Using machine learning for dynamic resource orchestration & task scheduling, in a radio access network based edge environment

Jude Kojo Amponsah Fletcher

St Cross College
University of Oxford

A thesis submitted to the Department of Engineering Science, in fulfilment of the requirements for the degree of Doctor of Philosophy

Trinity 2021

Abstract

The evolution in mobile wireless communication generations, continues to gift the world with new telecommunication capabilities towards how people live and work. For instance, the roll out of the fifth generation (5G) promises a much-enhanced cloud-native application service, through improved communication speed, higher bandwidth, improved capacity for more connected devices and many more. These exciting new 5G features have given numerous vertical and horizontal industries a reason to explore different ways of delivering value, through emerging cloud-native applications that run on access devices (i.e. the Internet of Things (IoTs) and mobile devices). These cloud-native applications such Virtual Reality, Vehicle to everything communication (V2X), artificial intelligence, video analytics and so on however have strict performance requirements for extremely low latency (10 milliseconds and below) and higher bandwidth, that 5G alone cannot deliver. Unfortunately, the traditional cloud computing and radio access network setups do not sufficiently address these key communication needs which are crucial to the performance of these cloud-native applications. There is therefore an urgency to enhance and optimize the traditional mobile telecommunication network architecture, to meet these performance needs. This thesis seeks to demonstrate how we have developed a facility called Multi-Access Edge Computing with Cloud Radio Access Networks (MECRAN) to address this issue. MECRAN is based on the edge computing paradigm and uses machine learning to optimize the round-trip delivery of data at low latency, by dynamically scheduling cloud-native applications, to run in close proximity to a mobile user, at the edge of the radio access network.

Keywords: Multi-Access Edge Computing, Mobile Edge Computing, Cloud-Based/ Centralized Radio Access Network (C-RAN), Artificial Intelligence, Federated Cloud, Security, Containers, Task Scheduling, Ceph, Cloud Computing, 5G, Energy Reduction, MEC Orchestration, eMBB, URLLC, mMTC

Related Published Abstracts

Whilst it is not the intention to submit this DPhil thesis in an integrated format, it is useful to highlight that this document extends some of the content published and / or submitted to international conferences and journals throughout the course of my study. I confirm all material including the core source codes are my original work and I remain the first author on all publications.

[1] Jude Fletcher, David Wallom. 2019. *Deep Learning based task scheduling in a Cloud RAN enabled edge environment.* SEC '19: The Fourth ACM/IEEE Symposium on Edge Computing Proceedings, November 7 {9, 2019, Arlington, VA, USA, 3 pages. https://doi.org/10.1145/3318216.3363319

[2] Jude Fletcher, David Wallom. 2019. *Using machine learning to orchestrate cloud resources in a RAN enabled edge environment.* SenSys '19: The 17th ACM Conference on Embedded Networked Sensor Systems Proceedings, November 10 13, 2019, New York, NY, USA, 3 pages. https://doi.org/10.1145/3356250.3361930

Abstract [1] was also accepted in a workshop at the Conference on Neural Information Processing (NeurIPS 2019). Two additional abstracts were also accepted at the 12th IEEE/ACM International Conference on Utility and Cloud Computing Conference (UCC, Auckland, New Zealand) and at the 12th IEEE International Conference on Service-Oriented Computing and Applications (SOCA, Kaohsiung, Taiwan) in 2019. I was unable to attend both UCC and SOCA conferences as they coincided with SEC [1] and SenSys [2] that was taking place in North America.

Using machine learning for dynamic resource orchestration & task scheduling, in a radio access network based edge environment

Jude Kojo Amponsah Fletcher

St Cross College

University of Oxford

Supervised by Professor David C. H. Wallom

A thesis submitted to the Department of Engineering Science, in fulfilment of the requirements for the degree of

Doctor of Philosophy

Trinity 2021

This thesis is dedicated to

My father Mr Isaac Fletcher,
for giving me a father's blessing and teaching
me the value of positive thinking and hard work.

My mother Mrs Rita Fletcher,
for instilling in us an appreciation for the value of education.

My brothers
Dr Gerald Fletcher,
Dr Eugene Fletcher,
Edward Fletcher & Roderick Obiri,
for being an effective support system and my reason to persevere.

Acknowledgements

I would like to thank my supervisor Professor David Wallom for his relentless support and much appreciated counsel, during the writing of my dissertation and indeed throughout my DPhil study. He kept me on the right track and I have thoroughly enjoyed this remarkable journey and growth at Oxford, under his supervision.

The Oxford e-Research Centre (OeRC), the Department of Engineering Science and St Cross College deserve my deepest gratitude for their pastoral care, academic advice and enduring diverse support which set me up for success at the university. The formidable staff across these departments and college are a treasured part of my time at Oxford and I am truly grateful for the role they played.

This project would not have taken place without the funding from the Ghana National Petroleum Corporation Foundation. Thank you Dr Kofi Koduah Sarpong (Ag. CEO – Ghana National Petroleum Corporation) for believing in me and effectively giving me the opportunity to study. I am eternally grateful and humbled by your consideration.

Finally, I am grateful for and to my family. My mother Mrs Rita Fletcher, you labored abundantly in prayer and persistently extended the love and support I needed for this journey. My brothers Dr Gerald Fletcher, Dr Eugene Fletcher, Edward Fletcher and Roderick Obiri; you motivated me to work hard and persevere each day and I would not be where I am without you. It is my sincere admission that aside the distracting consequences of the pandemic and the war in Ukraine, the most difficult time during the course of this study was losing our father Mr Isaac Fletcher. He has been a big part of this DPhil from the beginning and I am pleased to be able to bring it to a successful conclusion, knowing fully that, I have made him proud with the support of my family. Dada, may you continue to rest in peace.

Abstract

The evolution in mobile wireless communication generations, continues to gift the world with new telecommunication capabilities towards how people live and work. For instance, the roll out of the fifth generation (5G) promises a much-enhanced cloud-native application service, through improved communication speed, higher bandwidth, improved capacity for more connected devices and many more. These exciting new 5G features have given numerous vertical and horizontal industries a reason to explore different ways of delivering value, through emerging cloud-native applications that run on access devices (i.e. the Internet of Things (IoTs) and mobile devices). These cloud-native applications such Virtual Reality, Vehicle to everything communication (V2X), artificial intelligence, video analytics and so on however have strict performance requirements for extremely low latency (10 milliseconds and below) and higher bandwidth, that 5G alone cannot deliver. Unfortunately, the traditional cloud computing and radio access network setups do not sufficiently address these key communication needs which are crucial to the performance of these cloud-native applications. There is therefore an urgency to enhance and optimize the traditional mobile telecommunication network architecture, to meet these performance needs. This thesis seeks to demonstrate how we have developed a facility called Multi-Access Edge Computing with Cloud Radio Access Networks (MECRAN) to address this issue. MECRAN is based on the edge computing paradigm and uses machine learning to optimize the round-trip delivery of data at low latency, by dynamically scheduling cloud-native applications, to run in close proximity to a mobile user, at the edge of the radio access network.

Keywords: Multi-Access Edge Computing, Mobile Edge Computing, Cloud-Based/ Centralized Radio Access Network (C-RAN), Artificial Intelligence, Federated Cloud, Security, Containers, Task Scheduling, Ceph, Cloud Computing, 5G, Energy Reduction, MEC Orchestration, eMBB, URLLC, mMTC

Related Published Abstracts

Whilst it is not the intention to submit this DPhil thesis in an integrated format, it is useful to highlight that this document extends some of the content published and / or submitted to international conferences and journals throughout the course of my study. I confirm all material including the core source codes are my original work and I remain the first author on all publications.

[1] Jude Fletcher, David Wallom. 2019. *Deep Learning based task scheduling in a Cloud RAN enabled edge environment.* SEC '19: The Fourth ACM/IEEE Symposium on Edge Computing Proceedings, November 7 {9, 2019, Arlington, VA, USA, 3 pages. https://doi.org/10.1145/3318216.3363319

[2] Jude Fletcher, David Wallom. 2019. *Using machine learning to orchestrate cloud resources in a RAN enabled edge environment.* SenSys '19: The 17th ACM Conference on Embedded Networked Sensor Systems Proceedings, November 10 13, 2019, New York, NY, USA, 3 pages. https://doi.org/10.1145/3356250.3361930

Abstract [1] was also accepted in a workshop at the Conference on Neural Information Processing (NeurIPS 2019). Two additional abstracts were also accepted at the 12th IEEE/ACM International Conference on Utility and Cloud Computing Conference (UCC, Auckland, New Zealand) and at the 12th IEEE International Conference on Service-Oriented Computing and Applications (SOCA, Kaohsiung, Taiwan) in 2019. I was unable to attend both UCC and SOCA conferences as they coincided with SEC [1] and SenSys [2] that was taking place in North America.

Contents

List of Figures

List of Abbreviations

1G . . . . . First generation of wireless cellular technology.

2G . . . . . Second generation of wireless cellular technology.

3G . . . . . Third generation of wireless cellular technology.

3GPP . . . . . Third Generation Partnership Project.

4G . . . . . Fourth generation of wireless cellular technology.

5G . . . . . Fifth generation of wireless cellular technology.

AI . . . . . Artificial Intelligence.

AM . . . . . Amplitude Modulation.

AMPS . . . . . Advanced Mobile Phone System.

AR . . . . . Augmented Reality.

AP . . . . . Access point.

BBU . . . . . Baseband Unit.

BS . . . . . Base station.

CAPEX . . . . Capital expenditure.

CDMA . . . . Code Division Multiple Access.

CN . . . . . Core network.

C-RAN . . . . Cloud-Based/ Centralized Radio Access Network.

D2D . . . . . Device-to-device.

DC . . . . . Dual Connectivity.

EDGE . . . . . Enhanced Data GSM Evolution.

eMBB . . . . . enhanced Mobile Broad Band.

eNB . . . . . Evolved Node B.

ETACS . . . . European Total Access Communication System.

FDMA . . . . Frequency Division Multiple Access.

FM . . . . . Frequency Modulation.

Gbps . . . . . Gigabits per second.

GHz . . . . . Gigahertz.

GPRS . . . . . General Packet Radio Service.

GSM . . . . . Global System for Mobile communication.

HSDPA . . . . High Speed Downlink Packet Access.

HSE . . . . . High spectral efficiency or bandwidth.

HSPA+ . . . . High Speed Packet Access plus.

HSUPA . . . . High Speed Uplink Packet Access.

IaaS . . . . . Infrastructure-as-a-Service.

IMT-2020 . . . International Mobile Telecommunications for 2020 and beyond.

IoTs . . . . . Internet of Things.

IP . . . . . Internet Protocol.

ITU . . . . . International Telecommunication Union.

ITU-R . . . . . International Telecommunications Union-Radiocommunication Sector.

Kbps . . . . . Kilobits per second.

Killer Apps . . Killer Applications.

LTE . . . . . Long Term Evolution.

Mbps . . . . . Megabits per second.

MEC . . . . . Multi-Access Edge Computing (Formerly Mobile Edge Computing).

MHz . . . . . Megahertz.

MIMO . . . . . Multiple-input multiple-output.

mMTC . . . . massive Machine Type Communications.

mmWave . . . Millimeter wave.

ms . . . . . Milliseconds.

NMTS . . . . . Nordic Mobile Phone System.

NTT . . . . . Nippon Telephone and Telegraph company.

OPEX . . . . . Operational expenditure.

ORI . . . . . Open radio equipment interface.

P2P . . . . . Point to point.

PaaS . . . . . Platform-as-a-Service.

PHY . . . . . Physical.

PoC . . . . . Proof-of-concept.

RAN . . . . . Radio access network.

RF . . . . . Radio frequency.

RRC . . . . . Radio resource control.

RRH . . . . . Radio Remote Head.

RTT . . . . . Round Trip Time.

SaaS . . . . . Software-as-a-Service.

SMS . . . . . Short Message Service.

TACS . . . . . Total Access Communication System.

TTFB . . . . . Time to First Byte.

UMTS . . . . . Universal Mobile Terrestrial / Telecommunication Systems.

URLLC . . . . Ultra Reliability and Low Latency Communications.

UE . . . . . User equipment.

UP . . . . . User plane.

V2X . . . . . Vehicle to everything.

VoLTE . . . . voice over LTE network.

VR . . . . . Virtual Reality.

WAN . . . . . Wide Area Network.

WiGig . . . . . Wireless Gigabit.

WLAN . . . . Wireless local area network.

1 Introduction

Contents

1.1 Research Motivation

Technological advancements and research have led to the emergence of a number of technological trends, which are purposed to transform domains and solve problems, thereby improving business operations as well as enhancing society. These trends which include *Artificial Intelligence (AI)*, *Augmented Reality (AR)*, *Tactile Internet*, *the Internet of Things (IoT)* are also often referred to as *Killer Applications or Killer Apps* due to their data-heavy and compute-intensive nature. Of even greater interest, is the ability for these emerging technologies and telecommunication services to converge and in combination, solve more complex problems in both vertical and horizontal markets [3].

A typical example of such convergence is: an autonomous train operating by

combining IoT sensors (such as *LiDAR*), fifth generation mobile networks (*5G*) and AI. It has been established that the enormous scale and complex nature of the problems being addressed today require high computational power, high spectral efficiency or bandwidth, low latency and higher network quality of service [4]. Another relevant observation is that the adoption rate of access devices such as IoTs and mobile devices for social, domestic as well as industrial purposes is on the rise and predicted to continue to grow exponentially, to over 50 billion connected devices by the end of 2020 [3]. Therefore, it is becoming increasingly important for large number of access devices to be able to connect simultaneously to a network [5]. Unfortunately, the traditional cloud computing and radio access network setups do not sufficiently address these key communication issues that are crucial to the performance of these emerging technologies.

MEC is a novel paradigm that moves compute, network and storage closer to the source of data, mainly to reduce latency [4]. C-RAN is a virtualized network architecture, where baseband units are separated and centralized from individual base stations, enabling easier deployment and scaling [6]. MECRAN is well placed to address these network communication challenges. Thus, this research endorses the idea of MEC + C- RAN as a single unit [6] and more importantly, it addresses the issues of task scheduling and resource orchestration using neural network and machine learning techniques within this [6] improved system.

MECRAN addresses latency concerns in quite a wide variety of uses cases such the railway and smart factory use cases in chapter 7. Another scenario of personal interest that would benefit unequivocally from the capabilities of MECRAN is a smart health infrastructure in tropical countries for disease research and control. Citing malaria research in Ghana as an example, currently researchers (both local such as from the *Noguchi Research Institute* [7][8] and international partners such the *World Health Organization* [9]) have to travel to distant remote locations to collect blood samples for malaria research. Unfortunately the actual study does not commence immediately or the research cannot be performed in those remote locations due to a lack of computational resources and high internet cost [10]. As a

consequence, researchers need to travel long distances back to the capital city Accra to progress their work. This already puts a strain on the effectiveness of samples collected, the researchers and it also introduces delays in delivering time-sensitive analysis for urgent case treatment. MECRAN can address this need as it provides a platform that is capable of enabling a very effective data collection system especially in the Tropics where internet is hugely expensive, by not only ensuring lower data management costs, low latency and high bandwidth, but also providing the speed, convenience and an interoperable platform that could also allow the international partners or agencies to engage productively even remotely. MECRAN could be an essential addition to national digital health platforms for developing countries. As among other benefits, MECRAN opens up opportunities to overcome high internet costs [10] and it is an architecture that could support ambitious health improvement agenda such as the United Nation's plans to transform the world through its Sustainable Development Goals (SDGs) [11][12][13] and many other schemes by the World Health Organization (WHO) around HIV/AIDS and the 6 childhood killer diseases [14]. As per WHO, surveillance is an indispensable requirement in eliminating malaria [9] and the capabilities of MECRAN can make this and the Global Technical Strategy for Malaria 2016-2030 (GTS) [9] possible.

1.2 Key Challenges

Computer networking underpins many industries which operate and communicate with technology [5]. This includes a plethora of industries such as banking & capital markets, agriculture, supply chain, gaming, automotive, to mention but a few. As these industries with time evolve to meet business needs, there is a digital transformation appetite that needs to be satisfied to sustain their evolution. Consequently, network transformations are necessary as they support digital transformations in enabling system communication and moving data around [3]. One of the major digital transformation of our time is motivated by the scope and requirements of 3 service types upon which 5G has been developed: enhanced Mobile Broad Band (eMBB), Ultra Reliability and Low Latency Communications (URLLC), and massive Machine Type Communications (mMTC) [3] [15]. The biggest challenge facing the telecom industry as a result of this, is network resource orchestration to meet demand at a high standard of quality of service especially in use cases where application users are constantly mobile [16].

1.3 Research Questions

Due to the scale, heterogeneity and complexity of the emerging 5G networks, it is essential for the network management solutions to be largely automated and distinctly intelligent from a networking point of view [5]. The intelligence of the network should enable it to assemble massive amounts of high dimensional data, process the "relevant" data automatically and make a set of decisions for subsequent actions. As an over simplified example, video analytics applications using IoTs are transforming the security and surveillance industry. Consider a surveillance camera that operates 24 hours and 7 days a week. Instead of transmitting entire video streams to the core network for decision making, ideally the data processing should be done locally at the data source, based upon expected visual events and triggering appropriate actions. The decisions on actions based on the transformed

data, can then be transmitted to the core cloud. This has many advantages in terms of latency, bandwidth, energy savings and storing only important data points.

MEC and C-RAN are already capable of enabling use cases where the application runs from the same location (such as the security and surveillance use case) or use cases where application users are confined with a certain space with hardly any movement. There is however a challenge with use cases that have a high level of user mobility as they easily introduce distance and latency between users and the server hosting the tasks (*application, service or workload*) being accessed by users. For example, a user accessing an application whilst traveling on a train, augmented reality, robot performing operations on the shop floor of a large manufacturing/ logistics warehouse are all typical examples of use cases with high level of user mobility. The key question here is how to dynamically manage network resources to support such use cases without compromising on the quality of service.

1.4 Problem Statement

MEC and C-RAN technologies together address many of the key communication issues that are crucial to the performance of emerging technology use cases. However, for use cases that require some level of mobility, a novel way of managing network resources across both MEC and C-RAN is needed in order to maintain the necessary quality of service. MEC and C-RAN are promising technologies and it is worth investigating the possibility of taking advantage of their virtualised environment to introduce dynamic resource orchestration and task scheduling automation to tackle the high latency problem outlined earlier.

1.5 Research Goal & Key Contributions

ETSI in one of its latest publications [16], considers the idea of using the same C-RAN infrastructure for MEC as a topic of 'ongoing discussion'. Even though ETSI highlights some potential challenges in the same publication, there has not been any real practical implementations of this phenomenon to validate its feasibility. There are two goals in this research work: to practically validate the feasibility of consolidating MEC and C-RAN (MECRAN); devise innovate and efficient ways of performing resource orchestration and task scheduling in MECRAN.

The key contribution and focus of this work, is developing a new network management strategy for use cases with mobility requirements, by adopting innovative machine learning algorithms (i.e. neural networks and deep reinforcement learning) to effectively facilitate resource orchestration and task scheduling decisions. These decisions are based on task performance requirements (around compute, network and storage) versus available MEC and C-RAN resources.

2

Background / Literature Review

Contents

2.1 Introduction: Intelligent application scheduling at the edge of the radio access network

Many of the existing cloud-native applications, services, or workloads (collectively referred to as *applications* in this thesis) perform well with latencies greater than 10 ms (milliseconds) (as in Fig. 2.1) and therefore can be delivered by legacy networks [16].

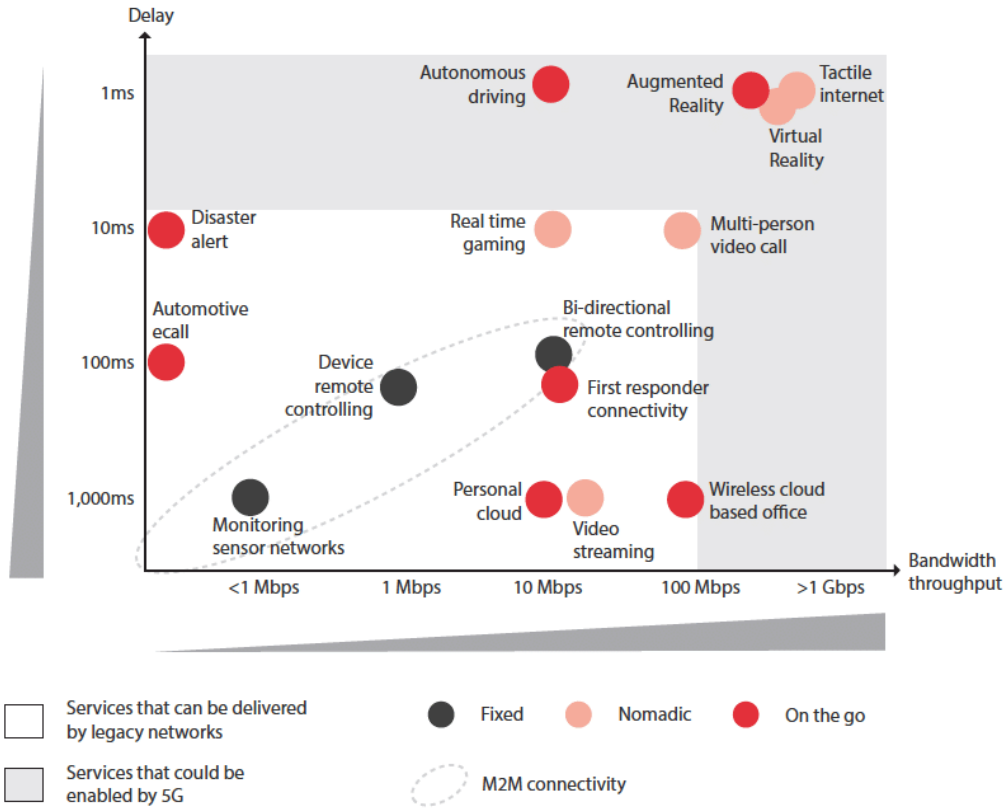

Figure 2.1: Latency and bandwidth requirements for some popular emerging digital applications [15][17].

The evolution and innovation in mobile wireless communication technology has enabled new digital opportunities [3], giving the mobile network community reason to explore more efficient ways of delivering value, through emerging game-changing cloud-native applications [5]. However, these emerging technologies such

as augmented reality, virtual reality, and artificial intelligence are sensitive to per-packet delivery times and therefore require much lower latencies (less than or equal to 10 ms, as in Fig. 2.1) as well as high bandwidth for performance [5][15][17]. Such paired requirements are not being delivered or met by the legacy cloud computing and radio access networks [16].

The current rollout of fifth generation (5G) mobile networks promises much-enhanced service capabilities focused on how people live and work; through improved capacity serving more connected devices, lower latency, higher bandwidth, and efficient energy usage [5]. However, despite the fact that 5G offers lower latency, the constant low round-trip latency required cannot be achieved with 5G alone [16]. This is because communicating clients and hosts will not always be in the same geographic location and therefore the physical distance between them can cause delays (latency) in exchanging data packets [6][16]. It is common for technology and cloud computing companies to have their master data centre in one country (such as the United States of America), and serve customers across different parts the globe from this data centre. How can the industry therefore supplement the cloud computing and radio access network architecture, to be able to meet the low latency and high bandwidth performance needs of emerging cloud-native applications, without compromising on the *quality of service* (QoS)?

Our research work tackles this problem using machine learning in a *Multi-access Edge Computing* (MEC) and *Cloud Radio Access Network* (C-RAN) environment, to reduce latency as well as facilitate highly efficient network operation, to offer a much-improved user experience. It is important to address this gap as this is critical for a large number of use cases pertaining to next generation or emerging applications such as autonomous vehicles or self-driving cars, to perform reliably [4]. An autonomous vehicle on the move typically processes the large volumes of collected data from its set of IoTs (such as *LiDAR*) in the cloud before making key navigation decisions [18][19]. This decision-making process needs to be in real-time so that a car can for example stop in time as it approaches pedestrians without causing fatal accidents [20]. To facilitate this real-time experience, these emerging applications

require latencies equal to or lower than 10 ms [3][5] and this thesis is designed to achieve this performance requirement. Using machine learning in a Multi-access Edge Computing (MEC) and Cloud Radio Access Network (C-RAN) collocated environment makes our approach unique and effectively facilitates this dynamism.

In this chapter, we will be expanding on how wireless communication technology has evolved over time, summarising all the new features and opportunities with 5G. We will also explore why 5G alone, coupled with the current cloud computing and radio access network landscape does not solve the latency problem. Additionally, we will be reviewing the unique features of machine learning, MEC and C-RAN to understand their collective advantage as an approach to the problem, over other approaches. Finally, we will review the various approaches and techniques other researchers have tried to solve the latency problem.

2.2 The foundations of mobile wireless communications technology

Wireless communication refers to communication between two or more entities over a distance via electromagnetic waves [specifically radio frequency (RF)] from a *transmitter* to a *receiver* [21]. This is because electromagnetic waves carry energy as they propagate through vacuum or space, and this energy can be used to communicate [21].

Michael Faraday in the early 19th century took huge strides in experimenting and laying the key foundations of an electric current, a magnetic field and mechanical motion [22]. These laws of electricity and magnetism today are expressed by the properties of electric and magnetic fields, as seen in equations (2.1), (2.2), (2.3) and (2.4). Due to Faraday's work, we understand that an electrically charged particle placed at a given point in space, generates an electric field around it and the strength of that field at a given position is represented by a vector [23]. The

magnitude and direction of this vector represents the magnitude and direction of the force that would experience a positive charge of 1 coulomb [24].

$$\nabla \circ \mathbf{D} = \rho \tag{2.1}$$

$$\nabla \circ \mathbf{B} = 0 \tag{2.2}$$

$$\nabla \times \mathbf{H} - \frac{\partial \mathbf{D}}{\partial t} = \mathbf{J} \tag{2.3}$$

$$\nabla \times \mathbf{E} + \frac{\partial \mathbf{B}}{\partial t} = 0 \tag{2.4}$$

In modern day notations, equations (2.1) [Gauss' Law], (2.2) [Gauss' Law for magnetism], (2.3) [Ampere-Maxwell Law] and (2.4) [Faraday's Law] are the four key expressions that outline the properties and relationship between electric and magnetic fields [25]. These are collectively referred to as *Maxwell's Equations*. The quantity **E** refers to the electric field whilst **D** refers to the displaced electric and also highlights how the electric charges become polarized [25]. Quantity **H** is the magnetic field whilst **B** is the magnetic field flux density [26] [25]. In conclusion, Maxwell's equations summarize how a change in electric field generates a change in magnetic field and vice versa [26].

In the second part of the 19th century James Clerk Maxwell however extended this work by Faraday and others [27], demonstrating that a change in electric field generates a change in magnetic field; so the oscillation of an electric field generates an oscillation of a magnetic field too [28]. This can be observed in Fig. 2.2. These changing fields form *electromagnetic waves* (EM). The entangled electric and magnetic waves simultaneously oscillate in perpendicular planes with respect to each other [24], as shown in the cosine or sine curve in Fig. 2.2. This forms the basis of how a signal is transferred through space at the speed of light (i.e., 3 x10^8 ms^{-1}) and essentially the basis of *mobile wireless communication technology* [21].

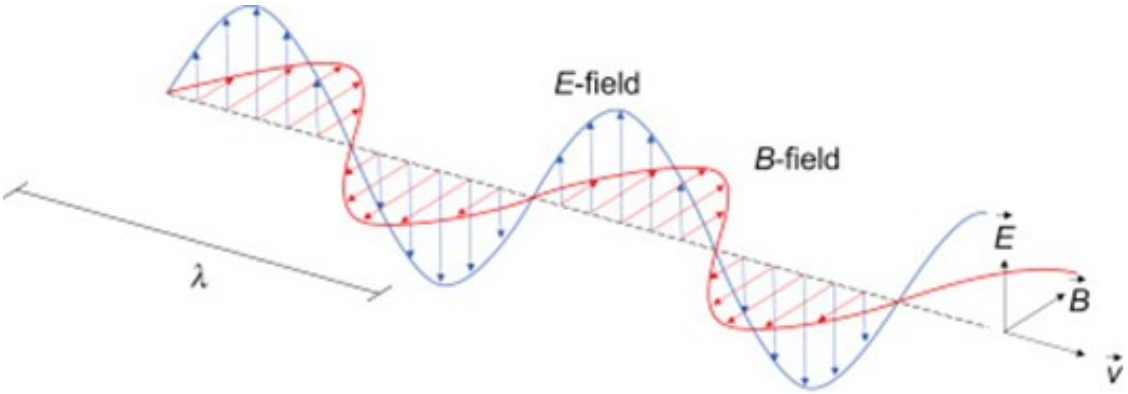

Figure 2.2: A 3D visual of an oscillating electromagnetic field, also known as electromagnetic waves (EM). Where **E** is the electric field and **B** the magnetic field [29].

During the experimentation of Faraday, he speculated that light was electrical in nature [21]. While Maxwell's work further provided unwavering grounds for this hypothesis, Heinrich Hertz finally proved this and summarised this idea in a spectacular way: "The connection between light and electricity is now established ... In every flame, in every luminous particle, we see an electrical process ... Thus, the domain of electricity extends over the whole of nature. It even affects ourselves intimately: we perceive that we possess ... an electrical organ-the eye." [21]. Only later in the 1990s did it become obvious that visible light was only but a small part of a vast spectrum of electromagnetic waves [21], as seen in Fig. 2.3.

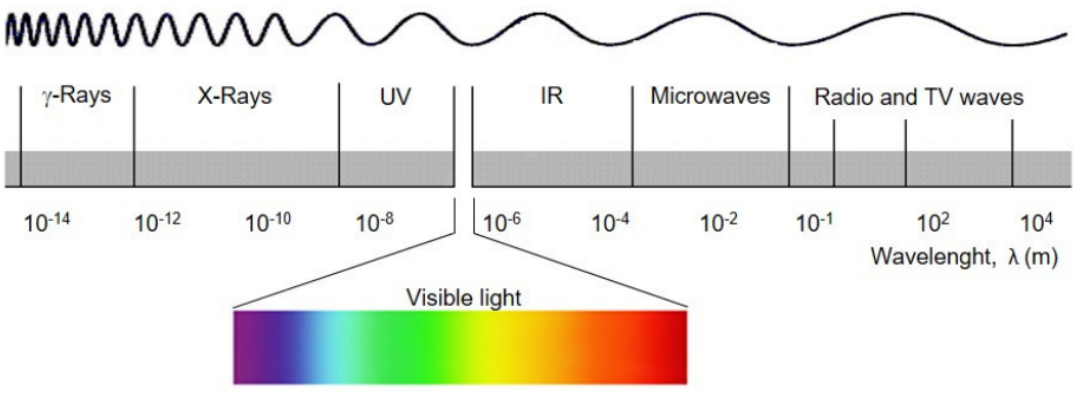

Figure 2.3: A diagram of the electromagnetic spectrum covering a wide range of electromagnetic waves, from gamma rays, in order of increasing frequency and decreasing wavelength.

All the electromagnetic waves (as in Fig. 2.3) travel at the speed of light in vacuum and encompasses a wide range of frequencies; from the lowest to the highest frequency (longest to shortest wavelength) contains but not limited to the following waves: *radio frequency (RF)*, *microwaves*, *infrared*, *visible light*, *ultraviolet*, *x-rays*, and *gamma rays* [21]. The discovery of the radio frequency was an important moment for mobile technology because a *radio signal* is propagated through space from one *Baseband Unit (BBU)* to the other, in what is known today as *wireless communication* [24] as seen in Fig. 2.4.

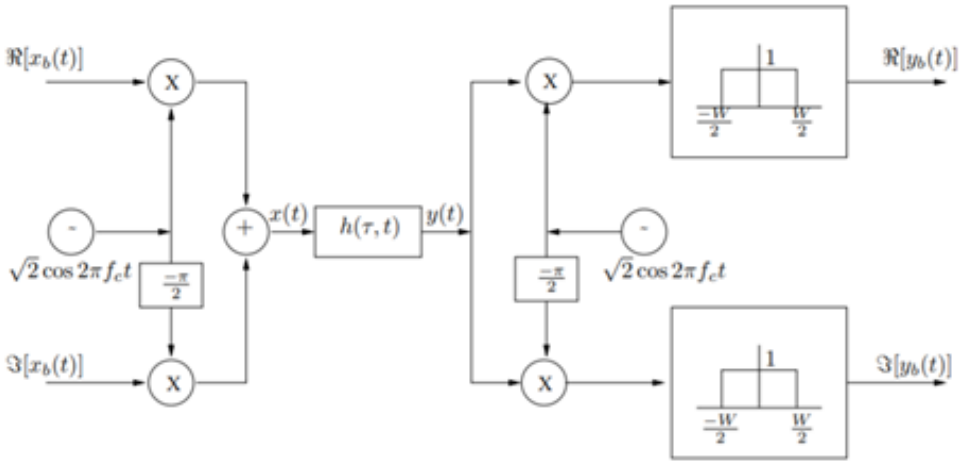

Figure 2.4: A technical representation of the transmitted $x_b(t)$ and received $y_b(t)$ baseband signal [6].

2.3 Evolution in mobile wireless communications technology

2.3.1 The beginning of wireless through to 4G LTE

During the latter part of the 1890s, the Italian inventor Guglielmo Marconi (1874–1937) developed the earliest commercial radio frequency (RF) communications called the *wireless telegraph* [30]. The wireless telegraph however had come over 50 years after the original commercial wired telegraph was initially demonstrated by Samuel F. B. Morse (1791–1872); in 1832 [30]. Marconi is also revered as the first person to broadcast radio signals to mobile receivers on ships at the beginning of the 1900s [31]. Even though wireless technology had been discovered after wired technology and costs more money, it gives the extra advantage of mobility such that users are able to receive and transmit information while on the move.

Marconi was able to wirelessly transmit Morse code signals as radio waves over 3.2 kilometres in 1985 [27]. This was a major breakthrough; after which many others from academia and industry have worked tirelessly exploring efficient ways of communication using radio frequency waves [30][31]. Point to point (P2P) telephone communication then became popular during the 20th century [27]. The need for an alternate way of communicating without depending on wired connection became dire, due to restriction in mobility and the propensity for wires to be effortlessly damaged [27]. An industrial engineer in the 1970s from Motorola called Martin Cooper, invented the first-generation mobile phone - a device (originally developed for use in a car) capable of being held in hand and allowing two-way wireless communication [27]. Cooper's work triggered a series of evolutions of many other technologies and standards. In fact mobile wireless communication technology has continued to evolve till today [22]. As demand surged for more devices and people to be connected around the globe, the mobile communication standards have been advanced rapidly to support various use cases. Cooper's work therefore was the main turning point in wireless communication [27][28].

In 1979, Japan was the first to see the deployment of the premier generation (first generation or 1G) of the mobile communicating network [28]. Nippon Telephone and Telegraph company (NTT) pioneered this deployment in Tokyo [28]. Within a few years after in the early 1980s, the first generation mobile communication system started gaining grounds and seeing adoption in the United States of America, Finland and other parts of Europe [26][28]. The system was based on analogue signals which disadvantaged the technology in many ways [26]. Examples of some of the popular 1G systems in the 1980s were the *Advanced Mobile Phone System* (AMPS), *Nordic Mobile Phone System* (NMTS), *Total Access Communication System* (TACS) and *European Total Access Communication System* (ETACS) [26]. 1G was known to have a frequency range between 800 MHz and 900 MHz as well as bandwidth of 10 MHz (666 duplex channels with bandwidth of 30 KHz). As mentioned earlier 1G was based on analogue switching technology and used *Frequency Modulation* (FM) for modulation [29][32]. 1G was characteristic of also being voice only and used *Frequency Division Multiple Access* (FDMA) as the main access technique [29]. There are some drawbacks that limited the adoption of 1G; these included poor voice quality due to interference; poor battery life and its phone was large in size, limiting the ability to carry around [26]. Other limitations included: inadequate system security as phone calls could be decoded using an FM demodulator [26]; the system was also disadvantaged in terms of the number of users, its cell coverage and roaming was not possible between similar systems [29].

The second generation (2G) mobile communicating network was developed as an enhancement to 1G to address its limitations. 2G therefore introduced *Global System for Mobile communication* (GSM), a new digital technology for wireless transmission [32]. Most of the current wireless standards were based on this new technology [32]. GSM supported between 14.4 to 64 Kbps data rate; enough data required for both Short Message Service (SMS) and email services [32][33]. In the 1990s Qualcomm also developed a new technology called *Code Division Multiple Access* (CDMA) which was adopted as part of 2G as it provided more breadth in scope than 1G especially in bandwidth size, number of users and data rate [33].

Unlike 1G, 2G brought internet to phones for the first time and at low data rates [33]. The 2G system also enabled digital system switching, allowing SMS and roaming services on mobile phones [32]. While 1G pioneered voice only, 2G had an extra advantage in terms of enhanced security by using an encrypted voice transmission methodology [33]. 2G allowed user mobility and limited number of users were assigned to a base station [32]. Due to the successful roll out of 2G, in no time there was a need for the technology to support higher data, leading to the development of *General Packet Radio Service* (GPRS). GPRS was then deployed and had the capacity to support relatively higher data rates up to 171 Kbps (maximum) [33]. There was another evolution of GPRS to improve data rate capacity; *Enhanced Data GSM Evolution* (EDGE) was developed. EDGE pushed the data rate capacity barrier up to 473.6 Kbps (maximum) [33]. The industry continued as per user needs and yet again built *CDMA2000* to facilitate higher data rates for CDMA networks; which had the capability to provide up to 384 Kbps data rate (maximum) [32].

The constant evolution of 2G due to industry needs led to the development of the third generation mobile communication standard (3G). 3G pioneered the *Universal Mobile Terrestrial / Telecommunication Systems* (UMTS). UMTS also supported a data rate of 384 Kbps and it introduced video calling on mobile devices [33]. Due to the enhanced capabilities of 3G which supports better communication in the workplace and at home, the adoption of smart phones increased extraordinarily across the globe [33]. New trends of applications supporting peer to peer communication started becoming more popular on smartphones; examples of the functions of some of these applications include video calling, gaming, multimedia content based chat, interactions through social media, location tracking, support for maps, better web browsing support, TV streaming, to mention but a few [34]. Various industries such as healthcare, built mobile interfaces that allowed them to reach customers directly from their phones [34]. As has been the trend, a desire for better data rates to support new mobile applications and use cases, saw the introduction of *High Speed Downlink Packet Access* (HSDPA) and *High Speed Uplink Packet Access* (HSUPA) technologies. 3.5G network was then

introduced and could support up to 2 Mbps data rate [34]. Further down the line, an improved version of 3.5G network with *High Speed Packet Access plus* (HSPA+) was introduced as 3.75G system [33]. Another evolution introduced 3.9G system as *Long Term Evolution* (LTE) [33]. In summary 3G provided more advantages over 2G, with a few key features being video calling, enhanced security, the ability for a higher number of users to connect to the network, better coverage and higher data rates [34]. The biggest concern of 3G at the time was cost. 3G was characterized by costly spectrum licenses, infrastructure, implementation and relatively expensive mobile phones that were compatible with the standard and frequency bands [33]. 3G also required higher bandwidth to support higher data rates.

The *International Telecommunication Union* (ITU) defined the standard for the fourth generation mobile communication standard (4G) to offer higher data rates and handle more advanced multimedia services [3][5]. Some key features of 4G are: an outstanding increase in data rate of up to 1 Gbps, reduced latency for time and mission critical applications, enhanced security and mobility [35]. 4G also supports high definition video streaming and gaming as well as *voice over LTE* network (VoLTE) [36]. 4G also uses LTE advanced wireless technology [36]. Assuredly, 4G is compatible with 3G, making its deployment and upgrade of LTE and LTE advanced networks a smooth transition [32]. 4G also enabled the concurrent transmission of voice and data with LTE system, as an effective way of significantly increasing data rate capacity [35]. Simultaneously wireless transmission technologies like *WiMax* are deployed with 4G system to improve data rate and network performance [37]. All the services that came with 4G such as high definition video streaming, voice services can be transmitted over *Internet Protocol* (IP) packets. Complex modulation schemes and carrier aggregation is used to multiply uplink / downlink capacity [37]. 4G also had a number of drawbacks, including costly hardware and infrastructure as well as expensive frequency spectrum [37]. Mobile phone hardware took advantage of the user experience 4G's features are able to provide users and developed expensive compatible mobile devices to the market. Another disadvantage of 4G is that its widespread deployment and upgrade can be time consuming [3].

2.3.2 5G and new opportunities for emerging technologies and use cases

It can be observed in sub-section 2.3.1 that the evolution of wireless network generations from 1G to 4G LTE have been consistently characterized by improvements in data rate, speed and other features that provide better quality of service in communication. This sub-section focuses on the fifth generation of cellular network technology (5G), and the key features of 5G that distinguishes this standard from previous standards. To mention but a few, 5G features lower latency, higher capacity, and increased bandwidth. These network improvements have far-reaching impacts on how people live, work and are entertained.

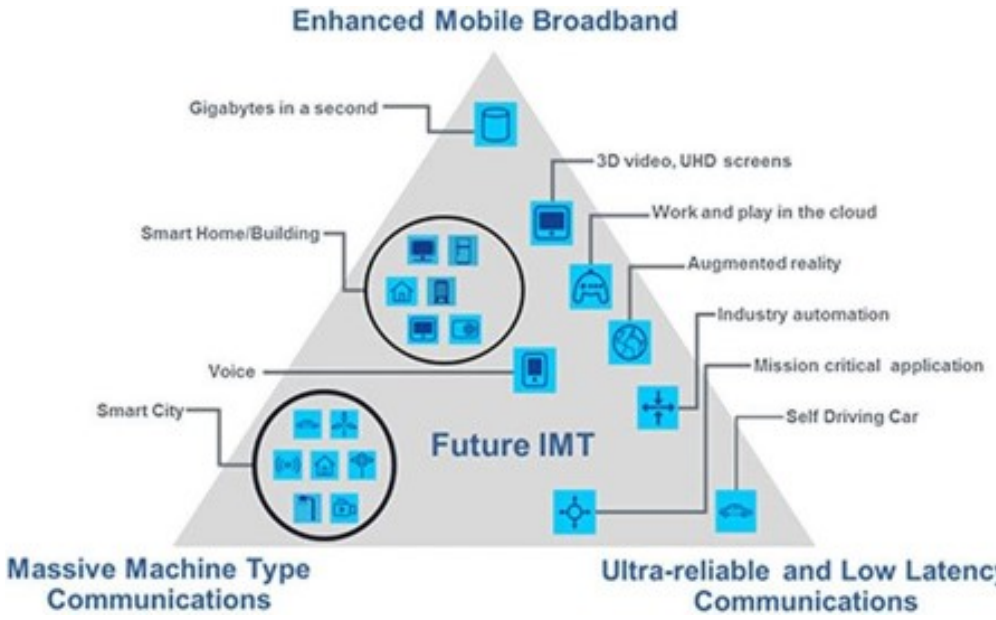

Figure 2.5: Enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC) and massive Machine Type Communications (mMTC) are the 3 main 5G use cases as defined by the International Telecommunication Union Radiocommunication sector (ITU-R) [3].

As per Fig. 2.5. from the ITU-R [3], 5G is used across three main types of connected services, including *enhanced Mobile Broadband* (eMBB), *Ultra Reliable Low Latency Communications* (URLLC) and *massive Machine Type Communications* (mMTC) [5]. Despite being associated with these identified benchmark use cases,

5G has also been designed to be forward compatible i.e. it is capable of addressing the needs of use cases that may emerge in the near future [5].

Enhanced Mobile Broadband (eMBB) focuses on improving mobile broadband experience for mobile application scenarios which consume high volumes of upstream data such as in high-definition video, gaming etc. [3]. As 5G is able to offer bigger bandwidth, higher and more uniform data rates as well as lower latency, it is well-positioned to support new immersive experiences such as with virtual reality and augmented reality powered services at lower cost-per-bit [16].

There are sub use cases or scenarios that depend on time and mission critical communication such as in a smart manufacturing environment, remote manipulation of critical infrastructure in tactile internet, interactive gaming, virtual reality and so on [3]. This real time communication feature is extremely fundamental for these scenarios as if specific time is not met, it leads to often unrecoverable system and process failure [16]. These mission / time-critical communications sub use cases which have a strong need for low end-to-end latency are categorized as Ultra Reliable Low Latency Communications (URLLC) [3]. As 5G is characterized by ultra-reliable, available, low latency, it is capable of enabling new services that can transform industries that depend on time and mission critical communication such as health, automobile and supply chain industries [5].

Last but not the least, massive Machine Type Communications (mMTC) are the sub use cases or scenarios for which 5G is capable of enabling the internet of things (IoTs) on a large scale [3][19]. 5G is capable of seamlessly connecting a massive number of IoTs and embedded sensors, for example for a smart city project [5]. In such a scenario a significant amount of highly localized data volume is constantly being collected and passed upstream through the network. 5G is well-endowed with features such as high bandwidth, low latency to manage the volume of data as well as meeting other requirements of the data such as not crossing domain boundaries because of ownership and privacy concerns [16].

Below are a few of 5G's features that allow it to support the 3 main uses cases in Fig 2.5. effectively:

- **Better Improved Speeds:** 5G is predicted to have enhanced speeds of up to 10 Gbps, providing more value to customers over 4G and the previous mobile wireless generations [5]. The new possibilities this enhanced speed brings to consumers are endless. 5G is capable of reach maximum data rate under normal circumstances for each user or device [3]. This means transferring a high-resolution digital asset at peak download speeds can now take seconds instead of minutes or hours. That time saving is a positive user experience or quality of service for the consumer. It is likely that when mobile network operators complete 5G roll out, many users and stakeholders may seriously begin to consider it as a potent alternative for fast broadband connections.

- **Lower Latency:** Round trip latency measured in milliseconds refers to the time in takes for a signal to go from its source to its receiver, and back. When latency is high users experience lags in how the response time of applications, and that is why it can be observed in the prior sub-section that each new wireless generation, an effort to reduce latency. 5G will have even lower latency than 4G LTE [5], making it possible for new use cases such as eMBB, URLLC and mMTC to come into existence, due to their requirement for extremely low latency of 10 ms or below [3].

- **Enhanced Capacity:** 5G can facilitate the mass scaling of a business or project from a connected device perspective [3]. 5G will increase the total number of connected and / or accessible devices per unit area (per km^2) by delivering up to a thousand times more capacity than 4G, enabling a potent environment for IoT development and advancement [5]. This capacity for hundreds or thousands of devices to seamlessly connect and exchange information has birth endless opportunities in many industries such as rural as well as urban development, agriculture, health and wellness, education and so on. Machines with a large number of sensors on a factory floor or

distribution centre environment can now collaborate by automating an entire manufacturing and supply chain management process, taking advantage of automatic predictive maintenance to minimize operational interruptions [16]. This will lead to high productivity and lower costs especially from a human resource point of view. This also creates room for AI and edge computing to engage and add value such as leveraging intelligence, bringing energy savings [3] in ways that have never been seen before, in use cases such as smart homes, smart cities etc [16].

- **Increased Bandwidth** An increase in speed and network capacity will lead to larger amounts of data being exchanged in 5G than in previous wireless network generations. 5G however addresses this with a bigger bandwidth capacity, allowing more data throughput per unit of spectrum resource and greater optimization of network traffic [16].

2.4 Why does latency continue to be a problem even with 5G?

Even though 5G features lower latency, the constant low round-trip latency required cannot be achieved with 5G alone [16]. This is because communicating clients and hosts are not always in the same geographic location and therefore the physical distance between them can cause delays (latency) in exchanging data packets [6][16]. It is common for technology and cloud computing companies to have their master data centre in one country (such as the United States of America), and serve customers across different parts the globe with this data centre.

Round-trip latency continues to be a problem even with 5G [3]; the further devices are apart, the longer it takes to transfer data packets between them. In physics, the special theory of relativity [38] [39] suggests that nothing travels faster than the speed of light in a vacuum: 3 x10^8 ms^{-1} [15][17]. This means that an increase in distance has a directly proportional consequence on travel time. For example, sunlight takes approximately 8 minutes and 20 seconds to reach the earth, as the sun is approximately 147.51 million km away [15][17]. Latency is a real-world speed limit.

2.5 What is Multi-Access Edge Computing and Cloud-Based/ Centralized Radio Access Network (C-RAN)?

2.5.1 Cloud Computing and its essential characteristics

John McCarthy was not only known as the Father of Artificial Intelligence (AI) but he has also been referred to as the man behind the idea that it can be possible to offer computing as a public utility [40] just like gas and electricity, since the 1960s. Today, this idea is manifested through the concept of cloud computing which continues to be adopted by individuals and organisations daily [40]. It is a model that facilitates access to computing resources such as servers, storage, databases, networking, software applications and so on, by a stakeholder over the

internet [41]. In other words, cloud computing allows computing resources to be accessed as a service by users and typical examples of services enabled by cloud computing include applications, servers (both physical servers and virtual servers), data storage, integrated development environments and networking capabilities [41]. Cloud computing remains one of the most used buzzwords or metaphors that paints a picture of how the interconnections between devices on a network such as the internet or "in the cloud" looks like, without revealing the complex infrastructure that sits underneath.

The philosophy of cloud computing originates from various other computing research such as from high performance computing, computer virtualization, grid and utility computing [40][42][43][44]. Therefore this section highlights some characteristics of cloud computing that distinguishes it from the other computing areas. Cloud computing is a TCP/IP network based model of enabling ubiquitous, reliable, on-demand network access to shared computer resources via the linkage of technologies such as microprocessor, computer memory, high-speed network and reliable data center technologies [32][40]. The server's role is vital and it forms the core of the whole processing environment [41]. The server hardware in a cloud computing environment also does not need to be very sophisticated as cloud computing technology takes advantage and optimizes the power of cheap hardware on a larger scale [40]. More businesses these days embracing cloud computing for many reasons, including easy access to information or resources from any computer and increasing efficiency [32]. The use of cloud computing also prevents users from easily losing data as a result of data corruption, data destruction and unauthorised data access [40].

