changing the current state to the desired state at a controlled rate. For example, we can set the number of containers we want to run at any time.

4. **Resource Management**: Kubernetes enables cluster administrators to define resources such as memory (RAM) and CPU that will be consumed by containerized applications. Based on the description of resources, Kubernetes takes care of the better management of these resources.

5. **Health Checking**: Kubernetes constantly monitors the health status of the containers. In case of failed containers, Kubernetes restarts or replaces containers that do not respond.

6. **Secret and configuration management**: Kubernetes enables its users to define container configurations and store and manage various sensitive information such as passwords, SSH keys, etc. without having to rebuild container images.

4.2.1 Kubernetes Components

Kubernetes manages clusters that consist of worker machines called nodes. Clusters have at least one node, as on these worker-nodes run the containerized applications which are managed within the smallest unit that Kubernetes can handle which are called pods and are a generalization of the container (we will analyze them in detail later). However, for the cluster to be functional, the nodes must run the following components:

- **Kubelet**: Kubelet is an agent that ensures that the containers inside the pods are running and are healthy.
- **Kube-proxy**: Kube Proxy is a network proxy that maintains network rules at the nodes, to achieve network communication between the pods, inside and outside the cluster.
- **Container runtime**: The container runtime is the software that is responsible for running the containers. Kubernetes supports various container runtimes such as Docker, containerd, CRI-O, and many more.

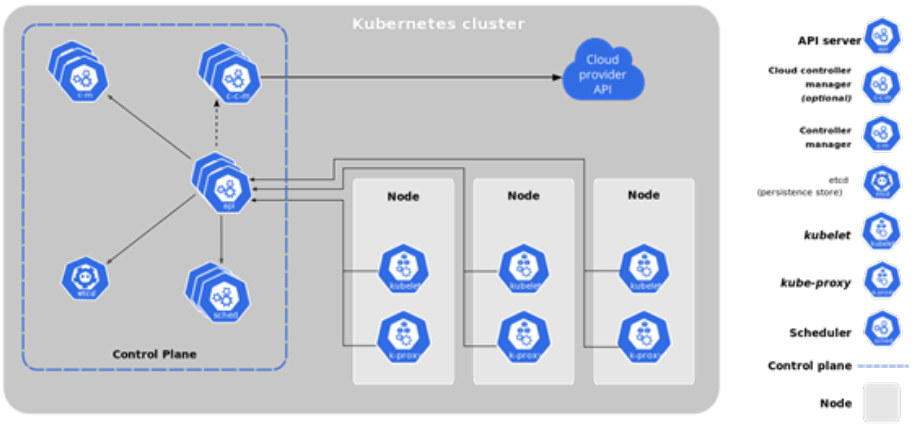

Figure 4.2: Kubernetes Components. [11]

Kubernetes has a system that checks the status of nodes and cluster objects such as pods and works to make the actual state of objects match the desired state. It also exposes the API and controls scheduling. This system is called Control Plane and operates on a node called a control plane node. Kubernetes clusters usually have only one control plane node, but there may be more for high availability. In such nodes operations are performed only for the control

operation of the cluster, therefore containerized applications are not scheduled on them. The control plane is made up of the following components:

- **kube-apiserver**: The kube-apiserver is a server that exposes the Kubernetes API. It is the main implementation of the Kubernetes API, as it is the front end of the Kubernetes control plane. There can be multiple instances of kube-apiserver in a cluster, to have load balancing between them for a smooth response to the requests of cluster administrators. The API is accessed and used either through a command-line interface such as kubectl or through a user interface such as the Kubernetes Dashboard.
- **etcd**: The etcd is a consistent and highly available, distributed data store. It's actually the Kubernetes backend, which contains the cluster information in key-value pairs.
- **kube-controller-manager**: The kube-controller-manager runs controller processes. More specifically it is a loop that constantly focuses on making the desired state equal to the current state for the Kubernetes objects in the cluster.
- **kube-scheduler**: The kube-scheduler is a controller that constantly assigns to the worker nodes the newly created pods. The assignment is based on multiple criteria such as resource requirements, hardware/software/policy constraints, and user specifications e.g. node affinity.
- **cloud-controller-manager**: The cloud-controller-manager links the cluster to the public cloud providers. If the cluster is local, the cluster does not have a cloud controller manager. The cloud controller manager runs controllers that are associated with specific cloud providers.

4.2.2 Kubernetes Objects

Kubernetes objects are entities provided by Kubernetes, for configuring, deploying, and scaling containerized applications. They are described in the form of YAML or JSON files which are passed to the Kubernetes API. Then, through the various components available to the Kubernetes nodes, the continuous control of the transition of the current state of these objects to the desired state is performed. These objects are:

- **Pod**: As already described, the pod is the smallest management unit in the Kubernetes. Multiple containers can run inside the pods, but it is common to have one pod for

one container. Kubernetes creates a virtual network that assigns new IPs to the newly created pods. These IP addresses are not permanent. Thus, in case there is a failed or restarted pod, a new IP address is created and assigned.

- **ReplicaSet**: The purpose of its operation is the desired number of pods that are specified, to always be in the running state. In short, it scales up and scales down the pods depending on the desired state.

- **Deployment**: Deployment manages the creation, deletion, and updates of pods. It can be considered as a higher level of abstraction of replicaSets, as it uses replicaSets to manage the pods. It allows for seamless application updates and downgrades through rollouts and rollbacks, and it directly manages its ReplicaSets for application scaling.

- **Service**: Service is configured to forward requests to a set of pods. Services have an IP address and this IP address automatically routes to a healthy pod. Due to the lack of permanent - static IP addresses of the pods, the services take advantage of this weakness and offer a more permanent solution in the communication between the end-users and the pods.

- **Job**: Job is the type of pod that is supposed to terminate on its own after execution. It is used to create and execute individual tasks.

- **StatefulSet**: StatefulSet is used specifically for stateful applications.Manages the deployment for the pod scaling and gives the pods a sticky identity that it maintains across any rescheduling.

- **PersistentVolume (PV)**: PersistentVolume is a part of storage space in the cluster that is provided either statically by the administrator using for example the resources from the worker nodes, or dynamically by the cloud provider.

- **PersistentVolumeClaim (PVC)**: PersistentVolumeClaim acts as a request to use the storage created by PersistentVolume. When the storage request is accepted, the volume is attached to the pod. Claims can request specific sizes and access modes.

- **ConfigMap**: ConfigMap is used to easily manage the configuration files used by containerized applications. It offers the ability to make dynamic changes, usually through environment variables, to the data during the runtime of the container.

- **Namespace**: It is responsible for the separation and organization of Kubernetes objects. A classic way to use it is to create different deployment environments.

4.2.3 Kubernetes Networking

A key part of Kubernetes functionality is networking. The basic communication scenarios between containers-pods in a cluster are the following:

- **Container to Container Communication**: Container to Container Communication: Containers are located inside the pod. Containers are connected to a physical network inside a pod via Docker bridge (with virtual interface) as we see in figure 4.3. So since from the moment they are connected to the same bridge interface they can communicate with each other without NAT (Network Adress Translation).