By reason that all of the information for a client is hosted at one physical site also allows for hardware and software to be managed more efficiently by a functional on-site team, dedicated in managing specifics such as updating hardware and software [45]. Essentially this process is practically seamless as there are no outages required in the cloud, and as such the average user would have no interruptions with their usage of the cloud resources [45]. Businesses find this managed-care of technology

hardware both cost-effective and convenient as it allows them to focus on the core aspects of their business [46]. They subscribe to the philosophy of the advantages of economies of scale in the sense that, other companies who host a bigger server operation are more likely to run it more effectively and efficiently than they (the businesses) would in house [46]. This also means that as more businesses are becoming more complex and opting for the cloud, the cloud is naturally growing and evolving swiftly into a whole new ecosystem not just for the technology industry [47].

Cloud computing services are broadly divided into these three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [32][40]. Cloud computing is also further divided into these six layers, which will be discussed later on: clients, services, applications, platform, infrastructure and storage [46]. The National Institute of Standards and Technology's (NIST) definition considers four different deployment models (Private, Public, Hybrid and Community Cloud), three different service models (Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), and 13 other characteristics [34].

Out of the 13 characteristics Wallom et al in their work [32] utilise a quantitative methodology to cluster different cloud projects and activities that are technically aligned and are likely to benefit from interactions and shared learning, to argue that only 5 characteristics define cloud computing more robustly:

1. On-demand self-service

This characteristic describes the ability for users to access cloud resources as and when needed, using a secure online control panel provided by the cloud vendor [32].

2. Broad network access

As a web based service, users can use any device with internet to access the cloud computing services [32].

3. Resource pooling

This characteristic describes how multiple users (also known as tenants) share scalable resources and services [32].

4. Rapid elasticity

As per customers evolving requirements, cloud computing provides the flexibility to scale resources both up and down [32].

5. Measured service

Cloud computing works on a pay-per-use billing model and therefore users are only billed by the resources and services used [32].

2.5.2 The purpose of Multi-Access Edge Computing (MEC)

Humans are technologically more connected now than we have ever been. Our needs and emerging lifestyles have encouraged many technology companies across the globe to invest in applications and physical devices that can make our lives easier. In fact, Gartner forecasted that there will be about 8.4 million connected "Things" by the end of 2017 whiles Cisco also believes there will be 50 billion connected devices by the end of 2020 [48]. With the help of traditional cloud computing, these devices can collect and send data over different communication media for computation and storage in the cloud [4]. Cloud computing also provides high computing power, cheaper cost of service, high performance, scalability, accessibility and availability of IT services and resources to handle such operations[46]. Cloud computing however, has its limitation when it comes to readiness for emerging technology use cases in 5G and access devices such as IoTs [4][19].

These emerging access devices raise new questions that are not adequately answered by the geographically distant centralized cloud compute architecture such as the need for high bandwidth and low latency [4]. Other concerns include non-adaptive machine to machine (M2M) communication, uninterrupted services

with intermittent connectivity, resource-constrained devices, enhanced security and capacity constraints including limited processing power, storage capacity and battery life [3] [5]. Also, in the traditional client to server concept, the ever-increasing number of IoTs and mobile devices cause high network load issues by placing additional load on both the radio access networks and the backhaul networks [4][16]. MEC as a novel paradigm based on virtualisation provides key features that complement the efforts of cloud computing combating the aforementioned issues, by moving compute, network and storage closer to the source of data [4], as in Fig 2.6 below.

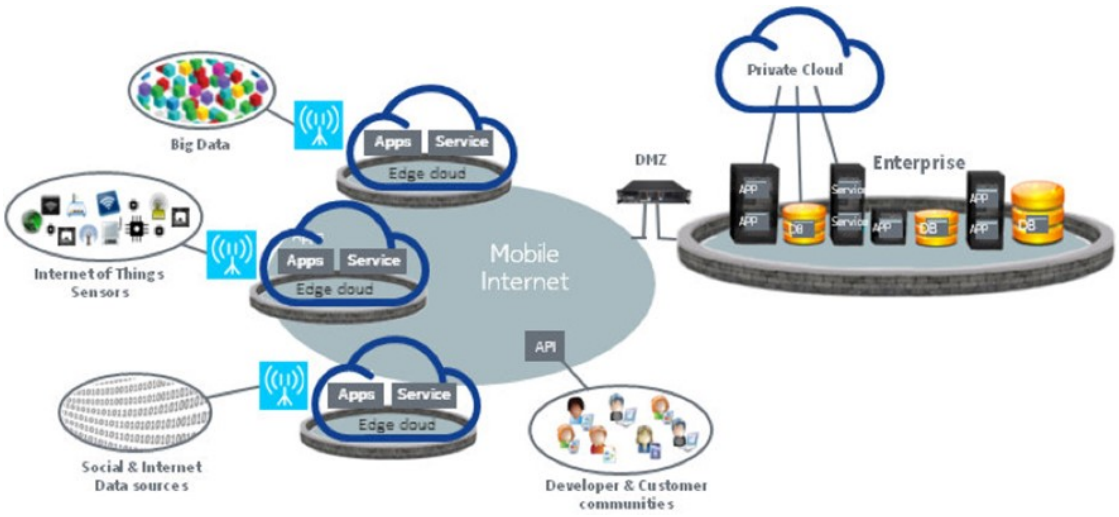

Figure 2.6: Basic Architecture for Multi-Access Edge Computing from [4].

2.5.3 Characteristics of Multi-Access Edge Computing (MEC)

The Internet of Things (IoTs) have a strong presence in our world today and it refers to the communication network that exists between a plethora of devices (mobile included) that produce and exchange enormous amount of data pertaining to real-world objects (i.e. things) [19][49]. From a practical point of view, sending this data back and forth the cloud from all these devices will require an exorbitantly high network bandwidth [4]. The idea of mobile (now multi-access) edge computing [MEC] is to facilitate faster responses and provide better quality of service than

what the traditional cloud computing architecture is able to offer [4]. This involves the data that has been generated by the access devices being processed at the network edge instead of transferring it to the centralized cloud infrastructure due to bandwidth and energy consumption concerns [4].

MEC facilitates computation at the earliest point of data ingestion and some of the motivations behind this includes:

1. Proximity, Low latency, high bandwidth — by hosting IT services and cloud computing capabilities close to the user, mobile resources are optimized to perform better [4][50].
2. Data optimization [50] — MEC brings intelligence to the edge of the network and such intelligence is useful for the data aggregation and filtering process. The motivation behind this is to only send relevant data to the cloud, thus saving on bandwidth cost and reducing latency on the wide area network (WAN) or the telecommunication network that extends over a large geographic area.
3. Energy savings — since only relevant data is sent to the core cloud and IoTs are not consistently forwarding data to the core, IoT device energy is saved [15][17].
4. Promising value chain — MEC enables IT and cloud computing services closer to the user which opens up numerous business opportunities for potential stakeholders such as over the top (OTT) service providers [4].
5. Real time radio access network (RAN) information — MEC stakeholders can benefit from real-time RAN information in enhancing their services and ultimately the user experience. Information such as network load and user's location can be used by content providers and application developers in providing context-awareness services to the mobile users [50].
6. Rapid provisioning — MEC takes advantage of virtual machine (VM) technology and therefore is more agile in terms of launching VM images to meet computing and network demands of users [16][51]. MEC is associated with access devices which often tends to be mobile and therefore MEC is dynamic enough with its resource provisioning due to user mobility. For example, a user who has just gotten

off a flight should be able to immediately resume connectivity to applications and services (context-aware) on the edge cloud without delay.

7. Virtual machine handoffs — A mobile device user moving away from the edge cloud will naturally affect its interactive responsiveness, especially as the logical network distance increases. In order to manage the effects of the user's mobility nature on the usability of the system, the offloaded services are migrated to the edge cloud closer to the user while still maintaining end-to-end network quality [50][51][52][53].

8. Connectivity — Mobile devices look for edge clouds to connect to. In a perfect world, these edge clouds will be distributed geographically and an access device will automatically to locate, choose, and assign itself with the appropriate edge cloud before it starts provisioning [16][50].

Multi-access Edge Computing (MEC) fundamentally brings compute and storage capabilities closer to a user or source of data, in a way that complements traditional cloud computing [16]. This emerging technology in other words, provides cloud-computing capabilities and an IT service environment at the edge of the network, to application developers and content providers [4]. As MEC complements cloud computing, some of the benefits it brings includes ultra-low latency, high bandwidth and also allows applications to access and use real-time radio network information [16]. Without MEC, the MNOs will not be able to support certain use cases classified under these 3 service types: eMBB (enhanced Mobile Broad Band), URLLC (Ultra Reliability and Low Latency Communications), and mMTC (massive Machine Type Communications) [3][5][34].

Some of these use cases include: video analytics, location services, augmented reality, optimized local content distribution and data caching [22][24][28]. The application and various services in these use cases can be customized according to the customer requirements and demands through APIs [29][54] built and maintained by ETSI MEC ISG. MEC also is extremely relevant in this brave new world of increasing numbers of IoTs that want to stay connected and share high volumes of data at extra ordinary speeds [19][48]. It is also worth mentioning that MEC is an earning

stream for MNOs and provides a new ecosystem and value chain. Such that MNOs allow authorized third-parties access to their Radio Access Network (RAN) edge, invariably making it possible for them to conveniently deploy innovative applications and services towards mobile subscribers, enterprises and vertical segments [4][16][22].

2.5.4 Cloud-Based/ Centralized Radio Access Network (C-RAN)

The architecture of a traditional Radio Access Network (RAN) consists of decentralised stand-alone base stations covering an area [55]. A base station processes and transmits signals to and from the mobile terminal (the hexagonal area in Fig. 2.7), and furthermore forwards the data payload to and from the mobile terminal and eventually out to the core network via the backhaul [56]. Every base station has both a Radio Remote Head (RRH) and a Base Band Unit (BBU) needed to accomplish its purpose, as in Fig. 2.7.

It is important to note that a base station maintains several components such as cooling, back haul transportation, backup battery, monitoring system, that keeps it running [16]. Maintaining these components form part of a mobile network operator's fixed operating costs and can be severe [56]. This is where C-RAN distinguishes itself and adds value.

C-RAN is a virtualized network architecture, where the BBUs are separated from the RRHs and centralised (see Fig. 2.7 B), optimising the use of these other components more efficiently (as well as at low cost) and enabling easier deployment and scaling of network resources. C-RAN is composed of high-performance programmable processors and real-time virtualization technology that perform baseband (PHY/MAC) processing [3]. BBUs are built on general purpose platforms and are interlinked by high bandwidth low-latency optical network to distributed Radio Remote Heads (RRHs), located at the remote site.

2.5.5 Merging MEC and C-RAN

MEC and C-RAN have a number of intersecting goals, interests and characteristics such that the same C-RAN architecture may be repurposed to deploy both C-RAN and MEC [5]. Virtualization is also a common attribute of their core setup [4][55]. Fundamentally, they both are purposed to optimise network function by maintaining reduced latency, increased bandwidth and lowered energy consumption levels [4][55]. MEC and C-RAN complement each other and collocating both, makes the economics of each other more attractive to a MNO [3]. This is why, this research endorses the more reliable solution of MEC + C- RAN as a single unit [16] and more importantly, develops an optimal network management strategy armed based on virtualisation; in the form of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) [58]. MEC, NFV and SDN also

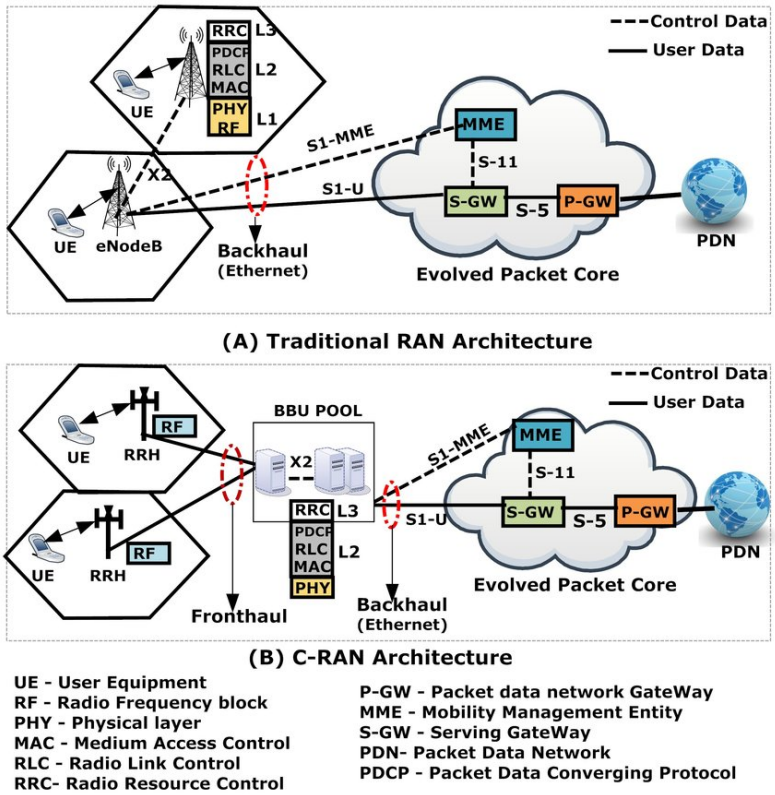

Figure 2.7: Traditional RAN vs C-RAN. Where in a traditional RAN, every BTS has both an RRH and a BBU and the whole BTS is distributed to provide service for a site. In a C-RAN, BBUs have been moved into a centralized location and RRHs are distributed to cover sites. Image taken from [57]

come together nicely as a fertile ground for deploying new services with increased flexibility, agility and quicker time-to-value [58][59]. This opens up an extremely exciting brand-new value chain in the telecom industry, but this is outside the scope of this project. This project focuses on using machine learning techniques are used to perform the network management routines of dynamic orchestration and resources scheduling in this novel architecture.

2.5.6 Containers - live migration of active Docker containers between nodes, using Checkpoint / Restore In Userspace (CRIU)

A container is a lightweight executable software package [60]; *Docker* containers are used in this research. Migrating a container live is a technique of moving a container from a source node to a destination node with minimal or no impact on the quality of service for the mobile application / network user [60] as in Figure 2.8.

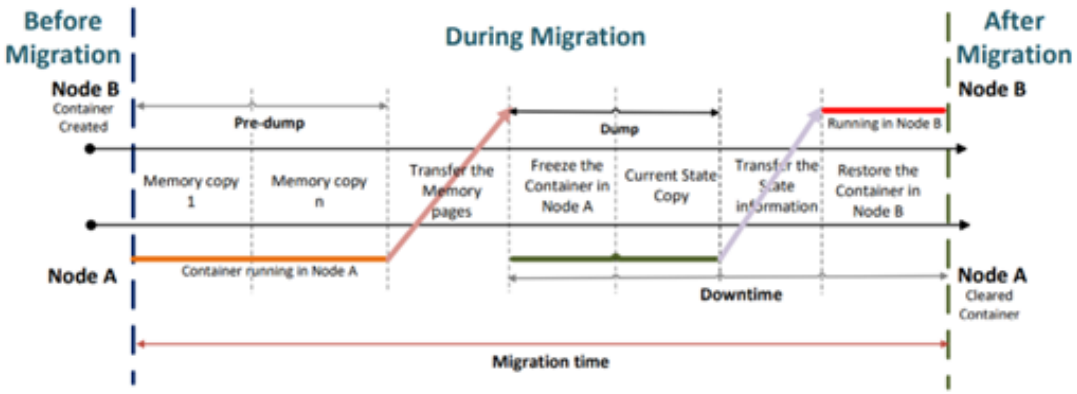

Figure 2.8: Live container migration process [60]

Being able to move containers from one node to the other has some advantages. Load balancing is a classic and popular use case for container migration [61]. This allows containers to be distributed across multiple servers / nodes (instead of them all operating on a single node) to improve container availability and prevent node overload [58]. Another advantage of container migration is to allow maintenance activities such as hardware and software upgrades to be temporarily

carried out on a particular compute node [61]. A third and not the least advantage of container migration is efficient resource use [61]. When host nodes are being underutilized, their containers may be clustered onto a single node for long-term cost savings benefits.

Containers were designed to be stateless when initially introduced as a way of packaging microservices with their necessary libraries and dependencies [58][62]. Stateless microservices are applications that have no post-requisites to persist data, and as such there is no stored knowledge of past interactions with the application [58]. That being said, containers have evolved and have been extended to stateful applications providing them with flexibility, speed, the ability to run in any environment, but with storage and the privilege of statefulness [59]. Where container state persistence is necessary, the state is usually stored separately in an external storage facility like a database [59]. In either scenario, TCP/IP based container run-time packages are used to migrate the Docker containers from a source node to a destination node using *Checkpoint and Restore In Userspace* (also known as CRIU) [63][64].

The way the migration technique works is: a checkpoint is initiated to start the migration process [60]. The CRIU technique freezes the running container at the source node whilst capturing metadata pertaining to the container such as CPU state, memory content, and information about the process tree [60]. The captured information is transferred on to the destination node via TCP/IP before the Restore process resumes container function [64]. The total migration time is a function of the checkpoint process time, metadata copy time, metadata transfer time, docker data volume and container restoration time [64]. This time is significantly reduced in MECRAN (especially for stateful applications) as all clusters already have access to a container's persistent data due to the cluster network's shared data access, an advantage of *Ceph Storage* [65] in MECRAN. Ceph Storage as a software defined storage system replicates and distributes the local registry as file storage [65] to each MECRAN cluster within the radio access network for peak performance.

2.6 Limitations in traditional Cloud and Radio Access Network architecture for today's emerging technologies / use cases

Cloud Computing and the radio access network have significantly improved the way we access and use computing services like never before. To such an extent that computing resources such as software applications, servers, storage, databases, networking and so on are now being delivered as a service over the internet ("*the cloud*") at a more economical rate [40][41].

Against this background, one major factor affecting this paradigm is physical distance limitations [4]. This consequently gives life to problems such as low spectral efficiency (low bandwidth) and high latency and therefore does not support the 5G uses cases which have been divided into three service types [15][17]: *enhanced Mobile Broad Band* (eMBB), *Ultra Reliability and Low latency Communications* (URLLC), and *massive Machine Type Communications* (mMTC) as part of the IMT-2020, *The International Telecommunication Union* (ITU) to define "5G" [3][5] – see Figure 2.6 above.

Emerging digital applications such as in self-driving cars, virtual reality, augmented reality and so on, have strict non-comprisable requirements for extremely low latency (10 ms or below) and high bandwidth for performance [4].Unfortunately, traditional cloud computing and radio access network architectures do not sufficiently address the need for low latency and high bandwidth [6], that are crucial to the performance of these cloud-native applications. There is, therefore, a need to enhance the mobile communication network to allow *mission* and *time critical* cloud-native applications to flexibly run on seemingly '*unlimited*' and reliable compute, network and storage, to meet these performance requirements. 'Unlimited' refers to the sense that applications have unconstrained access to the necessary network resources.

This work (Multi-Access Edge Computing with Cloud Radio Access Network or MECRAN) is a facility that uses machine learning at its core to augment or supplement the traditional cloud and RAN architecture in order to meet the demands of emerging technologies.

2.7 The role of Machine Learning

As emerging technologies and cloud-native applications communicate with the remote cloud through data exchanges, the extra latency introduced by the physical distance of communication severely affects the performance of some applications. Data exchange over physical distance is essential in wireless communication and cloud computing, but the resulting latency penalty is also a gap that 5G alone is unable to resolve [4].

The product of this research bridges this gap by leveraging machine learning in scheduling applications on host servers (base stations) in order to facilitate the data exchanges at low latency. *Neural Networks* and *Deep Reinforcement Learning* are the base techniques used in MECRAN as they have proven to be able to effectively discover regularities and recognise patterns from historical data for predictive purposes [66]. Being able to predict the best base station to schedule and run a cloud-native application from, in order to reduce latency is the goal of this work. MECRAN optimizes the roundtrip delivery of data and reduces latency by bringing the distant client and server closer to each other through scheduling at the edge of the radio access network and by the power of virtualization. Chapters 5 and 6 deep dive into the machine learning technique implementation in MECRAN.

2.7.1 Neural Networks / Deep Learning

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers i.e. at least an input layer, an output layer and hidden layer(s) [66]. A neural network (as seen in Fig. 2.9) attempts to mimic the behaviour of the human brain, enabling it to learn from vast amount of data.

Most regression and classification models such as *Support Vector Machines* (SVMs), comprise of linear combinations of fixed *basis functions* and the disadvantage of this is that, their practical application is restricted by the *curse of dimensionality* [66][67]. On the other hand, the advantage of neural networks is that it automatically adapts the basis functions to the data of interest using *parametric forms* in which the parameter values are adapted during training [66]. This is particularly important in

this research project as the training data (such as the use case, geographic area data, network data etc) for MECRAN could be different in different use cases and as such having a self adaptive model from the basis function perspective is an advantage. We use a feed-forward neural network or a *multilayer perceptron* model as seen in Fig. 2.9. The hidden layers helps the model to optimize and refine for accuracy. Deep learning has been used in many successful artificial intelligence applications and services such as in self-driving cars, fraud detection and in digital assistants [68].

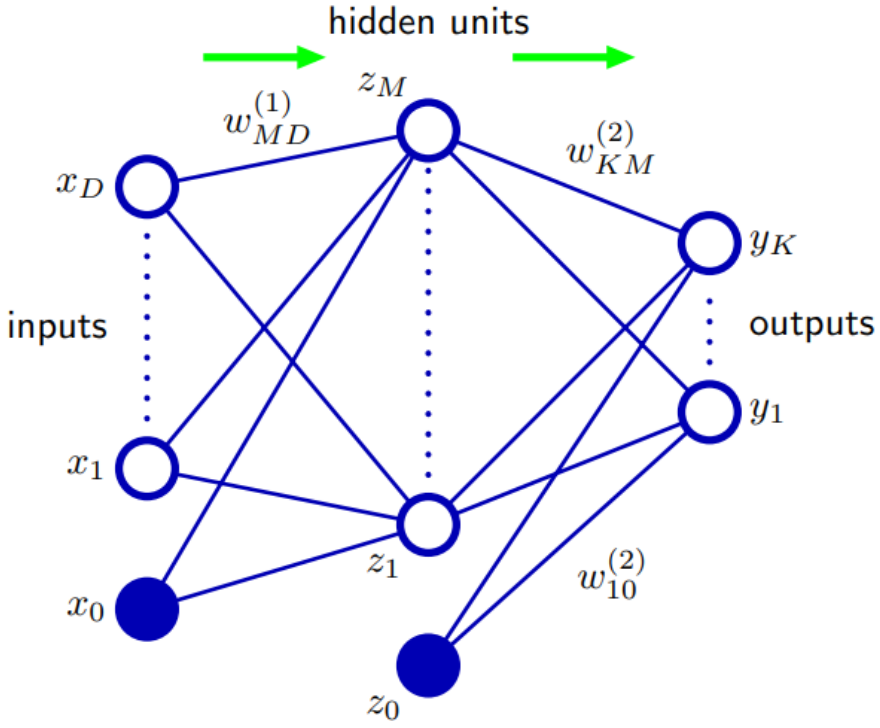

Figure 2.9: Neural Network from [66] showing an input layer, hidden layer and an output layer.

2.7.2 Reinforcement Learning

In machine learning, models are categorized under supervised learning and unsupervised learning [67]. Supervised learning models use labelled training data and predicts the testing data while unsupervised learning models discover patterns from the given testing data and attempts to glean the insights in that data without any training [67]. Reinforcement machine learning however focuses on finding suitable actions to take in a given situation in order to maximize a reward [66]. This is different from supervised learning where the objective is attaining good accuracy or on minimizing the cost function [66].

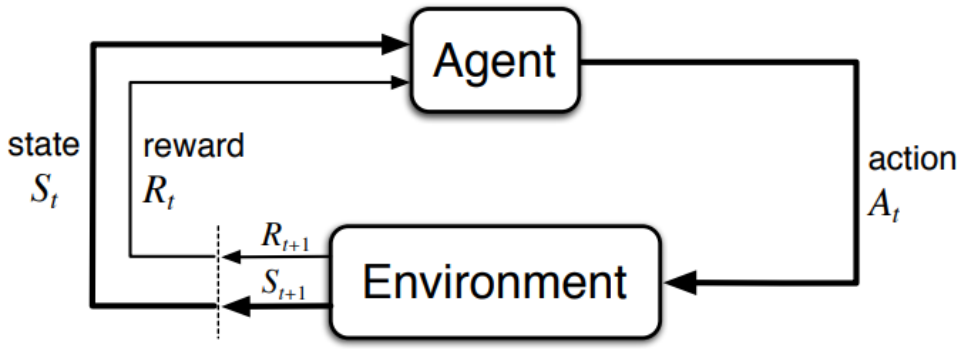

Figure 2.10: Reinforcement Learning model from [69] showing how an agent interacting with the environment (in a state) and continuously receiving rewards at every time step for every action taken in the previous state.

Reinforcement learning algorithms get feedback in the form of *rewards* (R_t) as an *agent* takes *actions* (A_t) or interacts with its environment in a particular *state* (S_t) [69] as seen in Fig. 2.10. An agent aims to maximize not only its immediate reward but future rewards as well in various states - states are different configurations of the environment in which an agent takes actions [69]. Based on an action performed by the agent in one state, the agent moves to another state in the environment and obtains a reward either positive or negative [66]. The goal of the agent is to maximize its total rewards [66]. In MECRAN, the reinforcement model is used to calibrate the impact of moving a cloud-native application to a range of base stations, maximizing or optimizing the chance of landing on base station(s) that offers the lowest latency.

Examples of reinforcement learning applications in the real world include: an agent such as a robot learning how to perform an action in the real world such as vacuuming a house, walking etc [69]. Other applications of reinforcement learning in the real world can be found in simulated game environments like Atari games, Chess, Go Game where it has successfully surpassed human learning capabilities and received recognition around the globe [70][71][72]. Reinforcement learning can also be applied in the banking and capital markets industry where it can be used to optimize investment portfolios to achieve higher rewards [73].

3

Strategy & Planning: Road Map & Methodology

Contents

3.1 Introduction & Purpose

The goal of this work is to build an optimal and sustainable solution that dynamically schedules cloud-native applications to be hosted and executed close to the user i.e. at the edge of the radio access network instead of physically at the original distant server, in order to reduce latency. This chapter walks through the road map or big picture of the proposed solution, explaining the various approaches, methodologies and processes that come together to form the solution.

Whilst this chapter focuses on elucidating the overall strategy (as per Fig.

3.3) for the proposed solution, the next 3 chapters deep-dive into the practical implementation of the various moving parts of this road map. The information in this chapter will therefore be referenced often over the next few chapters to ensure consistency and a smooth flow in the explanation of the strategy.

3.2 Approach

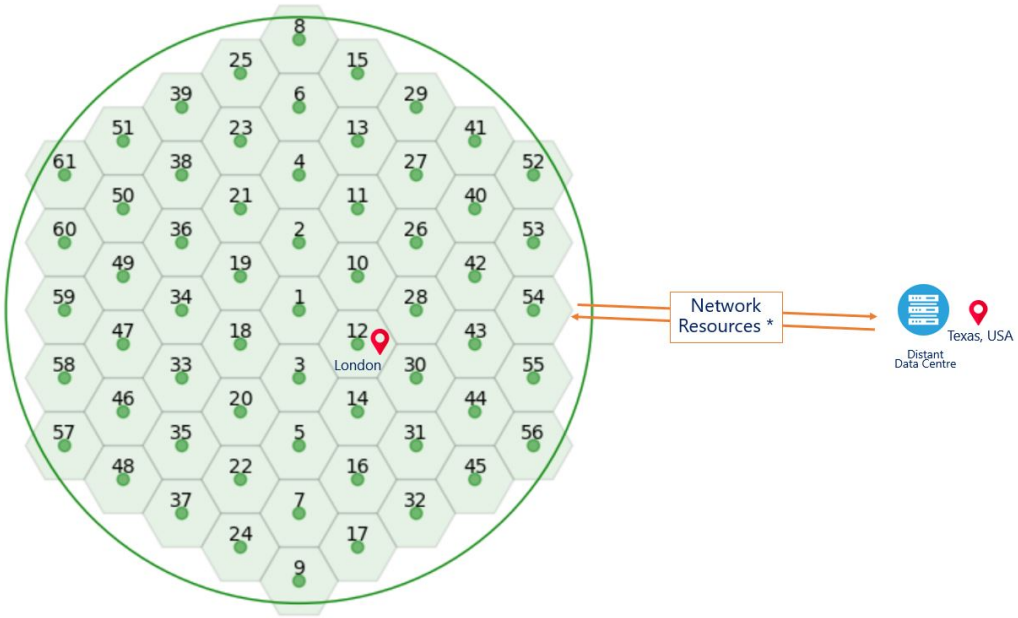

Figure 3.1: Radio Access Network with a London (UK) based user accessing a cloud-native application from a data centre in Texas, USA.
* Network resources such as switches, routers, internet service provider hubs, under sea fibre optic cables etc.

Assuming user A is based within the coverage area of base station 12 (as in Fig. 3.1) in London (UK) and wants to access an augmented reality based cloud-native application from a data centre in Texas (USA). Traditionally, the server hosting the application in the data centre is able to send data via a combination of intermediate network resources e.g. switches, routers, internet service provider hubs etc., then via a cluster of under-sea fibre optic cables [74], and eventually through a local base station / cell tower / mast, to user A's device [55][56]. These network resources are also linked by connection types (such as copper cables, fibre optic, digital subscriber

line (DSL), satellite) with different bandwidth capacities and lengths. Round-trip latency continues to be a problem even with 5G [3][5]; the further user A is away from the host server, the longer it takes to transfer data packets between them [6].

In physics, the *special theory of relativity* suggests that nothing travels faster than the speed of light in a vacuum, approximately 3 x10^8 ms^{-1} [38]. This means that any increase in distance has a directly proportional consequence on travel time. For example, sunlight takes approximately ∼8 minutes and 20 seconds to reach the earth, as the sun is approximately ∼147.5 million km away [38]. We have recorded an average of 110 ms latency between London (United Kingdom) and Dallas (Texas). Latency is a real-world speed limit.

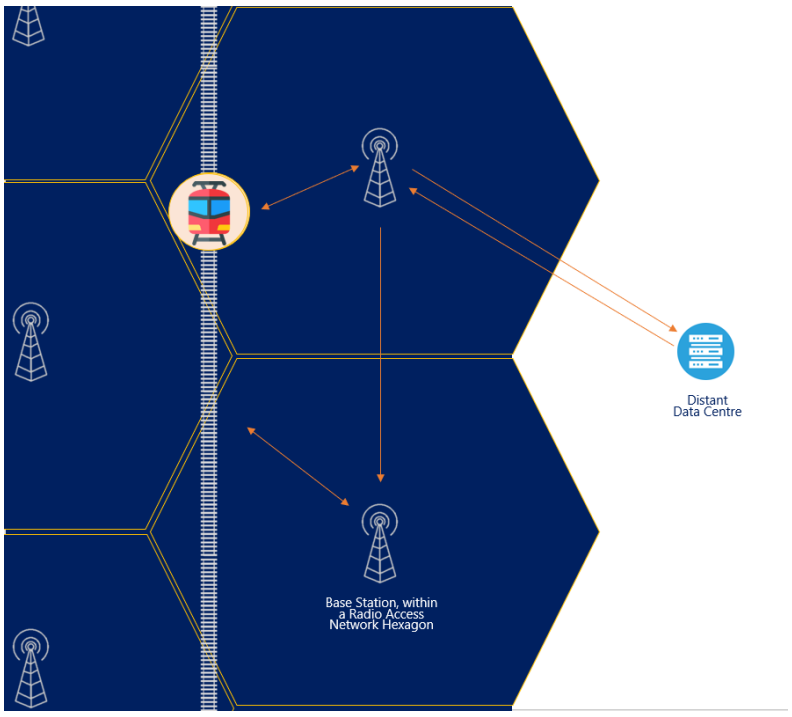

Figure 3.2: Radio Access Network - a closer look.

From the above perspective, reducing physical distance between communication devices, is crucial and a fundamental design principle of our approach. This is achieved by adopting the edge computing paradigm, instead of depending on the remote cloud / distant server. When a user A tries to access the augmented reality application from the Texas-based data centre, MECRAN takes the following approach:

1. MECRAN pings the host server to determine the average latency based on the current distance between user A's device and the host server. This derived latency is compared with latency required (10 ms and below) by the augmented reality application for performance.

2. If required latency by the augmented reality application is met:
   Then, MECRAN runs the application at the host server
   Else, MECRAN migrates the application in a container (such as Docker [75] or Kubernetes [76]), from the host server to be hosted at the edge (base station) of the radio access network as in Fig. 3.2, for dynamic scheduling.

3. MECRAN's machine learning-based data packet routing and scheduling routines work hand-in-hand to schedule the application on the "best" server. The best server is one that enables the application to meet its latency requirement best.

Based on the above approach, section 3.3, outlines a road map that encompasses this approach and delivers a machine learning based solution that tackles the latency problem in cloud-native applications.

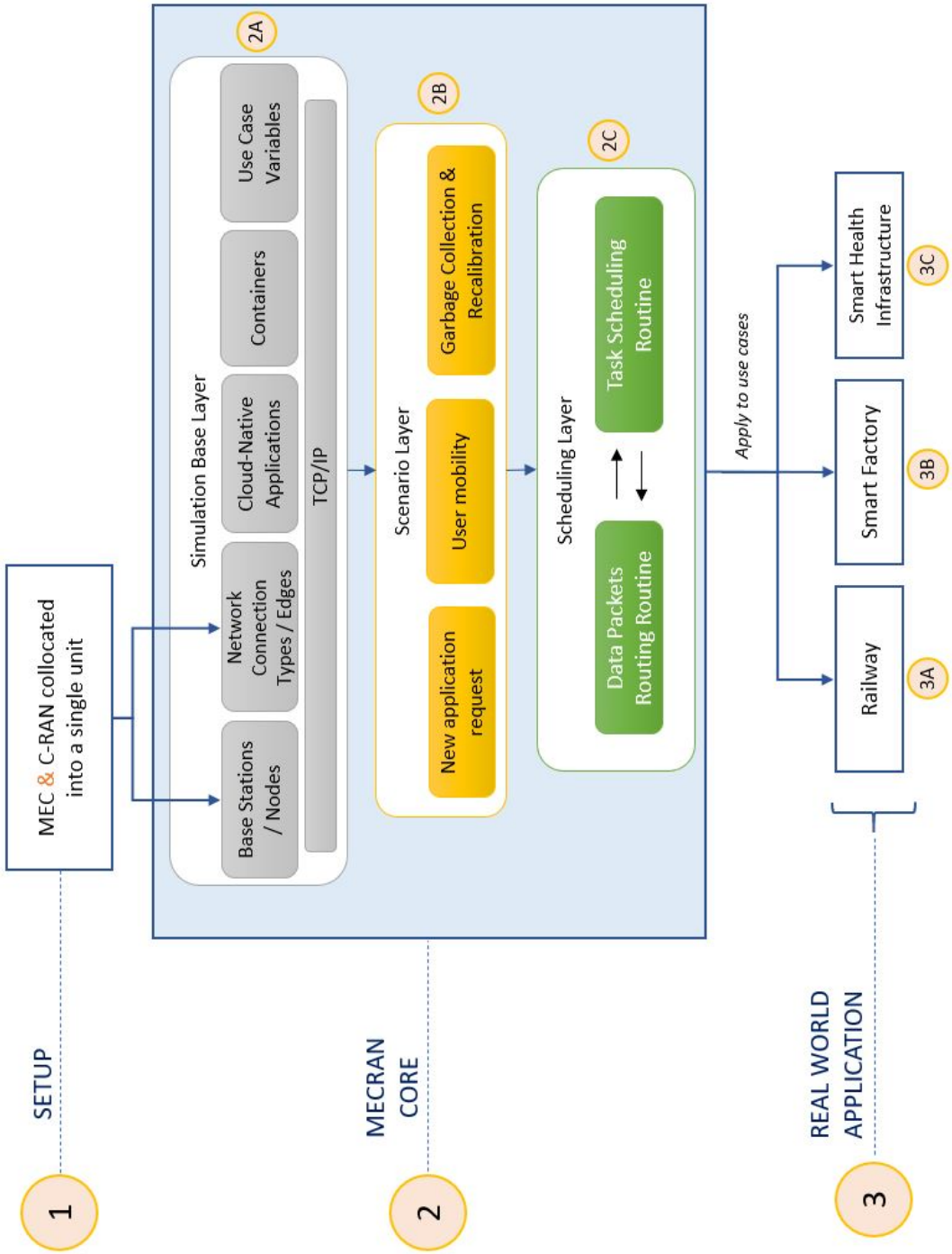

Figure 3.3: MECRAN implementation road map, highlighting the 3 main components of MECRAN, from research to application.

3.3 Proposed Solution Road Map

This research seeks to solve a practical problem and as such Fig. 3.3 outlines the practical application of science in tackling the cloud-native application latency problem. The work done can be described in 3 main parts:

- **Part 1 - MECRAN environment setup:** This illustrates the initialization and setting up of all the MECRAN components and artefacts, based on the research methodology in section 3.4.
- **Part 2 - MECRAN Core:** This is the heart of MECRAN which encapsulates a simulation environment (based on the earlier setup in Part 1) and the main operations of MECRAN i.e. optimizing data packets routing and scheduling cloud-native applications effectively. They complement each other and therefore co-exist in a symbiotic relationship within MECRAN. Efficient data packets routing predicts and alleviates bottlenecks from the network during the cloud-native application scheduling process, which aims to reduce latency. The MECRAN Core also contains a scenario layer (to handle important scenarios that underpin MECRAN) and the Transmission Control Protocol/Internet Protocol (TCP/IP) that facilitates the data packet movement across the network. See implementation of the MECRAN Core in chapter 4, the DPR in chapters 5 and the task scheduler in chapter 6.
- **Part 3 - Use cases / Real world application of MECRAN:** This part of the road map is crucial, as it demonstrates the value of this research in the real-world. The problem at hand is a practical one and consequently the outputs of this research must be applied. The section therefore highlights results of experiments done in conjunction with industry partners i.e. with Network Rail (United Kingdom) [77] testing smart railways and also with Nestle S.A (Switzerland) [78] testing MECRAN on smart factories. We are also in conversation with Noguchi Medical Research Institute (Ghana) [79] exploring the gap MECRAN fills as a smart health infrastructure towards Malaria elimination.

3.3.1 Overview of MECRAN Environment setup

In order to effectively schedule cloud-native applications at the edge of the radio-access network, an efficient end-to-end network model is needed as the foundation of the network environment. In this research, the working network model comprises of collocated Multi-access Edge Computing (MEC) and Cloud Radio Access Networks (C-RAN) resources. This is highlighted in Part 1 (of Fig. 3.3) with extended implementation details in Chapter 4. Chapter 4 further provides context and explains the details behind the full technical implementation of MEC and C-RAN as a single (collocated) unit.

MECRAN as a single unit embodies the full features of both MEC and C-RAN, necessary to deliver cloud-native applications and services at low latency and high spectral efficiency without compromising on the quality of service in an access device (i.e. mobile or IoT). The challenge at hand therefore is building and maintaining one management interface to coordinate scheduling and data packet functions across both MEC & C-RAN technologies.

3.3.2 MECRAN Core

The MECRAN Core is comprised of the *Simulation Base layer*, the *Scenario layer* and the *Scheduling layer*. These layers coordinate between themselves to host cloud-native applications at the required latency and bandwidth to the requesting mobile users. MECRAN Core is built based on the outputs of Part 1, which are used as the network nodes (base stations) and edge (network connection types) as shown in part 2 of Fig. 3.3.

Simulation Base Layer - 2A of Figure 3.3.

The simulation base layer establishes a base networking environment for MECRAN. This layer through the power of virtualisation, leverages the advantages of software-defined networking (SDN) and network functions virtualisation (NFV). The layer is made up of the following parts:

- **Transmission Control Protocol/Internet Protocol (TCP/IP):** Communication within the simulation base layer is facilitated by TCP/IP. TCP and IP are implemented to ensure that data is successfully sent and received at the intended destinations within the network. Typically a user's device communicates with a server hosting a cloud-native application of interest. However, when both devices communicate, data would have to be routed through other nodes first.

- **Base station / nodes:** The nodes in MECRAN's radio access network are mainly base stations which are leveraged as temporary host servers to facilitate edge behaviour. Virtual routers and switches are however introduced along the network to also assist with routing data around the network and to mimic the real world.

- **Network connection types / edges:** Fibre optic connection types are the main connection types used in MECRAN. However copper cables, fibre optic, digital subscriber line and satellite are introduced around the network based

on the specific needs of the use case in question. The connection types create links between nodes in the network and therefore are paths through which data can travel from a node to another.

- **Cloud-native applications:** This research focuses on delay-sensitive emerging cloud-native applications with strict requirements for low end-to-end latency for timely response and required performance. The International Telecommunication Union (ITU) has classified these emerging 5G applications into 3 service types: eMBB (enhanced Mobile Broad Band), URLLC (Ultra Reliability and Low Latency Communications), and mMTC (massive Machine Type Communications). Therefore depending on the use case of interest, the cloud-native applications may vary. Example of such applications include Artificial Intelligence (AI) based applications, Virtual Reality, Augmented Reality, High Definition Gaming, IoT connectivity, Industry Automation, Tactile Internet, to mention but a few. These thrive on very low latency, usually 10 ms or less, for performance [3]. Cloud-native applications are registered by their application IDs in 2A of MECRAN Core (Fig. 3.3) to allow them to be scheduled in MECRAN.

- **Containers:** The concept of containerization is a very interesting one. Containers allow the cloud-native applications to be packaged with all of its libraries and dependencies, and transported in easily portable 'containers' across any Linux environment [60]. This supports the mobile (constant moving) and agile nature of the MEC users. Containerization is another form of virtualisation, allowing applications to be available in a more distributed and scalable sense. Docker is used in our implementation; it is a software technology providing OS level virtualization, making it easier to create and run applications using containers [60]. Containers therefore have a very practical use case in MEC.

- **Use case variables:** There is a provision within the simulation environment to capture use case specific attributes or independent variables that may have

a statistically significant influence on the decision of which base stations we schedule the applications on, and consequently the latency. For example in our railway use case, we considered the diversity in rolling stock, speed of travel, track geometry, number of stations, traffic density, user equipment density, time and location. These together with the MECRAN's edge and RAN resources were fed into the deep learning based scheduling model, to approximate the best base stations to host a particular cloud-native application.

Scenario Layer - 2B of Figure 3.3.

MECRAN considers 3 main scenarios when scheduling cloud-native applications:

1. **User accesses cloud-native application for the first time** -
   When a user tries to access a cloud-native application for the first time, MECRAN compares the initial latency between the user and the distant host server to the latency required by the application in question.

   If the latency required by the application is greater than the tracked latency between the user and the distant server, MECRAN allows the cloud-native application to continue to be hosted on the distant data centre.

   However, if the latency required by the application is less than the tracked latency between the user and the distant server, MECRAN schedules or migrates the application to a base station at the edge of the radio access network and in close proximity to the user. By so doing, the latency between the user and the new temporary host server (i.e. base station) has a much higher chance of meeting the latency required by the application.

   MECRAN's task scheduler determines and schedules the application on to the base station at the edge, whilst the Data Packets Routing (DPR) routine provides intelligence to the scheduler about which routes to use when exchanging data between the user and base station to avoid bottlenecks. The

role of DPR is important as an application is scheduled on the base station whose position / location in the network allows the applications latency requirement to be met. This base station may not always be the closest (in terms of physical distance) to the user and therefore data being exchanged may have to go through other nodes before the intended destination. It is therefore crucial for MECRAN to know the best routes to use when facilitating data exchanges, in order to avoid bottlenecks. Bottlenecks arriving in the network makes scheduling the application redundant and therefore it is crucial for DPR is executed in its role to prevent them from happening, as the highest priority. Figure 3.4 shows an activity diagram of the above process.

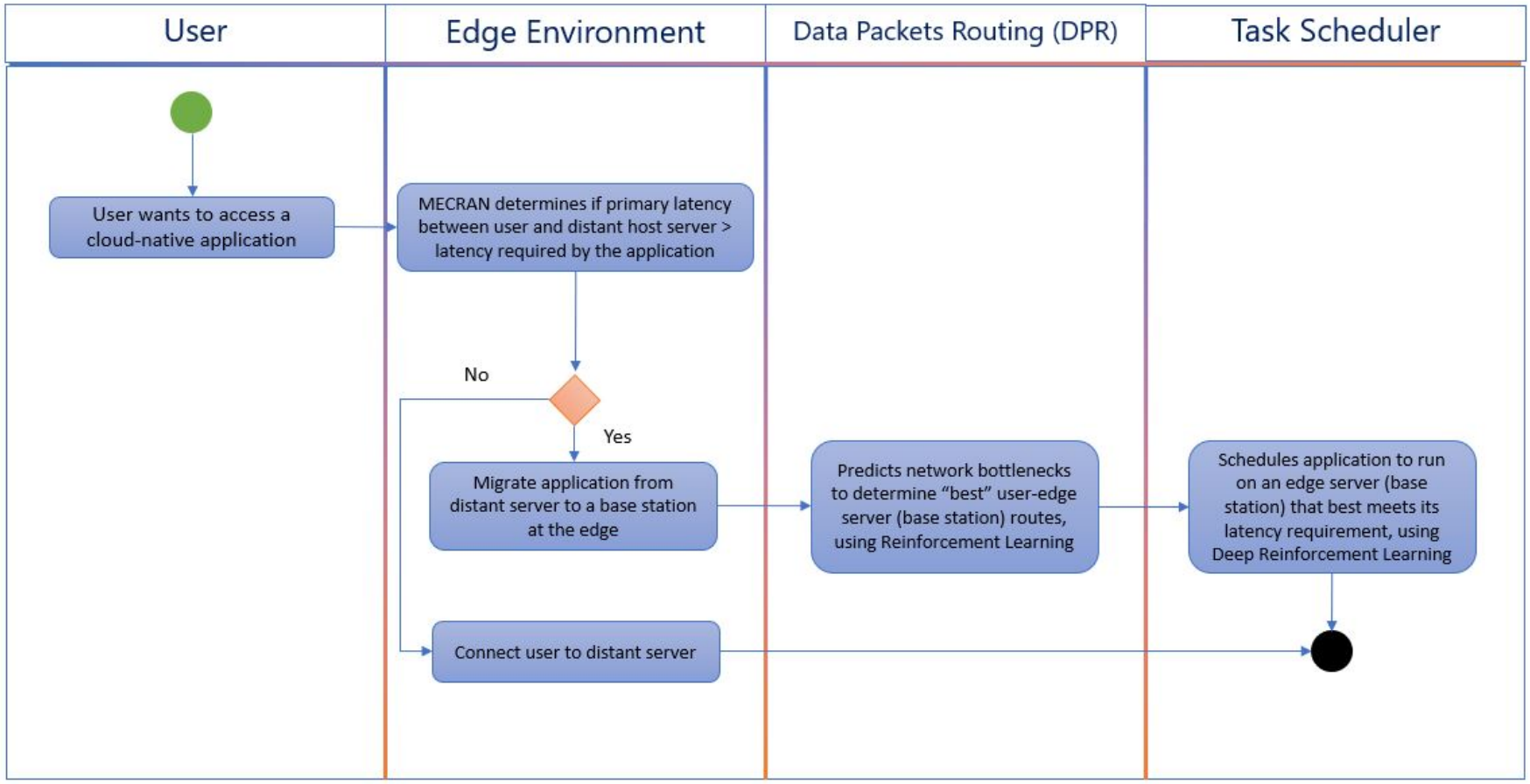

Figure 3.4: Scenario 1 Activity Diagram - User accesses cloud-native application for the first time.

2. **User location changes due to mobility** -

   One of the key aspects of the problem being tackled is the ability to support users that are constantly on the move. As an example a self-driving car conveying people from one place to the other or a user on a moving train playing a multi-player high definition game, are both considered to be "*mobile*" as the vehicle travels. Mobility is therefore an important requirement which is catered for by MECRAN.

   As a user exits the coverage area of a previous base station allocation due to mobility, MECRAN compares the primary latency between the user and the previously assigned base station, to the latency required by the application in question for performance. If the latency required by the application is greater than the derived latency between the user and the previously assigned base station, MECRAN allows the cloud-native application to continue to be hosted on the previously assigned base station.

   However, if the latency required by the application is less than the derived latency between the user and the previously assigned base station, MECRAN schedules / migrates the application to a new base station at the edge of the radio access network and in closer proximity to the user. By so doing, the latency between the user and the new temporary host server (i.e. base station) has a much higher chance of still continuing to meet the latency required by the application. Figure 3.5 shows an activity diagram of the above process.