- **Intra-Node Pod Communication**: Different pods on the same Node. Suppose the communication scenario between pod1 and pod2 is shown in figure 4.3. The packet leaves from pod1 and enters the Node's 1 Root Network, then it passed to the Linux bridge (cbr0). Cbr0 makes an ARP request to find the destination. Then it finds the destination and forwards the packet to veth1 and finally, the package reaches the pod2 network.

- **Inter-Node Pod Communication**: Different pods on different Nodes. Suppose the communication scenario between pod1 and pod4 is shown in figure 4.3. The initial process of sending the packet from pod1 Network to cbr0 is the same as the Intra-Node communication scenario. However, when the package reaches the root network of Node1 it cannot be forwarded unless there is a routing table. So with the routing table, the packet is forwarded to the cbr0 of Node2 and then reaches the pod4 network.

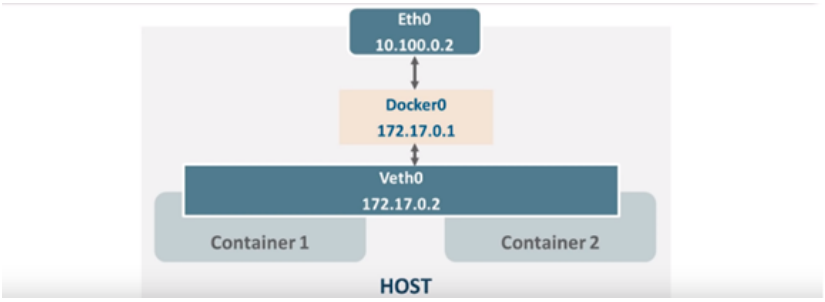

Figure 4.3: Container to Container Communication.

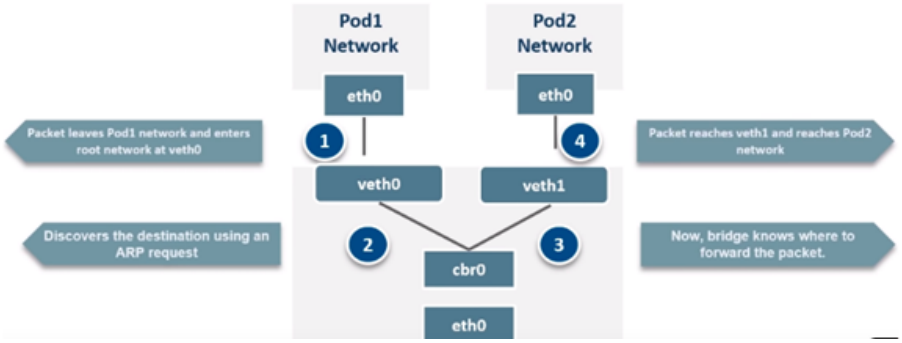

Figure 4.4: Intra-Node Pod Communication.

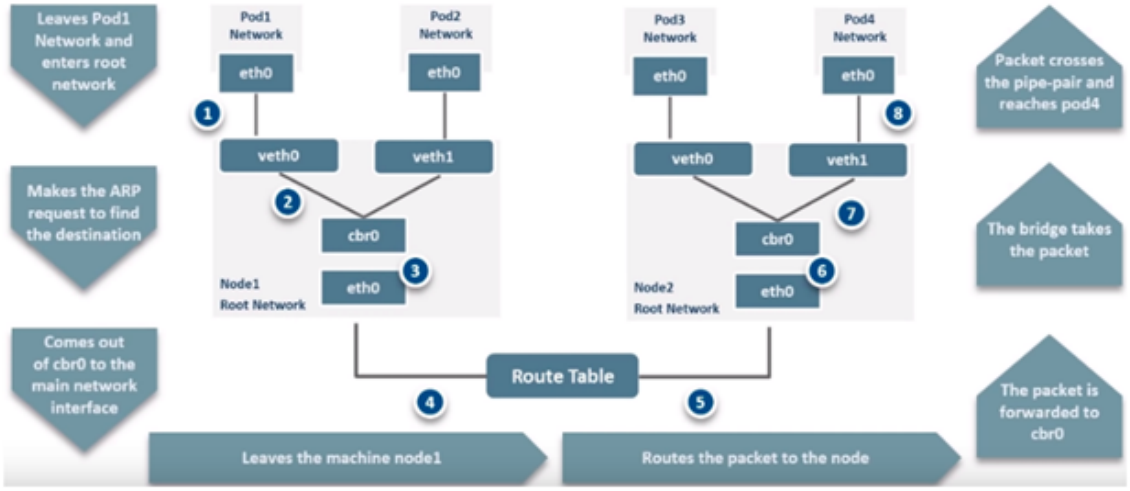

Figure 4.5: Inter-Node Pod Communication.

4.3 Multus CNI

By default, it is not possible to add more than one network interface to Kubernetes pods. This gap is being filled by Multus CNI [12]. Multus CNI is a container network interface (CNI) plugin for Kubernetes. Thanks to Multus CNI, attaching multiple network interfaces to pods is enabled. The addition and specification of one or more network interfaces are

done through the Kubernetes Network Custom Resource Definition, where the user through a YAML file can configure the additional network interfaces, and then the Kubernetes API will take care of the attachment.

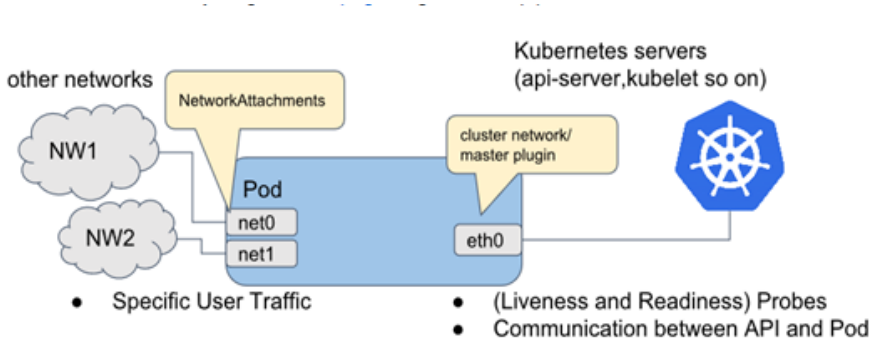

Figure 4.6: Secondary interface attached to a pod via Multus [12].

4.4 KubeVirt

As we know Kubernetes is a container orchestrator managing containerized applications in the most efficient way. By default, it does not support the management of virtualized technologies such as Virtual Machines. This weakness is covered by KubeVirt [13] which is a Kubernetes add-on that enables the management of libvirt virtual machines. KubeVirt is not a way to separate between containers and VMs, because KubeVirt delivers virtual machines as container workloads. Its architecture is based on the following components:

- **virt-api-server**: virt-api-server is the server that exposes the API of KubeVirt. It updates anything that has to do with virtualization flows through custom resource definitions. It is also responsible for the validation and defaulting of VMs.

- **virt-controller**: Manages the pods associated with VMs and is responsible for monitoring the status of VMs.

- **virt-handler**: virt-handler is similar to kubelet, as it runs on each worker node and monitors the state of VMs constantly trying to satisfy the desired state.