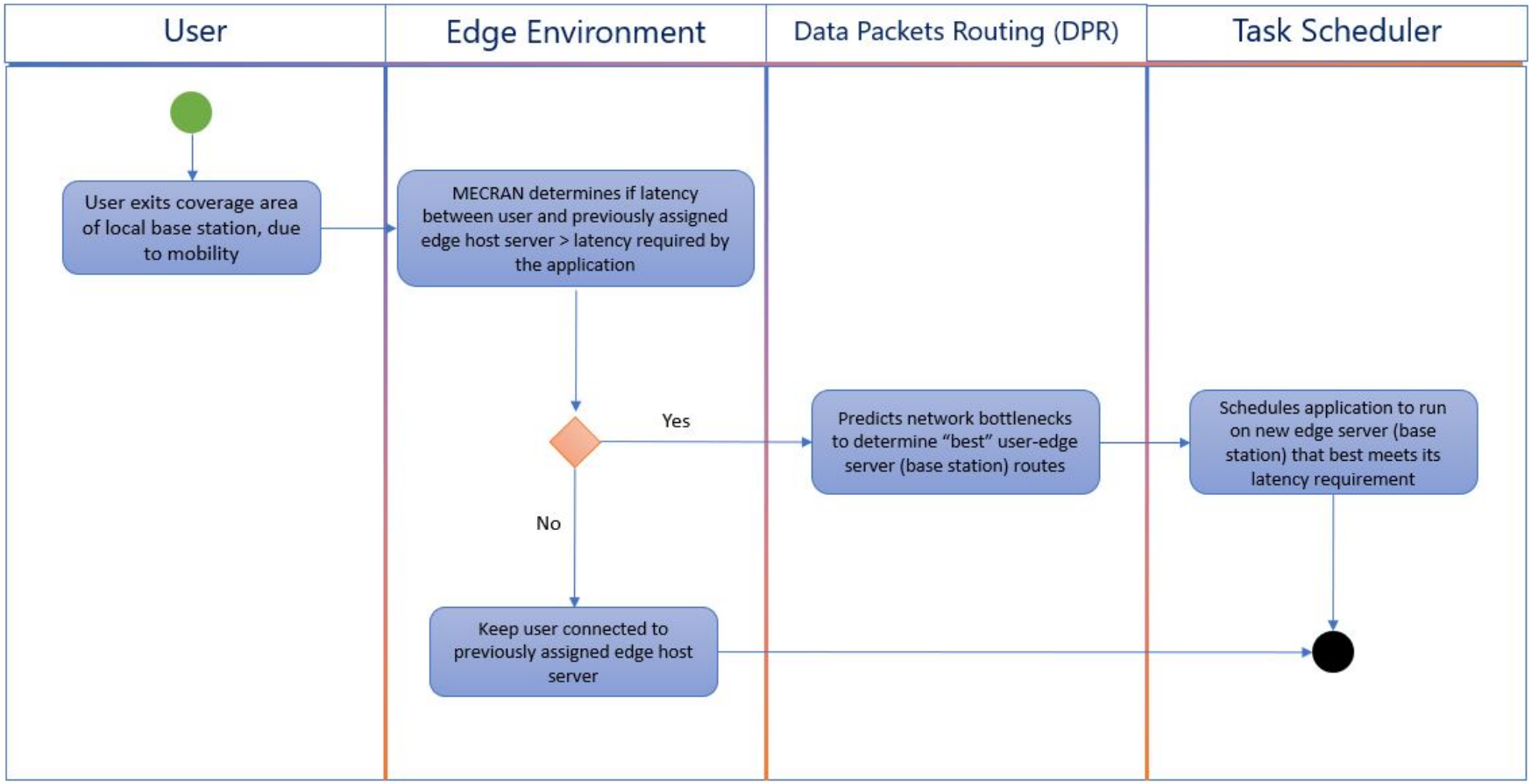

Figure 3.5: Scenario 2 Activity Diagram: User location changes due to mobility.

3. **Change in user traffic density, resulting in base station resources freeing up'**.

   MECRAN is designed to look after itself or to be self-sustaining. As users connect and disconnect from MECRAN, resource usage fluctuate and MECRAN therefore cleans up the environment by re-calibrating and clustering application-server allocations in order to make the best use of existing resources and also to take advantage of the benefits of the Cloud/Centralized Radio Access Network(C-RAN). This is a strategy for using resources optimally as well as reducing the operational cost for mobile network operators or owners of the network's infrastructure.

   As highlighted earlier in chapter 2, the architecture of C-RANs separates the Baseband Unit (BBU) from the Remote Radio Head (RRH). The RRHs remain at the cell site, while the BBUs are aggregated into a centralized office (also known as a BBU pool / hotel), with a centralized management that processes calculations digitally in a base station. From that perspective MECRAN cleaning up the environment when resources are freed up and clustering applications to be hosted in a pool, reduces the costs on the maintenance of the network. Some of the cost of maintenance pertains to cooling provision, power supply and a back-up battery, to mention but a few.

   Users over time retire from accessing their cloud-native applications from MECRAN. For example, users may alight at their planned destinations from a train and may discontinue accessing their cloud-native applications. As users stop accessing their applications from MECRAN, base station resources are freed up. MECRAN ascertains the resource capacities and states of all base stations and with this intelligence, it progresses to re-calibrate and aggregate the existing applications to be hosted in a centralised pool, if their latency requirements can be met by the centralised pool. This aggregation also prevents the under-utilisation of base stations across the network.

However, if the latency required by the application is less than the latency between the user device and the centralised pool, MECRAN schedules / migrates the application to a new base station at the edge of the radio access network and in closer proximity to the user. By so doing, the latency between the user and the new temporary host server (i.e. base station) has a significantly higher chance of meeting the latency required by the application. Figure 3.6 shows an activity diagram of the above process.

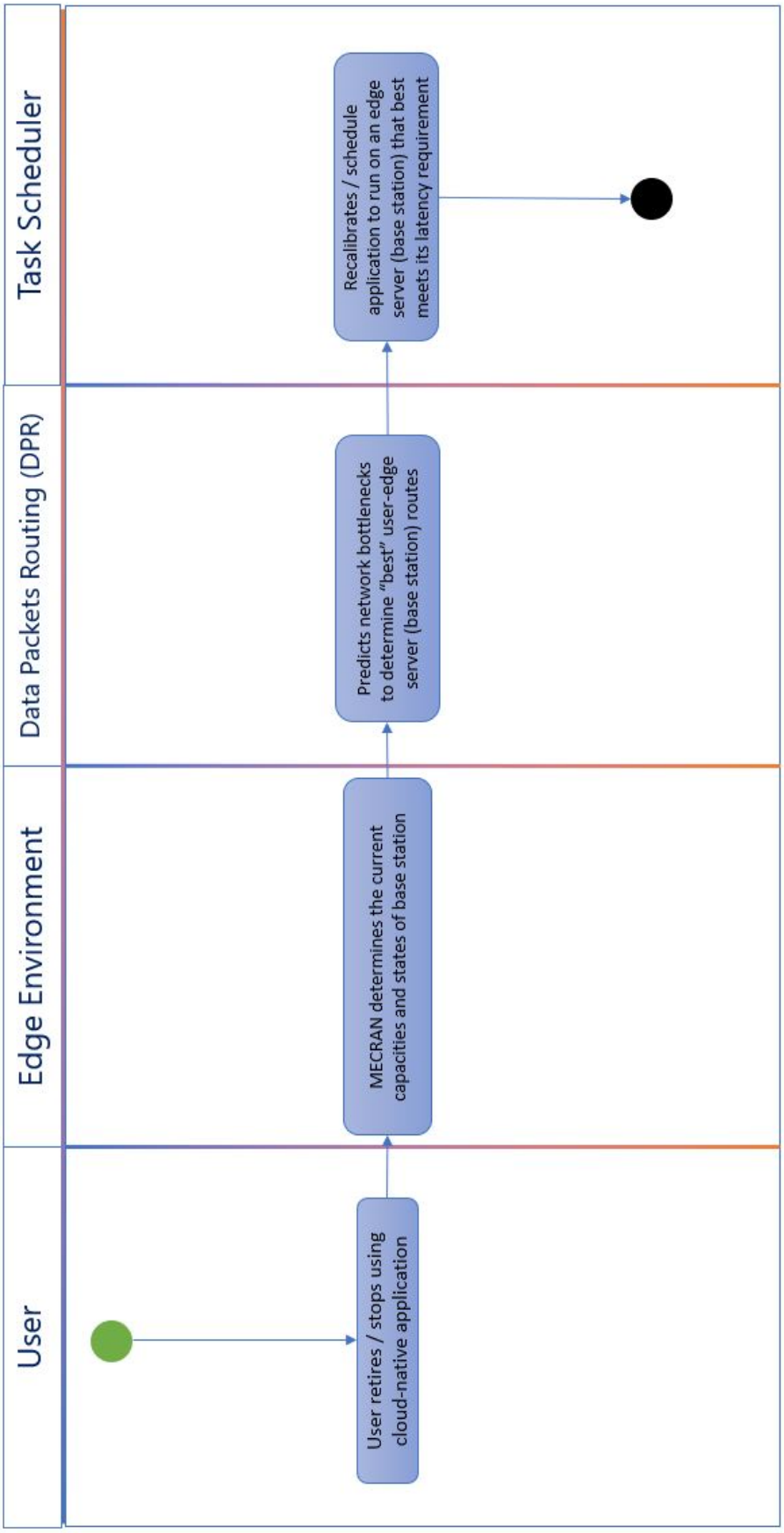

Figure 3.6: Scenario 3 Activity Diagram: Change in user traffic density, resulting to base station resources freeing up.

Scheduling Layer - 2C of Figure 3.3.

MECRAN uses the power of virtualisation to schedule cloud-native applications to be hosted in close proximity to their users in order to reduce latency and meet their QoS requirements. Whilst scheduling an application at the edge is important, it is also equally important to prevent bottlenecks from happening by managing the routes through which data flows from the base station to the user's device and vice versa. The cloud-native application scheduling in MECRAN is therefore facilitated by 2 routines:

1. **Data Packets Routing (DPR) Routine for bottleneck prediction, detection and prevention, using Spatial-temporal Graph Neural Network (STGNN)**

   Whilst the deep reinforcement learning based task scheduler focuses on "where" best to host a cloud-native application, it is also important to have an associated data packet routing routine that focuses on "how" best data travels within the network. This alleviates bottlenecks between the base station and the application user. Network nodes such as servers and routers process data packets stored temporarily in buffers, through several layers (Application Layer, Transport Layer, Network Layer, Datalink Layer) of the TCP/IP model for packet switching networks [80] - see implementation in section 5.3. Buffers can become full causing data loss, and data packets being processed can encounter some delays, resulting in bottlenecks across the network. Alleviating these bottlenecks and data losses within the network are the motivations behind the Data Packets Routing (DPR) routine, which complements the efforts of the task scheduling routine. The scheduler and routing routines therefore operate in tandem to achieve desire outcome.

   Te Spatial-temporal Graph Neural Network algorithm as described in chapter 2 and implemented in chapter 5, learns how data packets flow within the network, with an aim to understand what causes bottlenecks, predict them

and reroute data packets in conjunction with chosen base stations to meet the application's performance requirement.

2. **Task Scheduling (TS) Routine, based on Deep Reinforcement Learning**

   The Task Scheduling routine feeds a deep learning model with inputs as described in the simulation layer, to make a decision on where to schedule the cloud-native application. The routine takes inputs describing the cloud-native application (such as latency, bandwidth, priority), edge and the radio access network resources (such as available base stations, base station capacity, distance, weather temperature, wind-speed, user equipment density) and the use case variables (such as speed, railway traffic density, time and location, in the railway use case). The DPR recommends bottleneck-aware alternative data routing paths to the scheduler as a way of optimizing the network and maintain lower latencies. The scheduler therefore takes this as an input into the model.

   The scheduler model has two outputs - the best two base stations within which to run a particular cloud-native application. The first is the primary location to execute the application, with the second base station on standby as back-up. The back-up is important in this real-time environment where unforeseen situations occur. A typical example is; a server may have to adjust to a live application's operational needs by scaling and consuming more of its compute, network and storage resources. Assuming this server was predicted to host a new task moments ago by MECRAN, MECRAN will now need to fall on an alternative second base station as the current one may not have enough resources to serve the task in context. This ability for MECRAN to dynamically allocate resources, makes the system as a whole appear to have 'inexhaustive' or 'unlimited' resources.

   Chapter 6 explains that, in search for the best base station, MECRAN takes an exploratory or exploitative reinforcement action to test a range of

base station and observe the impact on the application's latency overtime. Depending on the results of how an application performed based on the route recommendation, the reinforcement learning routine, self-learns to improve its recommendation strategy.

Both routines complement each other in facilitating the scheduling of applications to run at the relevant base stations as well as also managing the best routes packets should travel from the host base station to the user. The Task Scheduling (TS) and Data Packet Routing (DPR) routines are deployed within this ecosystem to provide a performance-enabling, resource-rich and robust environment that allows applications to flexibly run on 'inexhaustive' and reliable compute, network and storage resources.

3.3.3 Use Cases / Real world application of MECRAN

The following two use cases have been identified as a way to review the practical benefits of MECRAN when deployed in all (greenfield and brownfield) environments:

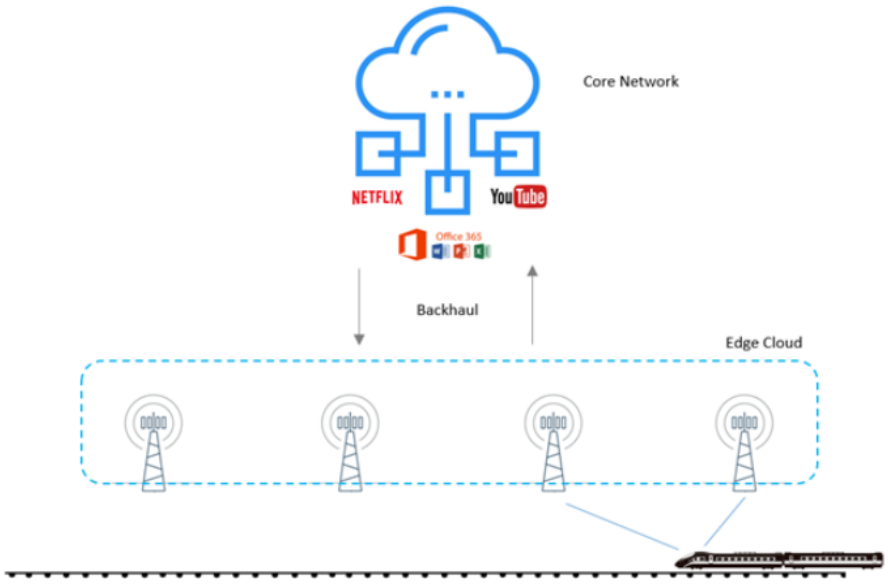

Figure 3.7: The railway use case diagram - this use case addresses both railway applications and consumer services .

1. **Railways of the future:** In the railway context we imagine a train or series of trains (i.e. Mainline, Regional, Suburban, Urban, Metro or Freight) equipped

with IoTs and 5G applications, traveling from Point A (for example London Paddington) to Point B (Newcastle upon Tyne), with evolving passenger headcount numbers depending on whether it is peak or off-peak time travel. With the railway IoT and 5G applications categorized as Critical, Performance Enhancing and Business application depending on their function on the train, MECRAN was able to maintain high application performance giving priority to the critical applications that the train depends on. This was done by dynamic scheduling of applications to run at the closest base station on route whilst the train travels at an average of 124 miles per hour, in such as way that the Quality of Service (QoS) to the operator is not interfered. Examples of some of the railway applications and IoTs include rail robots, drones, railway cameras, multi-train voice comms, real-time video, critical voice, critical data. These must execute alongside the cloud-native applications customers are also using on board, such as Google Stadia (a cloud gaming service that can allow players up to 4K resolution at 60 frames per second with support for high-dynamic-range), Netflix and so on.

2. **Smart Factory**.

   The adoption of 5G is enabling traditional manufacturing industries to explore smarter ways of delivering value through game-changing digital applications that can allow factory equipment to make safe and accurate autonomous decisions based on what is happening on the factory floor [5]. Some of the motivations behind manufacturers going smart are, to increase production, eliminate bottlenecks in the production process, reduce production lead times and to optimise production schedules [81]. By introducing emerging technologies such as the Internet of Things (IoTs), Artificial Intelligence (AI) and edge computing, manufacturers can seamlessly integrate all steps of the manufacturing process – such as design, manufacturing, supply chain and operations – thereby bringing flexibility and more importantly, reactivity to competitive markets [81]. These applications/ services however have uncompromisable requirements for low latency and high bandwidth, which 5G

alone cannot deliver and it is even more challenging in such an environment where there are a lot of moving parts on the factory floor; be it an AI-powered robot or factory staff performing an inspection using Augmented Reality HoloLens [5][82] [83].

Despite the benefits of emerging technology in manufacturing, latency continues to be a problem because, the further apart a client and host are from each other on the network, the longer it takes to transfer data packets between them. In the case of manufacturing environment, whilst clients and hosts are within the same building, scheduling is still important because some manufactures use a large number of micro cells (also known as *small cells*) to target coverage for specific areas as supposed to one large base station covering the entire manufacturing site, where many parts of the site may not need network coverage. From that perspective scheduling cloud-native application across multiple micro cells whilst the user is moving is key.

To further lower latency in private or hybrid cloud enabled manufacturing environments, MECRAN uses machine learning to optimize the round-trip delivery of data at low latency, through dynamic resource scheduling of cloud-native applications, to run close to a moving or mobile client in the factory, at the edge of the radio access network. In this use case, we have modelled the use of an Augmented Reality HoloLens [82] in a hybrid cloud of a food and beverage manufacturing environment. The HoloLens is used for factory remote auditing of refrigeration, ammonia, boilers, steam generations, air generation and nitrogen generation. It is also used for modifying productions lines in real-time.

3.4 Underlying Agile and Software Development Life Cycle-Based Methodology

The methodology in Fig. 3.8 is used throughout the sub-sections of the road map in chapter 3, in implementing MECRAN. The methodology follows a software development life cycle and it contains 3 main phases - Plan & Analyse, Design & Build, and Integrate - for building and the validation of the outputs.

The Plan & Analyse phase describes the entry and termination points of the methodology, reflecting the initial inputs as well as the final output of this research. The build dimension of the Plan & Analyse phase involves performing requirement analysis and planning the basic building blocks for collocating MEC and C-RAN as well as formulating a strong hypothesis for the Data Packets Routing (DPR) and the scheduling algorithms. The validation dimension of the Plan & Analyse phase analyses and formalises the results of the entire research work. The Plan & Analyse phase was used to plan the basic project approach, conduct feasibility study and eventually formalise the findings of the project.

Based on an agile methodology, the Design & Build phase is where all the work has been done. Here the work was delivered in 1 / 2 weeks' sprint cadences. My supervisor and I met on Mondays at 1 PM. These meetings were an opportunity to show any work that has been done, review what went well, what did not and recommendations on ways to improve the work done. We planned what the focus for the coming week should be, keeping an eye on new literature from the edge computing community. The build dimension of the Design & Build phase involved the iterative design, implementation and evaluation (through unit and function testing) of the machine learning based Data Packets Routing (DPR) and scheduling routines. The validation dimension of this phase was where both routines were tested against the 2 real world use cases. Publications are produced as parts of the outputs of step 5 where both DPR and the scheduling algorithms were implemented and also at step 8 during the validation of the algorithms against the use cases.

The integrate phases marked the end of the build dimension where all development work was consolidated in a presentable format for publication or presentation.

This included building a management interface showing real-time analytics on the performance of MECRAN - see chapters 5 and 6. The integrate phase as well marked the beginning of the validation dimension where developed work was validated on real world use cases - see chapter 7 for the full implementations.

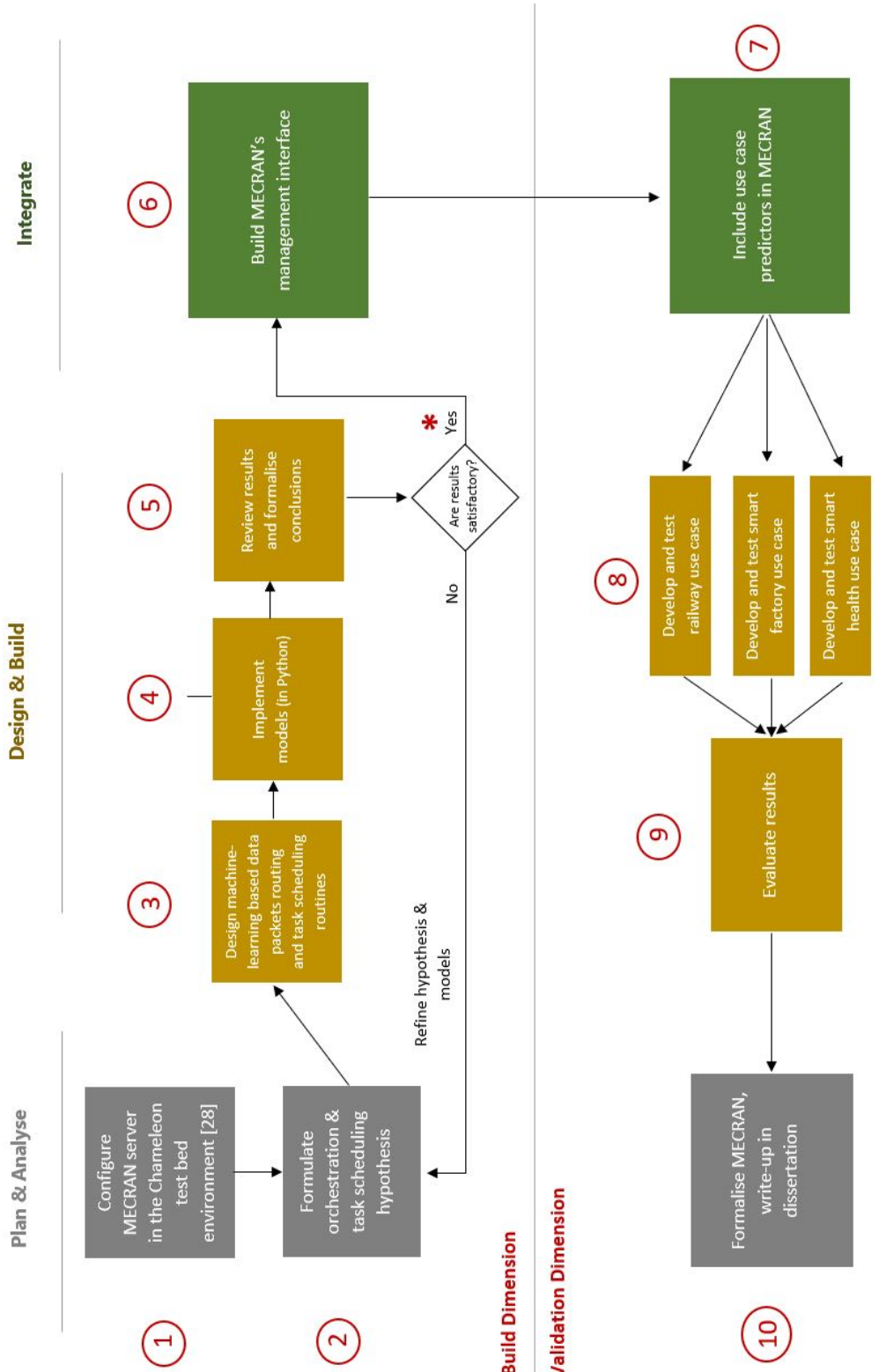

Figure 3.8: MECRAN's methodology - based on software development life cycle principles, to build and test the foundation fabric of MECRAN.

4

Implementation: MECRAN Environment Setup

Contents

4.1 Introduction & Purpose

The European Telecommunications Standards Institute (ETSI) in one of its initial publications [16], considers the idea of using the same C-RAN infrastructure for MEC as a topic of 'ongoing discussion'. Even though ETSI highlights some potential

challenges in the same publication, there have been no real practical implementations of this phenomenon in industry to validate its feasibility. The goal of this section is to practically validate the feasibility of collocating MEC and C-RAN (MECRAN). In chapters 5 and 6, we build and test the data packet routing and task scheduling routines in this MECRAN infrastructure based on this section.

4.2 Collocating MEC & C-RAN

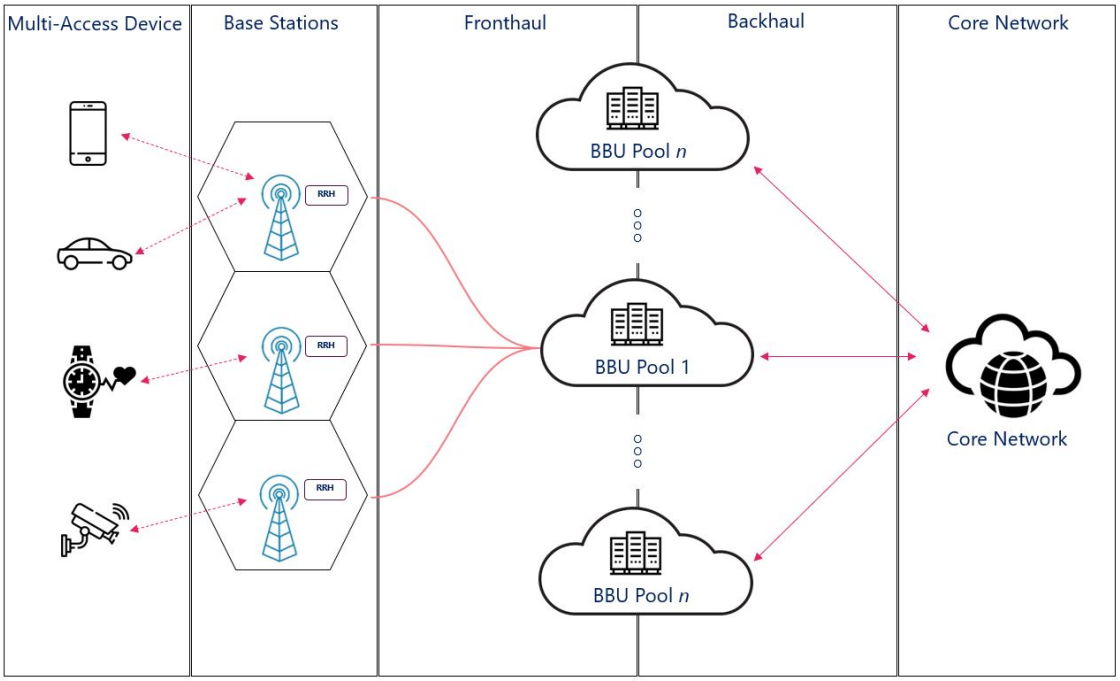

Figure 4.1: MECRAN's working telecommunication network.

We want to build a multi-access edge telecommunication network based on a cloud radio access network in line with our working telecommunication network in Fig. 4.1. We use the Chameleon testbed [**Chameleon2**][84] as a large-scale reconfigurable platform that supports bare metal reconfiguration. In order to mimic a real-world latency aware context, we set up our telecommunication network (as per Fig. 4.1) and the remote / distant data centre (as show in Fig. 3.2) whose latency we are optimizing in two different locations:

- **Our telecommunication network**: this setup is based at the Texas Advanced Computing Center (TACC) of the University of Texas at Austin

(represented by CHI@TACC)

- **The remote / distant data centre**: this setup is based at the University of Chicago (represented by CHI@UC).

The distance between both locations is approximately about 1000 *miles* with an average latency of approximately 40 *ms* on chameleon. The idea here being that MECRAN users are based in Austin (Texas, USA) trying to access cloud-native applications in Chicago (Illinois, USA). Cloud-native application in Chicago that have strict requirement for ultra-low latency are dynamically migrated using *Containers* [85][86] (and scheduled by MECRAN's DPR and Task Scheduler routines) to base stations closer to the users in Austin.

4.2.1 MEC

Chameleon is based on *Open Stack* [87] and we have extended this architecture with Ceph (an open source distributed storage system) [88] to enable the pooled BBU paradigm and separate out RRHs from base stations, as motivated by C-RAN. We established the following setup (Table 4.2.1 and Table 4.2.2) as per our telecommunication network in Fig. 4.1:

RAN Object	Bare Metal	Virtual Machine / Operating System	Ceph Component
Multi-access devices	Instance 1	VM 1 / Linux CentOS 7	Ceph Admin
Base station RRHs	Instance 1	VM 2 / Linux CentOS 7	Ceph Admin
Base station BBU 1 (part of BBU pool)	Instance 1	VM 3 / Linux CentOS 7	Ceph Monitor 1
Base station BBU 2 (part of BBU pool)	Instance 1	VM 3 / Linux CentOS 7	Ceph Monitor 2
Base station BBU 3 (part of BBU pool)	Instance 1	VM 3 / Linux CentOS 7	Ceph Monitor 3

Table 4.2.1 Telecommunication network at CHI@TACC

RAN Object	Bare Metal	Virtual Machine / Operating System
Remote data centre	Instance 2	VM 1 / Linux CentOS 7

Table 4.2.2 The remote / distant data centre at CHI@UC

To achieve the above, a project called *Dynamic Resource Allocation & Task Scheduling for Edge Cloud* (DRASTEC - CH-822670) with ID 352 was setup in Chameleon in order to assign a cloud resource to utilize the testbed sites. Bare metal nodes were reserved by creating a lease as per Fig. 4.2 - one at CHI@TACC and the other at CHI@UC. A bare metal node is a whole dedicated physical server allocated to a project. Bare metals are reserved for a fixed time period and they are initialized by creating an instance; which is a bare metal node that has been launched with an operating system image.

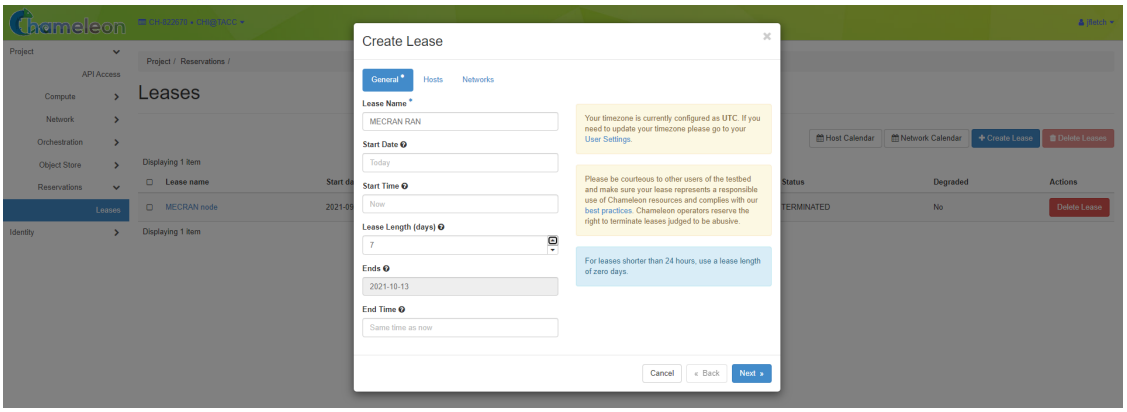

Figure 4.2: Creating a lease in Chameleon to reserve a bare metal node, which is a whole physical server.

In Chameleon, a bare metal instance is launched via a wizard (as in Fig. 4.3) in the Compute menu, where the user specifies configuration details of the instance.

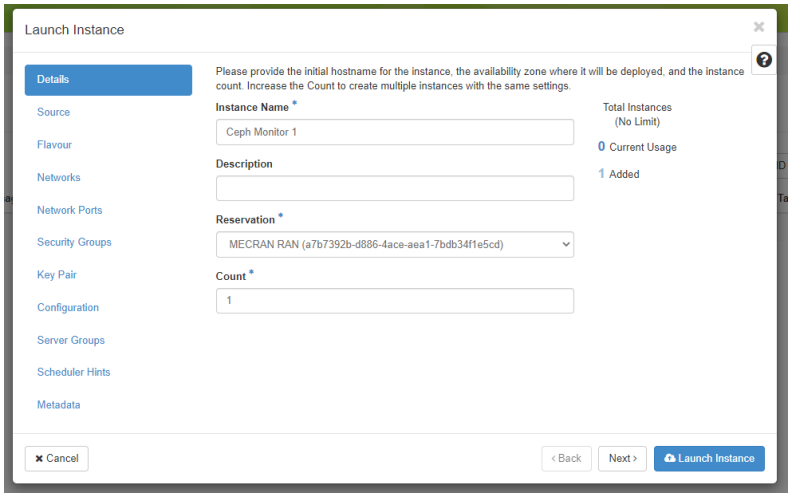

Figure 4.3: Creating an instance based on the leased / reserved bare metal in Chameleon.

The details cover domains such as instance specific details (e.g. instance name, description, reserved host), source (the template used to create an instance), flavour, network information and ports, security groups, key pair to allow secure remote access, a cryptographic network protocol, configuration, server groups and other metadata. The launch wizard allows users to associate an instance to the reserved host and use an image / instance source template to create an instance based on a specific operating system, as see in Fig. 4.4. All of our instances and virtual machines are using a CentOS 7.

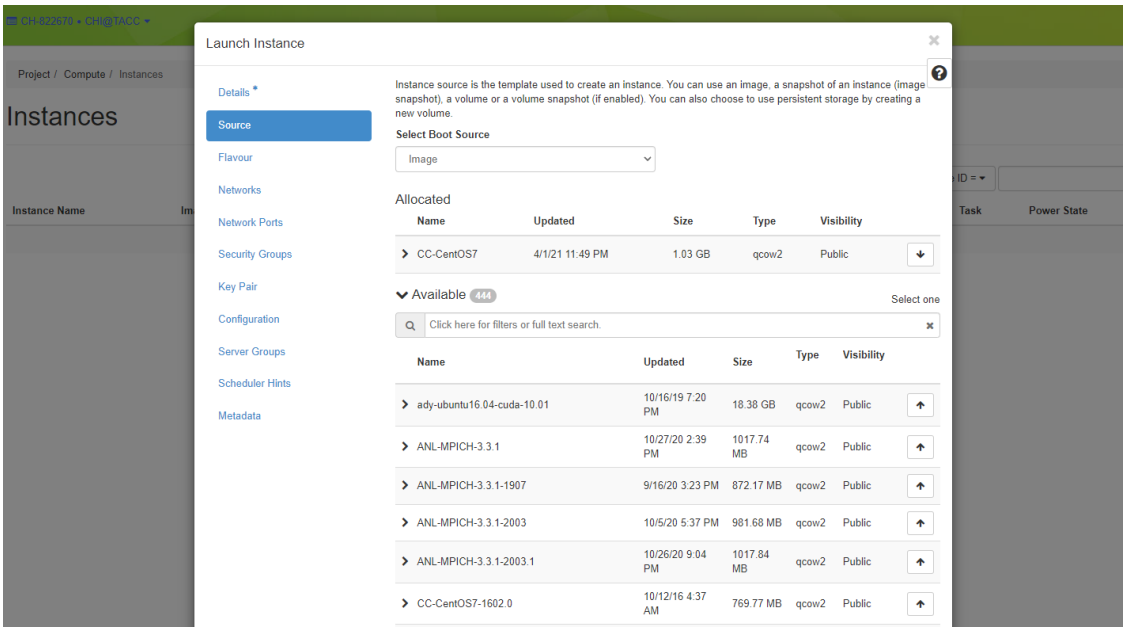

Figure 4.4: Associating an instance to the reserved host and using an image to create an instance on CentOS 7.

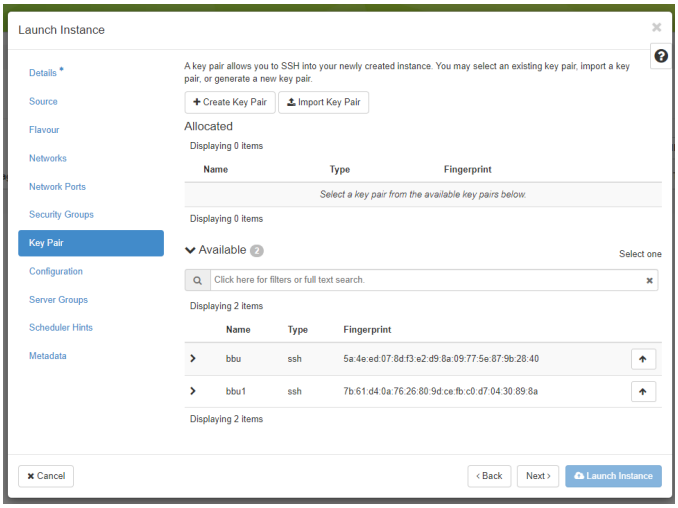

Figure 4.5: Creating public and private keypairs for our instance to allow remote log in into instances remotely via Secure Shell (SSH). Chameleon allows selecting an existing key pair, import a key pair, or generate a new key pair.

With the above setup, we introduced the hypervisor vSphere (from VMWare, formerly known as ESXi) [89][90] on both bare metal instances at CHI@TACC and CHI@UC to allow virtualizing servers on bare-metal and creating the virtual machines listed in the introductory tables of subsection 4.2.1 for our radio access network. See Fig. 4.6. Ceph is used to create the cloud cluster from the virtual machines created by the hypervisor. This cluster is an important setup to support the BBU pool as described in section 2.5.

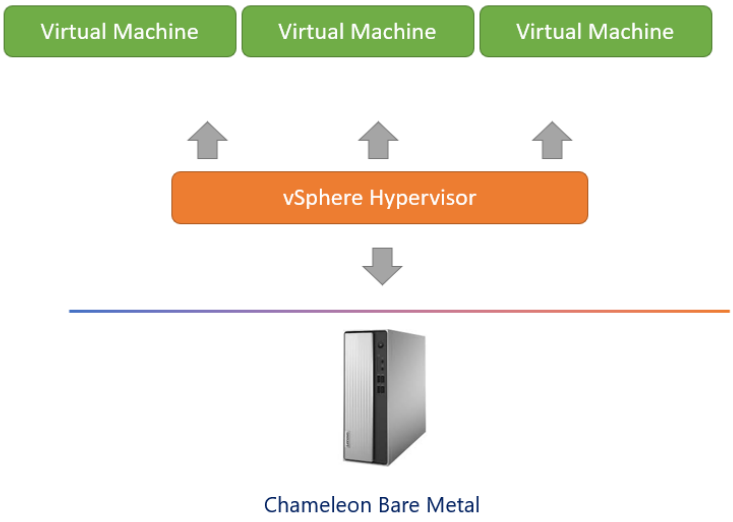

Figure 4.6: VSphere hypervisor on Chameleon bare metal instances to help further virtualize servers for MECRAN's RAN.

4.2.2 C-RAN

We know from sub-section 2.5.4 that the architecture of a traditional Radio Access Network (RAN) consists of decentralised stand-alone base stations covering an area. The way the base stations works is that it processes and transmits its own signal to and from the mobile terminal, and furthermore forwards the data payload to and from the mobile terminal and eventually out to the core network via the backhaul [55][56]. Every base station also maintains a number of components that keeps it running which form part of the operating costs mobile network operators incur [16], such as cooling, back haul transportation, backup battery, monitoring system, all sit with the base stations.

As IoT adoption increases and new types of applications such as Augmented Reality are being used, there is an increasing demand for more advanced intelligent wireless network architectures to cut networking cost and meet the future demands for lower latencies, higher bandwidth and lower power consumption [16]. It is with these motivations that China Mobile introduced the idea of moving the functions of base stations into an innovative the centralized cloud server, optimizing resource and energy consumption [56]. This centralized baseband pool regroups many BaseBand Units (BBUs), which are composed of high-performance programmable processors and real-time virtualization technology that perform baseband (PHY/MAC) processing. BBUs are built on general purpose platforms and are interlinked by high bandwidth low-latency optical network to distributed Radio Remote Heads (RRHs), located at the remote site.

C-RAN demands a fundamental architectural enhancement of the base stations in the radio access network. As shown in MECRAN's working telecommunication network in Fig. 4.1, we use Ceph to build BBU cloud clusters for the BBU pool. This allows separating out the RRHs from the BBU, an optimised network architecture based on C-RAN paradigm. The BBU pool and separation of base station component functions is the difference between a C-RAN and a traditional radio access network.

A cluster of 9 MECRAN nodes (Fig. 4.7, 4.8) has been setup using OpenStack, Ceph (an open source distributed storage system) on Chameleon. Within the

MECRAN Tier (Fig. 4.7), Node 0 (green) in Ceph terms is an Object Gateway Daemon. It keeps track of runtime analytics such as system load, performance metrics and storage usage. Nodes 1, 2 and 3 in orange are Monitors; they run maps on all the other components to monitor the state of the cluster. The Admin's (Node 4 in grey) responsibility is to configure the nodes and deploy the cluster. The role of the Object Storage Daemons (Nodes 5, 6, 7 in yellow) is to handle data storage, replication, rebalancing and recovery; while the Meta Data Server (Node 8 in blue) is needed for efficient cluster file storage capabilities.

The Ceph Monitors only represent the digital function unit of the base station or Baseband Unit (BBU). The Ceph Admin additionally also is used to simulate the base station RRHsT and multi-access devices to interact with the BBU – in

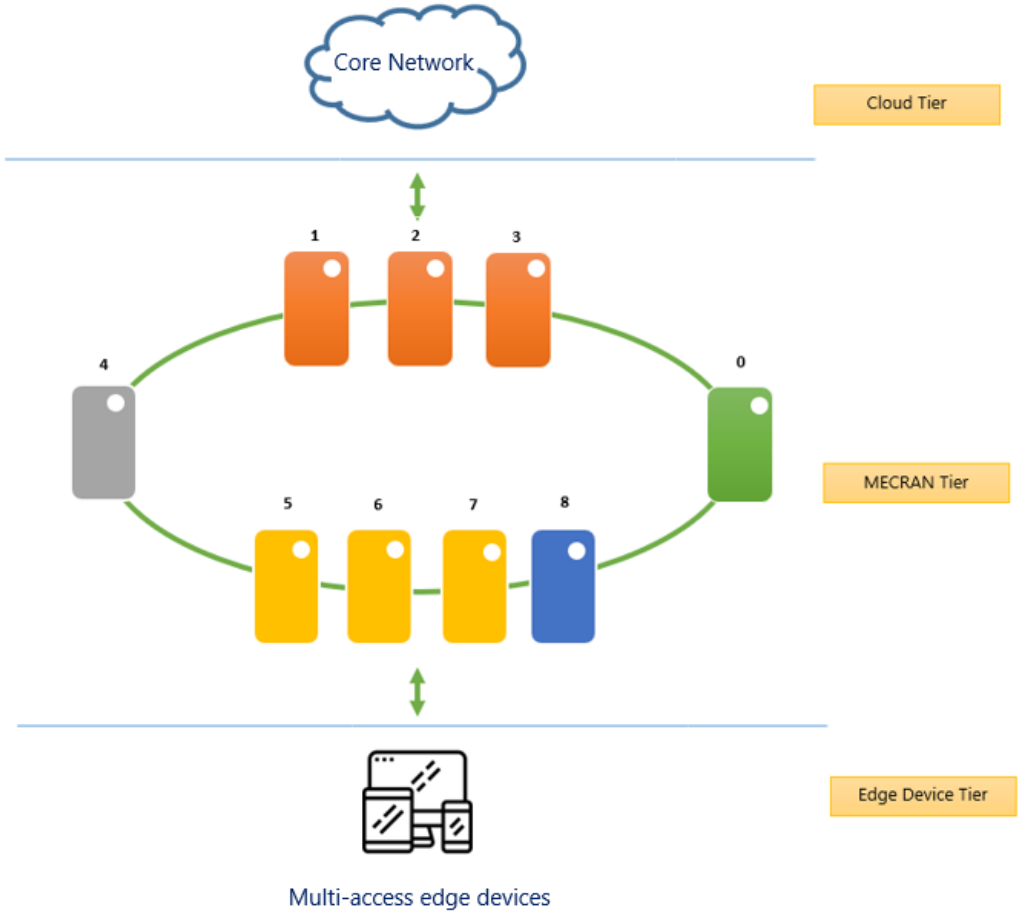

Figure 4.7: MECRAN BBU Pool, motivated by the C-RAN paradigm to separate RRHs from BBUs.

this case, the Ceph Monitors nodes.

Node	IP Address	Ceph Component
0	129.67.193.230	Object Gateway Daemon
1	129.67.193.231	Ceph Monitor 1
2	129.67.193.232	Ceph Monitor 2
3	129.67.193.233	Ceph Monitor 3
4	129.67.193.234	Ceph Admin
5	129.67.193.235	Ceph Object Storage Daemon 1
6	129.67.193.236	Ceph Object Storage Daemon 2
7	129.67.193.237	Ceph Object Storage Daemon 3
8	129.67.193.238	Ceph Metadata Server

Figure 4.8: MECRAN BBU Pool cluster detail.

Setting up MECRAN's infrastructure is an important first step, as this step is crucial in validating the viability of hosting MEC and C-RAN on a single infrastructure in order to establish a long term symbiotic relationship between both units or technologies. This setup continues to be important as the MECRAN telecommunication network is configured to possess full MEC and C-RAN capabilities, upon which the data packet routing and task scheduling routines will be performing. MEC and C-RAN have a number of intersecting goals, interests and characteristics. Virtualization is also a common attribute of their core setup. Fundamentally, they both are purposed to reduce latency, increase bandwidth as well as lower energy consumption levels. MEC and C-RAN complement each other and collocating both, makes the economics of each other more attractive to a mobile network operator [16].

4.3 MECRAN Core - Simulation Base Layer

We aim to build a dynamic application hosting scheduler, whose aim is to optimally reduce round-trip latency (even as a user moves) and does so making the most efficient use of the network resources. This is the primary purpose of MECRAN Core. This subsection elaborates on a fundamental layer of MECRAN Core called the Simulation Base Layer.

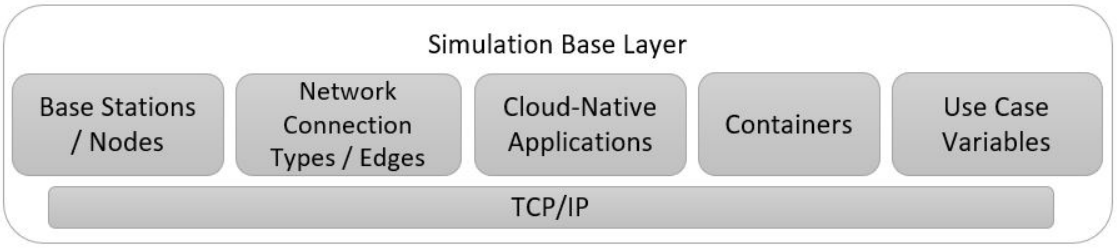

Figure 4.9: Simulation Base Layer as presented in the solution road map in Section 3 - based on virtualization, this Python-based environment, comprises of the basic building blocks that make up MECRAN

Base stations' Base Band Units (BBUs) are re-purposed as temporary servers to host cloud-native applications closer to users, mimicking edge behaviour. The base stations are connected to each other by edges or connection types such as fibre optics. Cloud-native applications are transported from their primary servers (in data centres) to the temporary servers (base stations) in containers. Given x base stations, MECRAN's scheduling routine's goal is dynamically scheduling cloud-native applications optimally on base stations to achieve reduced latencies, for these applications to be able to meet their performance requirements. Use case variables (such as trains in the railway use case) are factored into this layer and all elements in the simulation base layer communicate using TCP/IP protocol. The data network generated is also managed by MECRAN's data packet routing routine to prevent bottlenecks from occurring.

The simulation base layer is based on virtualization and comprises of the basic building blocks that offer an IT service environment and cloud-computing capabilities at the edge of the radio access network / mobile network. They are

compute, network and storage resources. This layer inherits the MEC & CRAN-based base station (Remote Radio Heads (RRHs) and Base Band Units (BBUs)) and their associated connection type blueprint described in the Setup stage of the road map in sub section 3.3.1. (which is also implemented in sub section 4.2).

The simulation base layer is the environment (developed mainly in Python 3) in which the other 2 MECRAN Core layers (Scenario and Scheduling) are built and tested. Figure 4.9 shows all the components of the simulation base layer and the next sub sections will deep dive into each element, including the data network that underpins the simulation base layer and facilitates communication between the elements in MECRAN. Network communication is important as it defines a set of protocols / rules / standards that allow the elements in the simulation base layer to talk with each other efficiently.

4.3.1 Geographic area coverage / Maps

Building from section 4.2 where we have shown that Multi-Access Edge Computing and Cloud-Based / Centralized Radio Access Network (C-RAN) can be symbiotic and therefore can be hosted within the same hardware environment, the next step here is to use this architecture as base stations in MECRAN. A key scoping question to answer at this point is *where* to setup the radio access network. This is important because geographies are different, which can influence how the radio access network should be architectured. In fact, the geographic location will determine whether we are dealing with a brownfield or greenfield site. The simulation base layer therefore has been built with the capability of simulating any geographic location to support the radio access network.

Cloud-native applications that users are trying to access are scheduled by MECRAN within a known geographic area, at the edge of the radio access network. This geographic element is key because the distance between where the user is based and the cloud-native application's primary data centre is important in determining the round-trip latency and whether that applications needs to be scheduled closer to the user to meet its performance requirements.

In order to simulate our benchmark scenarios (new application request; user mobility; garbage collection & re-calibration) from section 3.3.2, we assume a geographic area g_x. G_x has at least the following descriptors:

Descriptor	Description
Geographic area (g_x)	An area of size πr^2 is specified, where radius r is x *miles* between the centre of the geographic location and the circumference of the circle surrounding it, as shown in Fig. 4.10.
Base station (b_x)	A finite number of nodes or base stations (b_x) and their area coverage capacity.
Link / Edge / base station connection types (l_x)	A finite number of network edge or links (l_x). No base station is an orphan; meaning every base station has at least 1 connection or link to it.
Cloud-native applications (a_x)	A finite number of time-sensitive cloud-native applications (a_x) being accessed by users in the geographic area.
Users (u_x)	A finite number of cloud-native application users who are either mobile or stationary (u_x).
Data centre / remote servers (s_x)	A finite number of data centres or distant servers (s_x).