- **virt-launcher**: virt-launcher manages the namespaces to be used to host VMs. The virt-handler component signals the virt-launcher to start a VM, by passing the VM's CRD object to it.

- **libvirtd**: libvirtd is located inside each VM pod and is used by the virt-launcher to manage the life cycle of the VM process.

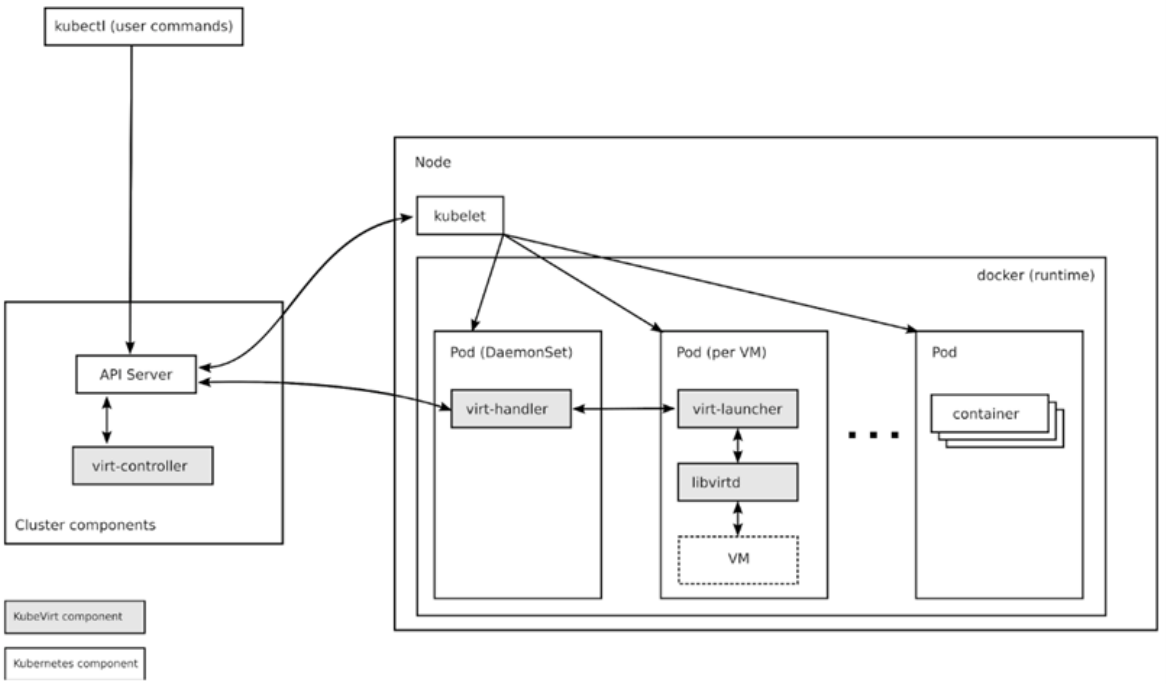

Figure 4.7: Kubevirt Architecture [13]

4.5 Prometheus

Prometheus [14] is an open-source monitoring and alerting framework. Prometheus is an ideal solution for collecting and monitoring multi-dimensional data, not only for machine-centric computer systems but also for architectures that support microservices (e.g. Docker, Kubernetes). It is capable of receiving large amounts of data every second and offers multiple benefits to computer systems. Some of these are:

1. A multi-dimensional, time-series data model which is defined by key/value pairs and a metric name.

2. An easily manageable query language named PromQL.
3. The collection of time series data is done autonomously through the HTTP protocol.
4. Targets are discovered via service discovery. Monitoring, graphing, and dashboarding support.

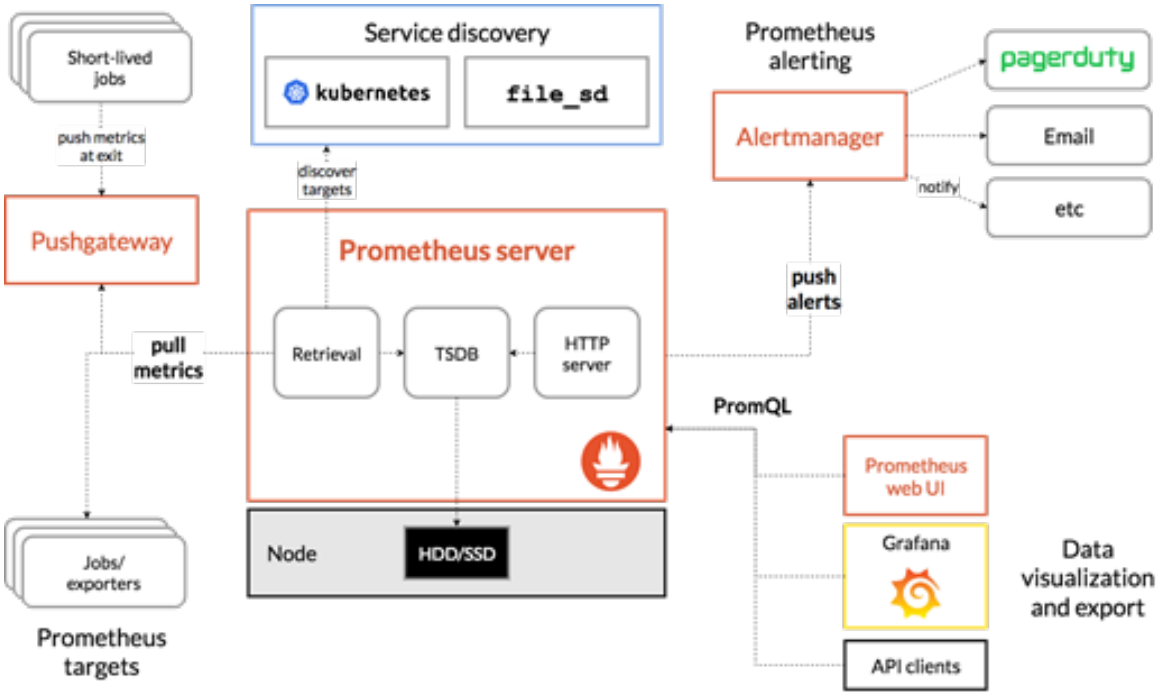

Figure 4.8: Prometheus Architecture [14].

4.6 Grafana

Grafana [37] is an open-source web application that provides interactive data visualization through charts and graphs and alerts. Ideally combined with time-series databases such as Prometheus. Used in machine-centric computer systems but also in systems that support architectural microservices such as Kubernetes. An ideal usage scenario is the visualization of the data that have been exported from containerized applications through Prometheus. Also, with the interactive dashboards available, Grafana can give the overall picture of the cluster to the user such as memory or CPU utilization of the nodes. Finally, it enables the user to create custom dashboards that visualize custom metrics that have been exported by Prometheus.

Chapter 5

Experimental Setup

5.1 Experimental Setup Resources

In this section, we are going to analyze thoroughly the testbed and the tools used for the experimental setup.