Table 4.1: The basic descriptors of a MECRAN geographic area

As referenced earlier, we take a geographic location (such as London) and a radius from this location to determine the range of interest, in terms of coverage. We are able to process a city / town / geographic address information into Latitude and Longitude coordinates (a process called *geocoding*) using Geopandas and Geopy libraries in Python. The radius r is added to the latitude-longitude coordinates to form the range. See code below:

```
# Import libraries
from geopy.geocoders import Nominatim
import math
import folium
import webbrowser
import os

# Define geographic area, lng, lat, scope/ coverage
geolocator = Nominatim(user_agent="my_user_agent")
geographicArea = "London, UK"
geographicAreaSize = sizeOfCity(geographicArea) # in miles
geographicAreaRadius = round(math.sqrt(geographicAreaSize / math.pi), 2)
```

```
# Get latitude and longitude values
areaLatCords = geolocator.geocode(geographicArea).latitude
areaLngCords = geolocator.geocode(geographicArea).longitude
geographicAreaMap = folium.Map(location=[areaLatCords, areaLngCords],
    zoom_start=10)

# Plot area on a map with a green circle highlighting the area as in
    figure 4.2 below
folium.Circle(location=[areaLatCords, areaLngCords],
    popup=geographicArea, fill_color='green',
    radius=geographicAreaRadius, weight=2,
    color="green").add_to(geographicAreaMap)

# Save map and visualize html file in a browser
geographicAreaMap.save('Resources/geographicAreaMap.html')
webbrowser.open(os.getcwd()+'/Resources/geographicAreaMap.html')
```

The above was validated using 382 cities' dataset [91] from the Office of National Statistics. Having identified a geographic location of interest, the next step is to setup the radio access network, with base stations covering the entire geographic area.

Figure 4.10: With London (United Kingdom) being our geographic area g_x, we consider a full base station around coverage of 19.10 *miles* radius from central London, marked by the green circle.

4.3.2 Base stations / Nodes

As per Fig. 4.11 below, our working mobile telecommunication network consists of 4 primary domains, which the simulation base layer enables:

- **Multi-access edge devices** - these include mobile and IoT devices being operated by their various users.
- **Radio Access Network (RAN)** - our RAN is cloud-based and therefore includes base stations with pooled BBUs, separated from their respective RRHs. The radio access network uses radio frequencies to facilitate wireless connectivity from the RRHs antennas to the multi-access edge devices.
- **Transport network** - the fronthaul and backhaul transport networks provide connectivity between the radio access network and the core network.
- **Core network** - the core network provides connectivity to the internet to allow the users with the RAN to access the cloud-native applications, hosted outside the radio access network.

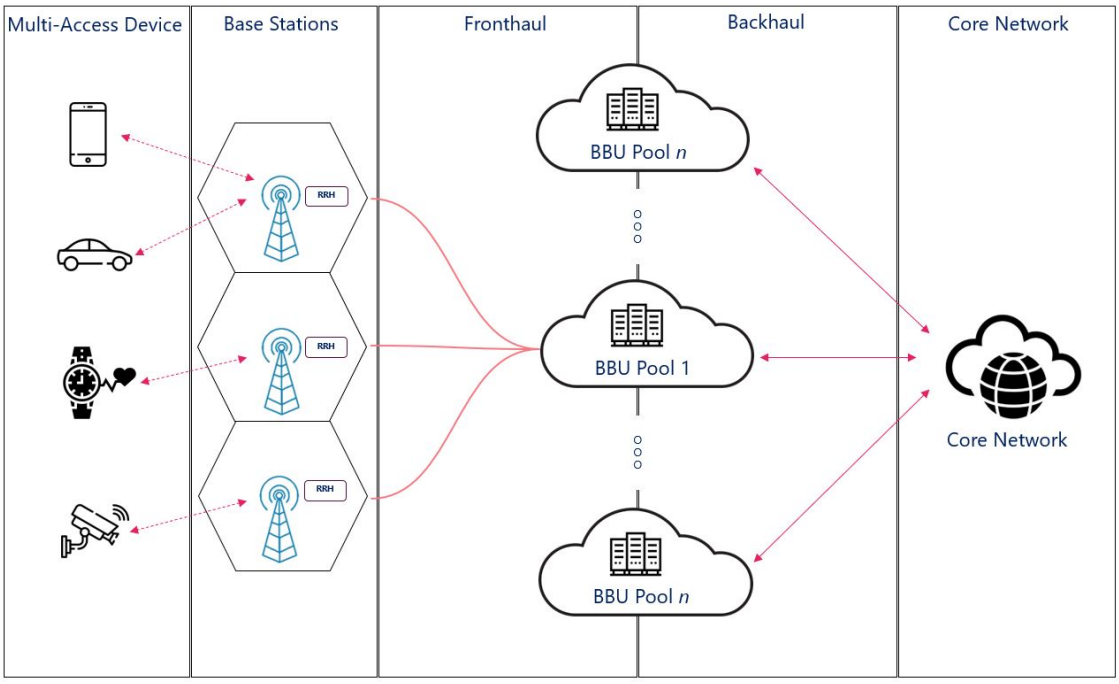

Figure 4.11: MECRAN's working telecommunication network.

Before going ahead to setup the radio access network in the simulation base layer, it is worth revisiting its 2 main components - the base station (i.e. baseband units (BBUs) and the antennas / remote radio heads (RRHs)) and the radio. In the C-RAN architecture the BBUs are separate from the RRHs. The RRHs remain at the cell site, while the BBUs are aggregated into a centralized pool / office / hotel, with a centralized management that processes calculations digitally in a base station. The radio converts digital information into signals that can be transmitted wirelessly to enable communication. In order for users to be able interact with the telecommunication network and access their cloud-native applications, there needs to be enough base station / cell coverage over the geographic area of interest.

Having determined the geographic location and the size of area (as indicated by the green circle in Fig. 4.10), the next step is setting up enough base stations to cover the location. Base stations coverage depth is indicated by the hexagon around the base station, as shown in Fig. 4.11. Therefore the next paragraphs will demonstrate how to approximate the number of hexagons of x length side, that can fit into a circle.

The key inputs here are - the radius of the circle and the length of the side (s) of the hexagon:

$$h_s = (x_s, y_s) = d(cos\ s\pi/3, sin\ s\pi/3) \quad (s \in \{0, ..., 5\}) \tag{4.1}$$

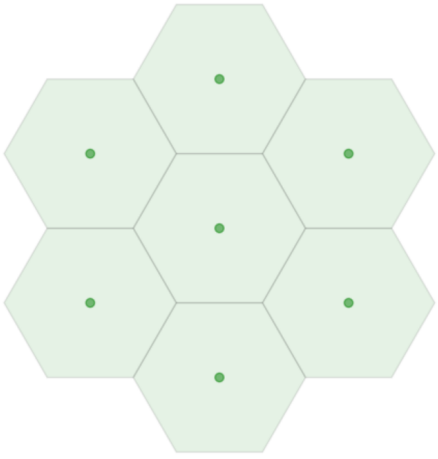

Figure 4.12: Base station coverage site hexagon plotted in Python 3 - a hexagon has 6 sides and each sides leads to another hexagon.

Where s is the number of sides of the hexagon and d is the incircle diameter of a hexagon cell. The deeper green points in the middle of each hexagon in Fig. 4.12 shows the centre of the hexagon. Starting from the first hexagon at the centre, whose origin is (0,0) each hexagon cell can be reached by walking along one of the six directions, as per below:

$$h = h_2 + 2h_5 - h_1 + 7h_2 + 2h_3 + ... \tag{4.2}$$

Further down the line, the statement above declines and drops off to integer multiples of two directions, based on the fact that h_3 = - h_0+... and h_2 = h_1 - h_0. We can therefore use (c_0,c_1) tuple as the centre coordinates of each hexagon and summarise h as:

$$h = c_0 h_0 + c_1 h_1 \tag{4.3}$$

We then conclude that the vertices of the starting hexagon in the centre of Fig. 4.12 is:

$$v_s = D(cos(2s+1)\pi/6, sin(2s+1)\pi/6) \quad (s \in \{0, ..., 5\}) \tag{4.4}$$

where D is the diameter of our geographic area circle as shown in Fig. 4.10 and translates as below, in relation to the d in the earlier hexagonal definition above:

$$D = \frac{2}{\sqrt{3}} d \tag{4.5}$$

All the base station coverage area hexagon vertices within the geographic area fulfills the below condition:

$$\|h + v_s\| \leq R \tag{4.6}$$

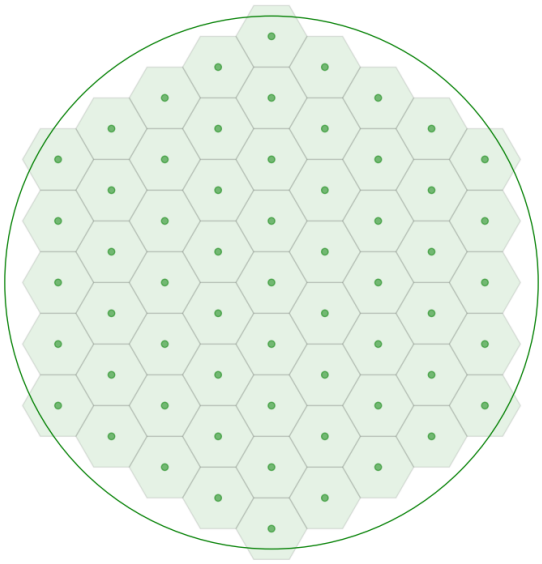

Figure 4.13: Base station nodes approximation and fitting within the geographic area of interest (demarcated by the green circle).

Given the size of the geographic area, we are able to simulate base stations to cover the entire geographic area as in Fig. 4.13. That being said, base station infrastructure owners may decide not to fully cover an entire area for many reasons such as small population to base station ratio, infrastructure costs etc. Whilst our setup is able to cover an entire area, it has the capability of eliminating coverage for sub-areas as per the requirements of the base station infrastructure owner. This also caters for areas that may have irregular or non-symmetrical shapes.

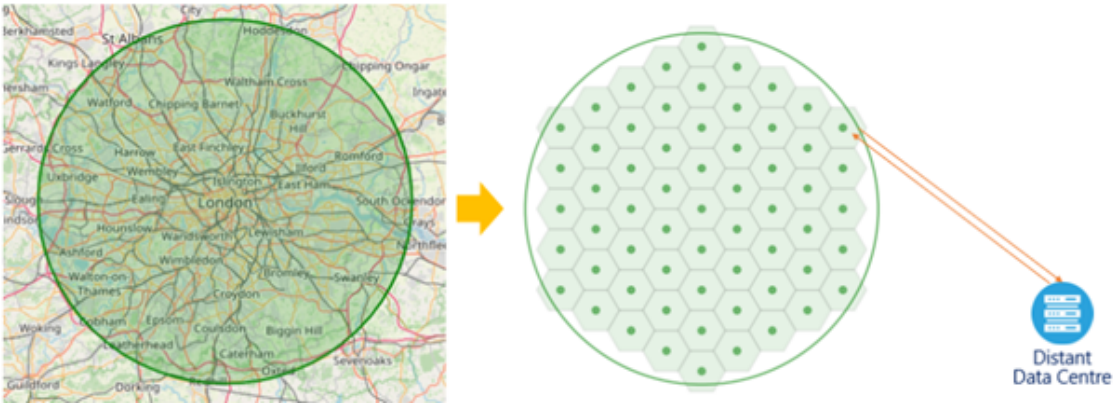

Figure 4.14: A model of the City of London's base station coverage area.

Having simulated cell coverage for a given geographic area (as shown in Fig. 4.14), the next step is to establish links / edges between. Not all base stations will have a direct connection to another base station and therefore we need to model this idea carefully in order to facilitate effective node to node communication.

4.3.3 Connection Types / Links

When a user tries to access an application, the user's device will be constantly communicating with the server / base station where the application is being hosted. As our task scheduler strives to dynamically host an application close to a moving user, it is important to understand the link or edges or relationships that exist between a network of base stations, to enable data routing between the user device and the host server.

In order to link base stations or establish a base station network, we first assign an identity to every base station in order to easily identify which base stations are physically closer or which base stations are neighbours. To identify neighbouring nodes, we introduce the z position coordinate into the hexagons. The z coordinate is an important data point that makes it possible to iterate over the 6 sides of the hexagon. We then take the 3D coordinates (as in Fig. 4.15) of a node and use the algorithm below to find all neighbours:

- **2 left-facing diagonal neighbours:** $(x,\, y-1,\, z+1)$ & $(x,\, y+1,\, z-1)$.
- **2 vertical neighbours:** $(x-1,\, y,\, z+1)$ & $(x+1,\, y,\, z-1)$.
- **2 right-facing diagonal neighbours:** $(x+1,\, y-1,\, z)$ & $(x-1,\, y+1,\, z)$.

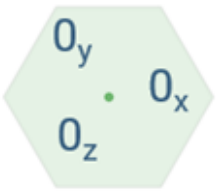

Figure 4.15: Identifying base station neighbours by matching their x, y, z coordinates on the grid.

Continuing from Fig. 4.13, the below are some extracts of the source code attached to this dissertation, showing an implementation of the above algorithm in identifying physical neighbours of all hexagons / base stations:

```
...

# Get / find node that matches the 3D coordinates
def find_node_of_these_coordinates(coord, dict_):
    for key, values in dict_.items():
        if values == coord:
            return key


# find all neighbours of base stations and reorganise them in
    hex_dict_reorganized dict
def find_hex_neighbours():
    global hex_dict_reorganized

    # add only coordinates; no colour
    for key, values in hex_dict.items():
        hex_dict_reorganized[key] = values[0]
        for value in values:
    print(hex_dict_reorganized)

    # create node to neighbours mapping
    for key, values in hex_dict_reorganized.items():
        neighbour_diagonal_1 = [values[0], values[1] - 1, values[2] + 1]
        neighbour_diagonal_2 = [values[0], (values[1]) + 1, (values[2])-1]
        neighbour_vertical_1 = [(values[0]) - 1, values[1], (values[2])+1]
        neighbour_vertical_2 = [(values[0]) + 1, values[1], (values[2])-1]
        neighbour_horizontal_1 = [(values[0])+1, (values[1])-1, values[2]]
        neighbour_horizontal_2 = [(values[0])-1, (values[1])+1, values[2]]

        hex_dict_reorganized[key] = list(filter(None,
        [
            find_node_of_these_coordinates(neighbour_diagonal_1,
                hex_dict_reorganized),
            find_node_of_these_coordinates(neighbour_diagonal_2,
                hex_dict_reorganized),
            find_node_of_these_coordinates(neighbour_vertical_1,
                hex_dict_reorganized),
            find_node_of_these_coordinates(neighbour_vertical_2,
                hex_dict_reorganized),
            find_node_of_these_coordinates(neighbour_horizontal_1,
                hex_dict_reorganized),
            find_node_of_these_coordinates(neighbour_horizontal_2,
                hex_dict_reorganized)
        ]))

    """ NEIGHBOUR RULE
    By the same logic we find the neighbours for tile x,y,z:
    By keeping x the same, we get two diagonal neighbours, x,y-1,z+1 and
```

```
    x,y+1,z-1.
By keeping y the same, we get two vertical neighbours, x-1,y,z+1 and
    x+1,y,z-1.
By keeping z the same, we get the remaining two diagonal neighbours,
x+1,y-1,z and x-1,y+1,z.
"""

print("Base stations and their neighbours:")
for a, b in hex_dict_reorganized.items():
    print(a, b)

return hex_dict_reorganized
```

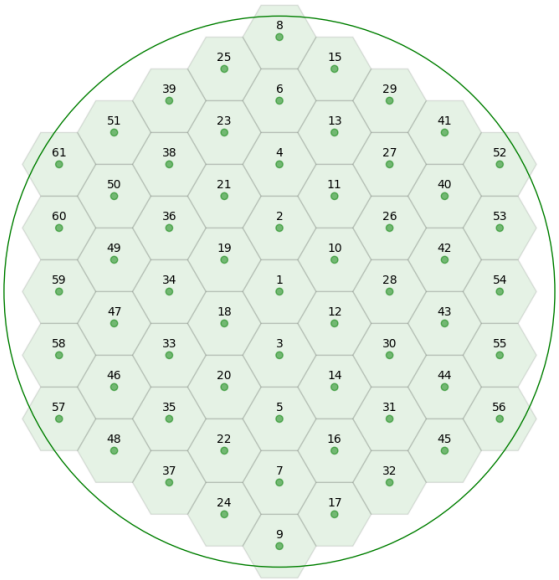

Figure 4.16: Successfully allocated identification numbers to all simulated base stations within the coverage area.

The base stations in the radio access network are linked by connection types (such as copper cables, fibre optic, digital subscriber line (DSL), satellite) with different bandwidth capacities and lengths. We define our link class based on the various attributes the distinguishes the connection types / links from each other:

```
...

class Link:
    def __init__(self, name, node_names, channel_type, link_speed):
        self.name = name
        self.node_names = node_names
        self.link_length = random.randint(10, 50) # in miles
```

```
        self.link_channel_type = channel_type
        if channel_type == "Fibre Optic":
            self.link_bandwidth = random.randint(100, 1000) # fibre
                optics have bigger bandwidths
        else:
            self.link_bandwidth = random.randint(50, 100) # others
                smaller bandwidth
        self.link_speed = link_speed

    @staticmethod
    # {'Fibre Optic': 70, 'Copper Cable': 30}
    def get_channeltypes_list(dict_of_channeltypes, number_of_links):
        channel_types = []
        for k, v in dict_of_channeltypes.items():
            # channel_type.append([k] * int(round((v / 100) *
                links_count, 0)))
            for i in range(int(round((v / 100) * number_of_links, 0))):
                channel_types.append(k)
            # print(k, v)
        return channel_types
```

Designers of the radio access network can therefore customise these connection types attributes based on the specific project / use case at hand. To summarise, by running the neighbour identification algorithm above, we are able to identify all the neighbours of all nodes and assign identities to them as in Fig. 4.16 above. Having established a set of base stations and a set of connection types, we can now link the base stations to build data network routes to enable communication within our telecommunication network.

4.3.4 Cloud-Native Applications

Containerization is introduced in MECRAN to facilitate the efficient migration of the cloud-native applications between nodes to reduce round-trip latency. The aim is to effectively keep latency at 10 ms or below, as shown in Fig. 2.1. The 10 ms threshold is key because the 3 main use cases / scenarios (Enhanced Mobile Broadband; Massive Machine-type Communications; and Ultra-reliable and Low Latency Communications) upon which 5G has been developed, require latency of 10 ms or below for optimal performance [5]. These scenarios informing the 10 ms threshold were defined by the International Telecommunication Union Radiocommunication (ITU-R) in their recommendation ITU-R M.2083 for 5G [3].

In terms of applications in scope for this work, we consider only applications that are cloud-native in nature (i.e. applications that are developed, deployed, and maintained on cloud infrastructure), such as the applications that constitute the aforementioned scenarios. Sub-section 2.3.2 gives further context on the scenarios and Fig. 2.5 also highlights many examples of these cloud-native applications. Most importantly, cloud-native applications in scope are characterised by a need for low end-to-end latency because physical limitations (such as distance and the speed of light) can severely interfere with how these applications perform or respond, when they are fully reliant on the remote cloud alone. These applications also require high bandwidth due to the fact that they turnover large volumes of upstream data; examples include high definition video sharing applications or applications that manage large numbers of IoT devices. We consider applications that operate in public, private or hybrid cloud environments. Further scope or context around resources the applications interact with such as containers and database are described below.

Our work is restricted to network devices communicating over Transmission Control Protocol/Internet Protocol (TCP/IP) only. This is to ensure that our work is compatible and interoperable with global standards on multi-access edge computing and fog computing as defined by European Telecommunications Standards Institute's Multi-Access Edge Computing Industry Specification Group (ETSI MEC ISG) and

OpenFog Consortium respectively. Our work is also restricted to single stateless or stateful container applications, but MECRAN can be easily extended in future work to also cater for multiple containers by running Docker Compose in a YAML file to configure and run all the required containers at once [92]. Docker Compose is a functionality for defining and running multi-container Docker applications [92]. Whilst this can technically be done in any other container environment, Docker is used in MECRAN because it specializes and optimizes the creation and starting of one or multiple containers [93].

For stateful containers, data generated is persisted in a storage facility called *Docker Volumes* to avoid data loss when the container is terminated. MECRAN persists the data by creating and mounting these Docker Volumes as they enable efficient back-up and migration of data. Volumes are stored in a directory on the file system of the base station. Volume drivers are used to safely encrypt the content of the volumes as a way of ensuring security in MECRAN when sharing volumes with other containers or with other nodes. The underlying Ceph Storage system (as discussed in sub-section 4.2.2) allows node clusters to have shared access to stateful container data.

Ceph Storage as a software defined storage system proactively replicates and distributes docker volumes as file storage to each MECRAN cluster within the radio access network for peak performance. This is key as it reduces the amount of data (and essentially migration time) to be transferred during container migration. This means that when the user moves around within the radio access network, the dynamically selected base station is able to access the stateful container's persisted data awaiting the migrant container. The replication and distribution of data around the MECRAN clusters using CRUSH algorithm [65] also prevents data loss when there is a single point of failure and avoids performance bottlenecks as well as avoids physical limits to scalability. To prevent data explosion of docker volumes, the third scenario from MECRAN's Scenario Layer (2B of Fig. 3.3) as part of garbage collection and freeing up resources, removes redundant volumes at the end of the container's life cycle.

4.3.5 Live Container Migration

The concept of live container migration has been tested as part of this research work for both remote moves (i.e. data transfer between 2 remote nodes) and local moves. For the remote moves, a node was based at the Texas Advanced Computing Center of the University of Texas at Austin (CHI@TACC) and the other node was based at the University of Chicago (CHI@UC), via the Chameleon Testbed [84]. The Chameleon Testbed houses a large-scale homogenous partition of approximately 15,000 cores, 5PB of total disk space across both CHI@TACC and CHI@UC sites. Our test nodes are bare metal on 64-bit Linux operating system with 32 cores, 64 GB RAM, a 500 GB SSD, a 1 TB HDD, reconfigurable Corsa switches, support for Infiniband and connected by 100 Gbps network. As indicated earlier, live container migration has also been tested within a cluster at the CHI@TACC to mimic local moves. In the experiment conducted and discussed below, the results show that data was moved 47% times faster locally than remotely as show in Table 4.2.

Container - Size	Local Moves (ms)	Remote Moves (ms)	How fast was moving the same data size locally vs remotely?
A - <1GB	1	1	0%
B - 2GB	3	5	40%
C - 3GB	5	9	44%
D - 4GB	7	16	56%
E - 5GB	8	21	62%
F - 6GB	9	19	53%
G - 7GB	9	24	63%
H - 8GB	12	27	56%
I - 9GB	16	35	54%
J - 10GB	18	26	31%
K - 20GB	24	58	59%
L - 30GB	46	73	37%
M - 50GB	74	138	46%
N - 100GB	103	290	64%

Table 4.2: The table summarizes and compares the time (in ms) it took for moving the same data locally versus remotely.

Sub-section 2.5.6 and Fig. 2.8 both highlight the fact that the cost of file (containing container state data) movement is the dominant part of the migration. The migration time is primarily dependent on this in-memory state transfer time. In the experiment above where the same in-memory state data is moved both locally and remotely, we can see from Table 4.2 that the cost of file movement locally is much smaller than when the same file is moved remotely. In fact in one example, 100 GB data size file was moved 64% times faster locally than remotely. In Table 4.2 it is also observed that the same files were moved 47% times faster locally (than remotely) on average. There are many variables (from network, hardware resources to container attributes) that can affect migration times. In MECRAN the container state data is migrated live, however the persisted data (volumes as described in sub-section 4.3.4) is proactively shared among cluster nodes separately from the container live migration, optimizing the migration time.

A Docker container image is a lightweight executable software package and also a lightweight alternative to virtual machines in many scenarios; therefore it lightweight nature is one of the key advantages of using a container [94] in MECRAN. We reviewed the average image size of 1 million randomly selected docker images and their corresponding layers from the *Docker Hub* [95] and we have also reviewed similar docker image analytics work done by other researchers [94] to conclude that over 90% of the images have compressed sizes less than 0.50 GB and uncompressed sizes below 1.5 GB. Docker Hub is a centralized image registry, holding approximately 4 million public image repositories with about 20 million layers [94].

14 containers with total metadata and memory content size from 0.5 GB to 100 GB were migrated between 2 remote nodes and also locally between nodes within the same cluster. Each container was migrated 100 times locally and 100 times remotely. The results observed below show the average total migration time per container for both local migration (Fig. 4.17) and remote migration (Fig. 4.18).

The aim is to migrate live containers with minimal or no impact on the quality of service to the mobile user. As discussed in the earlier chapters the cloud-native applications in scope for this research, have strict requirement for latency below

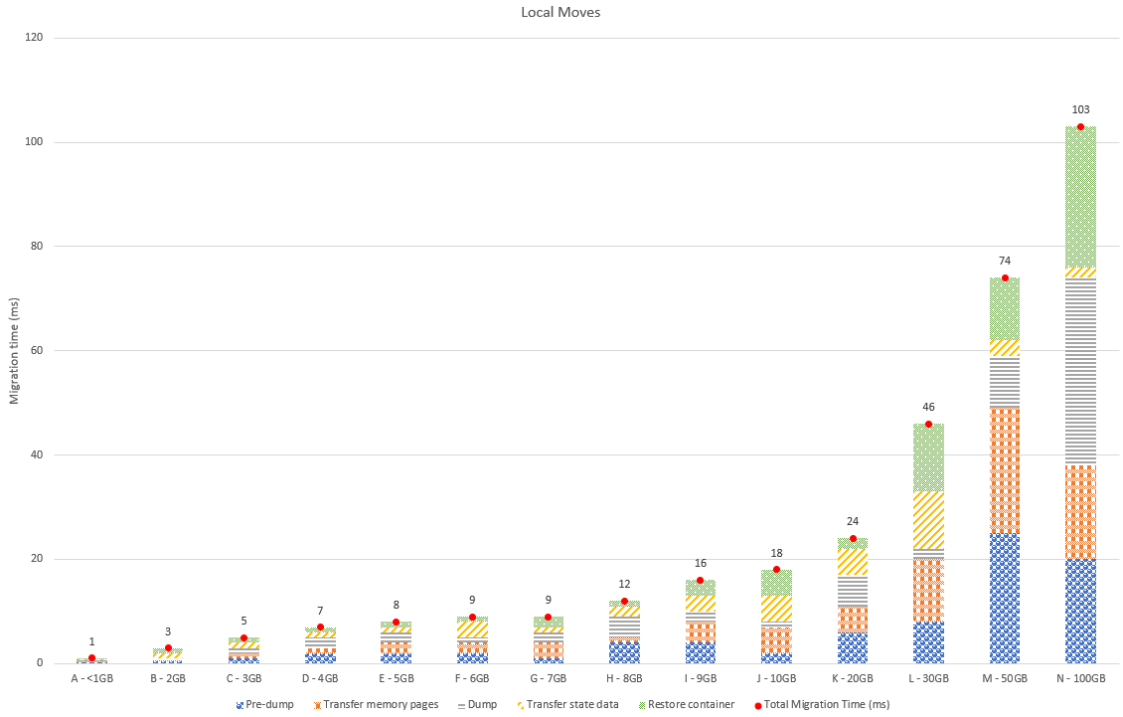

Figure 4.17: A Visualization of migration time vs size of metadata in MECRAN for local moves.

10 ms and this continues to be a benchmark for the container migration. In (Fig. 4.17) we observe that containers with migration sizes of approximately 7 GB and below were able to meet the less than 10 ms migration target. This validates the fact that more than 95% of images in Docker Hub can be migrated locally without impacting the user's experience.

In (Fig. 4.18) we observe that containers with migration sizes of approximately 3.5 GB and below were able to meet the less than 10 ms migration target. However as described in MECRAN's approach in sub-section 3.2, MECRAN performs the remote migration only once and at the beginning of the process when the user initially attempts to access the cloud-native application. This buffering experience or delay is therefore experienced before the user starts using the application and not during the use of the application; this is an important difference when it comes to the user experience and quality of service with regards to remote migration.

The above is to demonstrate that it is possible to maintain client connection in MECRAN with CRIU (based on TCP/IP) when cloud-native applications are

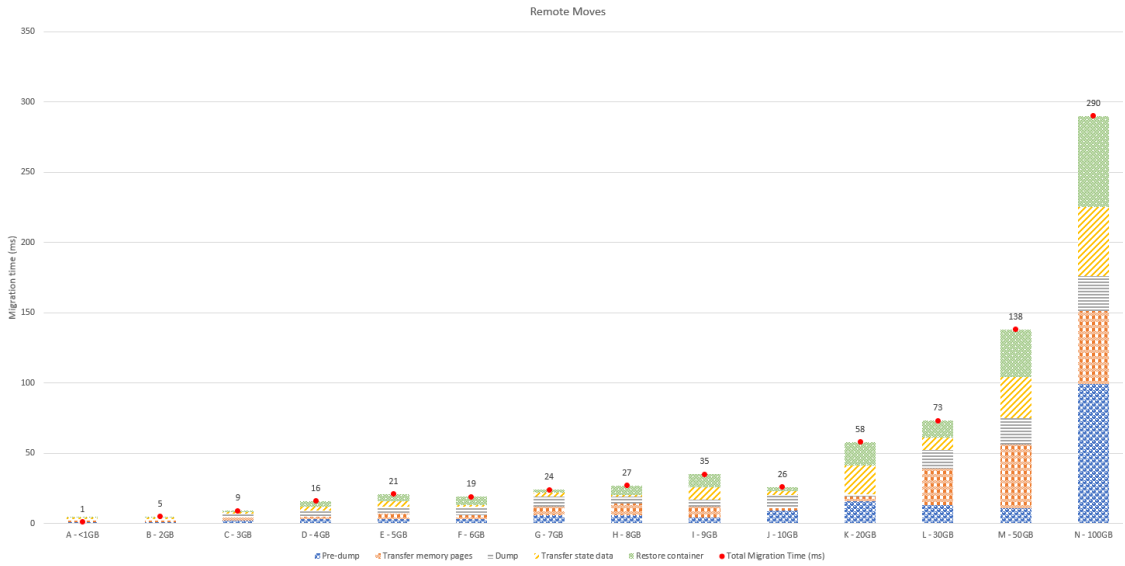

Figure 4.18: A Visualization of migration time vs size of metadata in MECRAN for remote moves.

migrated live between nodes using containers. It is also the aim to demonstrate that container metadata and memory content data movement is a dominant part of migration and to also highlight the relative smaller cost in local moves versus remote moves.

4.3.6 The data network, based on graph theory

With the setup in subsection 4.3.3, we are able to even simplify the problem further using *graph theory* to model the data network communication within the radio access network. We translate the base station data network map into graph nodes and edges and also based on a hybrid network topology.

We adopt a hybrid network topology for the structure of our inter-base station relationship map. The main advantage the hybrid network topology has over Tree, Mesh, Star, Bus and Ring topologies is scalability [96]. The setup in this research enables telecommunication network designers to easily scale base stations resources to meet the user demand while planning the radio access network. This means when base station infrastructure owners such mobile network operators adjust the network size, a hybrid topology is able to accommodate this change quickly. This is important for this piece of work as the requirements for base station roll-out in each geographic area vary and therefore we want a system that is highly reliable, flexible and available to meet these ever-changing demands.

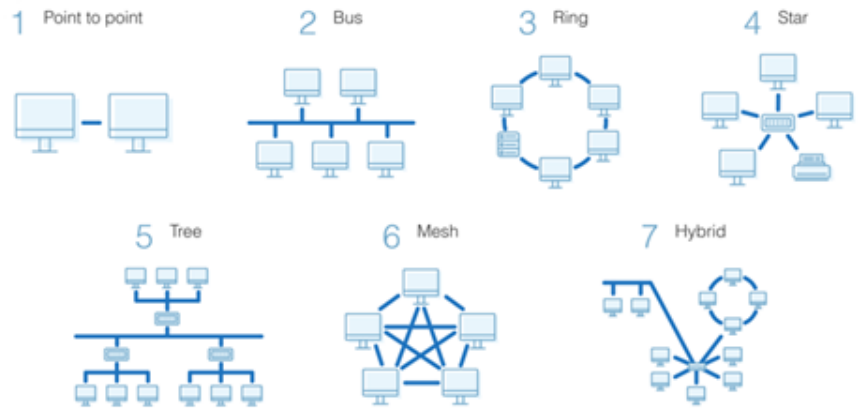

Figure 4.19: Network topology types [97]. Our work is based primarily on the Hybrid topology.

We designed the inter-base station relationships by creating a dictionary data structure (called hex_dict_reorganized) with the identifying numbers of the base stations being the keys (as demonstrated in subsection 4.3.3) and an array of the other base stations they are connected to being the values:

$$b_x : [b_{x1}, ..., b_{xn}]$$

We are able to generate a graph of these relationships by passing the dictionary to NetworkX [98]. NetworkX is a Python package for the creating and manipulating complex graphs as seen in our data network. Base station nodes definition are presented below.

```
...

hex_dict_reorganized = {} # inter-base station relationships dictionary

# where 'a' is our current base station and 'b' and array of linked
    neighbours

print("Base stations and their neighbours:")
for a, b in hex_dict_reorganized.items():
    print(a, b)

# Base station nodes attributes definition
class Node:

    def __init__(self, ip_add, n_type, _data):
        self.node_ip_add = ip_add
        self.node_type = n_type
        self.node_data = _data
        self.node_port = random.randint(0, 100)
        self.node_transmission_rate = random.randint(50, 100) # bits per
            sec
        self.node_buffer = queue.Queue(random.randint(2, 5))
        self.node_packet_processing_speed = random.randint(10, 20) #
            assume in ms
        self.node_data_loss_count = 0

    @staticmethod
    def tcpip_model(rt_position, routing_table, frame_,
        data_transfer_matrix): ...

    # Processing packets in buffer
    @staticmethod
    def process_buffer_items(Node_): ...

    # Peer to peer messaging
    @staticmethod
    def p2p_messaging(data_senders, layout, ax): ...
```

```
    # Export metadata about node to node data transfer to pickle for
        visualization
    @staticmethod
    def export_node_data_transfer_to_pickle(sender_, receiver_,
        dic_for_pickle): ...

    # Export metadata about node capacity to pickle for visualization
    @staticmethod
    def export_node_capacity_data_to_pickle(all_base_stations): ...
```

Our base stations (and indeed any other nodes such as routers) at a minimum have an ip address, a type, port number, buffer, data transmission rate, packet transmission speed, holds some data and tracks its data losses. It also has a number of static functions such as communicating using the TCP/IP model.

Implementing and initializing our base station data network based on the above definitions:

```
...

import sys
import networkx as nx #NetworkX module
import random
import queue
from random import choice
import matplotlib.pyplot as plt

# Main Program
if __name__ == '__main__':

Mecran_Graph = nx.Graph()
Node_list = [i + 1 for i in range(len(hex_dict_reorganized))] # number
    of base stations
Links = []
Edge_list = []
Channeltypes_source_list = {'Fibre Optic': 70, 'Copper Cable': 30} #
    Channel types and their % assignment
Channeltypes_list = []
Node_colour_list = []
Data_loss_tracker = []
All_going_well = True

# Generate edges dynamically or hard code edge list based on network
    design
for i in range(len(Node_list) * 2):
    choice1 = random.choice(Node_list)
```

```
    choice2 = random.choice(Node_list)

    if choice1 != choice2: # generate connections between base stations
        new_conn = tuple(sorted((choice1, choice2)))
        if new_conn in Edge_list:
            continue
        Edge_list.append(new_conn)

    Channeltypes_list =
        Link.get_channeltypes_list(Channeltypes_source_list,
        len(Edge_list)) # Get list of channel types

    for i in range(len(Edge_list)):
        Links.append(Link(
            str(Edge_list[i][0]) + "-" + str(Edge_list[i][1]),
            Edge_list[i], # node names
            Channeltypes_list[i] # channel type
        ))

# Graph
    Mecran_Graph.add_nodes_from(Node_list)
    Mecran_Graph.add_edges_from(Edge_list)
    pos = nx.spring_layout(Mecran_Graph)
    source = choice(list(Mecran_Graph.nodes()))
    destination = choice(list(Mecran_Graph.nodes()))
    shortestPath = nx.shortest_path(Mecran_Graph, source, destination)


    # Create dictionary
    Mecran_Nodes = []

    for i in range(1, len(Node_list) + 1):
        Mecran_Nodes.append(Node(i, "host" if i == source or i ==
            destination else "router", Data(random.randint(10, 100),
            Data.data_colour_generator())))
    # Plots
    fig, ax = plt.subplots(figsize=(6, 4))
    ani = animation.FuncAnimation(fig, animate3, frames=10,
        interval=1000, fargs=(Node_colour_list, pos, Mecran_Graph, ax))
    plt.show()
```

We have been able to build a base station data network which is easily customizable based on use case and user needs. The next subsection enables communication within this network based on Transmission Control Protocol/Internet Protocol (TCP/IP).

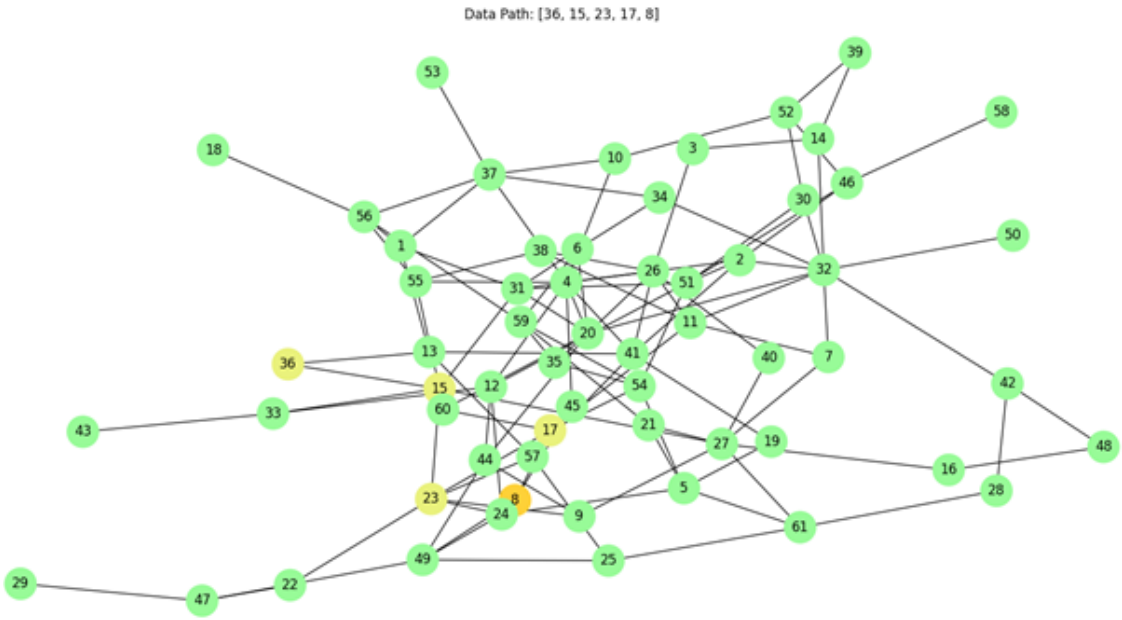

Figure 4.20: Base stations and their edge relationships visualization based on the implementation above. The image also shows base station 36 sending a data packet to base station 8 via 15, 23 and 17.

4.3.7 Communicating within the data network via packet switching / TCPIP

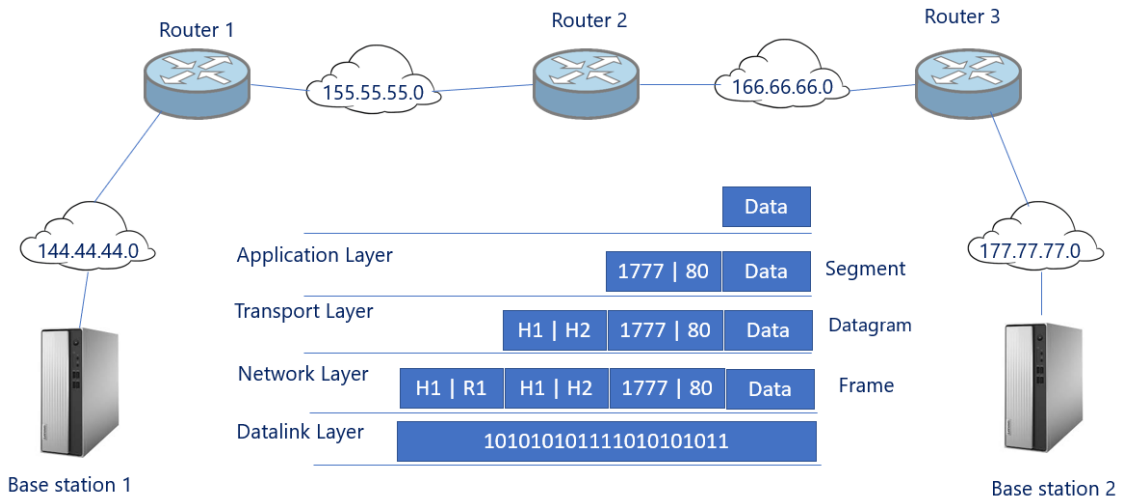

Figure 4.21: Data packets being sent from one base station to the other, across a number of intermediary network resources [99].

Let us review what happens when a single packet is being transported from host 1 (Base station 1) to host 2 (Base station 2). Host 1 and Host 2 could be in the same country / city or in different locations, but the process is still the same. The above setup illustrates the basic process that allows data communication across the internet. Suppose host 1 is a client device or base station, TCP/IP allows host 1 and host 2 to communicate across the internet whilst it appears to the application process on both hosts that they are directly connected to each other.

A data packet being sent from host 1 to host 2, it will travel via router 1, 2 & 3 (or some kind of intermediary hop point which in our use case, are all mainly base stations) and then finally to host 2. This illustrates one path a packet might travel across the internet. In reality there are many paths as seen in Fig. 4.21 above for data packets to travel across the network. This is what is known as packet switching as compared to circuit switching where data packets travels through only one path such as in the days of the Plain Old Telephone Systems (POTS). In packet switching, MAC address (usually of 48 bits) change as the packets move across the internet but the IP addresses (usually of 32 bits) stay the same.

The way TCP/IP works is that data (which happens to be html information) from the *application layer* of host 1 is sent down to the *transport layer*. At the transport layer, a header (containing a source port of host 1 and destination port of host 2) is added to the data. The transport header together with the data is called the *segment*; that entire segment is sent down to the *network layer*. At the network layer, a header is added to the segment. The network layer header contains the source IP of host 1 and destination IP of host 2. The network header together with the segment is called the *datagram*. This entire datagram is then sent down to the *datalink layer*. At this layer, a header is added to the datagram. This header contains the hardware source address of host 1 and hardware destination address of router 1 (not host 2). The data link layer header together with the datagram itself is called a *frame*. The frame is sent down to the datalink layer and is converted to 1s and 0s and sent to router 1, over the connecting link between these two nodes.

The frame is received by router 1 and checks if the hardware address matches its own, otherwise it ignores the frame. Assuming the frame was addressed to router 1, it removes the frame's header as this is no longer needed, and the datagram is passed up to the network layer. At the network layer, the router looks at the *routing table* to find out which node is next in line for the data packet to reach host 2 successfully. Assuming router 2 is next in line, router 1 sends the datagram down to its datalink layer and a new frame is added containing the same source hardware address of router 1 but now the destination hardware address becomes that of router 2. The frame is then sent to router 2 and this process continues until the data packet reaches host 2.

Assuming the frame has finally been sent out to host 2, host 2 looks at the frame to confirm the frame is addressed to it in order for host 2 to not ignore the frame. Host 2 then removes the frame's header and passes the datagram up to the network layer. At that layer, it sees that the destination IP matches its own. It therefore removes the network header as it is no longer needed, and the segment is further sent up to the transport layer. At the transport layer, host 2 looks at the destination port to know which application process is supposed to receive the data.

It therefore removes the header and sends the data the http application process in the application layer.

Implementation of the aforementioned TCP/IP model in MECRAN's simulation base layer. Where tcpip_model is a static method of our network node:

```
...

# Moving data via the 5 layers of the network architecture:
    # 5 - Application, 4 - Transport, 3 - Network, 2 - Datalink, 1 -
        Physical Medium
    @staticmethod
    def tcpip_model(rt_position, routing_table, frame_,
        data_transfer_matrix):
        routing_table[0].node_data.data_value = random.randint(50, 100)
        data = routing_table[0].node_data.data_value
        segment = []
        datagram = []
        frame = []
        frame_size = 0

        print("-- Processing Packet Transmission: SENDING...")

        # Create Data, Segment, Datagram and Frame
        if rt_position == 0:
            # Application Layer -> Transport Layer # Initialise segment
                by adding a header [source port, destination port] to the
                data
            if data:
                segment.append([routing_table[0].node_port,
                    routing_table[-1].node_port])
                segment.append(data)
                print("Data being sent: ", data)
                print("Application Layer -> Transport Layer: ")

            # Transport Layer -> Network Layer # Initialise datagram by
                adding a header [source ip, destination ip] to the segment
            if segment:
                datagram.append([routing_table[0].node_ip_add,
                    routing_table[-1].node_ip_add])
                datagram.append(segment[0])
                datagram.append(segment[1])
                print("Segment: ", segment)
                print("Transport Layer -> Network Layer: ", )

            # Network Layer -> Datalink Layer # Initialise frame by
```

```
            adding a header [ sending source hardware add,
            destination source hardware add] to the datagram
        if datagram:
            frame.append([routing_table[rt_position].node_ip_add,
                routing_table[rt_position + 1].node_ip_add])
            frame.append(datagram[0])
            frame.append(datagram[1])
            frame.append(datagram[2])
            print("Datagram: ", datagram)
            print("Network Layer -> Datalink Layer: ")

            # Get size of frame in BYTES to pass via physical layer
            frame_size = sys.getsizeof(frame)
            print("Frame Sent: ", frame)
            print("Frame Size: " + str(frame_size) + " bytes")
            print("Datalink Layer -> Physical Layer")

        routing_table[rt_position].node_buffer.put_nowait(frame)
    else:
        frame = frame_
        print("Frame being Sent: ", frame)

    # routing_table[rt_position].node_buffer.put_nowait(frame)

    # Packet(Frame) is ready to be sent; needs to be received
        successfully on the other end
    print("-- Processing Packet Transmission: RECEIVING...")

    # Receiving node checks if hardware address of sent frame matches
        its own
    if routing_table[rt_position + 1].node_buffer.full(): # if
        receiving node buffer is full, data loss!
        # track data loss
        routing_table[rt_position + 1].node_data_loss_count += 1
        Data_loss_tracker.append(frame)
        print(" *** Buffer Full *** ")
        print("Buffer Capacity: " + str(routing_table[rt_position +
            1].node_buffer.qsize()) + "/" + str(
            routing_table[rt_position + 1].node_buffer.maxsize))
        return
    else:
        if frame[0][1] == routing_table[rt_position + 1].node_ip_add:
            # upon finding the right next node
            frame_size = sys.getsizeof(frame)
            datagram = frame[1:]
            segment = datagram[1:]
            data = frame[-1]
            if rt_position + 2 < len(routing_table):
```

```
                # if rt_position != 0:
                    routing_table[rt_position].node_buffer.get_nowait()

                print("Yes, this frame's processing hardware address
                    matches mine!")
                print("Physical Layer -> Datalink Layer")
                print("Frame Received: ", frame)

                print("Frame Size: " + str(frame_size) + " bytes")
                print("DataLink Layer -> Network Layer")
                print("Datagram: ", datagram)
                frame.pop(0) # removes frame header
                frame.insert(0, [routing_table[rt_position +
                    1].node_ip_add,
                                 routing_table[rt_position +
                                     2].node_ip_add]) # add new frame
                                     header
                print("- Checks the routing table for the next base
                    station details -")
                print("New Frame: ", frame)
                routing_table[rt_position].node_buffer.get_nowait() #
                    remove frame to sending node's buffer
                routing_table[rt_position +
                    1].node_buffer.put_nowait(frame) # add frame to
                    receiving node's buffer

                print(str(routing_table[rt_position].node_ip_add) +
                      "'s buffer Capacity: " +
                          str(routing_table[rt_position].node_buffer.qsize())
                          + "/" + str(
                    routing_table[rt_position].node_buffer.maxsize))
                print(str(routing_table[rt_position + 1].node_ip_add) +
                      "'s buffer Capacity: " +
                          str(routing_table[rt_position +
                          1].node_buffer.qsize()) + "/" + str(
                    routing_table[rt_position +
                        1].node_buffer.maxsize))
            else:
                routing_table[rt_position +
                    1].node_buffer.put_nowait(frame)
                routing_table[rt_position].node_buffer.get_nowait()

                # print("I am base station: " +
                    str(routing_table[rt_position + 1].node_ip_add))
                print("Yes, this frame's processing hardware address
                    matches mine!")
                print("Physical Layer -> Datalink Layer")
                print("Frame Received: ", frame)
```

```
                print("Frame Size: " + str(frame_size) + " bytes")
                print("DataLink Layer -> Network Layer")
                print("Datagram: ", datagram)
                print("Network Layer -> Transport Layer")
                print("Segment: ", segment)
                print("Transport Layer -> Application Layer")
                print("Data Received: ", data)
                print("Data has reached it's final destination!")
                routing_table[rt_position + 1].node_buffer.get_nowait()

                routing_table[rt_position].node_data.data_value =
                    routing_table[rt_position].node_data.data_value
                    -frame[-1]
                routing_table[rt_position + 1].node_data.data_value =
                    routing_table[rt_position +
                    1].node_data.data_value + frame[-1]
                Node.export_node_data_transfer_to_pickle(
                routing_table[0].node_ip_add,
                routing_table[-1].node_ip_add, data_transfer_matrix)
                Node.export_node_capacity_data_to_pickle(Mecran_Nodes)
                print(str(routing_table[rt_position].node_ip_add) +
                    "'s buffer Capacity: " + str(
                    routing_table[rt_position].node_buffer.qsize()) +
                        "/" + str(
                    routing_table[rt_position].node_buffer.maxsize))
                print(str(routing_table[rt_position + 1].node_ip_add)
                    + "'s buffer Capacity: " + str(
                    routing_table[rt_position +
                        1].node_buffer.qsize()) + "/" + str(
                    routing_table[rt_position + 1].node_buffer.maxsize)
                    )

        return frame
```

The takeaway from this packet switching process (as per figure 4.21) is as follows:

- As the frame moves from one node to the other, only the frame header changes. The IP header is looked at but never changes.