5.1.1 NITOS Testbed

The target facility used for the development, application, and evaluation of this dissertation is the NITOS testbed [15], located in University of Thessaly, Greece. The testbed is providing in a 24/7 fashion remotely accessible resources, targeting experimentally driven research in wireless and wired networks. In this sub-section, we provide a very brief description of the capabilities of the testbed. The testbed is providing access free-of-charge to over 100 static physical nodes, equipped with key networking technologies:

1. All the nodes are high-end PCs, equipped with Core-i7 processors and 8 GBs of RAM each, featuring at least two IEEE 802.11 a/b/g/n/ac cards, compatible with Open Source drivers (e.g. ath9/10k) used for WiFi research.

2. Two commercial off-the-shelf LTE access points are available for experimentation, along with a commercial off-the-shelf Evolved Packet Core (EPC). Both femtocells and EPC are programmable through the available testbed services. About half of the nodes are equipped with LTE dongles, that allow the establishment of an operator-grade LTE network, using testbed-specific SIM cards.

3. Over 20 different SDR devices are installed in the testbed, which are compatible RF front-ends for open source implementations of 4G/5G and beyond base stations such as OpenAirInterface [38]

4. Six mmWave devices are installed in the testbed, reachable from all the nodes, supporting the creation of high-throughput wireless point-to-point links. The nodes support beam steering allowing the formation of different topologies over the mmWave links.

5. All the testbed nodes are interconnected through three hardware OpenFlow switches, organized in a tree topology.

Figure 5.2 illustrates the deployed testbed infrastructure. The testbed is organized in three different setups: An indoor RF-isolated, an outdoor setup prone to uncontrolled external interference and an office setup with mild interference settings. Resources can be mixed from the different locations in order to create a versatile experimentation environment.

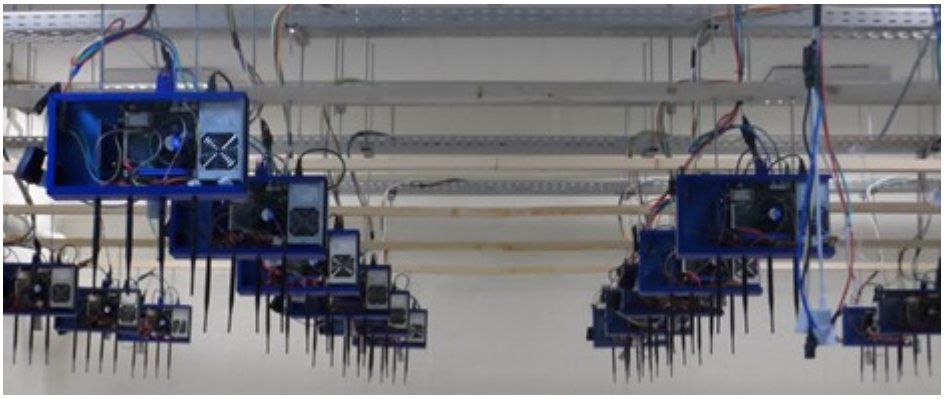

Figure 5.1: NITOS testbed overview; all the physical nodes are available to be used as bare metal machines. In the Figure, an overview of the indoor testbed nodes is shown. All the nodes of the testbed are in-terconnected under the same network. [15]

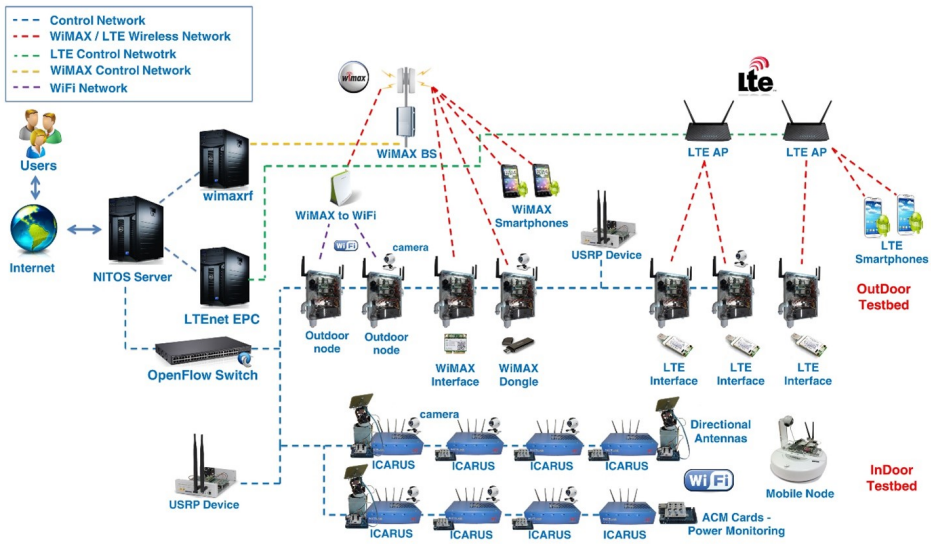

Figure 5.2: The ecosystem of NITOS testbed. [15]

5.1.2 OpenAirInterface

OpenAirInterface (OAI) [38] is an open-source platform which consists of 3GPP technologies such as LTE / 4G and 5G. The project includes developments for the core network (EPC) and access-network (EUTRAN) of 3GPP cellular networks. The software is written in the C programming language and can be compiled and executed on x86 computer systems. For the proper operation of Radio Access Networks, the computers must be equipped with SDR (Software Defined Radio) devices.

5.2 Experimental Setup Architecture

As shown in figure 5.3 our implementation comprises a Heterogeneous 5G Architecture with MEC functionality, which has been deployed in the Kubernetes framework. The experimental setup is based on the architecture of the specific research [8].

We leverage the Kubernetes framework which we analyzed extensively in chapter 4. The control-plane node is running as a Virtual Machine (VM) in the testbed infrastructure, which has direct network access to the wireless nodes of the testbed. Each of the testbed nodes is configured as a worker node for the K8s control-plane node, and therefore containers can be orchestrated on top of them. Regarding the selection of Kubernetes Network, we deploy the Flannel [39] CNI (Container Network Interface) plugin to the cluster. Flannel is a simple overlay network that acts as a network of containers and consequently as a pod network. By

default, it is not possible to add more than one network interface to Kubernetes pods. In figure 5.3 we see that the pods have more than one network interface. This feature is possible as we used the Multus plugin to add more than one interface to the pods. These network interfaces have been added to the pods as macvlan-static IPs, which means that the plug-in creates a sub-interface from the parent interface on the host. These host interfaces are bridge interfaces from a VLAN interface. The reason we used bridge interfaces, is that VMs that are deployed by the KubeVirt API only attach bridge-type network interfaces.

The architecture components end up being Kubernetes pods, as internally in them, run containerized applications based on Docker images that contain compiled OAI instances that together make a complete Heterogeneous 5G network. With the help of KubeVirt, we integrated the functionality of MEC in the architecture, as we deployed virtual machines as container workloads by expanding the ecosystem of our Kubernetes cluster. To accomplish a stateful Live Migration of the VMs, there must be L2 connectivity between the source and the target/destination VM. To achieve this we created bridge interfaces in Kubernetes Nodes. On these bridge interfaces, the VMs attached their own static IPs. Thus, in this way, we can maintain active IP / TCP connections during Live Migrations. Also to be able to test the functionality of Follow Me implementation that we analyzed in the section 3.6.2, we created mobility scenarios of the end-user by injecting constant delay in the interfaces of 3GPP and non-3GPP DUs with the help of the Linux Traffic Control tool (tc).