- The transport header and the data is never looked at by the non-destined nodes. The transport layer along with the application layer are called the end-to-end layers as they are only looked at by the end point host. See this behaviour showcased in Fig. 4.22 below.

- The network layer, data link layer and physical layer are called host-to-host layers as these layers are looked at (although not necessarily changed) by every host along the way.

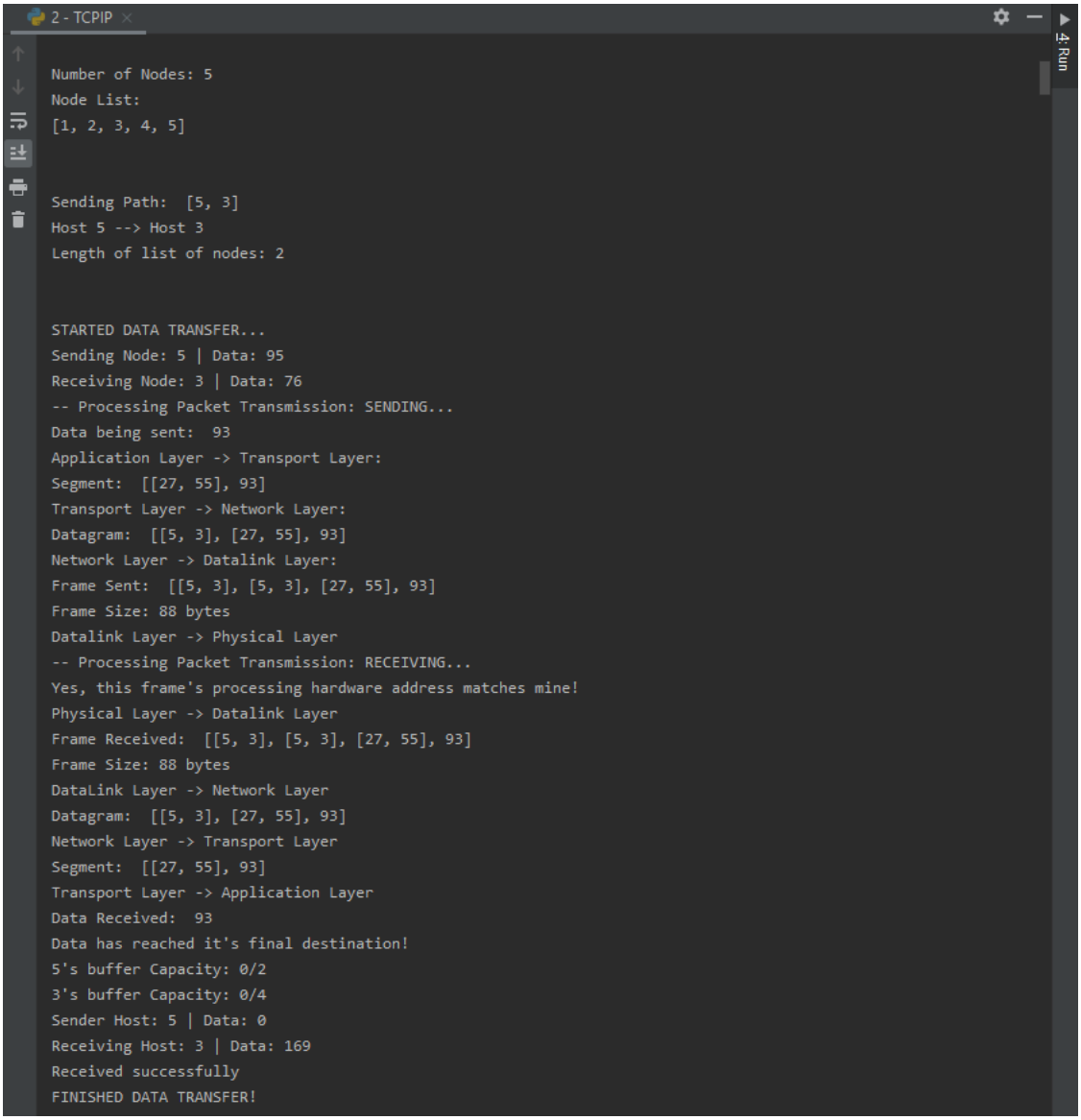

Figure 4.22: An output of the data packets switching process (as implemented above) with base station 5 sending data to base station 3 using the TCPIP protocol.

4.3.8 Latency Measurement, Delay, Loss & Throughput

In MECRAN latency is measured in two ways: mainly as Round Trip Time (RTT) or sometimes as Time to First Byte (TTFB). RTT shows how long (in milliseconds) it takes to send a data packet from an application user's device to a server (or base station) and back again to the application user's device. This could be seen in Fig. 4.23 - for illustrative purposes only, we determine the latency between Base station 1 and Base station 3 by sending a data packet from Base station 1 to Base station 3 and tracking the amount of time it took the packet to go from the former, through the various routers and eventually, to the latter. Similarly with TTFB we measure the amount of time it took for Base station 3 to receive the first byte of data sent by Base station 1. This measured latency is the key parameter MECRAN uses in the scheduling scenarios (in sub-section 3.3.2) to determine if a cloud-native application should be scheduled or not.

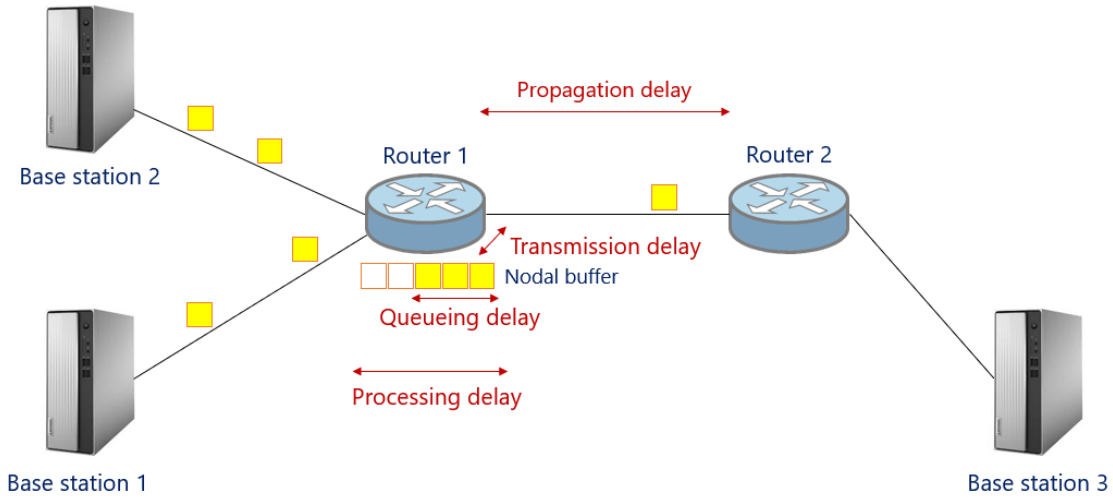

Figure 4.23: The concept of delays and losses in MECRAN during data packets switching. MECRAN however minimizes the occurrence of these bottlenecks effectively through the Data Packets Routing (DPR) routine.

Base stations communicate with each other exchanging data. This subsection highlights the data attributes and some of the delays and losses which happen as a result of the data exchange. Some of these delays and losses can cause bottlenecks and affect the overall latency between the sender and receiver of the data; thus MECRAN minimizes the occurrence of these bottlenecks effectively through the

Data Packets Routing (DPR) routine. See sub-section 3.3.2 for DPR context and chapter 5 for full DPR implementation. Below is a basic implementation of our data object in the simulation base layer:

```
...

# Payload or Data
class Data:
    def __init__(self, _data, _colour):
        self.data_value = _data
        self.data_trans_colour = _colour

    @staticmethod
    def transfer_data(link, list_of_nodes): ...

     @staticmethod
    def p2p_transfer_data(list_of_nodes, rand_frame): ...

    @staticmethod
    def data_colour_generator(): ...
```

Unfortunately, delays and losses do occur in packet switching networks. Consider Fig. 4.22 (a representation of packet switching activity in the MECRAN's simulation base layer) with two nodes sending data packets via an intermediary node (this could be a router or another base station). The intermediary node has a *buffer*, and the node can only process the data packets at a certain speed and only a certain number of packets per second. A buffer is a part of memory used to temporarily hold data packets while they are being sent from one node to another.

Delays

If the arrival rate of the data packets is faster than the node's sending rate, the node queues the arriving data packets whilst sending others until it has capacity to resume. In Fig. 4.23, we can see 2 more empty slots that can hold arriving packets, however we can also see 4 other data packets already on their way to this intermediary node from base stations 1 and 2. There is a possibility of delays happening and some of the sources of these delays include:

- **Packet transmission delay** – there is a delay associated with data packet transmission as indicated in Fig. 4.23. Upon the intermediary node receiving a data packets, it needs to inspect and process the packet (as we saw with the various layers in TCP/IP in the section above) before sending it along. The time it takes the node to put the data packet on on the wire or link is the transmission delay. Transmission delay is therefore:

$$l_p/r_t \tag{4.7}$$

  Where l_p is the length of the data packets (in bits) and r_t is the rate of transmission. The smaller the speed or rate of transmission, the bigger the transmission delay.

- **Packet processing delay** – As mentioned earlier, data packets need to be inspected and processed by the intermediary node. The time it takes to process the data packets (such as determining the next node from the routing table) could cause delays.

- **Packet queuing delay** – After data packets have been processed, they may be waiting in a queue before they are transmitted on to the wire (link or edge) connecting to the next base station. This is what is known as queuing delay and it depends on congestion in the buffer. Queuing delay highlights the traffic intensity to the node. Traffic density to the MECRAN nodes can be calculated as:

$$La/R \tag{4.8}$$

Where L is the length of the data packets (in bits), a is the average data packet arrival rate and R refers to the bandwidth of the wire / link / edge (in bits per second). When the traffic density (La/R) approaches 0, there is very little queuing delay. When (La/R) approaches 1, the queuing delay gets very large. There will be infinite amount delay if (La/R) > 1; meaning the node is constantly getting more data packets than it can process.

- **Packet propagation delay** – After the data packets have been transmitted from the intermediary node on to the wire to the next node, there may be delays in the time it takes to travel across the wire depending on the connection type and its associated distance of travel. Propagation speeds are fast (i.e. speed of light - $3 * 10^8 m/s$) with fibre optics connection type [38].

Delays can add up to a significant amount of waiting time. Total nodal delay therefore:

$$d_{nodal} = d_{transmission} + d_{processing} + d_{queuing} + d_{propagation} \tag{4.9}$$

Losses

With regards to losses, when the buffer is up to its full capacity and unable to accept or store any more data packets, this can lead to data packet losses. In other words if the storage space for data packets waiting to be sent over the wire is full, data packet losses may occur. In MECRAN, the nodes track "lost" data packets and the receiving node may request for these data packets to be resent to them. This is how the sending node tracks lost data packets:

```
...

# if receiving node buffer is full, data loss!
if routing_table[rt_position + 1].node_buffer.full():
    # track data loss
```

```
    routing_table[rt_position + 1].node_data_loss_count += 1
    Data_loss_tracker.append(frame)
    print(" *** Buffer Full *** ")
    # print(list(routing_table[rt_position + 1].node_buffer.queue))
    print("Buffer Capacity: " + str(routing_table[rt_position +
        1].node_buffer.qsize()) + "/" + str(routing_table[rt_position +
        1].node_buffer.maxsize))
    return
```

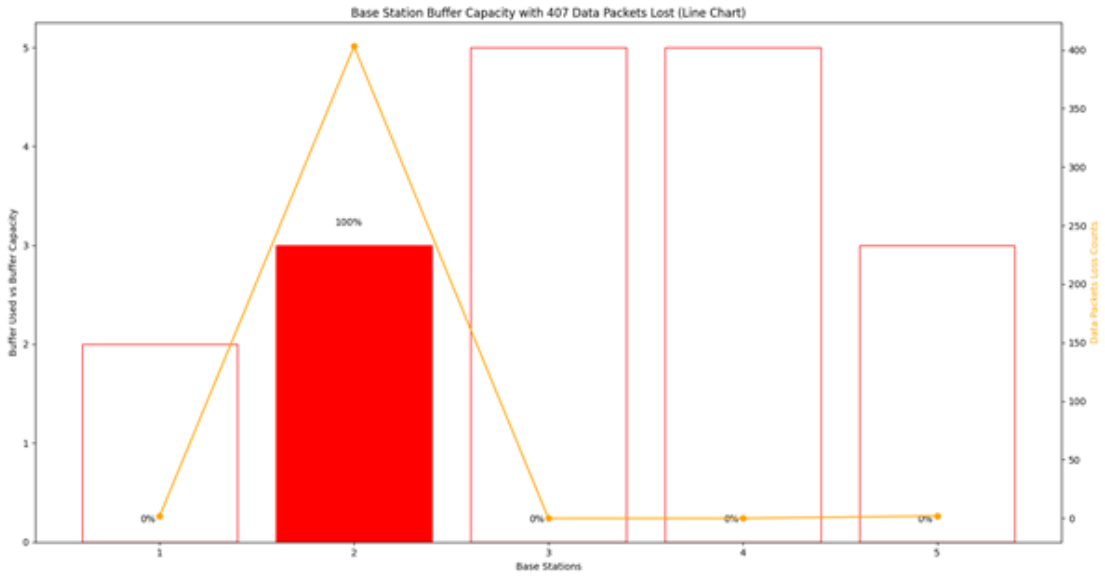

Figure 4.24: With no active scheduling in the data network, node 2 is seen to reach its buffer capacity quickly and losing over 400 data packets.

MECRAN monitors data losses within the radio access network. With no active scheduling in effect, Fig. 4.24 shows an $La/R > 1$ scenario in MECRAN, where node 2 reaches its full buffer capacity. This is because the incoming workload or data packets are constantly higher than the outgoing processed data packets.

Throughput

Throughput is also an important concept in data packet switching and indeed in communicating within any network. Throughput is therefore modelled and factored into the data network of MECRAN's simulation base layer. Throughput is the rate (bits / time unit) at which bits of data is transferred between one node and the other. In Fig. 4.25 below, we observe the links connecting the nodes to all have different capacities. Data being sent from node 2 to node 17 will be capped at a sending rate of 5 bits / second.

Throughput concept is synonymous to pipes that can carry fluid at a rate / time unit. Bigger pipes have the capacity to carry more fluid than smaller pipes. If both pipes are then connected and the bigger pipe flows fluid via a smaller pipe, the flow rate at the end is capped at the size of the smaller pipe. This is what is referred to as a *bottleneck*. In the same vein if the smaller pipe flows fluid into the bigger pipe, the flow rate / time unit is still capped at the size of the smaller pipe. That is why despite the fact that the capacity of the link between node 1 and 10 is bigger (10 bits / second) than the link (5 bits / second) from the sending node (2), there is still bottleneck that allows only 5 bits / second of data to be sent through other nodes (1 and 10) to node 17.

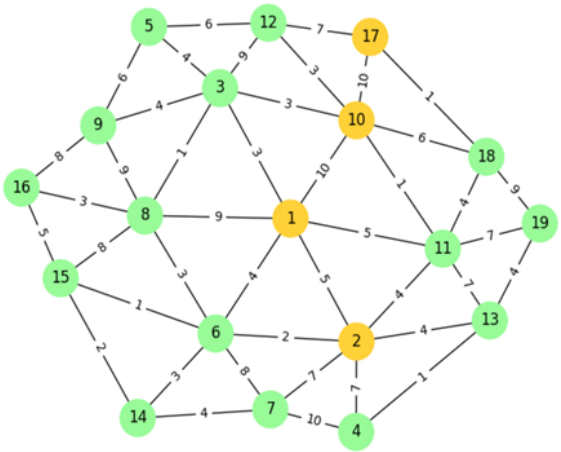

Figure 4.25: MECRAN sample data network, demonstrating the concept if throughput.

In summary, the bottleneck link is the link on the end-end path that constraints end-end throughput. Where R_{sender} is link capacity from sender node and $R_{receiver}$

is link capacity from receiving node, if:

- $R_{sender} < R_{receiver}$: then average end-end throughput is R_{sender}.
- $R_{sender} > R_{receiver}$: then average end-end throughput is $R_{receiver}$.

As seen above, delays, losses and throughput can cause bottlenecks and affect the overall latency between the sender and receiver of the data. MECRAN therefore addresses how to eliminate these bottlenecks effectively using the Data Packets Routing (DPR) routine. See section 3.3.2 for DPR context and chapter 5 for full DPR implementation. Below shows how we have factored in delays, losses and throughput by modelling variable node buffers as well as variable processing, transmission, queuing and propagation speeds during the initialisation of new nodes. Communication and buffer processing happen concurrently:

```
...

class Node:
    def __init__(self, ip_add, n_type, _data):
        self.node_ip_add = ip_add
        self.node_type = n_type
        self.node_data = _data
        self.node_port = random.randint(0, 100)
        self.node_transmission_rate = random.randint(50, 100) # bits per
            sec
        self.node_buffer = queue.Queue(random.randint(2, 5))
        self.node_packet_propagation_speed = random.randint(10, 20) #
            assume in ms
        self.node_packet_processing_speed = random.randint(10, 20) #
            assume in ms
        self.node_data_loss_count = 0

      @staticmethod
    def process_buffer_items2(Mecran_Nodes):
        send_path_objects = []

        print("All Nodes Capacity BEFORE:")
        for i in Mecran_Nodes:
            print(str(i.node_ip_add) + "'s capacity: " +
                str(i.node_buffer.qsize()) + "/" +
                str(i.node_buffer.maxsize))

        def process_buffer_items_sub(Node_):
            if not Node_.node_buffer.empty():
```

```
                get_frame = Node_.node_buffer.get_nowait()
                SendFrom = get_frame[0][0]
                SendTo = get_frame[0][1]
                send_path = [SendFrom, SendTo]
                Host1 = send_path[0]
                Host2 = send_path[-1]
                for x in range(len(send_path)):
                    send_path_objects.append(Mecran_Nodes[send_path[x] -
                        1])
                print("\n")
                print("Buffer Processing - Sending Path: ", send_path)
                print("Buffer Processing - Host " + str(Host1) + " -->
                    Host " + str(Host2))
                Data.transfer_data(edge, send_path_objects)

        with concurrent.futures.ThreadPoolExecutor(max_workers=5) as
            executor:
            future_to_url = {executor.submit(process_buffer_items_sub,
                MECRAN_NODE): MECRAN_NODE for MECRAN_NODE in Mecran_Nodes}
            for future in concurrent.futures.as_completed(future_to_url):
                MECRAN_NODE = future_to_url[future]
                try:
                    data = future.result()
                except Exception as exc:
                    print('%r generated an exception: %s' % (MECRAN_NODE,
                        exc))
                else:
                    ...

        print("All Nodes Capacity AFTER:")
        for i in Mecran_Nodes:
            print(str(i.node_ip_add) + "'s capacity: " +
                str(i.node_buffer.qsize()) + "/" +
                str(i.node_buffer.maxsize))


class Link:
    def __init__(self, name, node_names, channel_type):
        self.name = name
        self.node_names = node_names
        self.link_length = random.randint(10, 50) # in miles
        self.link_channel_type = channel_type
        if channel_type == "Fibre Optic": self.link_bandwidth =
            random.randint(100, 1000) # fibre optics have bigger
            bandwidths
        else:
            self.link_bandwidth = random.randint(50, 100)
        self.link_speed = random.randint(10, 50)
```

5

Implementation: MECRAN Core - Data Packets Routing (DPR) Routine

Contents

5.1 Introduction & Purpose

As discussed in Chapter 4, bottleneck is a reality of communication networks and therefore needs to be tackled effectively and carefully. Cloud-native application scheduling in MECRAN will be redundant if after scheduling, the user and the host server are unable to communicate effectively due to bottlenecks. That is why DPR

routine works in tandem with the Task Scheduling routine as shown in Fig. 5.1.

The occurrence of bottlenecks also has a direct impact on overall latency, therefore being able to forecast and minimize bottlenecks is an important part of enabling network communication and reducing latency within the network. This is the purpose of the *Data Packets Routing (DPR) Routine* which is based on the principles of the *Spatio-temporal Graph Neural Network (STGNN)* [100][101][102].

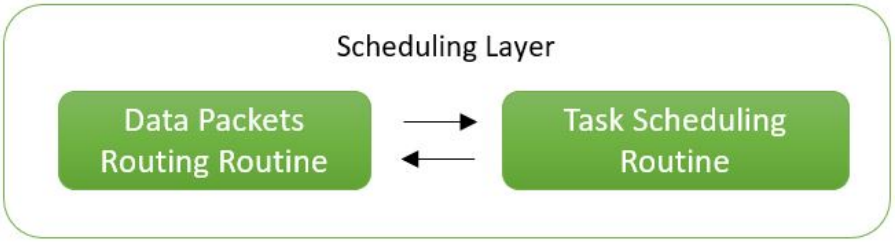

Figure 5.1: From the road map in Fig. 3.3, the DPR and the Scheduling routines compliment each other and make up the *Scheduling Layer*. This layer is activated by our 3 working scenarios (new application request, user mobility, garbage collection) as explained in subsection 3.3.2

We use STGNNs because it allows us to integrate spatial data (radio access network) with time series data (time series data on data packets being exchanged on the network) to predict bottleneck occurrence at individual base stations. In our single spatio-temporal graph model, base stations are the nodes while their connection types represent the *spatial edges* between the nodes. The base stations or graph nodes have the following features such as number of neighbours, buffer capacity and nodal delays, while the connection types or edge also have their own features such as connection type, length, throughput and bandwidth. These nodal and edge features through time, form the *temporal edges*. This chapter demonstrates how:

- We significantly reduce the occurrence of bottlenecks in the network by predicting bottlenecks using Spatio-temporal Graph Neural Network (STGNN), a deep learning model. This is built into our DPR routine.

- The DPR learns the bottleneck occurrence behaviour of our data packets switching network through time, to accurately predict future bottlenecks. The

benefit of this step in our work is to enable the DPR to enhance network communication by recommending less bottleneck-prone routes to both the task scheduler and to the packet switching network, to minimize bottlenecks building up.

- Our DPR outperforms other benchmark algorithms in terms of performance.

5.2 Predicting network bottlenecks using Spatio-temporal Graph Neural Network

In order to capture important spatial and temporal aspects of communication on our network to be able to forecast, we combine graph models (graph neural network) and time series models.

5.2.1 Formalising spatial and temporal features

As we have seen in section 4.3, graphs have been extremely useful in modelling our radio access network. Taking the graph $G(V, E)$ in Fig. 4.25 for example, it captures spatial / structural information about the base station entities and their relationships. We therefore pass this into a graph neural network model, to be able to capture other nodal non-spatial features (X_V such as nodal buffer capacity, nodal delays (as calculated in subsection 4.3.8), number of users / user equipment connected to the node) and edge non-spatial features (X_E such as connection type, length of connection type, bandwidth and throughput as discussed in subsection 4.3.8):

$$G(V, E, X_V, X_E) \tag{5.1}$$

At this point, we have been able to capture only the *static structure and features*. However to reflect an accurate picture of our radio access network, we need to factor in the time-varying *temporal features* such as number of data packets flowing through the network and the percentage of nodal buffer occupied at varying time steps, as in Fig. 4.24.

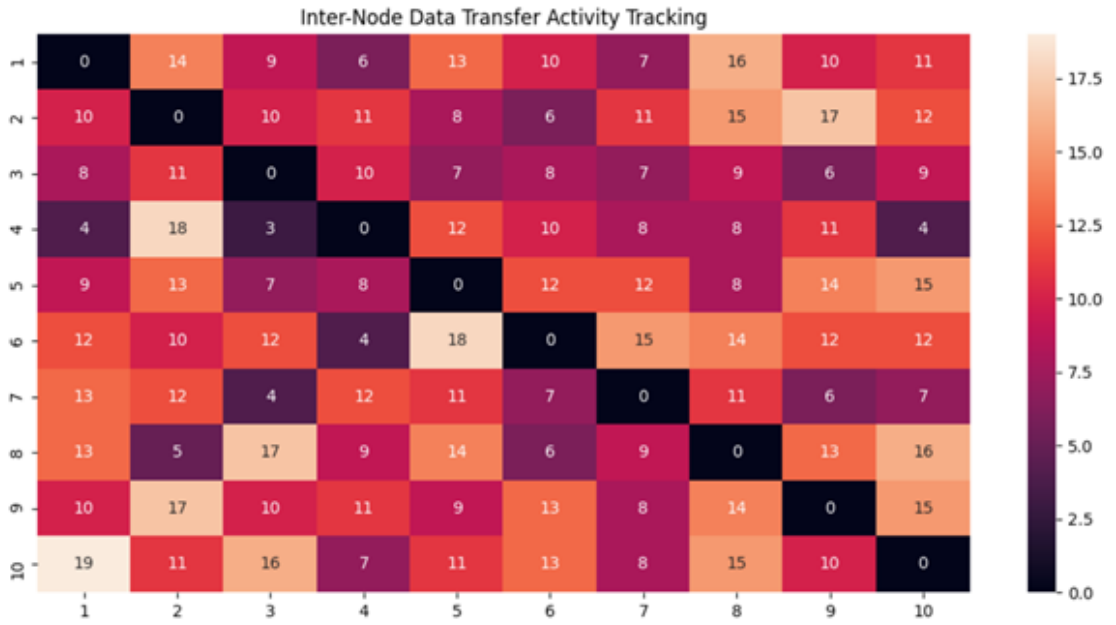

Figure 5.2: A heat map showing a snapshot of the number of data packets shared between nodes in the radio access network over time. This picture changes at each time step t.

This is because over time, our radio access network architecture *will not change*, however the number of data packets travelling through the network *will change or fluctuate*. The important question here therefore is how do we deal with graphs with static structures but with time-varying features? This is where the spatio-temporal graph principles [101] are incorporated:

$$G(V, E, X_V(t), X_E(t)) \tag{5.2}$$

This above approach (Eq. 5.2) allows us to sample our features over time as a spatio-temporal graph. We learn about the idea of the "*arrow of time*" from [103][104] which suggests that time series data of the past is highly correlated with future time series data. This is demonstrated in the *Autocorrelation Function* (ACF) plot in Fig. 5.3 below, based on the 10 communicating nodes in Fig. 5.2.

5.2.2 Data packets time series and autocorrelation

In Fig. 5.3's plot, we correlate the number of data packets as a variable to itself (hence autocorrelation) but with time lags.

To be able to generate the autocorrelation plot, we initially calculated the *Pearson* [11] correlation coefficients to measure the linear relationship between time and the number of data packets. Where the x variable is time and the y variable is the number of data packets being exchanged or travelling through the radio access network at that time:

$$\frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} \tag{5.3}$$

The above is based on *Least-Squares* fit and can be expanded as:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.4}$$

If the estimated correlation coefficient (as seen on the y axis of Fig. 5.3) is 1, it means there is a perfect positive relation between our normal distributed *time* and the *number of data packets* variables, such that when time increases, the number of data packets increases as well. -1 represents a negative relationship such that when time increases, the number of data packets decreases and 0 indicates there is no relationship between both variables.

We observe from Fig. 5.3 that our autocorrelation plot shows seasonality and decays slowly over time, meaning our time series data is *non-stationary*. Our autocorrelation plot with a 60 minutes' lag shows a clear pattern in how highly correlated the current number of data packets in the network at time t, is with the number of data packets previously t_{prev}. With this logic, we predict future data packets traffic volume based on historical network time series data, using Spatio-temporal Graph Neural Network which learns dynamic patterns from both the spatial and temporal data. Forecasting requires time series data to be *stationary*, so we transform our data accordingly using an approach called *differencing*, which

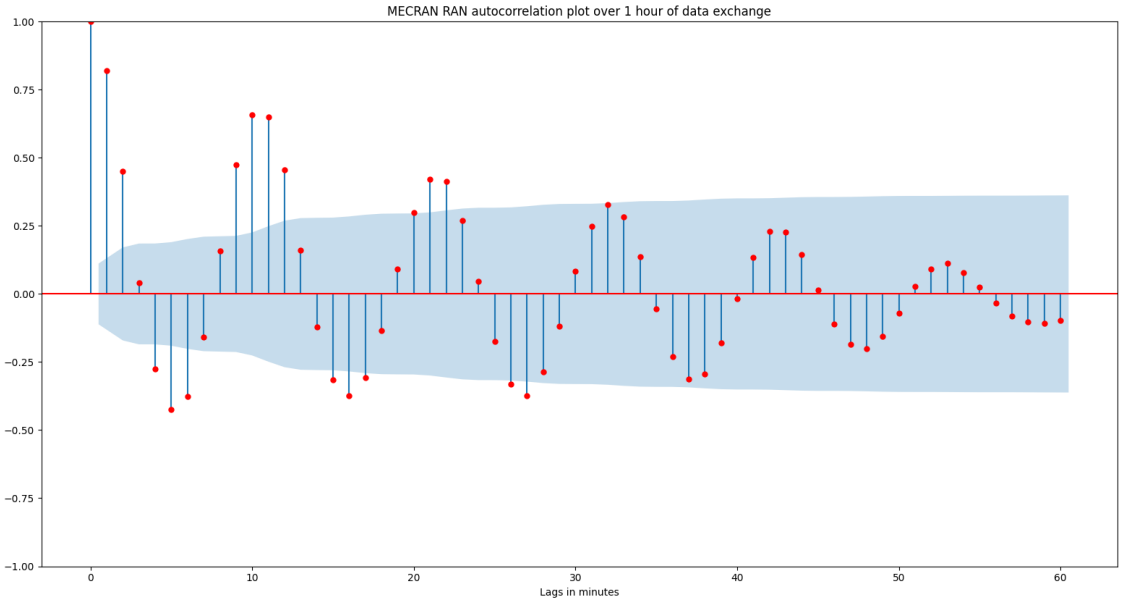

Figure 5.3: Autocorrelation Function (ACF) plot from the 10 nodes in Fig. 5.2 exchanging data over 60 minutes in the radio access network.

removes seasonality or trends from our data, leaving the mean and variance constant over time. We calculate 1 order difference between the current time series value and the previous value:

$$y'_t = y_t - y_{t-1} \tag{5.5}$$

The fundamental idea behind our time series modelling is to glean relationships between a number of consecutive radio-access network temporal feature sequences over time:

$$\sum_{i=0}^{a} \alpha_i y(t-i) = \sum_{i=0}^{b} \beta_i x(t-i) + \sum_{i=0}^{c} \mu_i \epsilon(t-i) \tag{5.6}$$

Throughout time t, we observe how data packets evolve and capture this observed evolution as a sequence $x(t)$ and other unobserved white-noise as $\epsilon(t)$, with leading coefficients α, β and μ respectively. Time series data often contains some white noise; the white-noise factor in Eq. (5.6) in MECRAN, is an opportunity to capture a very small proportion of new or unfamiliar future emerging cloud-native applications in this fast evolving cloud ecosystem that users may attempt to access from within MECRAN. It is likely that these applications' behaviour with regards

to generating and exchanging data packets, may be different from the portfolio of applications in scope. Another important consideration we have made however is that predicting the future data packets traffic at a base station does not only depend on the data packet time series but also on other spatial and temporal factors such as the number of neighbours, buffer capacity etc. That is why we incorporate both spatial features (radio access network structure) alongside the temporal features as a way of considering the holistic nature of radio access network and not underfit or overfit our model. This is why we take advantage of STGNNs (and look beyond other classical models such autoregressive integrated moving average (ARIMA), Kalman filters, regression models etc.) to model our RAN using graphs and capturing other important independent RAN variables as predictors of our target variable (bottlenecks based on data packets traffic).

5.2.3 DPR's STGNN-based model for data packets bottleneck prediction

Our model takes its foundations from a previously developed work for road traffic forecasting [101]. However the radio access network use case (which is the focus of our work) is very different as there is a high level of virtualization involved that we account for in our model, by their digital twins created in the simulated layer (Section 4.3). Due to its virtual nature the way objects or nodes interact within this network is different from the road use case. Most importantly, our goal is to predict the occurrence of bottlenecks in our network. This means predicting future data packet volumes alone is not enough to determine bottlenecks, so our work extends [101] to factor in other critical edge features such as bandwidth and throughput to achieve desired results.

Figure 5.5 describes the framework of our STGCN-based DPR model. In summary the model predicts data packet volumes at base stations' / network nodes' level, by passing our network temporal and spatial features through a 1 dimensional convolution network (Conv1D / 1D CNN) and graph neural network for learning. Our enhancements and contribution to this model allows the predicted

data packet volume at time t to eventually be used to estimate $(f(\hat{v}))$ nodal bottleneck for all base stations based on:

- the nodal delay.
- the nodal buffer capacity.
- and the connection types' throughput and bandwidth capacities

Given previous (M) data packets volume observations v_t of all our base stations or network nodes in the past, we predict values in the future (H) as:

$$f(v) = \log P(v_{t+1}, ..., v_{t+H} | v_{t-M+1}, ..., v_t), v_{t+1}, ..., v_{t+H} \tag{5.7}$$

We find the set of inputs v (base stations) for $f(v)$ within domain D (radio access network) that achieves the maximum function value; also known as $argmax$:

$$\{v | f(v) \geq f(w), \forall w \in D\}. \tag{5.8}$$

We consider v_t as an observation vector of a finite number of base stations at time step t. As a way of capturing detailed spatial and temporal features, our graph neural network is built to have 2 types of edges; one targeting the spatial domain and the other, the temporal domain. The spatial-based edge represents the network's connection type's length i.e. distance between 2 base stations connected by an edge. Whilst the temporal-based edge represents how the base station graph structure relates to itself, with regards to time.

This allows the graph's important features to be captured in a structured way through an ordered consecutive time step sequence. That is the base station graph at the current time step, is linked with the same graph at a previous and future time steps. See Fig. 5.4 for a visual representation of this concept. The data packets volume observations v_t at each time step is captured in a weighted graph structured data adjacency matrix W (as seen in Fig. 5.4), influencing the definition of our graph model:

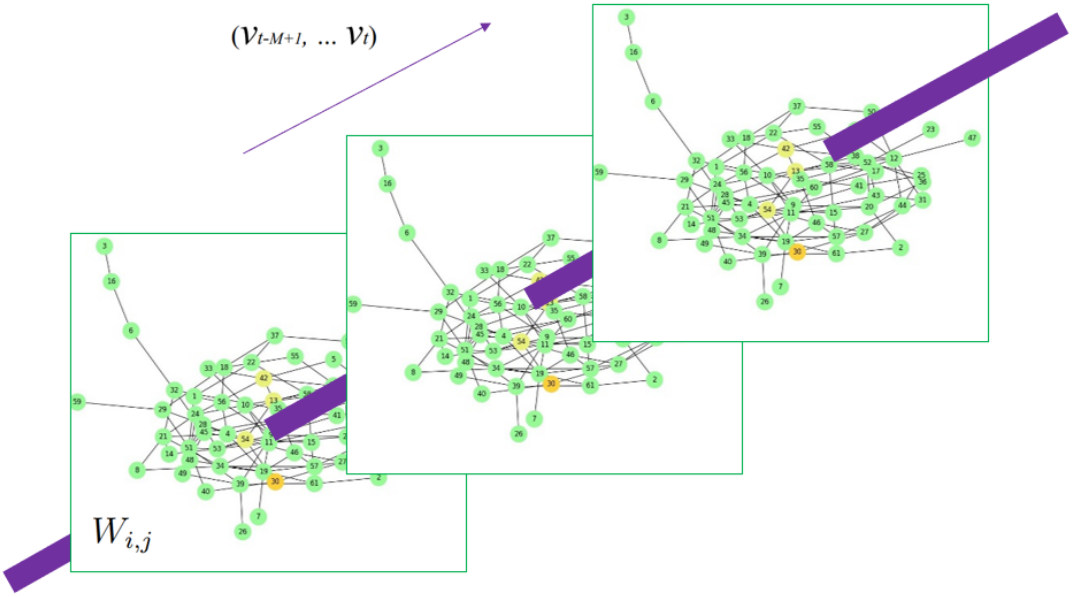

Figure 5.4: DPR Model Spatial and Temporal Edges. As a way of capturing detailed spatial and temporal features, our graph neural network is built to have 2 types of edges; one targeting the spatial (structural) domain and the other, the temporal (time) domain in the radio access network. The temporal / time edge is represented by the bold violet bar that runs across the 3 exemplary time frames shown.

$$G(V, E, X_V(t), X_E(t), W) \tag{5.9}$$

Our framework in Fig. 5.5 is based on an architecture of a spatio-temporal graph convolutional networks. It takes as inputs, past $(v_{t-M+1}, ..., v_t)$ data packets traffic data across the entire graph representing the base stations in our radio access network and also the weighted graph's structured data adjacency matrix W. Two of the larger blocks in our model include the *Spatio-temporal Convolution blocks* and *the fully connected output layer*. Each of these layers are composed of multiple operations as expanded out in Fig. 5.5. In the Spatio-temporal Convolution block, there are two main fundamental operations / layers that happens within this block to capture different characteristics of our radio access network. These are the *Spatial Graph Convolutions* and the *Temporal Gated Convolutions*.

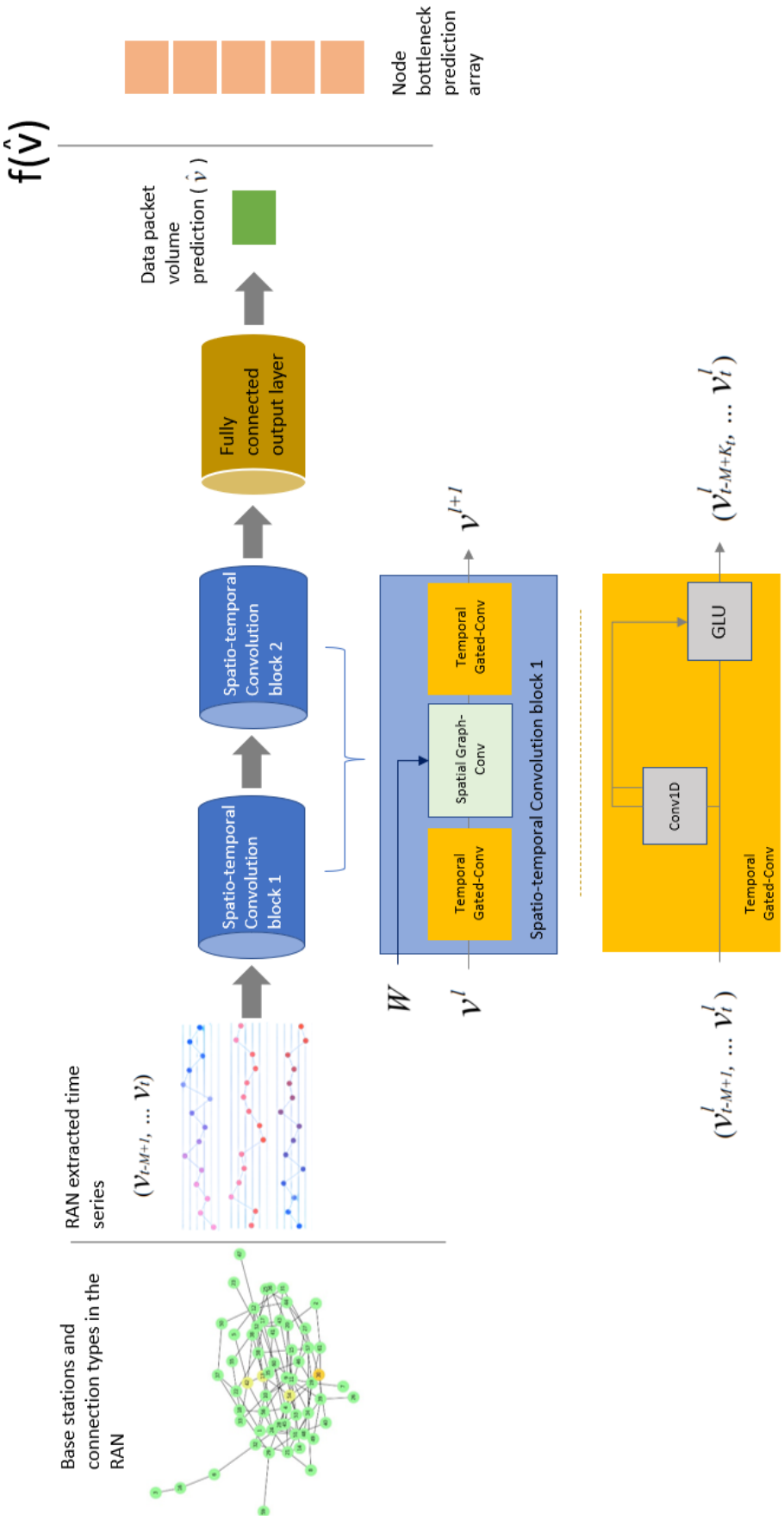

Figure 5.5: DPR's STGNN based model diagram for data packets bottleneck prediction.

The Spatial Graph Convolutions Layer

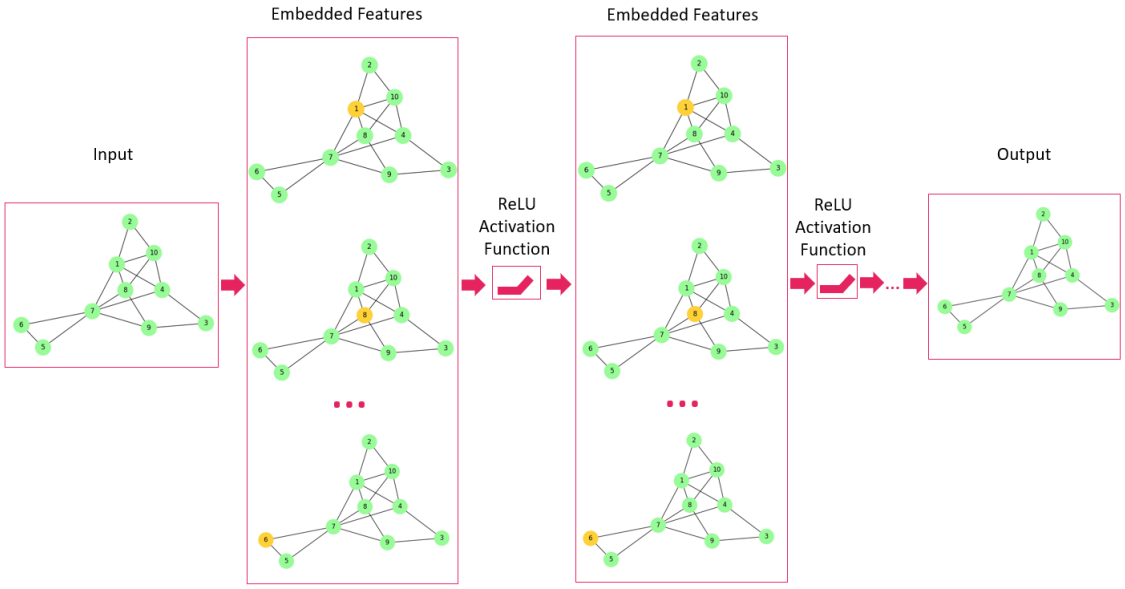

Figure 5.6: DPR Spatial Embedding Diagram explaining the operations done by the spatial graph convolutions layer on our radio access network data in Fig. 5.5.

The Spatial Graph Convolutions layer aims to capture local structures in our radio access network graph. It accomplishes this by creating a weighted combination of data packet volumes at all base stations and their neighbours. Taking base station 1 in Fig. 5.6 for example, its neighbours are base stations 2, 10, 4, 8 and 7. The Spatial Graph Convolutions layer creates a weighted sum (of data packet traffic) of all of these neighbouring base stations and propagates back to base station 1's value for the next layer. This operation is also known as *Spatial Embedding* and as our model iterates through the various base stations and layers, it learns to extract the most useful and relevant information of base stations and their neighbours. Our deep neural network continues to repeat this spatial embedding operations on our base stations and neighbours to glean more abstract information as informative independent variables for our intended forecast. Spatial embedding leverages *spectral graph theory*. Through spectral graph theory our model learns and extracts important graph properties using eigenvalues and eigenvectors of their associated graph adjacency matrices (as seen in figures 5.4 and 5.5) and the graph *Laplacian* which enables different ways of spectral clustering [101].

The Temporal Gated Convolutions Layer

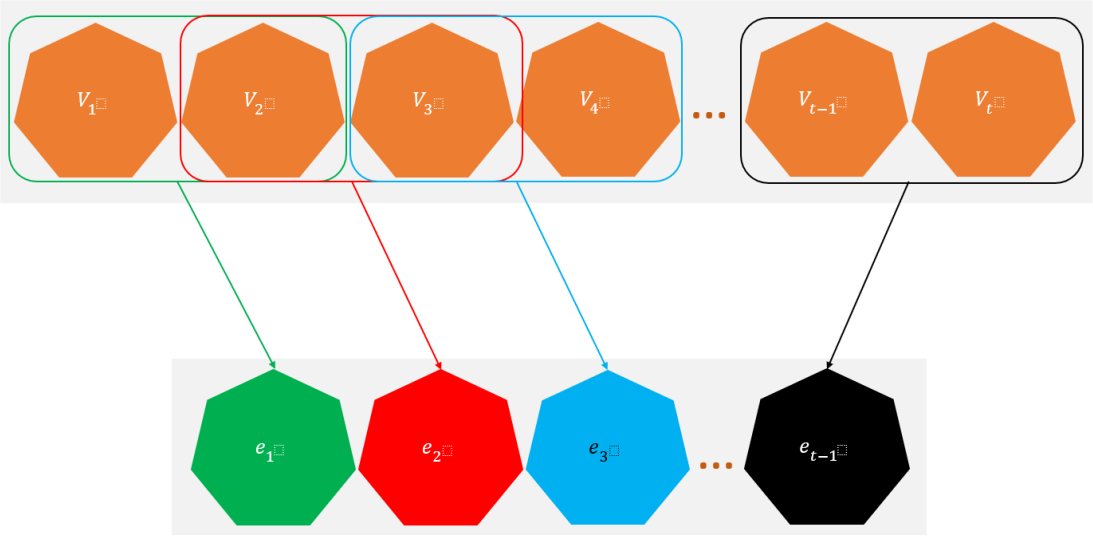

Figure 5.7: DPR Temporal Embedding Diagram explaining the operations done by the temporal gated convolutions layer on our radio access network data in Fig. 5.5.

The Temporal Gated Convolutions Layer as seen in our model, records details of our radio access network based graph as time passes using *temporal embedding*. This concept is best represented in Fig. 5.4, along the time axis of our radio access network based graph, v_t holds raw data packet volume data for all base stations in the graph. In our work, we sample the state (data packets volumes) of our RAN every 10 mins as seen in the autocorrelation plots in Fig. 5.3. We can think t in v_t as 10 minutes' interval discrete samples of all base stations. This convolution layer works as a slicer across the time axis of a RAN graph, combining the data packet observations in relation to time into temporal embedding as shown in Fig. 5.7. Assuming we have a size 2 slicer or window, the convolution begins by combining the values at v_1 and v_2 into a single temporal embedding e_1. It then iterates through the entire input history v_t to obtain the temporal embedding array (as seen in Fig. 5.7), whose size is $<$ than our observations' array. DPR's model was implemented in Python 3 using PyTorch's framework.

5.3 DPR Model Validation & Results

5.3.1 DPR Model Validation

We simulated a radio access network with 91 base stations across a geographic area size of 7023 km^2 (see Fig. 5.8) using the MECRAN RAN setup in sections 4.2 and 4.3.

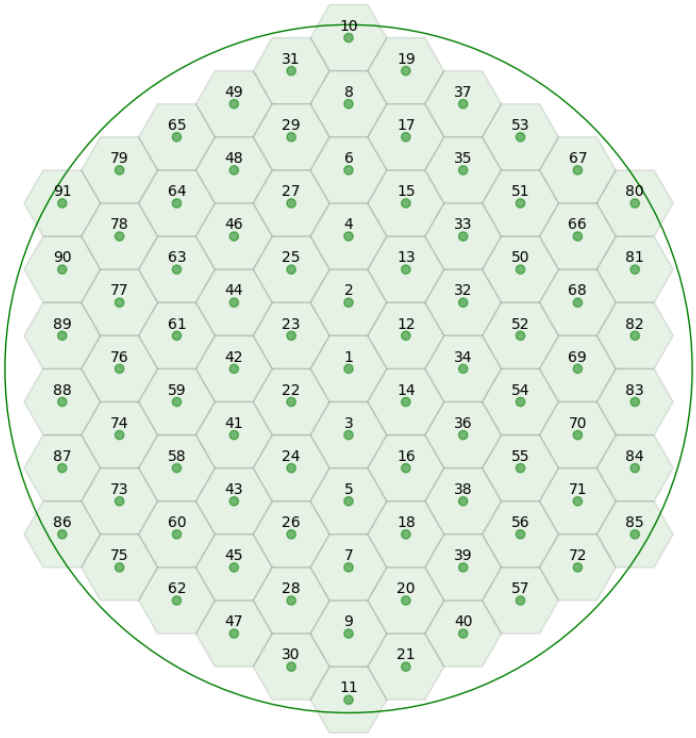

Figure 5.8: DPR model validation simulated base stations.

The base station network is translated into a graph as per implementation in 4.3.4, initializing the base station and connection types abstract classes with attributes and establishing links between between the base station in the network.