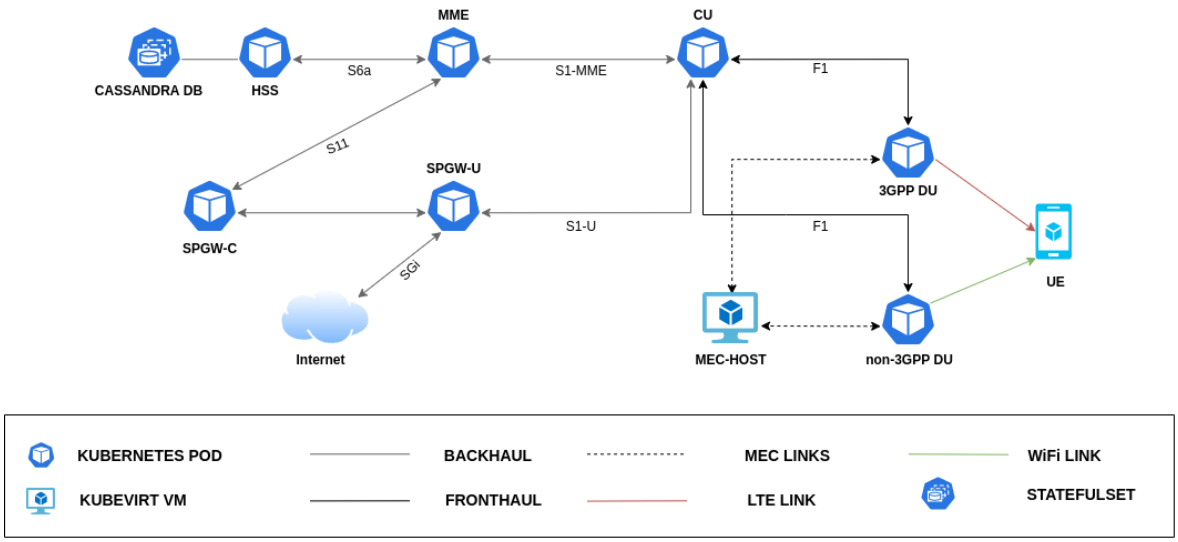

Figure 5.3: The deployment of Heterogeneous MEC-functional 5G Network on Kubernetes.

For the proper operation and management of our network from the backhaul to the fronthaul, specific Kubernetes objects were used for the respective architecture components. Below, we cover them extensively:

- **Cassandra DataBase**: It is the central database that contains information about all network operators' subscribers. It is a part of the operation of HSS. Therefore, to be able to communicate with the HSS pod, it runs a Kubernetes service so that the database is accessible from the HSS pod. Also, a StatefulSet is used to create the pods, to secure state information and other resilient data. Database configurations are parsed via ConfigMap. For the data persistence, local PV and PVC have used that mount the directory in which the data entries are written.

- **HSS**: HSS pod is responsible for session establishment support. To provide user authentication and access authorization it needs to communicate with the database and in our case with Cassandra database through its Service. To be able to access the Service running on the Cassandra Database, it has a ServiceAccount. Also through the ConfigMap the credentials for the connection to the database and configuration data to the application (e.g MME IP), are parsed. To create the Pod, we used the Deployment object as it provides us with the ability for scaling and better management of the pods. Finally, the HSS entity includes a Service for MME exchanging authentication information.

- **MME**: To exchange authentication information between the UE and the HSS, MME pod accesses the service running on the HSS pod via ServiceAccount. It includes network configurations for the multiple interfaces (S6a, S11, S1-MME) which are parsed via ConfigMap. In addition, it includes a Service over which the components (SPGW-U, SPGW-C) that share the S11 interface communicate with each other. The Deployment object was used to create the pods.

- **SGPGW-U**: It provides a Serving Gateway and PDN Gateway User plane. ConfigMap defines IPs for its interfaces (SGi, SPGW) and the UEs network IP. Also, accesses MME service through ServiceAccount. The pod is created through the Deployment object.

- **SPGW-C**: It provides a Serving Gateway and PDN Gateway Control plane. The SPGW-C pod is also created through the Deployment object. Since it is responsible for the UEs

IP address allocation, the UEs IP range and UEs DNS IP server are parsed in the configuration files of the application through ConfigMap.

- **CU**: CU pods are created through the Deployment Object and run the containerized version of OAI CU. The container runs a pre-built Image docker in which all the necessary libraries have been compiled and contains all the necessary configuration files for the proper operation of the Central Unit. It supports communication with non-3GPP DU technology like WiFi DU.

- **LTE DU**: Kubernetes Deployment object is used to create LTE DU pods. As we know, LTE DU includes both baseband processing and RF functions. Therefore, these pods are deployed only on Nodes that have a USRP device. To be able to map the USRP device port (USB port) of a Node to the pod, in the description file (YAML file), we run the container privileged and mount the path of the USB device in the container. With its creation, LTE DU is connected to MEC-Agent and to CU, which they should already be running on specific IPs and ports.

- **WiFi DU**: It is deployed through the Deployment object. Inside runs the WiFi Access Point with the help of Hostapd. Thus, these pods are deployed only on Nodes that have wireless chipsets (Atheros in our case) and drivers (Ath9k in our case). To be able to attach the WiFi device from the Node, we run the pod with the host network enabled. Also, the pod runs a server that is connected to the MEC agent along with the CU. The WiFi configuration utilizes 802.11n channels.

- **MEC Host**: The MEC host is a VM instance defined by the KubeVirt API. The VM contains a proper cloud-init network setup with static IPs. To replicate a large number of the VM workloads, we used containerDisk ephemeral storage.

5.3 Experimental Results

5.3.1 Live Monitoring of Resources

For the deployment of the heterogeneous containerized 5G network, we used 2 Kubernetes nodes which are Nitos testbed nodes. At one node we deployed the Backhaul components (Core Network & Central Unit), while at the other node we deployed all fronthaul

components including LTE DU and WiFi DU as well as the VM of MEC Host.

With the help of Prometheus, we exported some basic metrics that summarize the utilization of resources based on the respective deployment that takes place in Kubernetes Nodes. These measurements concern CPU, Memory, and Disk I / O. Then through Grafana we visualized these metrics in real-time.

In figure 5.4 we see the memory usage for the deployment of backhaul components. While in figure 5.7 we see the memory usage for the deployment of all fronthaul components. The memory usage is more in the core deployment due to the database running in the HSS component. However, as shown in figure 5.7 fronthaul deployment has more CPU usage than the corresponding CPU usage of core deployment 5.5. This is due to the processing power required by the antennas from WiFi, USRP devices. Disk I / O and Disk Usage do not exceed the limit and remain relatively low on both deployments as shown in the figures 5.6 and 5.9. From the following figures, we can conclude that given the resources available to Nitos nodes we can scale the 5G components, as the Memory / CPU / Disk usage is relatively low. This is due to the benefits of using application containers, as they have a lower virtualization overhead and are lightweight.

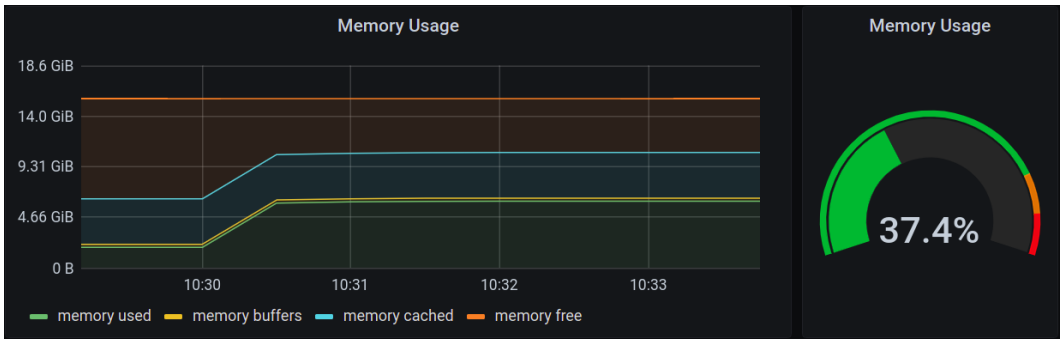

Figure 5.4: Memory Usage of the node on which the Core Network has been deployed.

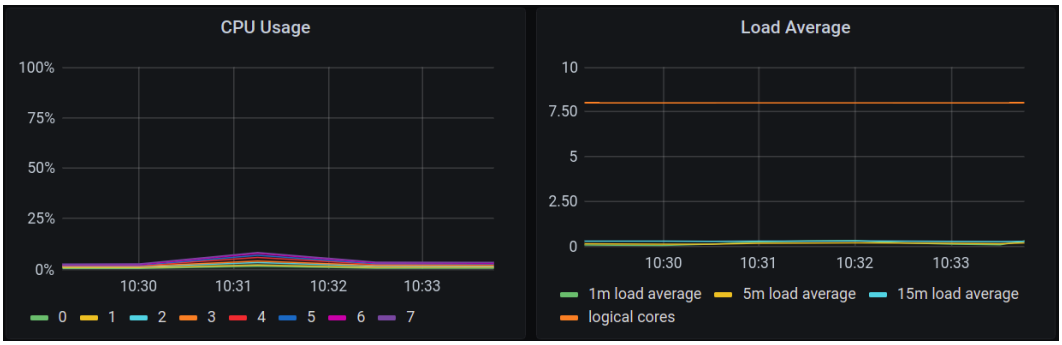

Figure 5.5: CPU Usage of the node on which the Core Network has been deployed.

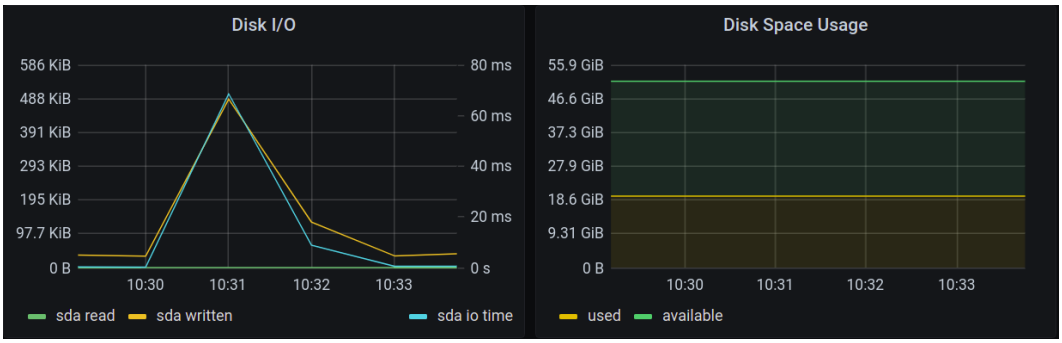

Figure 5.6: Disk I/O & Disk Usage of the node on which the Core Network has been deployed.

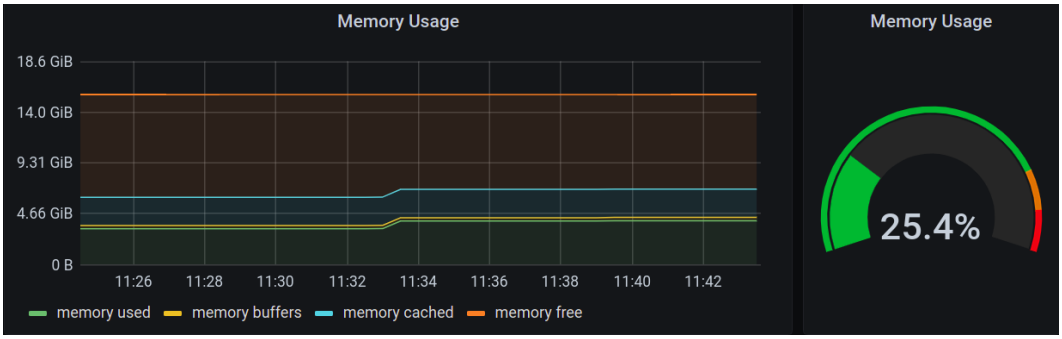

Figure 5.7: Memory Usage of the node on which the Fronthaul components have been deployed.

Figure 5.8: CPU Usage of the node on which the Fronthaul components have been deployed.

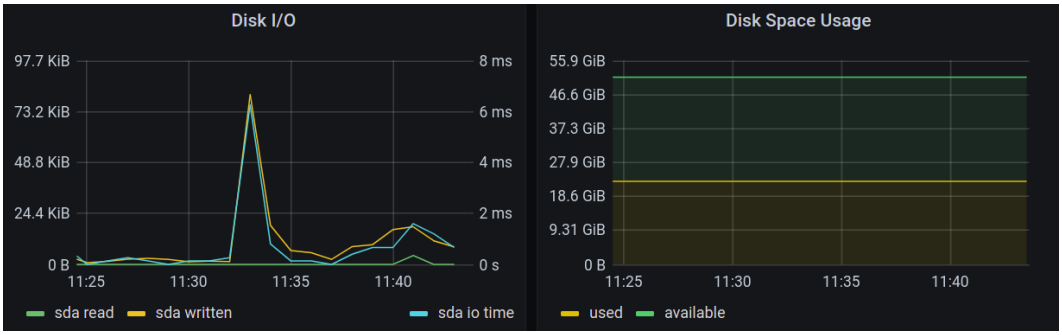

Figure 5.9: Disk I/O & Disk Usage of the node on which the Fronthaul components have been deployed.

5.3.2 Latency Measurements

For the evaluation part of our MEC deployment, we focus on measuring the overall latency for accessing the MEC services. The measurements are based on the latency between the multihomed UE (connected to LTE and WiFi DU) and the MEC service which is deployed either to the fronthaul or to the core network. We noticed that the latency measurements 5.1 of MEC services deployed on the fronthaul are slightly better than the MEC services deployed on the core network. This is because in the Nitos Testbed topology shown in figure 5.10 the core network container instances run on node081 which is relatively close to the fronthaul container instances running on node055 (UE operates on node050). In real-world scenarios, the core network is usually very far from the fronthaul. To emulate real-world scenarios, we tuned the delay on the link between the CU and the EPC by injecting 20ms delay. In addition, we can conclude that WiFi outperforms LTE for the cases of latency as shown in figure 5.11.