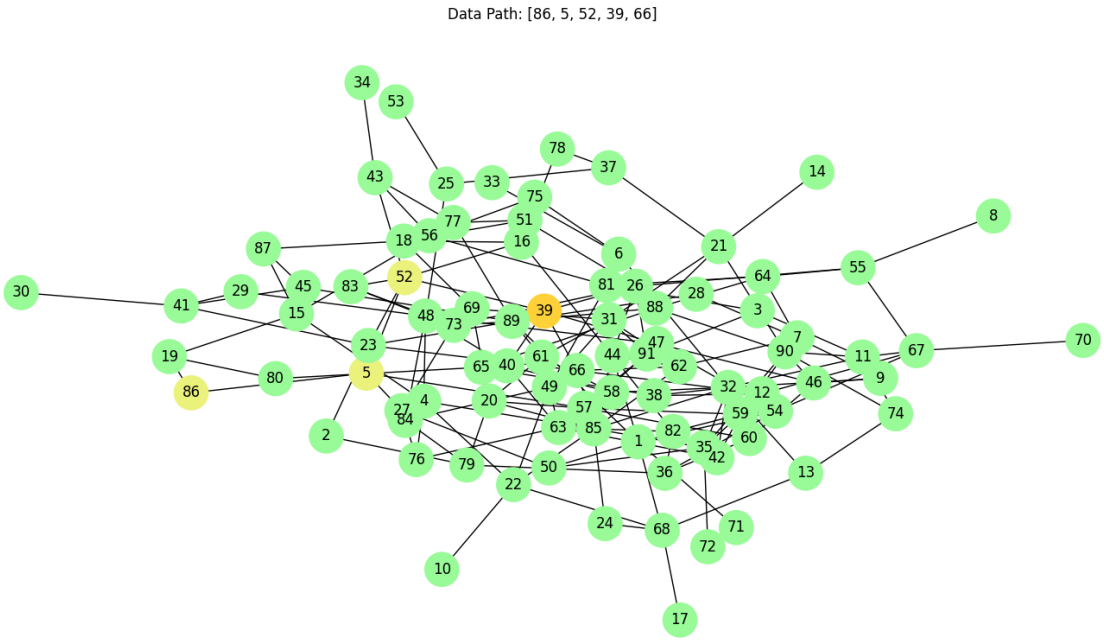

Figure 5.9: DPR Model Validation RAN graph. Figure 5.8 is translated into a graph to take advantage of the deep learning model discussed in sub section 5.2.3.

Additionally, we were able to generate 7 days' worth of 10 minutes discrete interval time series data (Fig. 5.10) as a result of base station communication using packet switching from the TCIP protocol implemented in Chapter 4.

Category	Status	Item	Unit	00:00	00:10	00:20	00:30	00:40	00:50	01:00	01:10	01:20	01:30
Base Station 6	New	Number of data packets	data packet		0	0	0	0	0	0	0	0	0
Base Station 7	New	Number of data packets	data packet	0	1	0	0	0	0	0	0	0	0
Base Station 7	New	Number of data packets	data packet		1	0	3	3	0	1	0	0	0
Base Station 7	New	Number of data packets	data packet		0	0	0	0	0	0	0	0	0
Base Station 8	New	Number of data packets	data packet	0	0	0	0	0	0	0	0	0	0
Base Station 8	New	Number of data packets	data packet		0	0	3	0	2	0	-3	2	4
Base Station 8	New	Number of data packets	data packet		0	0	0	0	0	0	0	0	0
Base Station 9	New	Number of data packets	data packet	23	17	23	23	10	8	10	6	10	9
Base Station 9	New	Number of data packets	data packet		10	5	14	8	5	13	11	22	10
Base Station 9	New	Number of data packets	data packet		0	0	0	1	0	0	3	1	0
Base Station 10	New	Number of data packets	data packet	2	1	2	1	3	0	2	2	0	0
Base Station 10	New	Number of data packets	data packet		0	0	0	1	-1	1	0	0	2
Base Station 10	New	Number of data packets	data packet		0	0	0	0	0	0	0	0	0
Base Station 11	New	Number of data packets	data packet	0	0	1	0	0	0	0	0	0	0
Base Station 11	New	Number of data packets	data packet		0	2	2	1	1	0	0	1	-2
Base Station 11	New	Number of data packets	data packet		0	0	0	0	0	0	0	0	0

Figure 5.10: A subset of MECRAN's data packets traffic dataset for time series forecast and bottlenecks estimation.

The captured data from base stations exchanging data across the MECRAN's radio access network contains all the relevant base station and connection type spatial and temporal features, which are passed through DPR's bottleneck prediction model. We took the 80/20 train-test approach. So we trained our model on 5 days 12 hours (with 10 minutes intervals) worth of data) and as a result of parallelization and combination of the extended CNN and graph convolutions, DPR's model was capable of processing multi-channel input and arbitrary batch sizes. Our model was consequently trained using a batch size of 1, a Rectified Linear Unit (ReLU) activation function and an Adam optimizer for 250 epochs.We also tested the trained model on 1 day 12 hours worth of data. Our initial 1 dimension convolution neural network translated 1-channel time series into multi-channel graph features. The accompanying graph convolution layers all possessed their own 16 output channels. Additionally we used a learning rate of 0.001 and a droprate of 0.18.

5.3.2 Results

We captured the results of applying DPR to the MECRAN's RAN (green line chart in Fig. 5.11) vs the RAN without DPR (red line chart in Fig. 5.11) across a 12 hour period and there is a clear improvement in data exchange across the network.

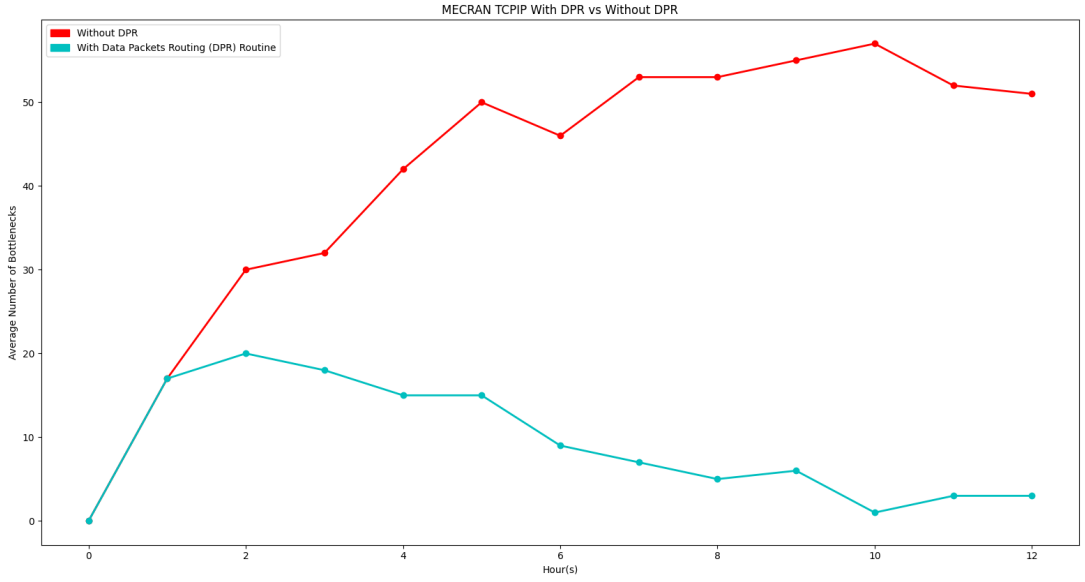

Figure 5.11: MECRAN TCPIP With DPR vs Without DPR.

Before DPR was applied, the network was allowed to build up bottlenecks for the first hour. There were a similar number of bottlenecks built up by the end of the first hour. Both experiences therefore started with an average of 17 bottlenecks at time 1 hour as shown in Fig. 5.11. From this point on average, we see the number of bottlenecks grow over the cause of the 12 hour period with a few not significant dips at hours 3, 6, 11 and 12 without DPR.

The inverse is true when DPR is applied to the network with the aim of predicting bottlenecks for MECRAN to suggest more efficient routing routes for data exchange. With DPR applied from hour 1, on average we see a constant fall in the number of bottlenecks across the 12 hour period apart from a few also insignificant average increases of the number of bottlenecks at hours 2, 5, 9 and 11. Across the 12 hour period MECRAN was able to reduce the number of bottlenecks by a approximately 78%! As with machine learning models, with more training and tuning the model has the potential of even yielding improved and better results.

5.4 Compare DPR model performance results with benchmark algorithms

Long short-term memory (LSTM), Sequence-to-sequence learning (Seq2Seq) and Spatio-Temporal Graph Convolutional Networks (STGCN) are the baseline models we are benchmarking DPR against. Running the same dataset through the baseline models on an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, with GPU: NVIDIA GeForce GTX 1080), we were able to decide and set the below hyperparameters to measure and evaluate the performance of all models. Below is a justification of why these specific hyperparameters were chosen.

Hyperparameters	**LSTM**	**Seq2Seq**	**STGCN**	**DPR**
Learning Rate	0.001	0.01	0.01	**0.001**
Loss	Mean Squared Error	Mean Squared Error	Mean Squared Error	**Mean Squared Error**
Batch size	32	1	32	**1**
Epochs	20	10	10	**250**
Dropout rate	0.18	0.18	0.18	**0.18**
Optimizer	Adam	Adam	Adam	**Adam**

Table 5.1 Model Hyperparameters

Our model choices use the hyperparameters as shown in in Table 5.1 to optimize the model for best performance. *Gradient Descent* is the adopted method to iteratively optimize DPR's model in producing the best bottleneck estimates. Gradient Descent methods multiply their gradients by a scalar value called *learning rate* in order to determine the step size of every iteration while approaching a minimum of a *loss function*. This is key as focusing on a strategy that approaches the minimum of a loss function enables DPR to make predictions with higher accuracy and consequently fewer errors. A *Mean Squared Error* (MSE) loss function is used to determine the size of error between the predicted and actual value as it is the distance between the actual data point and fitted line. MSE derives an average of the square of the difference between actual and predicted bottleneck value.

A learning rate of 0.001 was chosen as a much smaller learning rate takes the model a longer time to learn and a higher learning rate introduces fluctuations / instabilities in the quality of learning. The next paragraphs in this sub-section explain the effects of testing different learning rates. Due to our large training data size, it was enough to use 1 example of training data to train the model in each iteration; hence a batch size of 1. The complete training dataset was passed through the model 250 times before we saw a satisfactorily minimized model error; hence a batch-size of 1 and an epoch of 250.

A dropout rate of 0.18 is introduced to prevent overfitting our model to the current network data so that the model is generalized enough to be able to deal with any future network structure or data, maintaining a high degree of performance. An optimizer is finally needed to derive the value of the weights that best minimize the error when mapping inputs to outputs for better prediction accuracy. The *Adam Optimizer* is used mainly because of its faster computation capabilities and it also requires few tuning parameters.

Additionally, to fairly compare all models in scope we measured and evaluated their performance using the *Mean Absolute Error* (MAE) and the *Root Mean Squared Error* (RMSE) performance metrics. In the same vein, we took the 80/20 train and test approach by training our models 80% of our data, amounting to 5 days 12 hours (with 10 minutes intervals) worth of data) and testing with the outstanding 20% data which amounts to 1 day 12 hours (also with 10 minutes intervals) worth of data. The results below show the mean and standard deviations from the performance metrics, after 3 runs through all models.

The training dataset is generated and validated based on the most widely accepted and adopted set of rules that governs how computer systems interact on the internet - Transmission Control Protocol and the Internet Protocol (TCP/IP) [99][105]. In order to predict how and where traffic flows within the RAN network, a digital twin of network data is produced using an algorithm purely based on TCP/IP as described in sub-section 4.3.5. The dataset describes network nodes (i.e. source, intermediary and destination nodes including its descriptors such as

IP addresses), connection types, throughput, bandwidth, connection type sizes, data packets, data packet sizes, network layers (Application layer, Transport layer, Network layer and Datalink layer), network layer inputs as well as outputs (such as Data, Segment, Datagram and Frame) and network traffic.

Models	**MAE**	**RMSE**
DPR	**8.13 ±0.23**	**17.64 ±0.14**
LSTM	9.64 ±0.04	20.11 ±0.02
Seq2Seq	9.33 ±0.42	19.84 ±0.12
STGCN	8.54 ±0.34	18.24 ±0.13

Table 5.2 Models Performance Metrics Evaluation

Learning and error function optimization for accurate bottleneck prediction can be observed in Fig. 5.12 to Fig. 5.19 where the training dataset is the red line chart and the test dataset is the green line chart. In order to identify the best learning rate for our DPR model, a number of learning rates spanning the high, middle and low end learning rate spectrum (usually between 0.0 and 1.0) have been tested iteratively: 1.0, 0.1, 0.01, 0.001, 0.0001, 1e-05, 1e-06 and 1e-07. Testing a number of learning rates helped to tune the model as a way to optimally minimize the estimated error when the model weights are updated in each iteration.

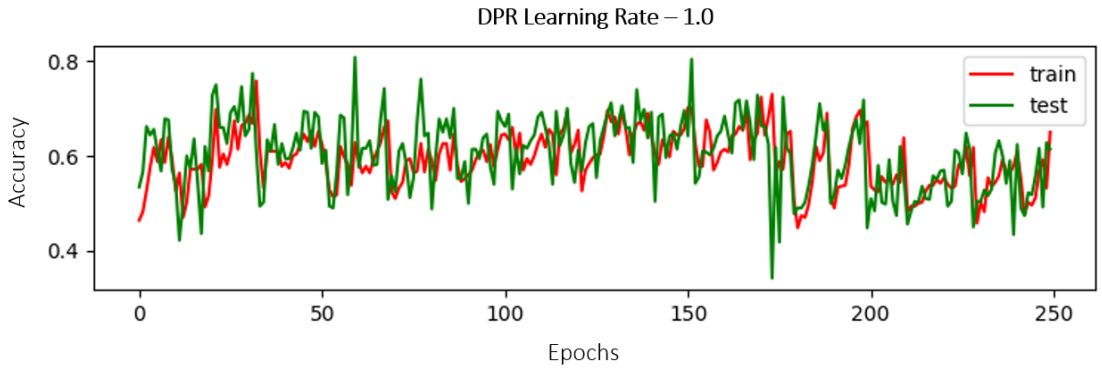

Figure 5.12: DPR Learning Rate - 1.0

We see that very large learning rates such as 1.0 and 0.1 (in Fig. 5.13 & 5.14 respectively) has seen large learning fluctuations, converged to a sub-optimal solution relatively fast and has used fewer epochs to train and learn.

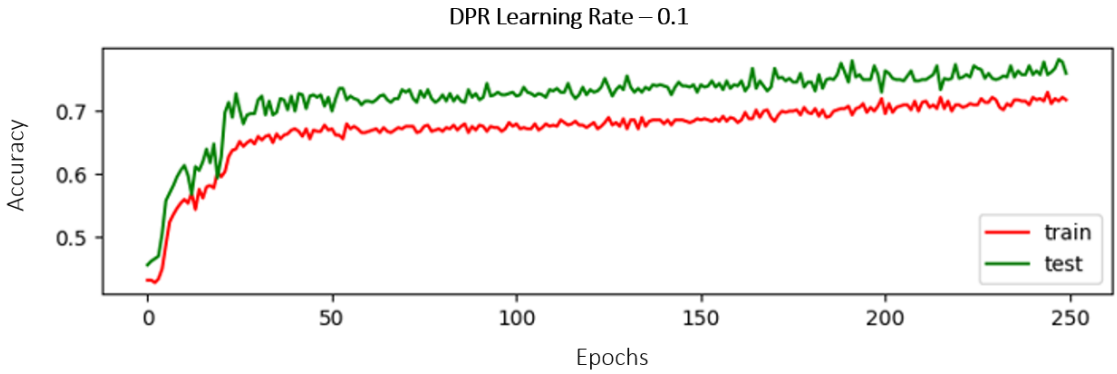

Figure 5.13: DPR Learning Rate - 0.1

The model learns better with learning rate 0.01 and learns best with learning rate 0.001. The learning accuracy of these rates can be seen to improve in figures 5.14 and 5.15. Learning accuracy starts to decline with learning rate 0.0001 (as in Fig. 5.16) which initiates the lower end of the learning rates spectrum.

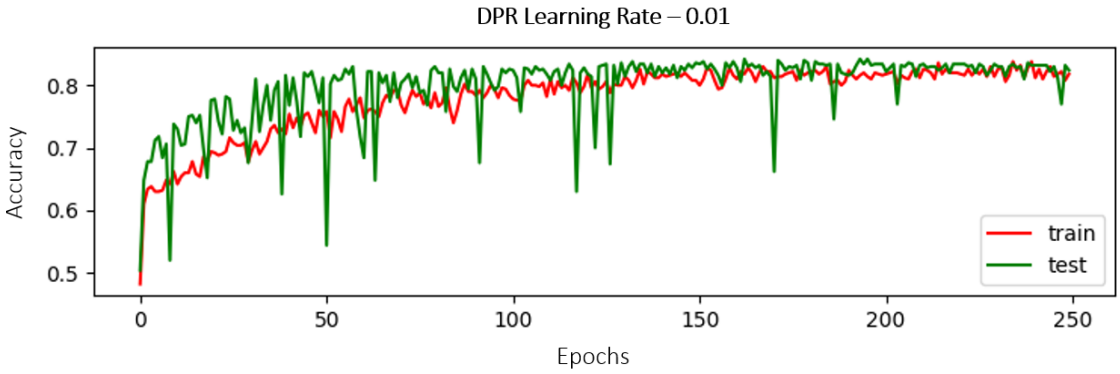

Figure 5.14: DPR Learning Rate - 0.01

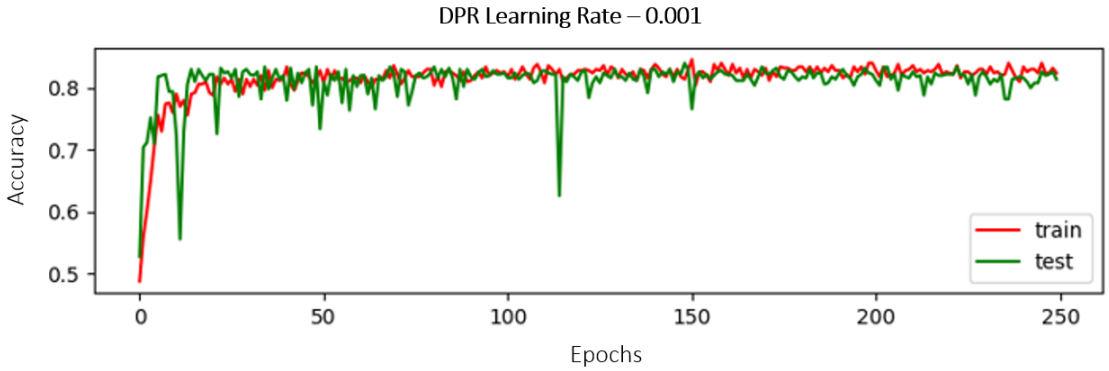

Figure 5.15: DPR Learning Rate - 0.001

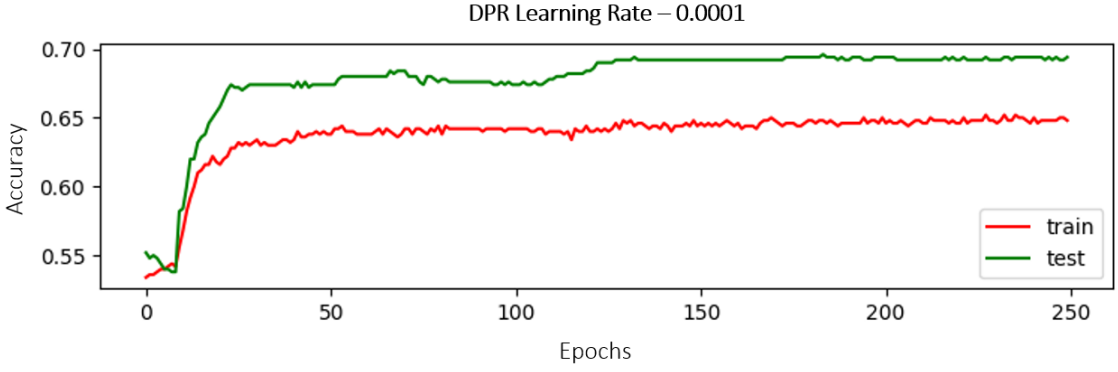

Figure 5.16: DPR Learning Rate - 0.0001

On the other hand with very small learning rates (such as 1e-05, 1e-06, 1e-07), because smaller updates are made to the weights in each iteration, they require larger epochs to train and learn. Very small learning rates may experience performance issues as a results of the large amount of computations that are required. We see less promising results with these learning rates as they produce the highest error margin as seen in figures 5.17 to 5.19.

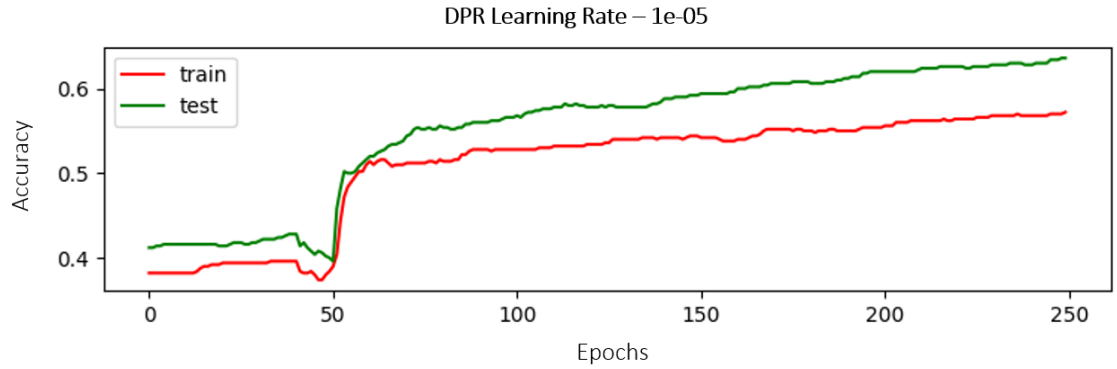

Figure 5.17: DPR Learning Rate - 1e-05

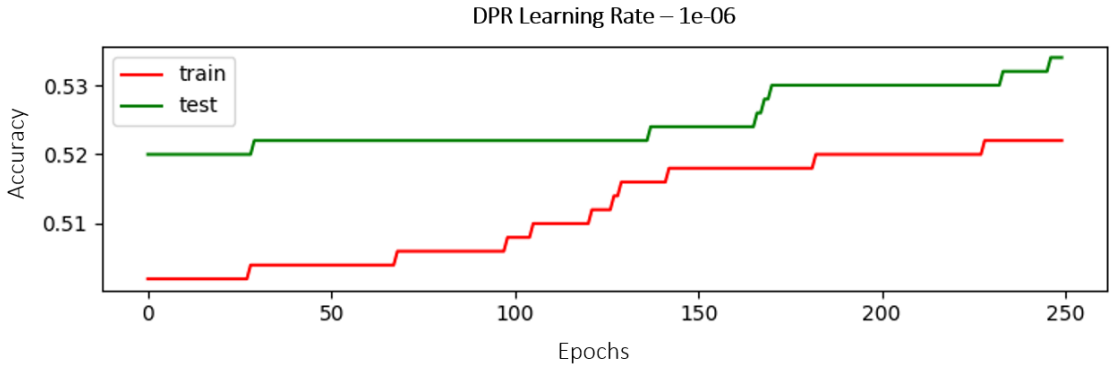

Figure 5.18: DPR Learning Rate - 1e-06

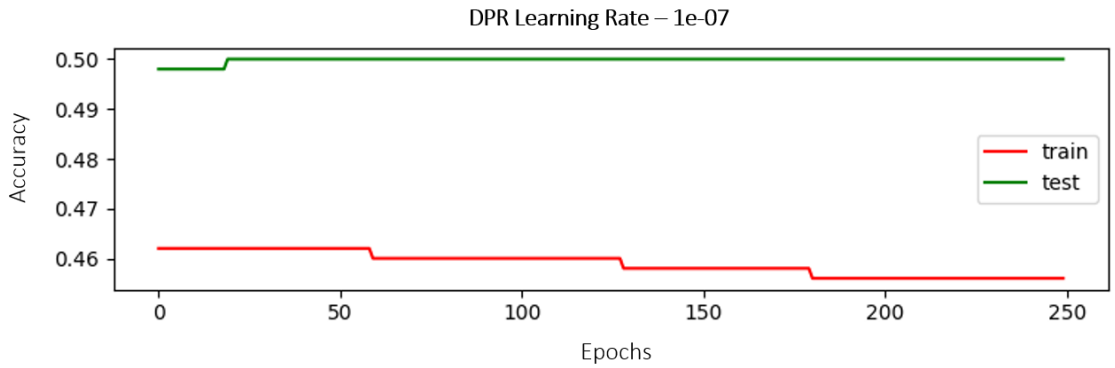

Figure 5.19: DPR Learning Rate - 1e-07

5.5 Summary

After evaluating all baseline models with the same performance metrics and similar hyperparameter definitions, we observe our DPR model outperforms LSTM, Seq2Seq and STGCN models (as shown in Table 5.2) using a learning rate of 0.001. Extremely large learning rates have resulted in unstable training and the network does very little learning with small learning rates.

As indicated earlier, cloud-native application scheduling in MECRAN will be redundant if after scheduling, the user and the host server are unable to communicate effectively due to bottlenecks. Therefore it is a critical part of MECRAN architecture's towards reducing latency. And after achieving such encouraging results with DPR, the next and final concluding step is to build the task scheduler, which will depend on the DPR for insights on the best or less bottleckneck-prone paths for scheduling cloud-native applications.

6

Implementation: MECRAN Core - Task Scheduling Routine

Contents

6.1 Introduction & Purpose

The main goal of this dissertation is to optimize the round-trip delivery of data at low latency, through dynamic resource scheduling of cloud-native applications, to run in close proximity to a mobile user, at the edge of the radio access network.

Having setup all radio access network tools and resources, we have also managed to successfully put in place an important pre-step towards scheduling: minimizing

bottlenecks. We are now going to build the *Task Scheduling Routine*, using Deep Reinforcement Learning (i.e. Reinforcement Learning and Deep Q-network (DQN) algorithm). As articulated from our road map in Fig. 3.3, our task scheduler will take insights from the DPR routine in order to schedule cloud-native applications, based on network resources that have the best chance at ensuring the long-term low latency needs of a connected mobile user.

Our key contribution here is enhancing the reinforcement learning model with binary search for improved performance.

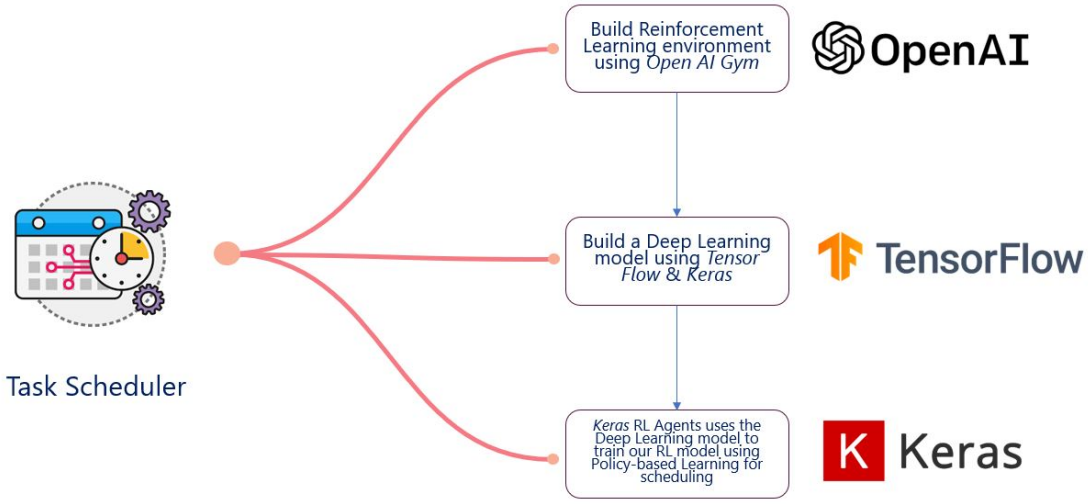

Figure 6.1: Task Scheduling Routine Development Activities

In this chapter as per Fig. 6.1, we are going to do the following in order to establish a durable fit for purpose task scheduler:

- Create a Reinforcement Learning environment, using *Open AI Gym* [106].
- Build a Deep Q-network (DQN) model using *Tensor Flow* [107] and *Keras* [108].
- Train model using *Keras-RL Agents' policy based learning* [108].
- Evaluate results and compare with baseline algorithms

6.2 Build Task Scheduling Routine using Deep Reinforcement Learning

6.2.1 Formalizing task scheduling as a Reinforcement Learning problem

In reinforcement learning terms, we consider MECRAN as an *Environment* in which the task scheduler acts as an *Agent* and takes scheduling *actions* A_t at certain *states* S_t on cloud-native applications, to maximize their chances of achieving low latency, reaping the *reward* R_t of the cloud-native application being able to meet its latency requirements for performance. Reinforcement learning judges actions by their rewards and therefore when an agent takes an action, there are 3 reward outcomes: a decrease in latency, no change in latency and an increase in latency. Knowing (from the DPR routine) the set of eligible nodes in scope for scheduling, the task scheduler takes an action to "*adjust*" where an application should be hosted and learns overtime based on the feedback loop in Fig. 6.2.

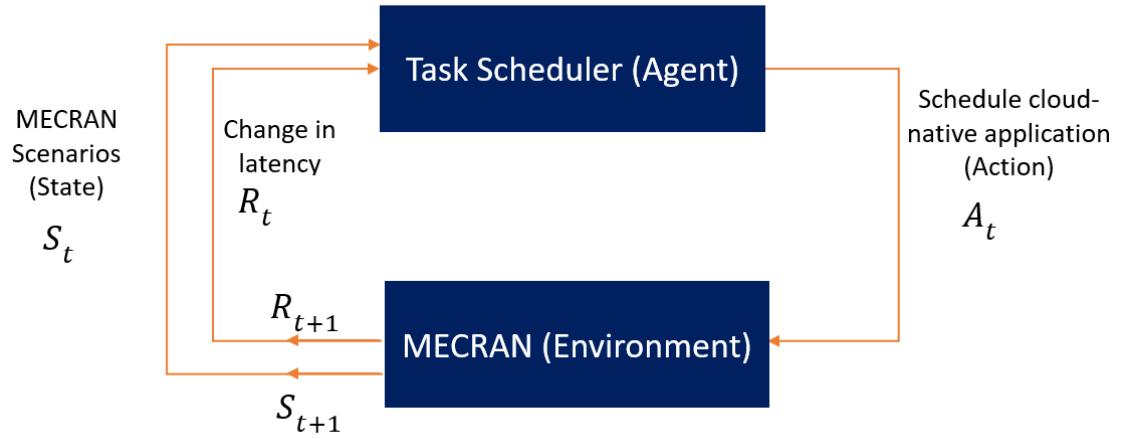

Figure 6.2: Reinforcement Learning formalization - the MECRAN context

The task scheduler agent takes actions based on the state. The states are the 3 scenarios (as specified in the scenario layer of the our road map in Fig. 3.3 and explained in sub-section 3.3.2): user accesses cloud-native application for the first time; user location changes due to mobility and lastly change in user traffic density,

resulting to base station resources freeing up. Scheduling actions taken by the task scheduler changes the state of the environment and this transition is communicated back to our task scheduler agent through a scalar *reinforcement signal* r. Therefore the state of MECRAN environment at any given time is determined by the state transitions or transition function $T(s, a, s')$ and action taken prior:

$$P(S_t = s' | S_{t-1} = s,\ a_t = a) = T(s, a, s') \tag{6.1}$$

This is based on the *Markov Decision Process* (MDP) [109]. MDP's aim is to iterate from current state to final state [109]. Through trail and error our task scheduler learns to take scheduling decisions and actions that have a high probability of increasing the long-term sum of values of the reinforcement.

With scheduling being a Markov decision process, each state (scheduling scenarios) has a set of actions that are performed in the specific state. For example , we refer to Fig. 3.4 which outlines the set of actions that needs to be performed in our first scenario (i.e. when a user accesses cloud-native application for the first time). Essentially when a user tries to access a cloud-native application for the first time, MECRAN compares the primary latency between the user and the distant host server to the latency required by the application in question. If the latency required by the application is greater than the derived latency between the user and the distant server, MECRAN allows the cloud-native application to continue to be hosted on the distant data centre. However, if the latency required by the application is less than the derived latency between the user and the distant server, MECRAN schedules / migrates the application to a base station at the edge of the radio access network and in close proximity to the user. By so doing, the latency between the user and the new temporary host server (i.e. base station) has a much higher chance of meeting the latency required by the application.

This is our policy in our reinforcement model that enables us to map states to actions within the environment. Based on the triggering scenario, our policy function $\Pi : S \mapsto A$ selects the appropriate action $a \in A$ given the current state $s \in S$.

Sometimes it is not instantly clear the effect of the action being taken for a particular scenario / state using the policy and therefore we make adjustments using reinforcement learning to the policy when the reward R_t is undetermined. One of the adjustments we make is to introduce *discount rate* γ to calculate a weighted estimate of future rewards:

$$\sum_{t=0}^{\infty} \gamma^t r_t \tag{6.2}$$

At any particular time t, our discounted rate γ is between 0 and 1, which means the formula above associates bigger rewards as time increases to handle the future reward uncertainty issue. We articulate this better with our earlier transition function in perspective:

$$V^{\Pi}(s) = R(s) + \sum_{s'} T(s, a, s')\gamma V^{\Pi}(s') \tag{6.3}$$

The next sections of this chapter will be focusing on the task scheduling routine implementation. In order to build and train our deep reinforcement learning based task scheduling model, we will first have to create the MECRAN scheduling environment. Following that is to build a deep learning model and train this model using a *Deep Q-network (DQN)* reinforcement learning agent for scheduling. Scheduling here essentially means dynamically adjusting / calibrating the location (base station node / server) where an cloud-native application can be hosted to meet its latency requirement, considering the distance the user and base station and the network resources available. The available network resources are suggested by the DPR routine from chapter 5.

6.2.2 Creating Reinforcement Learning environment, using Open AI Gym

Before creating the environment we define conceptually the goal, activities and properties of the environment.

Cloud-native application scheduling is an optimization problem. Our goal in this environment is to build a model that keeps the optimal latency for as long as possible for a specific cloud-native applications performance. Assuming an application requires at least 10 milliseconds (ms) latency for performance, the scheduler adjusts where to host the application at different distances from the user to see which base station has the best chance of delivering at least 10 ms latency long-term.

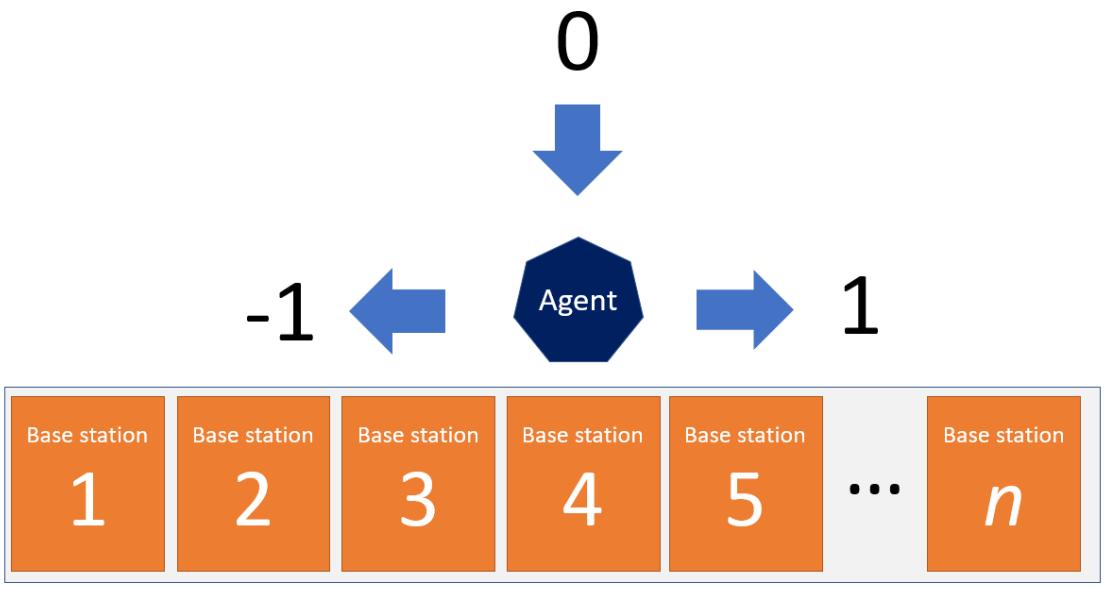

Figure 6.3: A diagram showing the Task Scheduler agent taking actions -1 (move left), 0 (stay put) or 1 (move right) as a way of exploring the optimal base station range for optimal latency required by the cloud-native application.

The main key activity is the scheduler receives from the DPR, a sorted matrix (of eligible base stations) by distance from the user. Based on the latency required by the cloud-native application, the scheduler estimates the distance that would deliver that latency. Following that the scheduler uses *binary search* to narrow down the optimal range of base stations from the list to deliver the optimal latency. The scheduler has a fixed time limit to make a decision about where to schedule

the application by exploring the optimal range of base stations to see which one provides the optimal latency to support the application.

The environment therefore needs the following properties:

- **States**: MECRAN scheduling scenarios.
- **Actions**: $(A_t - 1)$, where A_t is 0, 1 or 2. The action taken will determine how to move up or down the base station list. The outcome of these actions are -1 (move left), 0 (stay put) or 1 (move right) on the optimal base station list to find the optimal required latency. See Fig. 6.3.
- **Optimal latency**: a range from 0 to the latency required by the specific cloud-native application
- **Fixed time limit**: The length of time the scheduler has to schedule. This is set at 5 seconds in our implementation.
- **Episodes**: 1,000 full episodes or sequence of states, actions and rewards, which ends with terminal state.
- **Base stations list**: The list of eligible base stations is received from the DPR. This gives the assurance that it is unlikely for applications scheduled on these base stations to experience communication delays as a result of bottlenecks in the paths connecting the base station to the user.

The reinforcement learning environment is built in Python 3 and uses the following dependencies: OpenAI Gym [106], TensorFlow [107], Keras-rl2 [108].

The OpenAI Gym API is a defacto environment for developing reinforcement learning algorithms and it also offers a variety of simulated environments (such as Atari, Box2D, Robotics etc.) for training reinforcement learning agents such as our scheduler. The first step is therefore installing all the above 3 dependencies using pip / conda install (depending on your environment):

```
!pip install tensorflow==2.3.1 gym keras keras-rls
```

Upon installing the dependencies, we import the necessary libraries from those dependencies to support the creation of the environment; such as gym spaces which allows us to define the actions and current state within our MECRAN environment:

```
import random
import numpy as np
import matplotlib.pyplot as plt
from gym import Env
from gym.spaces import Discrete, Box


class MECRANSchedulingEnv(Env): # by passing Env to the class, we are
    inheriting methods and properties from OpenAI gym Environment class
    i.e. Env this is an example of Python class inheritance
    def __init__(self):... # initializing actions, observations space,

    def binary_search(self, baseStationList, target):...

    def calculateNodalLatency(self):... # estimate latency on the list
        of base stations provided by the DPR

    def step(self, action):... # defines what we do whenever we take a
        step within the environment and how we treat actions

    def render_model(model):...

    def reset(self):... # reset our env after each training run / episode
```

Env allows us to build our MECRAN environment and we define our actions and optimal latency using the *Discrete* and *Box* spaces. We build and initialize an environment class with 6 important functions and the aforementioned properties as well. The first function is the initialization function:

```
   def __init__(self):
       self.action_space = Discrete(3) # 0, 1, 2
       self.observation_space = Box(low=np.array([0]),
           high=np.array([100])) self.requiredLatency = 10
       self.listOfBaseStationsToUserLatencies = listOfBaseStationsFromDPR
       self.state = self.observation_space
       self.scheduling_length = 5
```

In the initialization function we initialize our actions, observation space, required

latency, scheduling length and get a list of eligible base stations from DPR. Our actions ($A_t - 1$) are three values (0,1,2): 0 - 1 = -1 ; 1 - 1 = 0; 2 - 1 = 1. This can be interpreted as 0 - choose a closer base station; 1 - stay with current base station; 2 - choose a further base station.

The binary search function allows the scheduler to quickly determine the optimal range of base stations to find or explore the optimal latency required. Assuming the DPR has provided a list of 100 base stations, the task scheduler has a time limit to find the optimal base station and therefore introducing binary search here helps the task scheduler to significantly narrow down the list for exploration:

```
def binary_search(self, baseStationList, target):
    lower = 0
    upper = len(baseStationList)
    while lower < upper:
        x = lower + (upper - lower) // 2
        val = baseStationList[x]
        if target == val:
            return x
        elif target > val:
            if lower == x:
                break
            lower = x
        elif target < val:
            upper = x
```

The final function worth explaining is the step function, which is what runs whenever the task scheduler takes a step within the MECRAN environment. This function starts of by taking the required latency and applying binary search to determine the list of optimal base stations to explore. This function also allows the agent (scheduler) to take actions and it calculates the rewards of the actions taken within the limited time allocated:

```
def step(self, action):
    if self.state > self.requiredLatency:
    self.state =
        self.binary_search(self.listOfBaseStationToUserLatencies,
        self.requiredLatency)
    else:
```

```
            self.state =
                self.binary_search(self.listOfBaseStationToUserLatencies,
                self.state)

        self.state += action -1
        # Reduce scheduling length by 1 second
        self.scheduling_length -= 1

        if self.state < 0:
           self.state = 0
        elif self.state >= len(self.listOfBaseStationToUserLatencies) - 1:
           self.state = len(self.listOfBaseStationToUserLatencies) - 1

        # Calculate reward
        if self.listOfBaseStationToUserLatencies[self.state] <=
            self.requiredLatency:
            reward = 1
        else:
            reward = -1

        # Check if Task Scheduler is done
        if self.scheduling_length <=0:
            done = True
        else:
            done = False

        # Return step information
        return self.state, reward, done, info
```

6.2.3 Building and Train the Deep Q-network (DQN) model using Tensor Flow and Keras

Having created our MECRAN environment, we use the environment to build our DQN model using sequential and dense fully connected layers within our neural network. As with the DPR model, we will be using the *Adaptive Moment Estimation (Adam)* optimizer as a main method for stochastic optimization in the task scheduler model. We also use the *Rectified Linear (ReLu)* activation function for the first 2 layers and a linear activation function for the last layer.

It is important to explain and justify some of the hyperparameters used in the task scheduler. In Keras there are two ways to build models; *sequential* and *functional*. Sequential was chosen as its API allows a developer to build models layer-by-layer. This is extremely beneficial for this research work as this capability arms the researcher to design a unique model for a unique problem. Our scheduler needs to process and transform its input (i.e. states and actions) into a reliable output (base station to schedule cloud-native application). In neural networks, hidden layers are introduced to manage this transformation. Therefore dense layers with a rectified linear (ReLu) activation function are introduced with 24 neurons each matching the number of nodes in the RAN.

A ReLu activation function is a simple, commonly used activation function and has been a reliable feature in facilitating predictions for the task scheduler as it converts the inputs into a minimum of zero before multiplying them by the weights and summing them together. A linear activation function is used in the output layer as we are trying to predict a quantity as output. An optimizer is finally needed to derive the value of the weights that best minimize the error when mapping inputs to outputs for better prediction accuracy. The Adam optimizer is used here as well mainly because of its faster computation capabilities and it also requires few tuning parameters. A *Mean Absolute Error* (MAE) loss function is used to determine the magnitude of error between the predicted and actual value as it is the distance between the actual data point and fitted line.

Here is how we build our model:

```
from rl.callbacks import ModelIntervalCheckpoint, FileLogger
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

import tensorflow as tf

states = env.observation_space.shape
actions = env.action_space.n

def MECRAN_scheduler(states, actions):
    model = Sequential()
    model.add(Dense(24, activation='relu', input_shape = states))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model

model = MECRAN_scheduler(states, actions)
model.summary() # to visual the model
```

We now go ahead and train our model using our DQN agent. We have employed the *BoltzmannQPolicy* because whilst exploring, we would also like to explore all the information that exists in the Q values of our network. BoltzmannQPolicy uses *Softmax* over the existing networks estimated value for each action, therefore actions taken by the scheduler has a weighted probability attached to it, which is advantageous long-term. We do our training over a 100,000 steps, 100 full episodes and a learning rate of 0.001:

```
from rl.agents import DQNAgent
from rl.policy import BoltzmannQPolicy
from rl.memory import SequentialMemory # allow us to maintain memory
    efficiently

def MECRAN_agent(model, actions):
    policy = BoltzmannQPolicy()
    memory = SequentialMemory(limit=50000, window_length=1)
    dqn = DQNAgent(model = model, memory = memory, policy = policy,
        nb_actions = actions,
                   nb_steps_warmup=10, target_model_update=1e-2)
    return dqn
```

```
# We use our dqn agent to train our model
dqn = MECRAN_agent(model, actions)
dqn.compile(Adam(learning_rate=1e-3), metrics=['mae']) # learning rate
    of 0.001

dqn.fit(env, nb_steps=100000, visualize=False, verbose=1)

# let's print out the trained model figures
scores = dqn.test(env, nb_episodes=100, visualize=False)
print("Mean Reward: " + str(np.mean(scores.history['episode_reward'])))
    # print mean reward

Visualise model training history
visualiseModelTrainingHistory(history)
print(scores.history.keys())

print(scores.history['episode_reward'])

print(scores.history)
```

With regards to the training dataset for our DQN model, we use the reinforcement learning capabilities of the OpenAI Gym library to model and synthesize data on which nodes (with available resources and appropriate network traffic status) within close proximity to a mobile user will yield the lowest latency, based on the latency required by the specific cloud-native application in question. The inputs of the reinforcement learning model are based on work done in chapters 4 and 5. This dataset generated is then used to train the deep learning model.

6.3 Task Scheduler Model Validation & Results

Using an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, with GPU: NVIDIA GeForce GTX 1080), we trained and tested our task scheduling model over a number of a learning rates across 100,000 steps. This means it took 10,000 steps per interval / episode, learning how to schedule efficiently to achieve an optimal latency. This is based on 2 fully connected dense layers with 24 units each and a final dense layer that has 3 units. Based on our setup, our agent gets a reward of 1 if it makes a good and -1 if it makes a bad decision. It also has 5 secs to make a scheduling decision, which means the maximum reward score an agent can have within the limited time period is 5 rewards and so far we are seeing a mean reward of approximately 4.3.

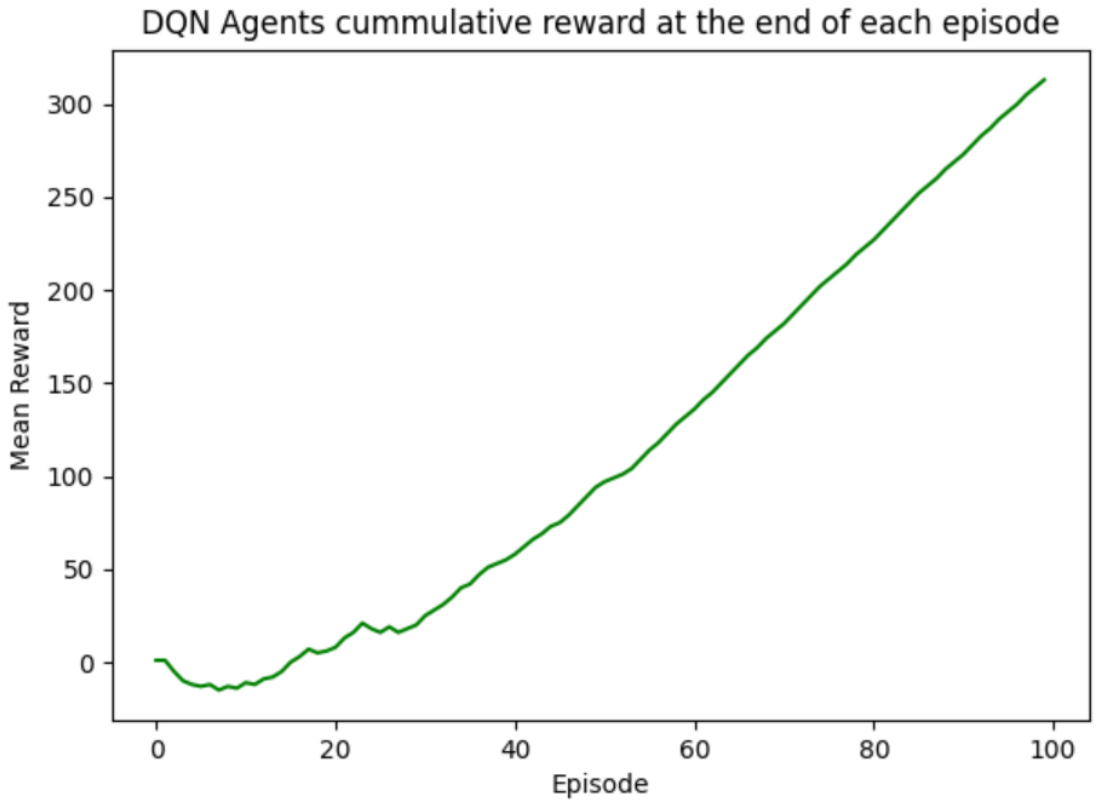

Figure 6.4: Task Scheduler DQN model cumulative rewards over 100 episodes using a learning rate of 0.001

Within the 5 seconds, the scheduler could make 5 good decisions or 5 bad decisions or a mixture of both within that same time period. In Fig. 6.4 we observe

the cumulative sum of average rewards based on the decisions made per episode. As part of its learning it can be observed how the scheduler made some bad decisions in the first 15 episodes yielding poor rewards. We can also observe how after the 15th episode, the scheduler having learnt from its experience started to make better decisions and reaping positively high rewards until the last episode. For the most part, the DQN agent exploits what it already knows based on current estimated value to get the most rewards. There are a few scenarios such as seen in episodes 20 and 25 where the DQN agent also tries an exploration strategy. Here the DQN agent makes risky decisions (different from its current experience) as a way of gain new knowledge / experience. Figures 6.5 and 6.6 supporting Fig. 6.4, highlight the mean rewards and mean losses respectively per episode.