Table 5.1: Benchmark Characteristics (in ms)

	LTE to MEC-APP	WiFi to MEC-APP	LTE to EPC	WiFi to EPC	LTE to EPC (20ms)	WiFi to EPC (20ms)
Avg. RTT	25.6	5.28	27.9	5.88	46.03	26.08
Min. RTT	18.76	3.09	22.04	3.21	45.4	25.2
Max. RTT	32.3	12.8	40.8	13.4	54.07	34.9

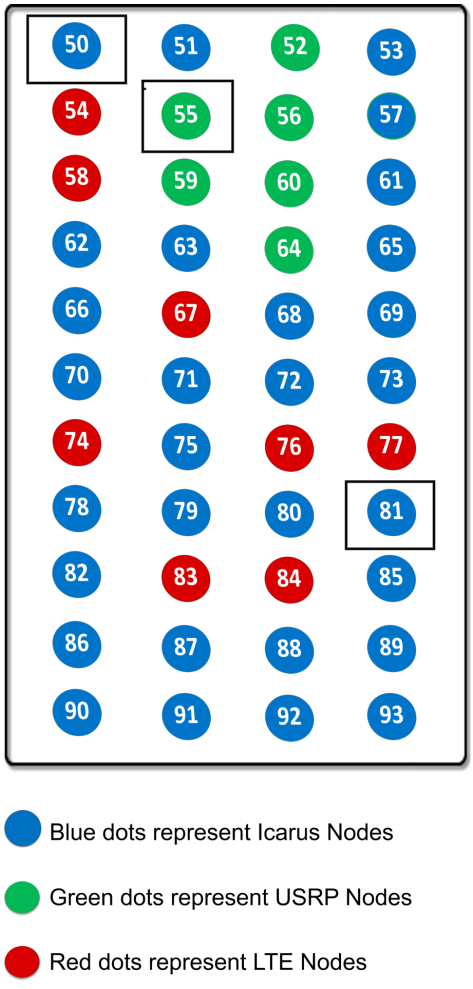

Figure 5.10: Nitos indoor testbed topology.

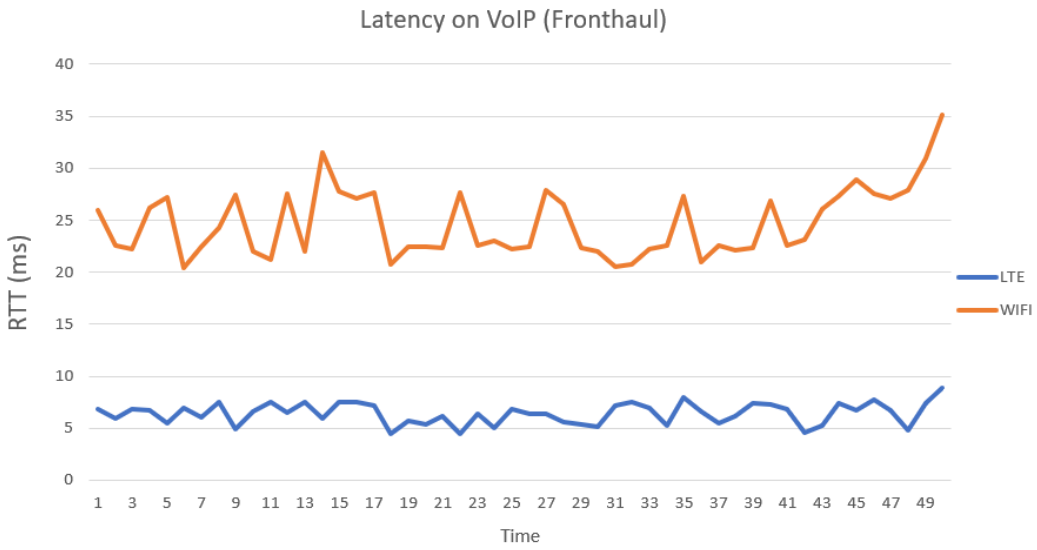

Figure 5.11: Latency on Fronthaul (VoIP application).

5.3.3 Migration Measurements

To test the functionality of our follow-me implementation, we used 2 MEC services. One was a simple chat application based on a TCP / IP socket and the other was an application called SIPp [40] that uses Session Initiation Protocol (SIP) to transfer VoIP packets. The SIP protocol can be carried by several transport layer protocols including Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Stream Control Transmission Protocol (SCTP).

To measure migration time, we created some scenarios:

- **S1**: VM includes MEC Agent and MEC Controller (Does not include MEC service)
- **S2**: The VM includes MEC Agent, MEC Controller and SIPp as MEC service.
- **S3**: The VM includes MEC Agent, MEC Controller and a text chat as MEC service.
- **S4**: Empty VM

In the following figure 5.12 we see the migration time of live migration of the VM for each of these scenarios. In all cases the migration throughput was 64 MiB / s.

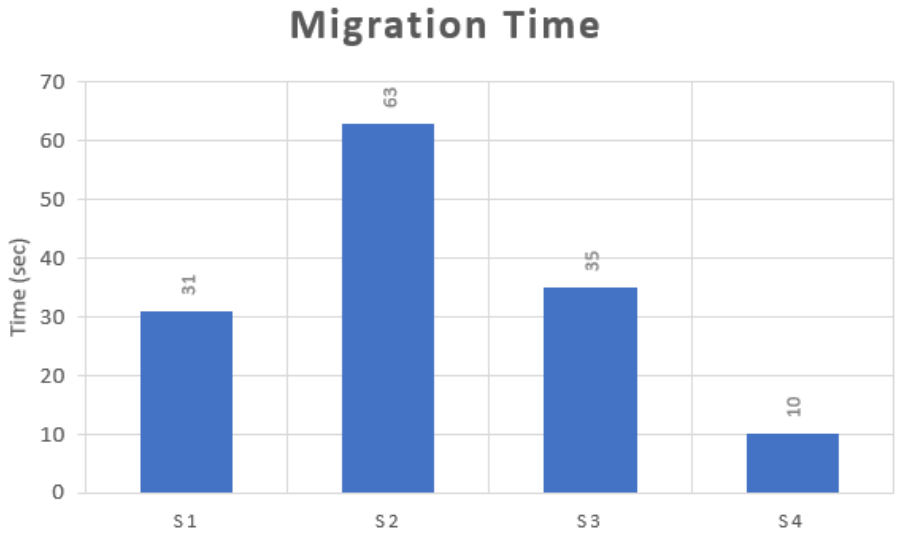

Figure 5.12: Migration Time for each scenarios.

Chapter 6

Conclusions

6.1 Summary and Conclusions

In this thesis, we deployed a Heterogeneous 5G Network with MEC functionality and implemented a MEC Follow-me scheme all in the Kubernetes Environment. We took advantage of the capabilities of the Kubernetes framework as we had the ability to scaling, load balancing, and monitoring the 5G components. In addition, with the KubeVirt add-on, we took advantage of the benefits of VMs in a containerized environment and use hybrid solutions offered by the coexistence of VMs and Containers. This combination was our basis for deploying a MEC ecosystem with Follow-me mechanisms.

With the Follow-me scheme that we implemented, we managed to keep the latency of the services and the migration downtime low. We also successfully maintained stateful connections during live migrations of MEC services. At the same time, we exploited the advantages of a heterogeneous 5G network by utilizing all the available transmission paths. Then with the help of Prometheus, we exported some metrics that summarize the utilization of resources and give us statistical information about the migration of services.

6.2 Future Work

The future work will enhance the proposed Follow-Me scheme with more intelligence to adapt to different network conditions and will add tracking user's position capabilities that will allow predicting user movement by using deep learning models. In addition, other policies will be added that will trigger the live migration of MEC services such as channel quality

indicator (CQI). This will allow the UE to give feedback on the quality of its connection and thus lead us to a better algorithm for deciding when migrations take place.

Bibliography

[1] Mohamed Habaebi, Jalel Chebil, A Al-Sakkaf, and Taha Dahawi. Comparison between scheduling techniques in long term evolution. *International Islamic University Malaysia Engineering Journal*, 14, 04 2013.

[2] K.V.N. Kavitha, Saikat Ghosh, Ankit Keetey, and Sibaram Khara. Error rate analysis of stbc-ofdm system with efficient channel coding technique at low snr. *International Journal of Applied Engineering Research*, 9:973–4562, 07 2014.

[3] Wireless tech. `https://wirelesstech.info`.

[4] Mohsin Khan and Valtteri Niemi. *AES and SNOW 3G are Feasible Choices for a 5G Phone from Energy Perspective*, pages 403–412. 01 2018.

[5] Mansoor Shafi, Andreas Molisch, Peter Smith, Thomas Haustein, Peiying Zhu, Prasan Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment and practice. *IEEE Journal on Selected Areas in Communications*, PP:1–1, 04 2017.

[6] Line M. P. Larsen, Aleksandra Checko, and H. Christiansen. A survey of the functional splits proposed for 5g mobile crosshaul networks. *IEEE Communications Surveys & Tutorials*, 21:146–172, 2019.

[7] Nikos Makris, Christos Zarafetas, Pavlos Basaras, Thanasis Korakis, Navid Nikaein, and Leandros Tassiulas. Cloud-based convergence of heterogeneous rans in 5g disaggregated architectures. 05 2018.

[8] Nikos Makris, Virgilios Passas, Christos Nanis, and Thanasis Korakis. On minimizing service access latency: Employing mec on the fronthaul of heterogeneous 5g architectures. 07 2019.

[9] Michael Hines and Kartik Gopalan. Post-copy based live virtual machine migration using pre-paging and dynamic self-ballooning. pages 51–60, 01 2009.

[10] Docker. `https://docs.docker.com/get-started/overview/`.

[11] Kubernetes. `https://kubernetes.io/docs/concepts/overview/components/`.

[12] Multus. `https://github.com/k8snetworkplumbingwg/multus-cni`.

[13] Kubevirt. `https://kubevirt.io/`.

[14] Prometheus. `https://prometheus.io/`.

[15] Nitos testbed. `http://nitos.inf.uth.gr`.

[16] Sriganesh Rao and Ramjee Prasad. Impact of 5g technologies on industry 4.0. *Wireless Personal Communications*, 100:1–15, 05 2018.

[17] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, PP:1–1, 01 2020.

[18] Multi-access edge computing. `https://www.etsi.org/technologies/multi-access-edge-computing`.

[19] A. Abioye, M. Joseph, and Hendrik Ferreira. Comparative study of 3g and 4glte network. *Journal of Advances in Computer Networks*, 3:247–250, 01 2015.

[20] S.S. Hussain, S.M. Yaseen, and Koushik Barman. An overview of massive mimo system in 5g. 9:4957–4968, 01 2016.

[21] Nikos Makris, Christos Zarafetas, Kostas Choumas, Paris Flegkas, and Thanasis Korakis. Virtualized heterogeneous 5g cloud-ran deployment over redundant wireless links. 07 2019.

[22] Nikos Makris, Pavlos Basaras, Thanasis Korakis, Navid Nikaein, and Leandros Tassiulas. Experimental evaluation of functional splits for 5g cloud-rans. *IEEE Conference Record - International Conference on Communications*, 05 2017.

[23] Alberto Martínez Alba, Jorge Humberto Gomez Velasquez, and W. Kellerer. An adaptive functional split in 5g networks. *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 410–416, 2019.

[24] T. Taleb, Konstantinos Samdanis, B. Mada, H. Flinck, Sunny Dutta, and D. Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19:1657–1681, 2017.

[25] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE Access*, PP:1–1, 01 2020.

[26] Etsi gs mec. `https://www.etsi.org/deliver/etsi_gs/MEC/001_099/026/02.01.01_60/gs_MEC026v020101p.pdf`.

[27] Nikos Makris, Virgilios Passas, Thanasis Korakis, and Leandros Tassiulas. Employing mec in the cloud-ran: An experimental analysis. pages 15–19, 10 2018.

[28] Miguel Xavier, Marcelo Neves, Fábio Rossi, Tiago Ferreto, Timoteo Lange, and Cesar De Rose. Performance evaluation of container-based virtualization for high performance computing environments. pages 233–240, 02 2013.

[29] Tung Doan, Giang Nguyen, Hani Salah, Sreekrishna Pandi, Michael Jarschel, Rastin Pries, and Frank Fitzek. Containers vs virtual machines: Choosing the right virtualization technology for mobile edge cloud. pages 46–52, 09 2019.

[30] Tarik Taleb, K. Samdanis, Badr Eddine Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge architecture & orchestration. *IEEE Communications Surveys & Tutorials*, PP:1–1, 05 2017.

[31] Prakash Nayak, Deepak Garg, Abhishek Shakya, and Poonam Saini. A research paper of existing live vm migration and a hybrid vm migration approach in cloud computing. pages 720–725, 05 2018.

[32] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[33] Jintian Hu, Gaocai Wang, Xiaotong Xu, and Yuting Lu. Study on dynamic service migration strategy with energy optimization in mobile edge computing. *Mobile Information Systems*, 2019:5794870, Oct 2019.

[34] Run Yang, Hui He, and Weizhe Zhang. Multitier service migration framework based on mobility prediction in mobile edge computing. *Wireless Communications and Mobile Computing*, 2021:6638730, Apr 2021.

[35] Tung V. Doan, Alexander Kropp, Giang T. Nguyen, Hani Salah, and Fitzek Frank H. P. Reusing sub-chains of network functions to support mec services. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–8, 2019.

[36] Cong Shen, Cem Tekin, and Mihaela van der Schaar. A non-stochastic learning approach to energy efficient mobility management. *IEEE Journal on Selected Areas in Communications*, 34(12):3854–3868, 2016.

[37] Grafana. `https://grafana.com/`.

[38] Navid Nikaein, Mahesh K. Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. Openairinterface. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[39] Flannel repository. `https://github.com/flannel-io/flannel`.

[40] Sipp. `http://sipp.sourceforge.net/`.