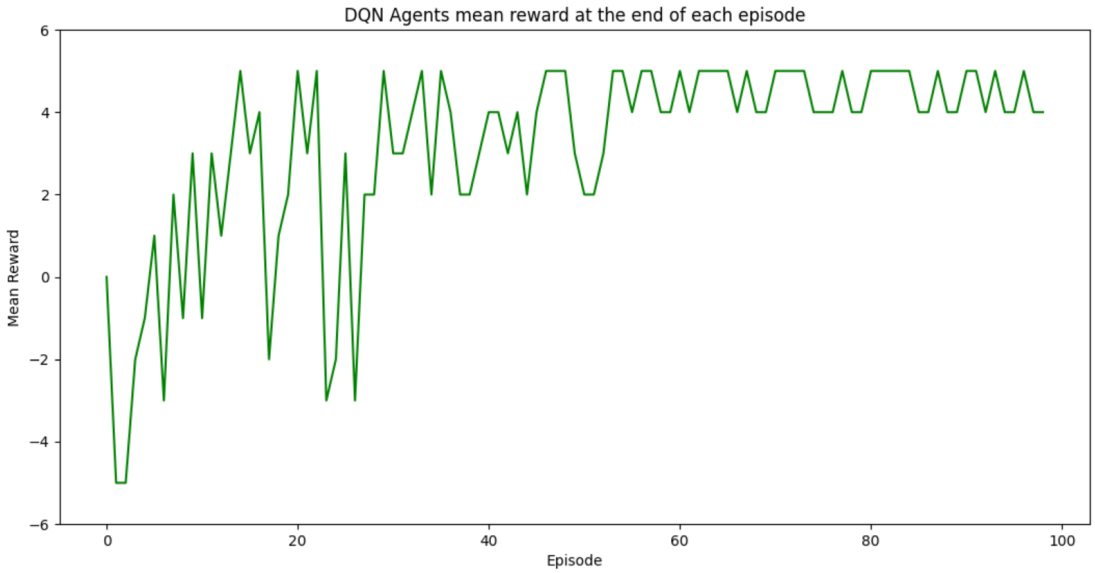

Figure 6.5: Task Scheduler DQN model mean rewards per episode

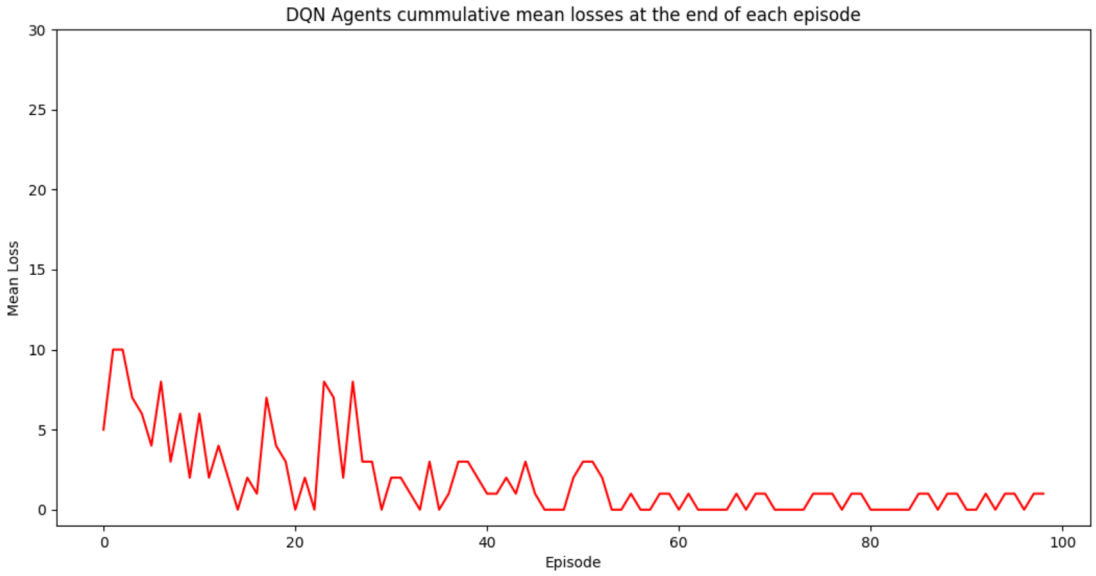

Figure 6.6: Task Scheduler DQN model mean losses per episode

Learning and error function optimization for accurate scheduling can be observed from Fig. 6.7 to Fig. 6.14. In order to identify the best learning rate for our DPR model, a number of learning rates spanning the high, middle and low end learning rate spectrum (usually between 0.0 and 1.0) have been tested iteratively: 1.0, 0.1, 0.01, 0.001, 0.0001, 1e-05, 1e-06 and 1e-07. Testing a number of learning rates helped to tune the model as a way to enable the DQN agent to make optimal overall long-term reward decisions based on experience, in order to minimize the estimated long-term losses. This means losses in some episodes may not necessarily be bad if that compromise was made for the DQN agent to reap maximum reward long-term.

The larger learning rates such as 1.0, 0.1 and 0.01 (in figures 6.7, 6.8 and 6.9 respectively) are seen to experience large learning fluctuations, converged to a sub-optimal solution relatively fast.

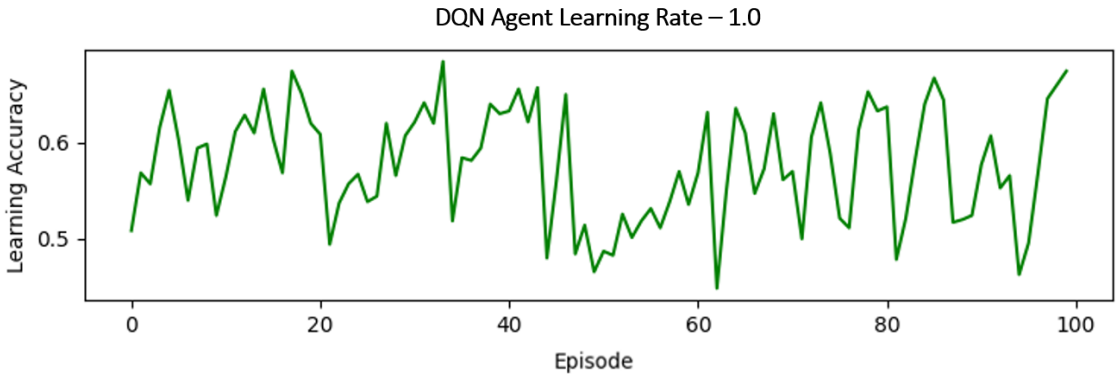

Figure 6.7: Task Scheduler DQN agent learning at 1.0 rate over 100 episodes

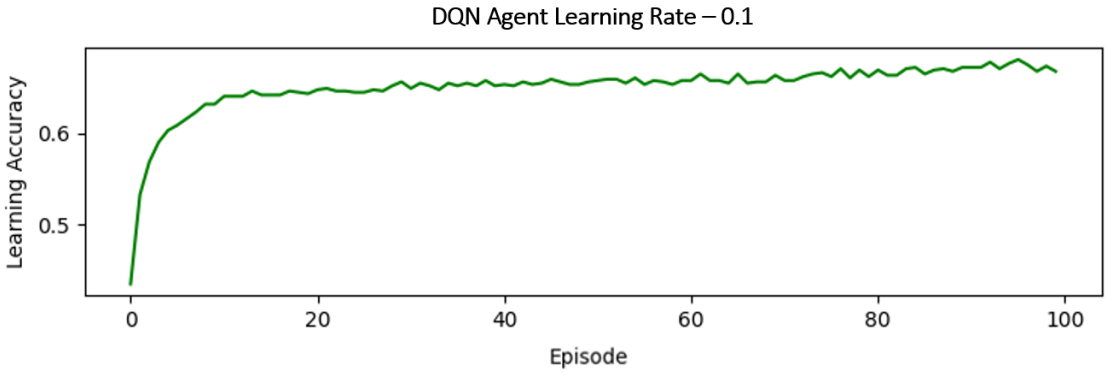

Figure 6.8: Task Scheduler DQN agent learning at 0.1 rate over 100 episodes

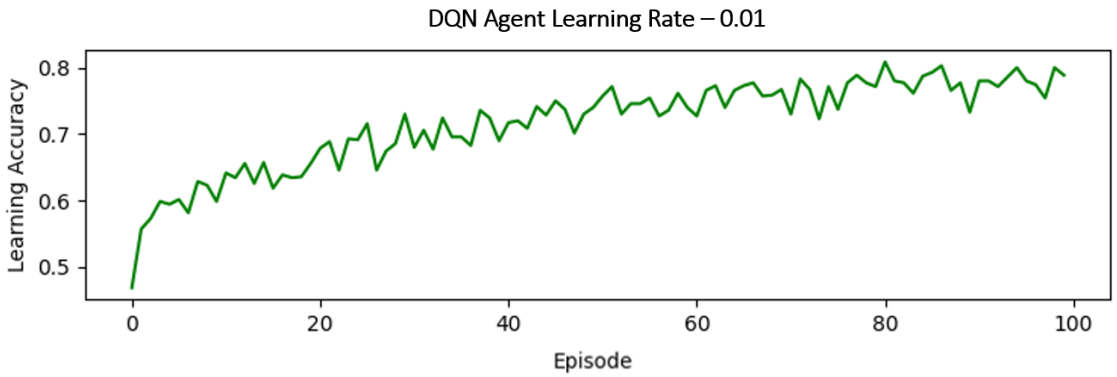

Figure 6.9: Task Scheduler DQN agent learning at 0.01 rate over 100 episodes

The highest learning accuracy was achieved with a learning rate of 0.001 as in Fig. 6.10. The model learns better with learning rate 0.01 and learns best with learning rate 0.001. Learning accuracy starts to decline with learning rate 0.0001 (as in Fig. 6.11) which initiates the lower end of the learning rates spectrum.

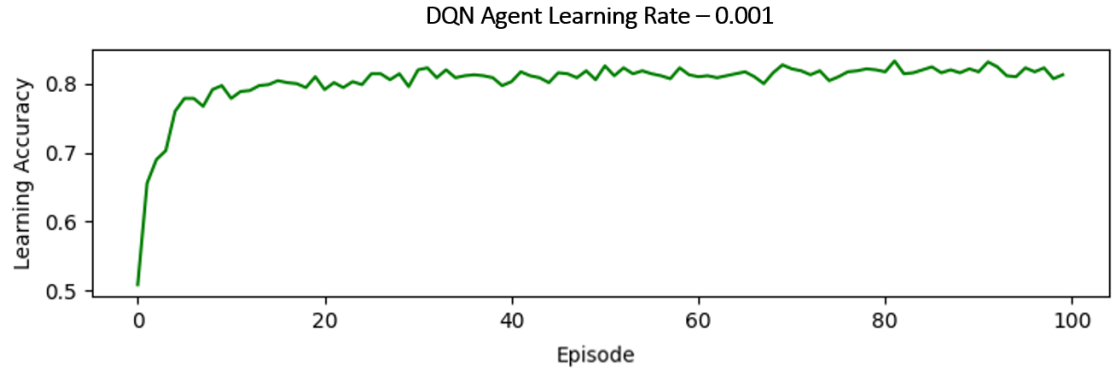

Figure 6.10: Task Scheduler DQN agent learning at 0.001 rate over 100 episodes

On the other hand with very small learning rates (such as 1e-05, 1e-06, 1e-07), because smaller updates are made to the weights in each iteration, they require larger epochs to train and learn. Very small learning rates may experience performance issues as a results of the large amount of computations that are required. We see less promising results with these learning rates as they produce the highest error margin as seen in figures 6.11 to 6.14.

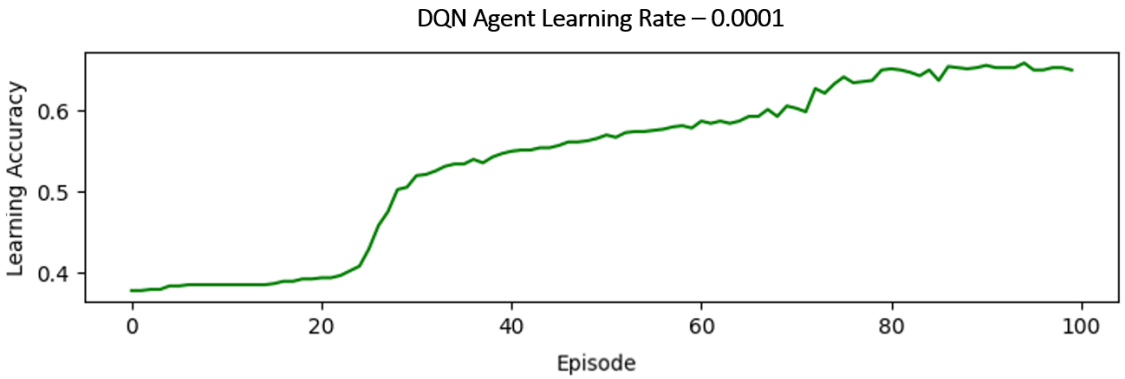

Figure 6.11: Task Scheduler DQN agent learning at 0.0001 rate over 100 episodes

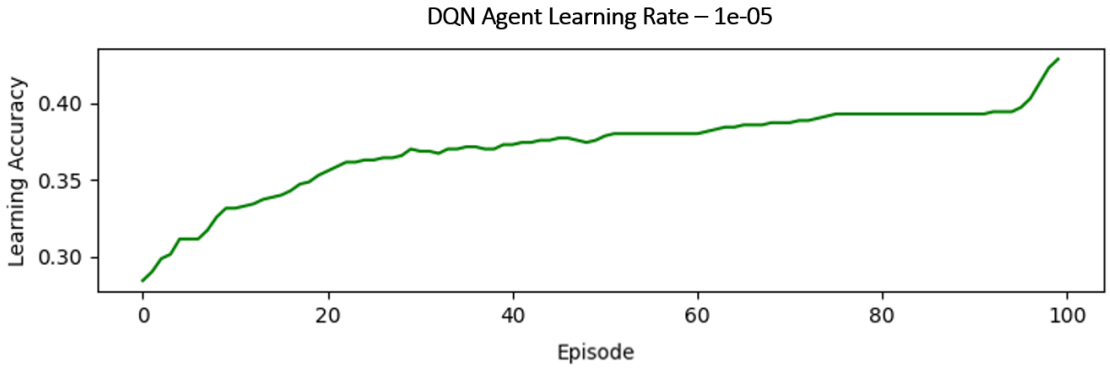

Figure 6.12: Task Scheduler DQN agent learning at 1e-05 rate over 100 episodes

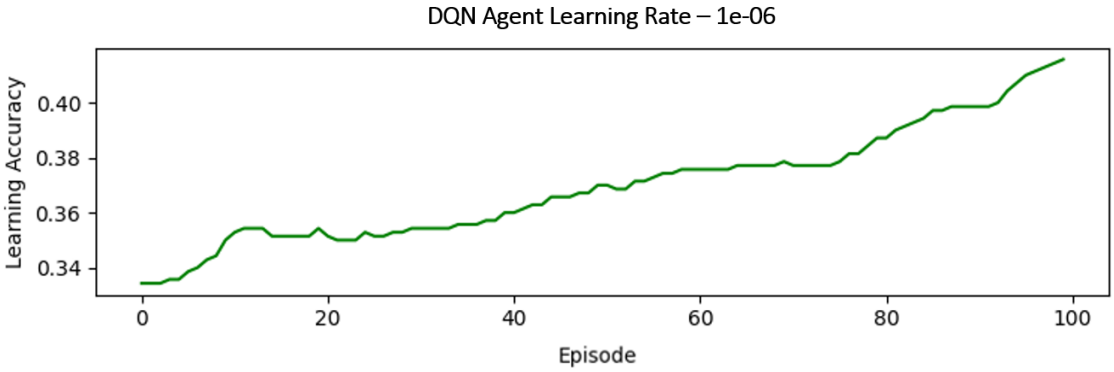

Figure 6.13: Task Scheduler DQN agent learning at 1e-06 rate over 100 episodes

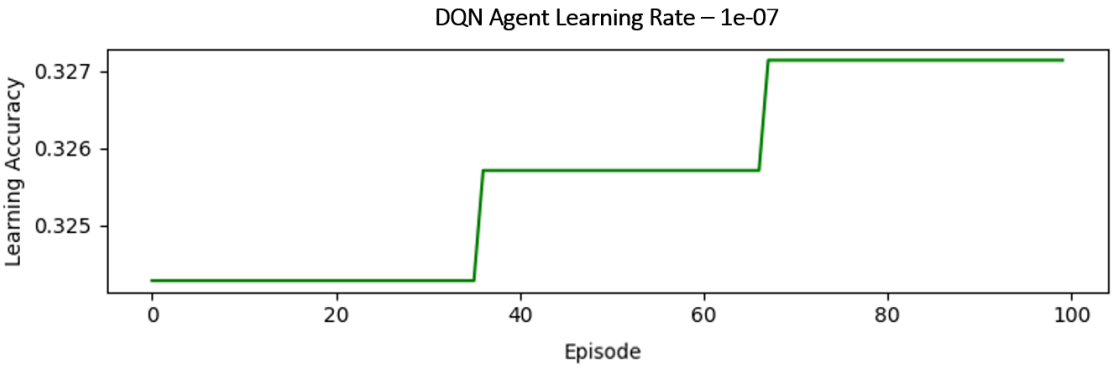

Figure 6.14: Task Scheduler DQN agent learning at 1e-07 rate over 100 episodes

At the end of the training we see an average loss of approximately 1.02, a MEA of about 3.03 and the training took 654.923 seconds. As indicated the scheduler has 5 seconds to explore and make a scheduling decision - which amounts to 1 second to explore 5 (5 steps) base stations for the optimal latency before making a decision.

6.4 Summary

We have been able to successfully build an effective and productive task scheduler using deep reinforcement learning. This task scheduling routine collaborates with the data packets routing routine to be able to find the best base station to temporarily host a cloud-native application, at the edge of the network in order at the application's required performance latency.

Our task scheduler is built with Python 3 using TensorFlow, OpenAI Gym and Keras-rl2. We were able to create a reinforcement learning scheduling environment from Open AI Gym. We then built a deep learning model and trained this model using a Deep Q-network (DQN) reinforcement learning agent for scheduling. Our scheduler narrows down the optimal base station list using binary search. The scheduler is able to schedule by dynamically adjusting the location (base station node / server) where an cloud-native application can be hosted to meet its latency requirement, considering the distance the user and base station and the network resources available.

The next chapter will be applying the complete MECRAN setup to 2 use cases to demonstrate the value of this research with real-world use cases in industry.

7

Validation: Real World Application Use Cases

Contents

7.1 Introduction & Purpose

The aim of this chapter is to apply the research work planned in chapter 3 and implemented in chapters 4, 5 and 6, to solve real world problems in industry.

Our road map in Fig. 3.3 highlights the 3 main use cases in scope - smart railway and smart factory. All these use cases have needs which can be solved by our researched solution MECRAN.

As part of testing this work in industry, a 6 weeks' internship by the Internship Office of the University of Oxford was embarked on to work alongside industry

leaders in Network Rail [77] in September 2020. During this internship, this use case was aligned with Europe's long-term railway technology, research and development strategy called *Shift2Rail* [110]. Shift2Rail is funded jointly by European Union members and a European Commission research fund called *Horizon 2020* [110]. Network Rail is one of the founding partners of Shift2Rail and Network Rail also plays an active role in four of its five Innovation Programme.

The rest of this chapter will expand on how MECRAN attempts to solve latency issues in our 2 main use cases.

7.2 Smart Railway

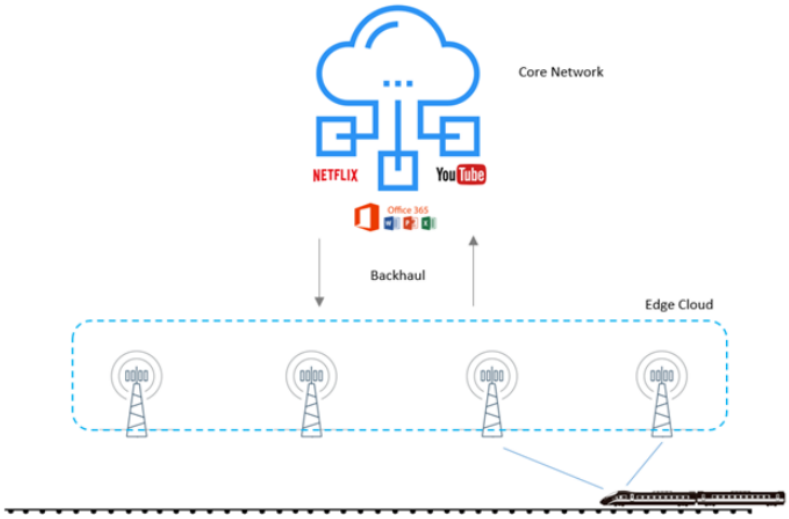

Figure 7.1: Smart Railway Use Case.

Emerging digital applications for future railways such as smart train technology, unmanned aerial vehicles (drone technology) for site inspection and rail robots for track inspection, have strict non-comprisable requirements for extremely low latency (<10 milliseconds) and high bandwidth for performance [3][5]. Unfortunately, traditional cloud computing and radio access network architectures do not sufficiently address the need for low latency and high bandwidth, that are crucial to the performance of these cloud-native applications [4].

There is therefore, a need to enhance the mobile communication network architecture to allow mission and time critical cloud-native applications to flexibly

run on seemingly 'unlimited' and reliable compute, network and storage, to meet these performance requirements. 'Unlimited' in the sense that applications have unconstrained access to the necessary network resources.

Emerging smart train applications require low latency to operate and we apply MECRAN to deliver cloud-native applications on smart trains at low latency.

Railway Scenario

To apply MECRAN to the railway scenario, the following assumptions were made. We consider Great Britain train lines from sectors Mainline, Regional, Suburban, Urban, Metro and Freight due to diversity in rolling stock, speed of travel, track geometry and surrounding environment. These trains are equipped with safety critical digital applications, and traveling at a maximum speed of 124 miles per hour, from *Point A* to *Point B*, with evolving passenger headcount numbers, depending on whether it is a peak or non-peak travel time. The cloud-native railway applications are categorised into Critical, Performance Enhancing and Business applications.

Critical applications (such as European Train Control System (ETCS), Computer Based Train Control (CBTC), signaling applications, Critical Video Applications, Critical Voice Applications, Critical Data Applications) are necessary for train movement. Performance enhancing applications such as environmental sensors, drones, rail robots and telemetry improve the performance of the railway operation, whilst business applications (including customer connectivity) support the railway business operation. In our scenario, drones and rail robots required latency of 10 ms or less, whilst the vast majority of the other applications required latency between 50 – 100 milliseconds.

Even though we consider a public network for 4G, we primarily consider 5G network slicing, such that an isolated end-to-end network slice is dedicated to a rail operator. Finally, we assume that there is enough cell site coverage between the two location points.

7.2.1 Approach

Network Rail shared a dataset (see a preview of this data in Fig. 7.2) describing the railway cloud-native applications in scope. The fields in the dataset includes Applications, Type of Application, Content Type, Symmetry Up, Symmetry Down, Latency_Min (ms), Latency_Max (ms), Data Rate_Min (kbps), Data Rate_Max (kbps), Setup Time (secs), Operational Speed of Device (km/hr).

Applications	Type of Application	Content Type	Symmetry Up	Symmetry Down	Latency_Min (ms)	Latency_Max (ms)
Multi-train voice coms (train-to-train) for train staff, drivers & ground users	Critical	Bi-directional Voice	50	50	10	100
Multi-train voice coms (train-to-train) for train staff, drivers & ground users	Critical	Uni-directional Voice	0	100	10	100
Multi-train data coms (train-to-train) for train staff, drivers & ground users	Critical	Bi-directional Data	50	50	10	100
Multi-train data coms (train-to-train) for train staff, drivers & ground users	Critical	Uni-directional Data	0	100	10	100
Voice recording	Critical	Uni-directional	0	100	100	500
Retreival of voice recording	Critical	Uni -directional	100	0	100	500
Data recording	Critical	Uni -directional	0	100	100	500
Retreival of data recording	Critical	Uni -directional	100	0	100	500
Train integrity monitoring data communication	Critical	Bi-directional Data	50	50	0	100
Real-time video uplink	Performance	Uni-directional	20	80	10	100
Real-time video downlink	Performance	Uni-directional	80	20	10	100
Real-time video UL / DL	Performance	Bi-directional	50	50	10	100
Real-time video uplink	Critical	Uni-directional	20	80	10	100
Real-time video downlink	Critical	Uni-directional	80	20	10	100
Real-time video UL / DL	Critical	Bi-directional	50	50	10	100
Communications with remote maintenance vehicles (rail robots)	Performance	Bi-directional	20	80	0	10
Communications with drones	Performance	Bi-directional	20	80	0	10
Communication for building and facilities security & management	Performance	Bi-directional	50	50	100	500
Wireless data communications for trackside staff	Performance	Bi-directional	50	50	100	500
Wireless real-time internet for passengers	Business	Bi-directional	50	50	10	100
Wireless non-real-time internet for passengers	Business	Bi-directional	20	80	100	500
Wireless real-time internet for passengers on platforms	Business	Bi-directional	50	50	10	100
Wireless non-real-time internet for passengers on platforms	Business	Bi-directional	20	80	100	500
Wireless real-time internet for cargo	Business	Bi-directional	80	20	10	100
Wireless non-real-time internet for cargo	Business	Bi-directional	80	20	100	500

Figure 7.2: Network Rail Data Preview

We combine this dataset from Network Rail with local authority data from the Office of National Statistics [111] and Google Maps. We also simulated 61 base stations to cover the geographic area. As a result we simulated model trains that travel using the Google maps routes via (local authority) cities and towns around England, with cloud-native applications users on board. The railway sectors for our model trains included: *Mainline*, *Region*, *Suburban*, *Urban*, *Metro* and *Freight*.

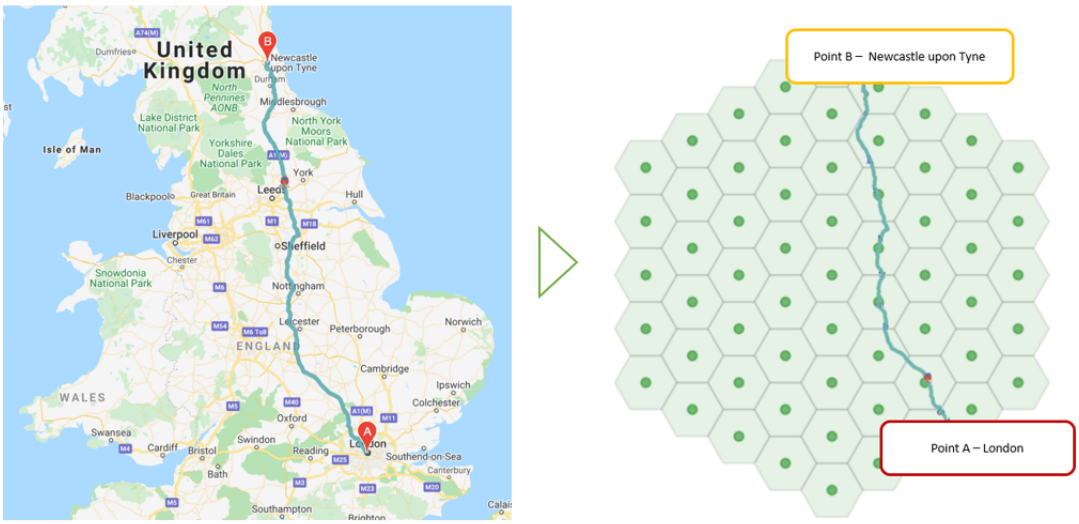

Figure 7.3: Railway Scenario - Smart train travelling from London to Newcastle.

The cloud-native applications in scope included:

- **Critical Applications:** European Train Control System (ETCS) and Computer Based Train Control (CBTC), Signalling Applications, Critical Video Applications, Critical Voice Applications, Critical Data Applications.
- **Performance Enhancing Applications:** Drone, Rail robot, environmental sensors, telemetry.
- **Business Applications:** Various mobile applications such as high definition gaming

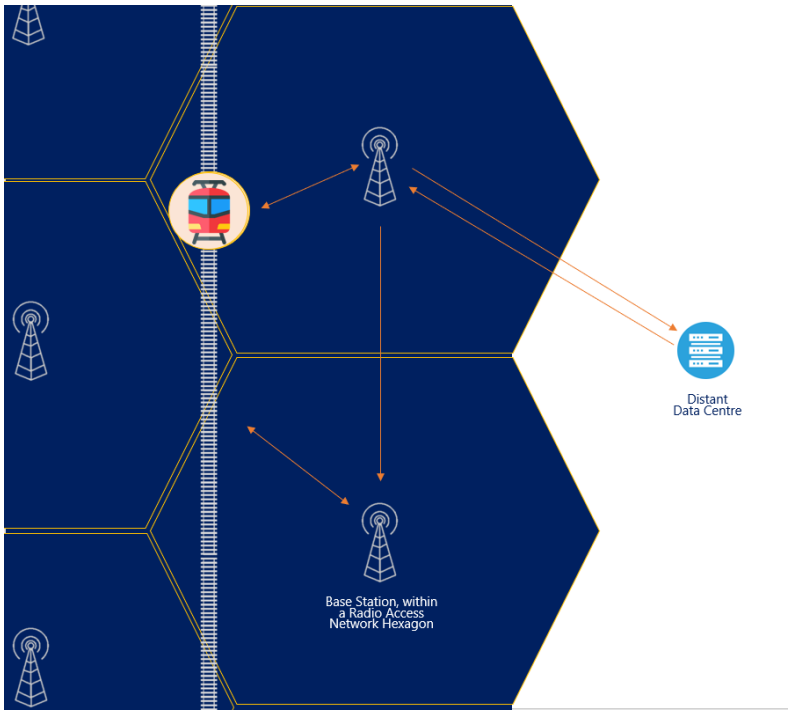

Figure 7.4: Railway Scenario Scheduling.

Based on MECRAN's scenario layer, we address the smart railway need through these three fundamental steps:

1. Based on the cloud-native application requested, MECRAN pings the original distant host server (as shown in Fig. 7.4) to determine latency (at current distance) and compares that with latency required by the task for performance.

2. If required latency is met:

   - MECRAN's user connects with the application from the distant server.
   - Else, MECRAN migrates the cloud-native application to the user's local edge for dynamic scheduling.

3. Dynamic scheduling + Data Packet Routing

7.2.2 Training and Testing MECRAN

Based on the above railway context, we built a model / virtual train that travelled around England (as seen in Fig. 7.3) at approximately 124 miles/hour for 96 hours, while MECRAN performed scheduling and trained the model. Throughout this period of travel, users requested to use a number of the applications in scope and where necessary MECRAN performed active scheduling to maintain high quality of service to the users. The data MECRAN trained from described:

1. The cloud-native applications (such as latency, bandwidth, priority).
2. The train (such traffic, speed, traffic density, time and location).
3. The edge and RAN resources (such as available base stations, base station capacity, distance, weather temperature, user equipment density).
4. Finally, user device (such as operational speed, type of device, setup time).

Overtime, MECRAN learnt how best to schedule the applications and route data packets within the network, at low latency.

7.2.3 Results

After successfully training MECRAN, it was then validated on *single track*, *double track* and *quadruple track* trains from Oxford (United Kingdom) to London (United Kingdom). We also benchmarked the performance of MECRAN against the *Packet Fair Queuing* (PFQ) [112][113] algorithm, a well-known computer networking methodology for scheduling data packets. Below is some context around why this model was chosen.

In wired networks, *Fluid Fair Queueing* (FFQ) model stands out as one of the most popular and effective packet scheduling algorithms [113]. In this algorithm, data packets are modeled as fluid flowing through a medium over a particular time period, with each flow assigned some weight. A packet level implementation of FFQ is the *Weighted Fair Queueing* (WFQ) model [114][115]. Our work is based on an

adaptation of WFQ specifically for wireless networks [113] as this model addresses two major drawbacks in WFQ (and FFQ) at large. For simplicity in language, we continue to refer to this model as Packet Fair Queuing (PFQ) or benchmark algorithm in this thesis. The drawbacks this model address are: (1) bursty channel errors and (2) location-dependent channel capacity and errors.

As part of validating our work, we compared application performance in 3 scenarios: when there is no active scheduling also referred to as the worst case (WC); scheduling based on the benchmark algorithm (Packet Fair Queuing) and scheduling using MECRAN (the best case). Where "application performance" here represents how often an application was successfully scheduled by MECRAN such that it met its required latency for performance. For example if a drone cloud-native application required 9 milliseconds of latency, $x\%$ of the time MECRAN was able to schedule the application on the right base station to meet this latency requirement. Below are the results of our tests.

Application performance on single train tracks

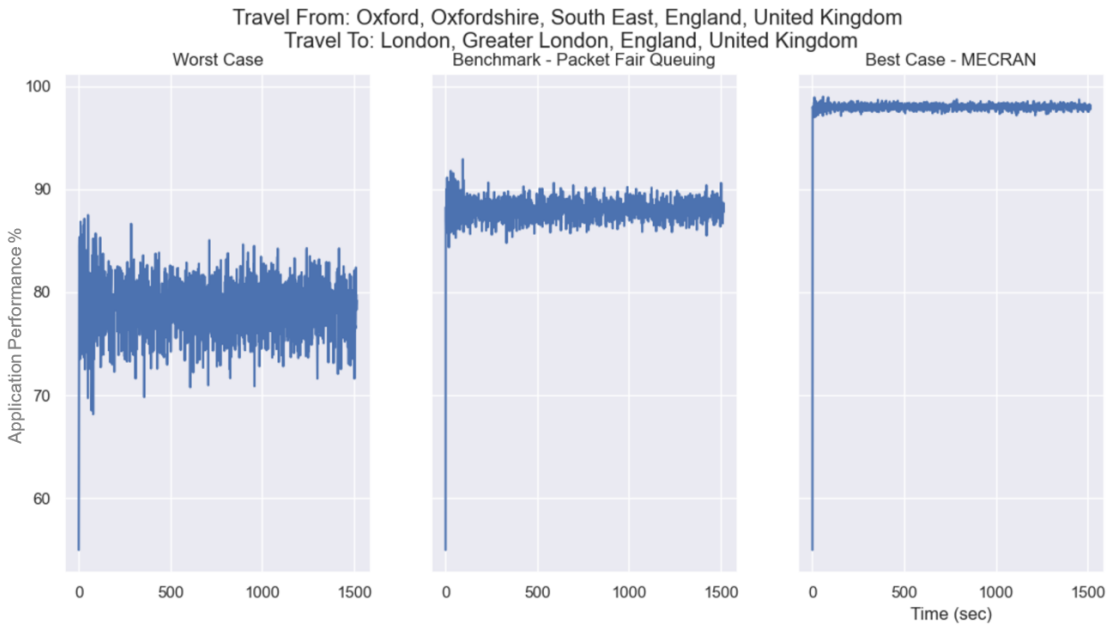

Figure 7.5: Single Track Performance Comparison - This shows how often cloud-native applications were able to meet their latency requirement either through no active scheduling (worst case), Packet Fair Queuing (benchmark) or MECRAN (best case).

Figure 7.5 begins with the worst case scenario where no active scheduling is done on the journey from Oxford to London. Here, we observe that cloud-native applications were able to meet their latency requirements between 68% to 96% of the time and averaging (weighted) at approximately 77%. This is unacceptable for time and mission critical applications [3][4][5][16]. When the same test data was passed through the Packet Fair Queuing algorithm, applications were able to meet their latency requirements 85% to 91.5% of the time, averaging (weighted) approximately at about 87%. When the same test data was passed through MECRAN however, applications were able to meet their latency requirements 97.5% to 98.5% of the time, with a weighted average of about 98%.

Application performance on double train tracks

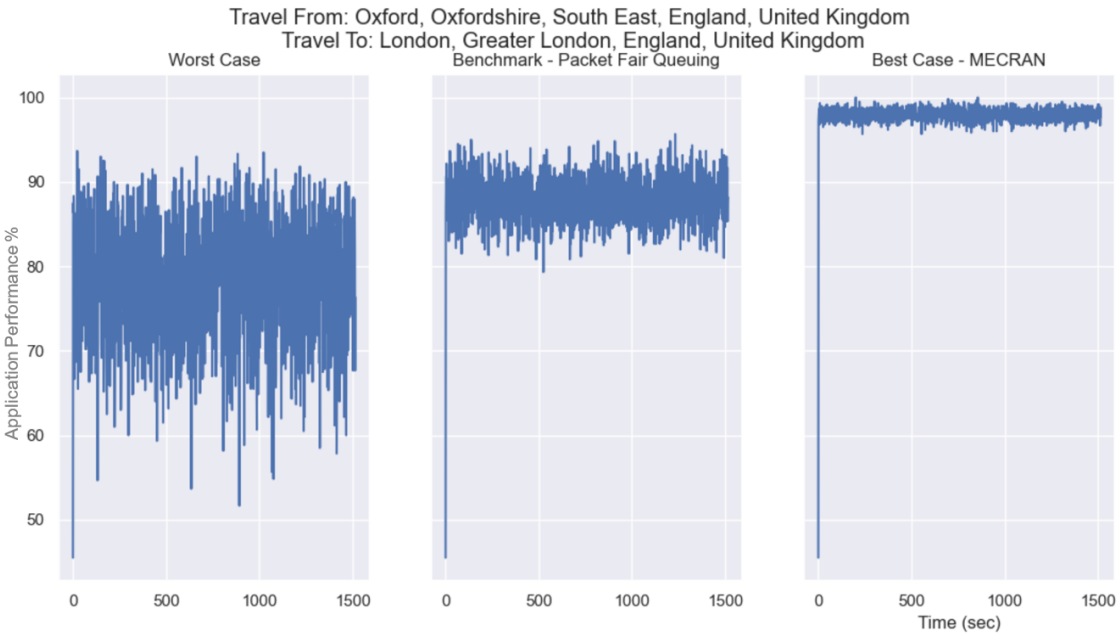

Figure 7.6: Double Track Performance Comparison - This shows how often cloud-native applications were able to meet their latency requirement either through no active scheduling (worst case), Packet Fair Queuing (benchmark) or MECRAN (best case).

In the case of the double track trains, Fig. 7.6 begins with the worst case scenario where no active scheduling is done on the journey from Oxford to London. Here, we observe that cloud-native applications were able to meet their latency requirements between 53% to 93% of the time and averaging (weighted) at approximately 75%. When the same test data was passed through the Packet Fair Queuing algorithm, applications were able to meet their latency requirements 79.5% to 93.5% of the time, averaging (weighted) approximately at about 85.4%. When the same test data was passed through MECRAN however, applications were able to meet their latency requirements 96.5% to 99.5% of the time, with a weighted average of about 97.8%.

Application performance on quadruple train tracks

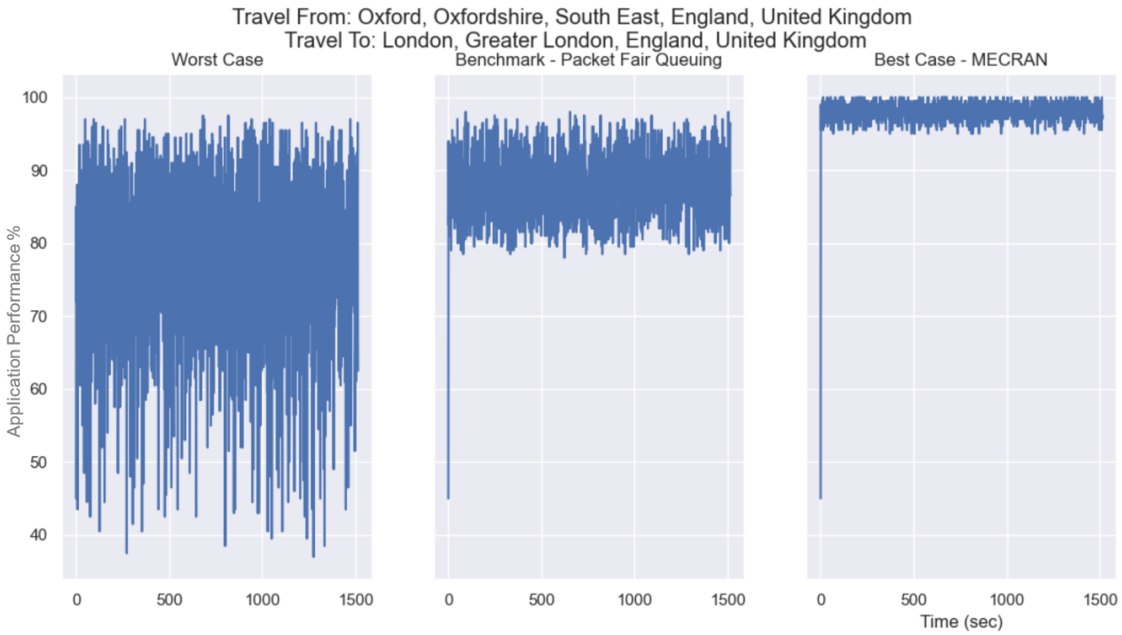

Figure 7.7: Quadruple Track Performance Comparison - This shows how often cloud-native applications were able to meet their latency requirement either through no active scheduling (worst case), Packet Fair Queuing (benchmark) or MECRAN (best case).

In the quadruple track scenario, Fig. 7.7 highlights the application performances here as well. This scenario also begins with the worst case scenario where no active scheduling is done on the journey from Oxford to London. Here, we observe that cloud-native applications were able to meet their latency requirements between 36.7% to 97.5% of the time and averaging (weighted) at approximately 70.8%. When the same test data was passed through the Packet Fair Queuing algorithm, applications were able to meet their latency requirements 78.5% to 98.2% of the time, averaging (weighted) approximately at about 83.7%. When the same test data was passed through MECRAN however, applications were able to meet their latency requirements 95.6% to 99.8% of the time, with a weighted average of about 97.2%.

Best Case Scenario - Application Performance Probability Density Function (PDF) Cumulative Distribution Function (CDF)

Based on the individual worst case, benchmarked and best case scenarios, we analyzed overall application performance distribution across these segments regardless of the train track type to understand the real application performance advantages of the scenarios aforementioned.

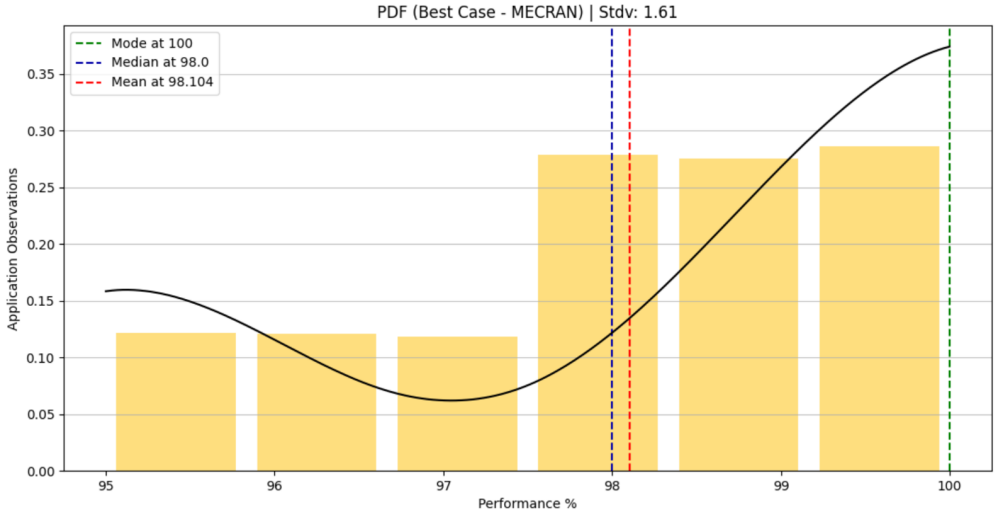

Figure 7.8: Best Case Scenario: Probability Density Function of simulations (Single, Double & Quadruple Tracks), showing the relationship between application performance observations and their probabilities.

Across all application performance in the best case scenario, Fig. 7.8 highlights an encouraging mean performance of approximately 98%. Most of the other application performance are within 1.6 standard deviations from this mean, which is exactly the sort of performance one would want for mission critical and time critical applications. Especially that most applications performed at 100% (mode) - this can also be observed in the best case scenario's cumulative distribution function in Fig. 7.9.

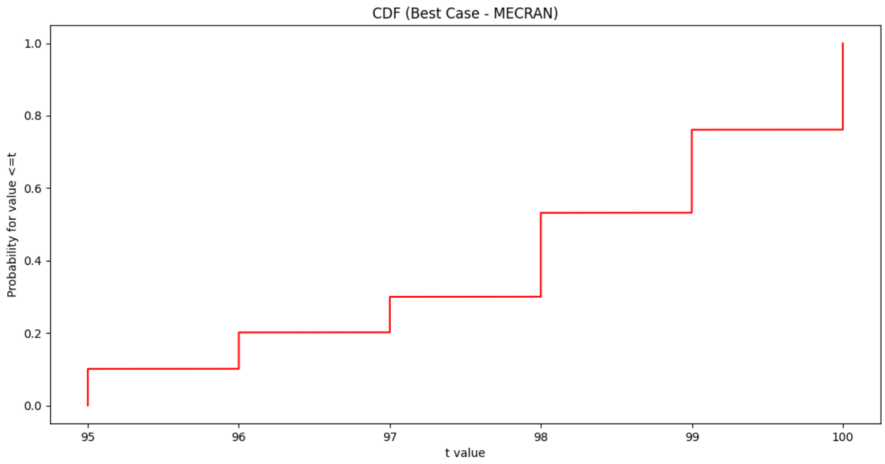

Figure 7.9: Best Case Scenario: Cumulative Distribution Function of simulations (Single, Double & Quadruple Tracks), showing the relationship between application performance observations and their probabilities.

The CDF in Fig. 7.9 describes the distribution of application performances as random variables. It brings to attention that over 80% of applications that were scheduled by MECRAN were able to meet their latency requirements 99% of the time. Concurrently, over 50% of applications that were scheduled by MECRAN were able to meet their latency requirements 98% of the time.

Benchmark Scenario - Application Performance Probability Density Function (PDF) Cumulative Distribution Function (CDF)

Focusing on applications scheduled by the Packet Fair Queuing algorithm in the benchmark case, the PDF in Fig. 7.10 reveals some important highlights. Unlike MECRAN, the mean application performance is approximately 89%. Majority of the other application performances are within 5.8 standard deviations from this mean, which is quite significant decline in performance compared to MECRAN, especially performances on the left side of the mean. Mission critical and time critical applications will struggle to perform at an acceptable quality of service if they were meeting their latency requirements 89% (and below) of the time.

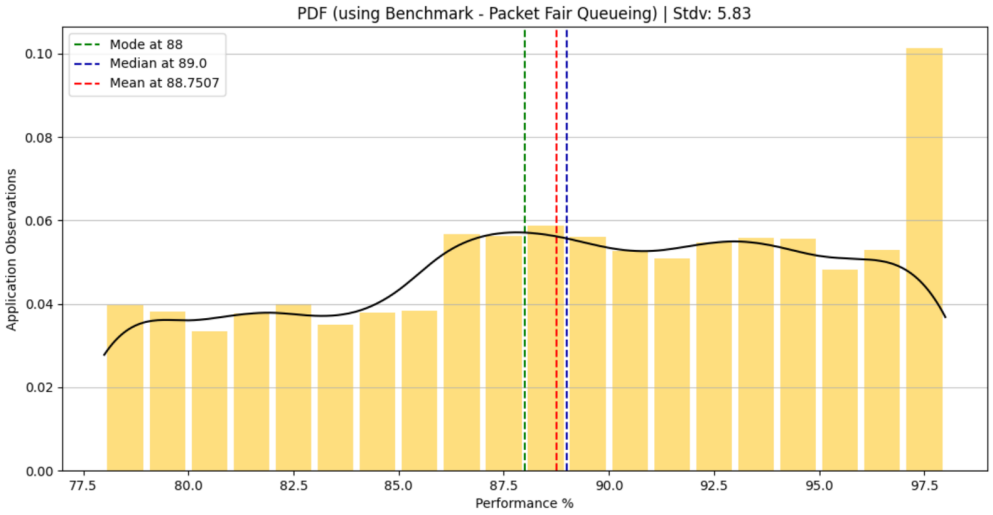

Figure 7.10: Benchmark Scenario: Probability Density Function of simulations (Single, Double & Quadruple Tracks), showing the relationship between application performance observations and their probabilities.

The CDF in Fig. 7.11 describes the distribution of application performances as random variables. It informs that over 80% of applications that were scheduled by MECRAN were able to meet their latency requirements 95% of the time. Over 50% of applications that were scheduled by PFQ were able to meet their latency requirements 89% of the time.

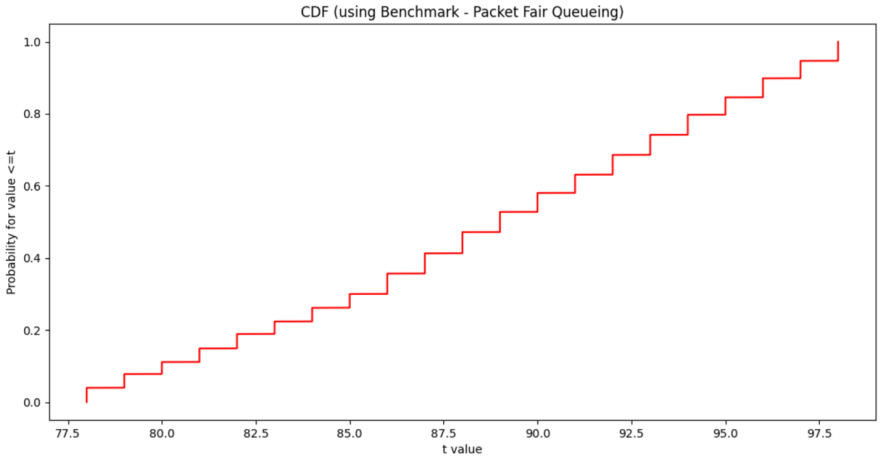

Figure 7.11: Benchmark Case Scenario: Cumulative Distribution function of simulations (Single, Double & Quadruple Tracks), showing the relationship between application performance observations and their probabilities.

Worst Case Scenario - Application Performance Probability Density Function (PDF) Cumulative Distribution Function (CDF)

Last but not least, we analyze performances (as shown in Fig. 7.12) of these same applications with no active scheduling and comparing their current performance with when they were scheduled and managed by MECRAN. Unlike MECRAN, the mean application performance is approximately 67%. Majority of the other application performances are within 17.12 standard deviations from this mean, which is quite significant decline in performance comparing to MECRAN, especially performances on the left side of the mean. Mission critical and time critical applications will not survive with such performances.

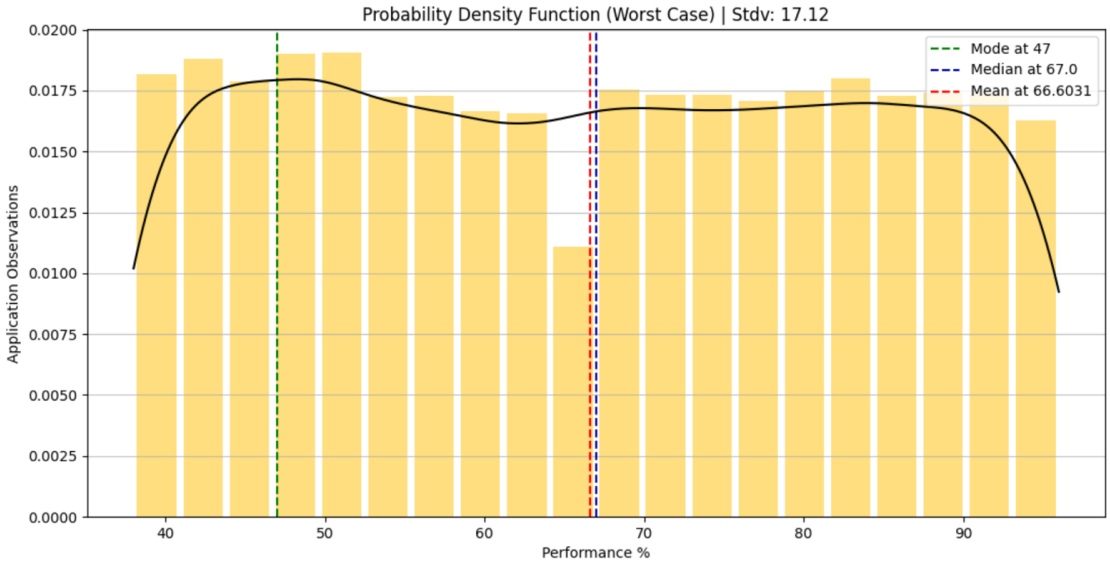

Figure 7.12: Worst Case Scenario: Probability Density Function of simulations (Single, Double & Quadruple Tracks), showing the relationship between application performance observations and their probabilities.

The CDF in Fig. 7.13 describes the distribution of application performances as random variables. It underlines the fact that over 80% of applications that were scheduled by MECRAN were able to meet their latency requirements 81% of the time. It reiterates the fact that over 50% of applications in the worst case scenario were able to meet their latency requirements 67% of the time.

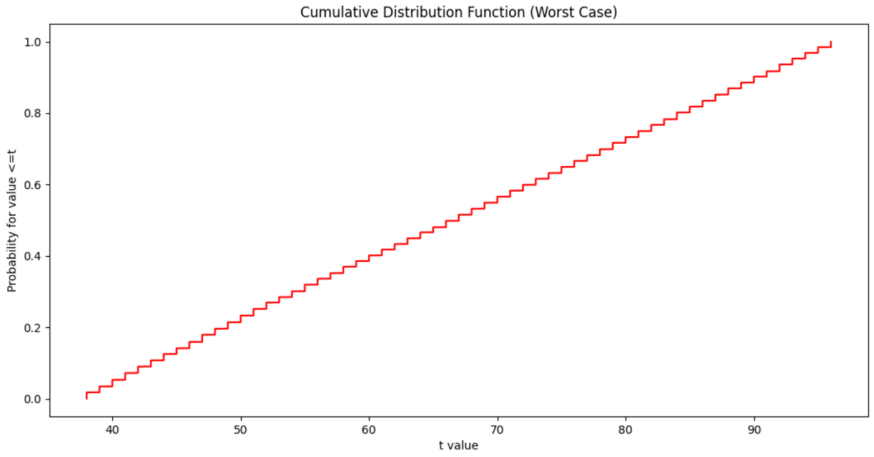

Figure 7.13: Worst Case Scenario: Cumulative Distribution Function of simulations (Single, Double & Quadruple Tracks), showing the relationship between application performance observations and their probabilities.

7.3 Smart Factory

While the use of software in manufacturing is not new, the adoption of smart or emerging technologies such as Augmented Reality and Artificial Intelligence is becoming popular in improving efficiency, as well as reducing the cost of factory operations by minimizing down times and effectively increasing production [116]. These emerging technologies however have strict performance requirements for low latency and high bandwidth [16], that 5G alone cannot meet due to physical distance [4]. Inter-node communication through data exchanges over a physical distance is part of the basic fabric of wireless communication and cloud computing, and therefore the weighted latency introduced by the physical distance of communication severely affects the performance of emerging smart factory services or applications.

In collaboration with Nestle S.A. [78] it was understood that the Swiss headquartered business was exploring various ways of using emerging technologies to deliver value in their factory environment as well as distribution centres. Some of the needs within their factory environment include conducting remote auditing of refrigeration, ammonia, boilers, steam generation, air generation and nitrogen generation. To address this need, a number of technologies had been explored by Nestle theoretically but the augmented reality hardware HoloLens2 [117] is of most interest and is being tested in their factory and distribution centre spaces. While it is showing potential in meeting the aforementioned needs of the factory, the product research team testing the technology experienced high latency (over 70 ms on average) whilst walking through the large factory or distribution centre floors to perform remote auditing. Therefore MECRAN will seek to dynamically host the HoleLens cloud-native application [83] in a server or cell closer to the tester as the tester moves or walks around. This will ensure lower latencies as the HoleLens hardware can interact with a host close by, reducing the distance for exchanging data. Another cause of the high latency was that when a user moves around the factory or distribution centre, the augmented reality device stayed connected to the same wireless checkpoint adding to the high levels of latency when the user moves apart from cell.

Augmented reality service is characteristic of supporting high data rates and performs best under low latency [3]. Like many time and mission critical emerging applications, high latency causes a severe disruption in the quality of service as they require latency of 10 ms or less. This use case is a unique opportunity to test how MECRAN is able to bring low latency to HoloLens in the factory and distribution centre environment in Lausanne, Switzerland.

One of the reasons for choosing this use case is because it posed a different challenge from the railway use case and it also operated a heterogeneous radio access network due to fluctuating network traffic demand. The factory and distribution centre are located in the outskirts of the small town and therefore do not completely benefit from the standard eNodeB base station but rather the factory supplements the radio access network with micro and pico cells [118] as shown in Fig. 7.14. In both manufacturing and supply chain environments, whilst clients and hosts are within the same building scheduling is still important because the factory uses a large number of micro cells (also known as small cells) to target coverage for specific areas as supposed to one large base station covering the entire manufacturing site, where many parts of the site may not need coverage. From that perspective scheduling cloud-native application across multiple micro cells whilst the user is moving is critical.

7.3.1 Approach

Modeling the factory's radio access network was the primary step. The factory had 33 cells of varying capacities across a geographic area size of approximately 4000 m^2 as shown in figure 7.14. Not all areas in the factory has network coverage; some cells are targeted towards specific areas and it is important to dynamically schedule application in such an environment with extremely limited RAN resources.

The modeled RAN (Fig. 7.14) of this hybrid cloud environment was translated into a graph in Fig. 7.15 using the graph translation methods implemented in sub-section 4.3.2. This new graph is consumed by our DPR and task scheduling routines for scheduling. This graph model is used to capture the spatial and temporal features of the RAN for our graph neural network.

7.3.2 Training and testing MECRAN

Nestle is planning to initially deploy 20 HoloLens devices in its factory and distribution centre. For this test however we simulated 100 of these augmented reality devices. In our simulation application users take random walks within the factory over the 6.7 hour (400 minutes) period to train the MECRAN models.

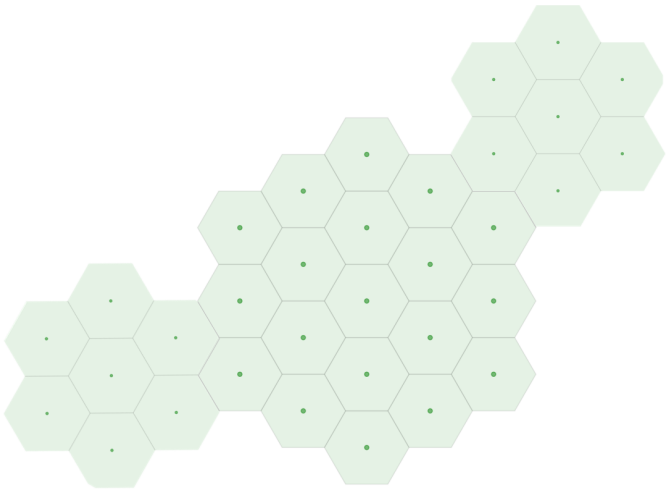

Figure 7.14: Factory's heterogeneous radio access network spread across approximately 1 acre of land.

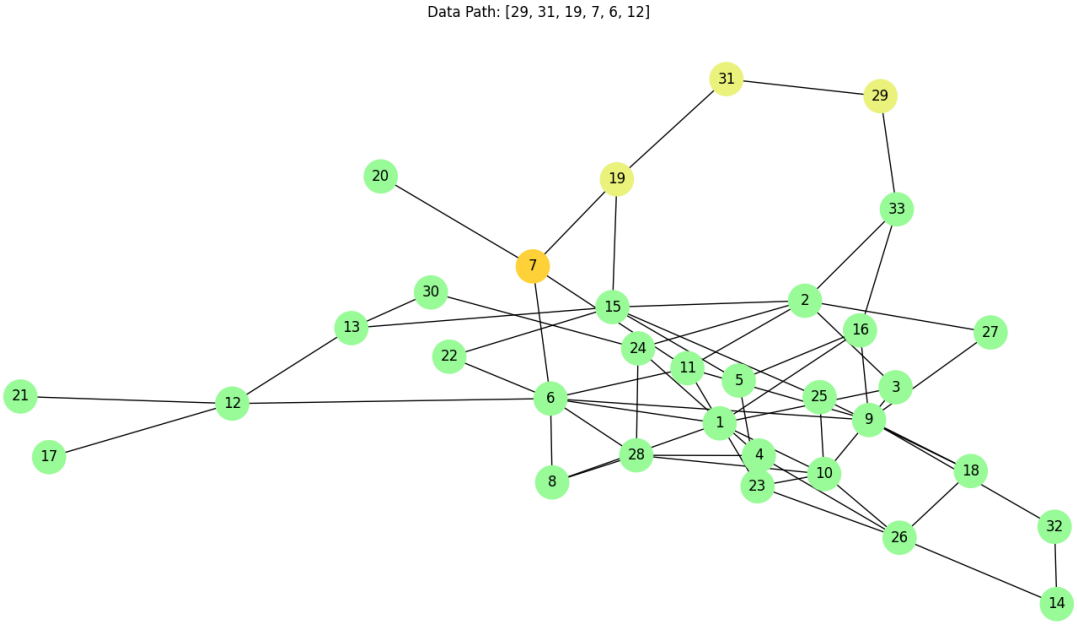

Figure 7.15: Smart Factory RAN translated into a graphical structure featuring nodes (base stations) and edge (connection types).

Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, with GPU: NVIDIA GeForce GTX 1080 was used to train MECRAN models at a learning rate of 0.001. As described in chapter 6, the task scheduling module was trained over 50,000 steps (taking 10,000 steps per interval) and the task scheduler was limited to making a scheduling decision within 10 seconds. The model was tested on 10 full episodes, receiving a mean reward of 9.8 (out of a maximum amount of 10 rewards). The scheduler has 10 seconds to explore and make a scheduling decision - which amounts to 1 second to explore 10 (10 steps) cells for the optimal latency before making a decision. With more training and tuning the MECRAN has the potential of outperforming itself further.

Smart technology in manufacturing has many benefits such as to increase production, eliminate bottlenecks in the production process, reduce production lead times and to optimise production schedules. MECRAN is therefore aligned to activate the full potential of smart technology by enabling them to meet their latency performance requirement through scheduling.

7.4 Results

In a remote smart factory environment that benefits from a heterogeneous radio access network, augmented reality applications experienced high latency (about 70 milliseconds on average) due to user mobility across the factory floor. Upon modeling the environment and scheduling the 100 cloud native applications over a 6.7 hour period, we observe latencies when there is no active scheduling taking place, when the scheduling is done by the Packets Fair Queuing networking algorithm (Benchmark case) and when applications are scheduled using MECRAN (best case scenario). Latencies plotted below are average application latencies at every minute of the 6.7 hours scheduling period.

Worst Case Scenario - No active scheduling involved

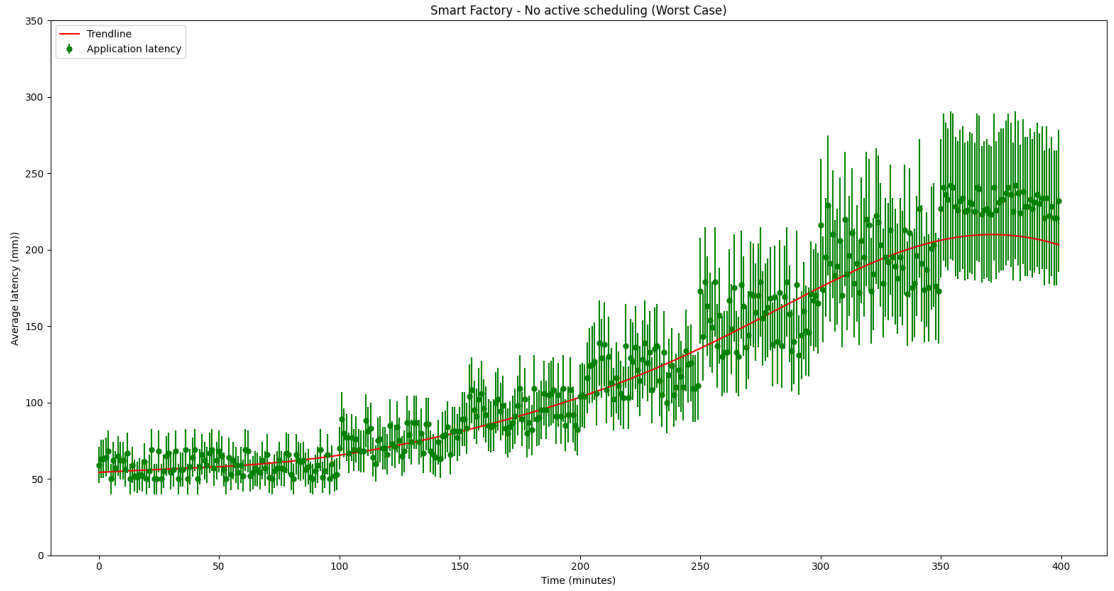

Figure 7.16: HoloLens application latency in a smart factory environment when no active scheduling is involved - Worst case scenario.

When there is no active scheduling taking place, the HoloLens' application latency has a value of 50 ms which increases over the 6.7 hours when users are trying to interact with the cloud-native application that powers the device. This can be observed in Fig. 7.16 and ideally Nestle is expecting to reduce the latency

to 10 milliseconds or below in order to improve the QoS and user experience on their devices.

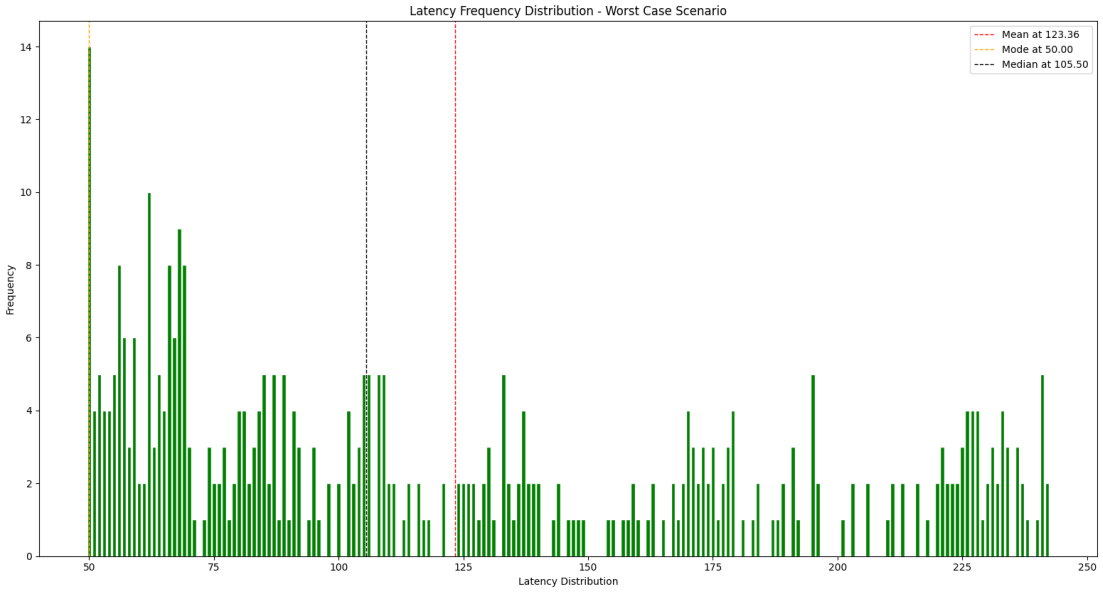

Figure 7.17: HoloLens application latency distribution showing mean, mode and median latency when no active scheduling is involved - Worst scenario.

Analyzing the latency distribution further in this scenario in Fig. 7.17, the mean latency is approximately 123 ms whilst the mode is 50 ms. Such latencies render emerging cloud-native applications not fit for purpose due to poor response time. In such a factory or distribution centre environment where tasks are time and mission critical, slow responding applications introduce inefficiencies into their operational processes.

Benchmark Scenario - scheduling using Packet Fair Queuing method

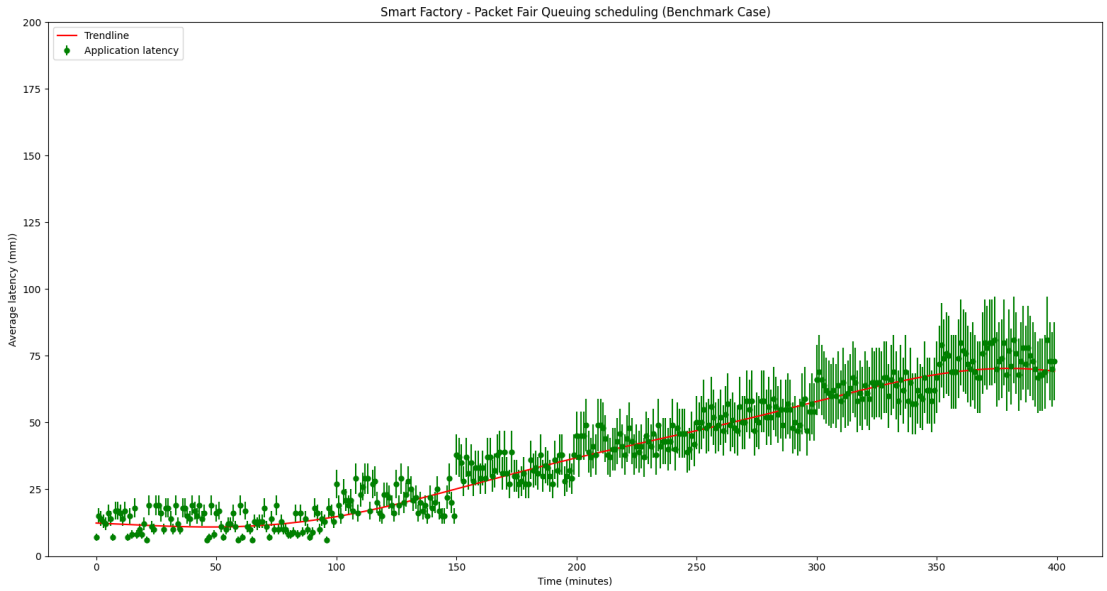

Figure 7.18: HoloLens application latency in a smart factory environment when scheduled by Packet Fair Queuing algorithm - Benchmark case scenario.

The worst case can be improved by introducing active scheduling. The same HoloLens application data is therefore passed through the Packet Fair Queuing method for scheduling and the latencies reached can be seen in Fig. 7.18. This method is able to achieve much lower latencies compared to the worst case scenario. This method is able to achieve 10 ms (the required latency) sometimes during the first 2 hours of scheduling. Unfortunately as usage increases and more users interact with the environment, we see application latencies increase continuously. At the end of the simulation period we observe latencies averaging 75 ms. Which is unacceptable for time and mission critical applications.

Figure 7.19 highlights the distribution of latency from the benchmark scenario further. The mean latency is approximately 39 ms - a huge improvement from 123 ms in the worst case scenario. However the HoloLens application requires constant latency of 10 ms or below. The most frequently occurring latency in this benchmark scenario is 16 ms (much improved than 50 ms mode from the worst case scenario. Even though there has been a significant improvement in latency using the Packet Fair Queuing method, the recorded latencies overtime will also render emerging

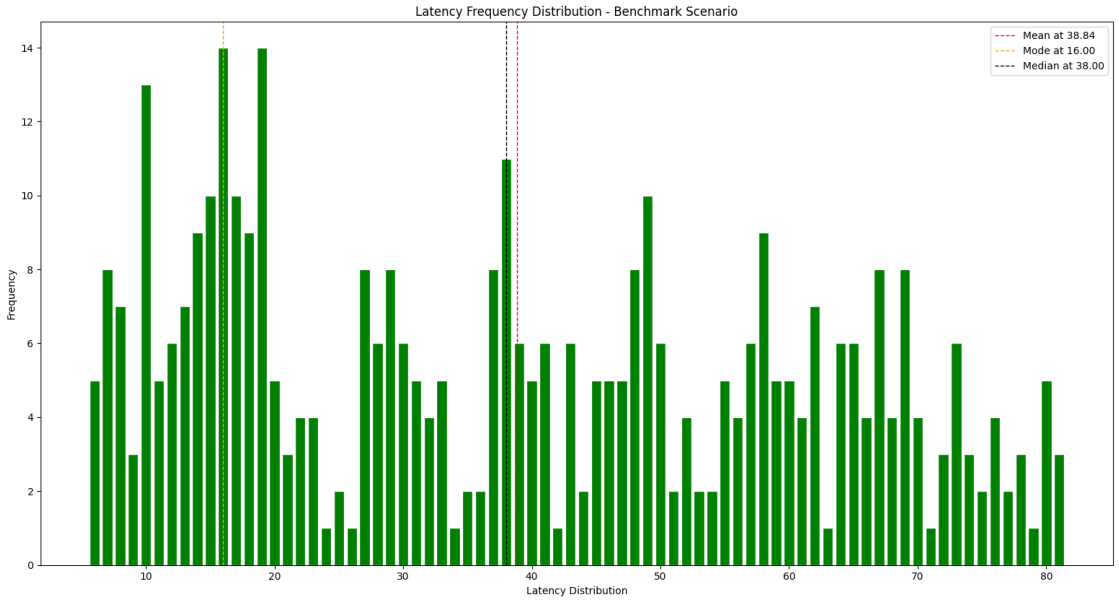

Figure 7.19: HoloLens application latency distribution showing mean, mode and median latency when scheduled by Packet Fair Queuing algorithm - Benchmark scenario.

cloud-native applications not fit for purpose due to how slow they take to respond. The latency required for best performance is 10 ms and below.

Best Case Scenario - scheduling using MECRAN

MECRAN as an optimization solution is designed to keep latencies at its lowest by making the best use of the available compute, network and storage resources. In this scenario, the exact same HoloLens application data is also passed through MECRAN scheduling. MECRAN starts off with latencies of about 6 ms. As more and more users start to interact with the environment in the first 2 hours, we see in Fig. 7.20 that MECRAN starts to learn and adjusts its scheduling routine to accommodate new or more traffic. Having learnt from its experiences from the past minutes, MECRAN consistently keeps latency at about 6 ms across the entire 6.7 hours, safely meeting the latency requirement for HoloLens to perform effectively.

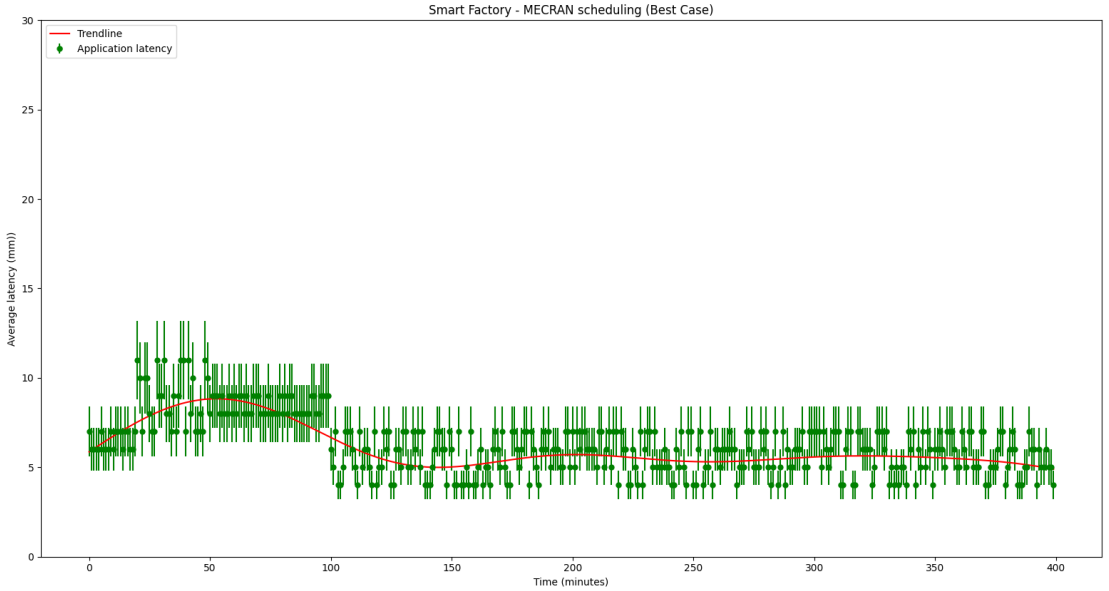

Figure 7.20: HoloLens application latency in Nestle's factory when scheduled by MECRAN - Best case scenario.

Figure 7.21 also highlights the distribution of latency from the best case scenario further. The mean latency is approximately 6.3 ms - an improvement over the 39 ms in the benchmark scenario and 123 ms in the worst case scenario. The most frequently occurring latency in this benchmark scenario is 7 ms - another improvement over the 16 ms in the benchmark scenario and 50 ms mode from the worst case scenario.

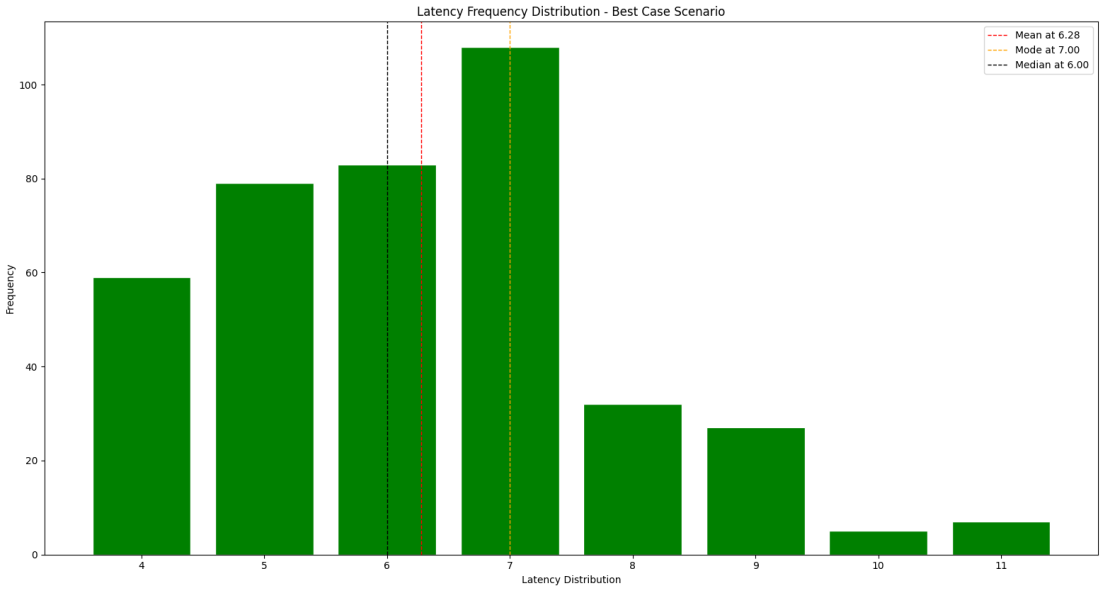

Figure 7.21: HoloLens application latency distribution showing mean, mode and median latency when scheduled by MECRAN - Best case scenario.

From the above results MECRAN is not only seen to improve latencies for HoloLens, it is also able to adjust effectively to increasing traffic due to its learning capability. MECRAN is also seen to be able to sustain good performance over a long period of time which is an encouraging observation that could benefit new future use cases.

8 Discussion & Conclusion

Contents

8.1 Summary

MECRAN has been effective at reducing bottlenecks in a data networks through its data packets routing routine and it has also been efficient at reducing latency through its task scheduling routine.

The DPR and task scheduler have achieved extraordinary results during their training and testing. When MECRAN was finally applied to the smart train use case, it outperformed other baseline models for single, double and quadruple track scenarios. MECRAN also outperformed baseline models when applied to the smart factory use case.

Figures 7.5, 7.6 and 7.7 show applications' performance percentages for single track, double track and quadruple track journeys over time. Where application performance percentage here describes how much of each application's latency and bandwidth requirements were met as a result of MECRAN's scheduling. As

an example, if an application required 10 ms latency and 50 Mbps bandwidth for performance, the above figures show how often such this required latency is achieved. Figures 7.5, 7.6 and 7.7 also compare how applications performed in the worst case scenario (with no active scheduling), the benchmark scenario and the best case scenario (with MECRAN). It can be observed that applications performed best under MECRAN.

Figures 7.8, 7.10 and 7.12 are the applications performance Probability Density Functions (PDFs) that supports how applications have performed in figures 7.5, 7.6 and 7.7 for the best, benchmark and worst case scenarios. The PDFs show the relationship between application performance observations and their probabilities of occurring. With a mean of 66.6% performance in the worst case scenario (Fig. 7.12), we also observe a huge number of latencies spreading away from the mean. The lowest and highest results being approximately 38% and 95% respectively, correlates with the relatively high standard deviation of 17.12. The benchmark scenario (Fig. 7.10) has a mean of approximately 88.8% performance and a standard deviation of 5.83. MECRAN (Fig. 7.8) shows a mean of 98.6% and a standard deviation of 1.57. This means the lowest and highest performance results were approximately 95% and 100% respectively.

Similarly in the smart factory use case too MECRAN's DPR routine lowered the occurrence of bottlenecks over time whilst the task scheduling routine delivered latencies below 10 milliseconds 99% of the time and outperforming other base models. MECRAN is not only seen to improve latencies in the smart factory use case, it is also flexible and able to adjust effectively to increasing traffic due to its learning capability and MECRAN is also seen to be able to sustain good performance over a long period of time.

MECRAN as a self-learning facility continues to outperform other algorithms and itself overtime, achieving the desired performance required by railway cloud-native applications. This effectively makes MECRAN a highly reliable performance-enabling, resource-rich and robust environment that allows applications to flexibly run on 'unlimited' and reliable compute, network and storage resources. This

research work takes an innovative approach in delivering on these key performance requirements through dynamic resource orchestration and task scheduling in a non-traditional radio access network environment where Multi-Access Edge Computing (MEC) is paired with Cloud-Based/ Centralized Radio Access Network (C-RAN). This collaborative work between academia and industry is a good example of research to practice partnerships.

8.2 Future Work

The results from this research have been encouraging and future work will be more geared towards applying MECRAN to other use cases to provide value to an even wider scope of stakeholders.

MECRAN currently considers single stateless or stateful container applications, but MECRAN can be easily extended in future work to also cater for multiple containers. Also, while independent singular models have been successfully used to develop MECRAN, a future opportunity would be to explore model ensembles for MECRAN explore any opportunities in improving performance.

8.3 Conclusion

5G wireless technology has enabled telecommunication and many other industries to explore better ways of delivering value by enabling game-changing digital applications, which could not have been considered in the past due to latency and data rate constraints. However most of these emerging digital applications have strict non-comprisable requirements for extremely low latency (10 ms and below) and high bandwidth for performance. When these applications communicate with the remote cloud through data exchanges, the extra latency introduced by the physical distance of communication, severely affects the performance of some applications. Data exchange over physical distance is a hard reality in wireless communication and cloud computing, but the resulting latency penalty is also a gap that 5G alone cannot resolve. With physical distance being the bottleneck, this research has set out, successfully achieved and tested a way to minimize the physical distance in order to enable low latency.

At the genesis of this research, efforts were made to practically validate an idea from ETSI to host MEC and C-RAN using the same physical hardware for many practical and financial benefits. This idea was implemented as part of this research and also formed the base infrastructure for this research.

In addressing the latency issue, this research set out to develop a machine learning based solution that would optimize cloud-native applications to be dynamically scheduled closer to a mobile / moving user or close where the application data is generated. This goal led to the development of the data packets routing (DPR) routine (using Spatio-temporal Graph Neural Network) and the task scheduling routine (using Deep Reinforcement Learning). The DPR predicts and alleviates bottlenecks from the network during the cloud-native application scheduling process, which aims to reduce latency. The task scheduler gets insight from the DPR routine in order to schedule cloud-native applications, based on network resources that have the best chance at ensuring the long-term low latency needs of a connected mobile user.

The MECRAN models have been tested against other baseline models with similar conditions and outperformed them. MECRAN models have then been applied to real-world use cases through collaborations with industry, as a way of validating the value of this research to industry as well.

References

[1] Jude Fletcher and David Wallom. "Deep Learning Based Task Scheduling in a Cloud RAN Enabled Edge Environment". In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. SEC '19. Arlington, Virginia: Association for Computing Machinery, 2019, pp. 283–285. URL: https://doi.org/10.1145/3318216.3363319.

[2] Jude Fletcher and David Wallom. "Using Machine Learning to Orchestrate Cloud Resources in a RAN Enabled Edge Environment: Poster Abstract". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. SenSys '19. New York, New York: Association for Computing Machinery, 2019, pp. 452–453. URL: https://doi.org/10.1145/3356250.3361930.

[3] M Series. "IMT Vision–Framework and overall objectives of the future development of IMT for 2020 and beyond". In: *Recommendation ITU* 2083 (2015), p. 0.

[4] Yun Chao Hu et al. "Mobile edge computing—A key technology towards 5G". In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[5] 5GPPP. "5G Vision". In: (2015), p. 16.

[6] David Tse and Pramod Viswanath. *Fundamentals of wireless communication*. Cambridge university press, 2005.

[7] Linda Amoah et al. "Probing the composition of Plasmodium species contained in malaria infections in the Eastern region of Ghana". In: *BMC Public Health* 19 (Dec. 2019).

[8] Benedicta Mensah et al. "Prevalence and risk factors associated with asymptomatic malaria among school children: repeated cross-sectional surveys of school children in two ecological zones in Ghana". In: *BMC Public Health* 21 (Sept. 2021).

[9] World Health Organization. "Global technical strategy for malaria 2016-2030," in: (2015).

[10] International Telecommunication Union. "Measuring Digital Development: ICT Price Trends 2019". In: (2019).

[11] Pisani Freedman. *Purves and Adhikari,"Statistics–Second Edition 4th Edn"*. 2007.

[12] Claus Stig Pedersen. "The UN Sustainable Development Goals (SDGs) are a Great Gift to Business!" In: *Procedia CIRP* 69 (2018). 25th CIRP Life Cycle Engineering (LCE) Conference, 30 April – 2 May 2018, Copenhagen, Denmark, pp. 21–24. URL: https://www.sciencedirect.com/science/article/pii/S2212827118300040.

[13] United Nations. *The UN Sustainable Development Goals. United Nations, New York*. https://sdgs.un.org/goals. Accessed: 2022-06-30. 2015.

[14] Z.A. Bhutta and M.A. Saeed. "Childhood Infectious Diseases: Overview". In: *International Encyclopedia of Public Health* (Dec. 2008), pp. 620–640.

[15] D Warren and C Dewar. *Understanding 5G: perspectives on future technological advancements in mobile. GSMA Intelligence*. 2014.

[16] A. Reznik et al. "Cloud RAN and MEC: A Perfect Pairing". In: *ETSI MEC* 23 (2018), p. 25.

[17] Arif Ahmed and Ejaz Ahmed. "A survey on mobile edge computing". In: *2016 10th International Conference on Intelligent Systems and Control (ISCO)* (2016), pp. 1–8.

[18] Julius Ziegler et al. "Making Bertha Drive—An Autonomous Journey on a Historic Route". In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20.

[19] Lanfranco Zanzi et al. "Evolving Multi-Access Edge Computing to Support Enhanced IoT Deployments". In: *IEEE Communications Standards Magazine* 3 (June 2019), pp. 26–34.

[20] Simon Hecker, Dengxin Dai, and Luc Van Gool. "End-to-end learning of driving models with surround-view cameras and route planners". In: *Proceedings of the european conference on computer vision (eccv)*. 2018, pp. 435–453.

[21] David J Griffiths. *Introduction to electrodynamics; 4th ed.* Re-published by Cambridge University Press in 2017. Boston, MA: Pearson, 2013. URL: https://cds.cern.ch/record/1492149.

[22] Jim Al-Khalili. "The birth of the electric machines: A commentary on Faraday (1832) 'Experimental researches in electricity'". In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 373 (Apr. 2015).

[23] G. Temple. "Electricity and Magnetism. By B. I. Bleaney and B. Bleaney Pp. xiv 676. 63s. 1957. (Clarendon Press, Oxford)". In: *The Mathematical Gazette* 43.345 (1959), pp. 229–230.

[24] I.S. Grant and W.R. Phillips. *Electromagnetism*. Manchester Physics Series. Wiley, 2013. URL: https://books.google.co.uk/books?id=Wi073n5G-8oC.

[25] Graham Turnbull. "Maxwell's equations [Scanning Our Past]". In: *Proceedings of the IEEE* 101.7 (2013), pp. 1801–1805.

[26] J.D. Jackson. *Classical Electrodynamics*. Wiley, 2012. URL: https://books.google.co.uk/books?id=8qHCZjJHRUgC.

[27] J. C. Maxwell. "XXV. On physical lines of force". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 21.139 (1861), pp. 161–175. eprint: https://doi.org/10.1080/14786446108643033. URL: https://doi.org/10.1080/14786446108643033.

[28] J. Clerk Maxwell. "A Dynamical Theory of the Electromagnetic Field". In: *Philosophical Transactions of the Royal Society of London* 155 (1865), pp. 459–512. URL: http://www.jstor.org/stable/108892 (visited on 07/08/2022).

[29] C.A. Bunge, M. Beckers, and T. Gries. *Polymer Optical Fibres: Fibre Types, Materials, Fabrication, Characterisation and Applications*. Woodhead Publishing Series in Electronic and Optical Materials. Elsevier Science, 2016. URL: https://books.google.co.uk/books?id=-EG9BgAAQBAJ.

[30] J. Carr. *Microwave and Wireless Communications Technology*. Elsevier Science, 1997. URL: https://books.google.co.uk/books?id=1j1E541LKVoC.

[31] L. Freed and P. Gralla. *How Wireless Works*. How it works series. Que, 2002. URL: https://books.google.co.uk/books?id=ZHdGAAAAYAAJ.

[32] Neil Caithness, Michel Drescher, and David Wallom. "Can Functional Characteristics Usefully Define the Cloud Computing Landscape and is the Current Reference Model Correct?" In: *J. Cloud Comput.* 6.1 (Dec. 2017). URL: https://doi.org/10.1186/s13677-017-0084-1.

[33] David C. H. Wallom et al. "Federating Infrastructure as a Service Cloud Computing Systems to Create a Uniform E-infrastructure for Research". In: *2015 IEEE 11th International Conference on e-Science*. 2015, pp. 155–164.

[34] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. en. 2011-09-28 2011.

[35] Suk Hui and Alan Yeung. "Challenges in the Migration to 4G Mobile Systems". In: *Communications Magazine, IEEE* 41 (Jan. 2004), pp. 54–59.

[36] Jalal Al-Muhtadi, Dennis Mickunas, and Roy Campbell. "A lightweight reconfigurable security mechanism for 3G/4G mobile devices". In: *Wireless Communications, IEEE* 9 (May 2002), pp. 60–65.

[37] Y. Yamao et al. "Radio Access Network Design Concept for the Fourth Generation Mobile Communication System". In: vol. 3. Feb. 2000, 2285–2289 vol.3.

[38] A. Einstein. *Die Grundlage der allgemeinen Relativitätstheorie (German) [The theory of relativity]*. Annalen der Physik und Chemie. J.A. Barth, 1916. URL: https://books.google.co.uk/books?id=0WHMngEACAAJ.

[39] J. Gribbin and M. Gribbin. *Einstein's Masterwork: 1915 and the General Theory of Relativity*. Icon, 2015. URL: https://books.google.co.uk/books?id=lvDQrQEACAAJ.

[40] Ian Foster et al. "Cloud computing and grid computing 360-degree compared". In: *2008 grid computing environments workshop*. Ieee. 2008, pp. 1–10.

[41] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities". In: *2008 10th IEEE international conference on high performance computing and communications*. Ieee. 2008, pp. 5–13.

[42] David F Bacon, Susan L Graham, and Oliver J Sharp. "Compiler transformations for high-performance computing". In: *ACM Computing Surveys (CSUR)* 26.4 (1994), pp. 345–420.

[43] Rich Uhlig et al. "Intel virtualization technology". In: *Computer* 38.5 (2005), pp. 48–56.

[44] Pradeep Padala et al. "Adaptive control of virtualized resources in utility computing environments". In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. 2007, pp. 289–302.

[45] Tharam Dillon, Chen Wu, and Elizabeth Chang. "Cloud Computing: Issues and Challenges". In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. 2010, pp. 27–33.

[46] Michael Armbrust et al. *Above the clouds: A berkeley view of cloud computing*. Tech. rep. Technical Report UCB/EECS-2009-28, EECS Department, University of California ..., 2009.

[47] Michael N Huhns and Munindar P Singh. "Service-oriented computing: Key concepts and principles". In: *IEEE Internet computing* 9.1 (2005), pp. 75–81.

[48] S.J. Shackelford. *The Internet of Things: What Everyone Needs to Know®*. What everyone needs to know. Oxford University Press, 2020. URL: https://books.google.co.uk/books?id=g9XTyQEACAAJ.

[49] Yating Wang, Ing-Ray Chen, and Ding-Chau Wang. "A Survey of Mobile Cloud Computing Applications: Perspectives and Challenges". In: *Wirel. Pers. Commun.* 80.4 (Feb. 2015), pp. 1607–1623. URL: https://doi.org/10.1007/s11277-014-2102-7.

[50] Michael Till Beck et al. "ME-VoLTE: Network functions for energy-efficient video transcoding at the mobile edge". In: *2015 18th International Conference on Intelligence in Next Generation Networks*. 2015, pp. 38–44.

[51] Zhen Xiao, Weijia Song, and Qi Chen. "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment". In: *IEEE Transactions on Parallel and Distributed Systems* 24.6 (2013), pp. 1107–1117.

[52] Mahadev Satyanarayanan. "A Brief History of Cloud Offload: A Personal Journey from Odyssey Through Cyber Foraging to Cloudlets". In: *GetMobile: Mobile Comp. and Comm.* 18.4 (Jan. 2015), pp. 19–23. URL: https://doi.org/10.1145/2721914.2721921.

[53] Yuan Ai, Mugen Peng, and Kecheng Zhang. "Edge computing technologies for Internet of Things: a primer". In: *Digital Communications and Networks* 4.2 (2018), pp. 77–86. URL: https://www.sciencedirect.com/science/article/pii/S2352864817301335.

[54] ETSI. *Collaborative tools for standardized technologies.* 2022. URL: https://forge.etsi.org/.[Accessed:21-Jun-2022].

[55] Guangjie Li et al. "Architecture of GPP based, scalable, large-scale C-RAN BBU pool". In: *2012 IEEE Globecom Workshops*. 2012, pp. 267–272.

[56] Aleksandra Checko et al. "Cloud RAN for Mobile Networks—A Technology Overview". In: *Communications Surveys Tutorials, IEEE* 17 (Jan. 2015), pp. 405–426.

[57] Venu Vinnakota et al. "Fronthaul Design In Cloud Radio Access Networks: A Survey". In: *Advanced Computing and Communications* (Dec. 2019).

[58] Giuseppe Carella et al. "Mobile cloud networking: From cloud, through NFV and beyond". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*. 2015, pp. 7–8.

[59] Van-Giang Nguyen et al. "SDN/NFV-Based Mobile Packet Core Network Architectures: A Survey". In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1567–1602.

[60] Shunmugapriya Ramanathan et al. *A Comprehensive Study of Virtual Machine and Container Based Core Network Components Migration in OpenROADM SDN-Enabled Network*. Aug. 2021.

[61] Jiao Zhang et al. "Load Balancing in Data Center Networks: A Survey". In: *IEEE Communications Surveys Tutorials* 20.3 (2018), pp. 2324–2352.

[62] Shunmugapriya Ramanathan et al. "Live Migration of Virtual Machine and Container Based Mobile Core Network Components: A Comprehensive Study". In: *IEEE Access* 9 (2021), pp. 105082–105100.

[63] CRIU Community. *Checkpoint/Restoration In UserSpace (CRIU),* 2019. URL: https://criu.org/.[Accessed:21-Jun-2022].

[64] CRIU TCP. *CRIU TCP Connection,* 2019. URL: https://criu.org/TCP_connection.[Accessed:21-Jun-2022].

[65] Sage A. Weil et al. "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data". In: *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. 2006, pp. 31–31.

[66] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. URL: https://books.google.co.uk/books?id=qWPwnQEACAAJ.

[67] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. URL: https://books.google.co.uk/books?id=eBSgoAEACAAJ.

[68] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[69] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[70] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[71] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533.

[72] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[73] Gang Huang, Xiaohua Zhou, and Qingyang Song. *Deep reinforcement learning for portfolio management*. 2020. URL: https://arxiv.org/abs/2012.13773.

[74] AquaComms TeleGeography. *Submarine Cable Map,* 2022. URL: https://www.submarinecablemap.com/.[Accessed:13-July-2022].

[75] Carl Boettiger. "An Introduction to Docker for Reproducible Research". In: *SIGOPS Oper. Syst. Rev.* 49.1 (Jan. 2015), pp. 71–79. URL: https://doi.org/10.1145/2723872.2723882.

[76] Brendan Burns et al. "Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade". In: *Queue* 14.1 (Jan. 2016), pp. 70–93. URL: https://doi.org/10.1145/2898442.2898444.

[77] *Network Rail Limited*. https://www.networkrail.co.uk/. Accessed: 2020-09-30.

[78] *Nestle S.A, NESN (SWX)*. https://www.nestle.com/. Accessed: 2020-09-30.

[79] *The Noguchi Memorial Institute for Medical Research (NMIMR)*. https://noguchimedres.org/. Accessed: 2020-09-30.

[80] V. Cerf and R. Kahn. "A Protocol for Packet Network Intercommunication". In: *IEEE Transactions on Communications* 22.5 (1974), pp. 637–648.

[81] Joachim Sachs and Krister Landernas. "Review of 5G capabilities for smart manufacturing". In: *2021 17th International Symposium on Wireless Communication Systems (ISWCS)*. 2021, pp. 1–6.

[82] *Microsoft HoloLens 2*. https://www.microsoft.com/en-us/hololens/. Accessed: 2022-06-30.

[83] Dorin Ungureanu et al. *HoloLens 2 Research Mode as a Tool for Computer Vision Research*. 2020. URL: https://arxiv.org/abs/2008.11239.

[84] Kate Keahey et al. "Lessons Learned from the Chameleon Testbed". In: *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*. USA: USENIX Association, 2020.

[85] Alin Calinciuc et al. "OpenStack and Docker: Building a high-performance IaaS platform for interactive social media applications". In: *2016 International Conference on Development and Application Systems (DAS)*. 2016, pp. 287–290.

[86] Inc Docker. *Docker*. https://www.docker.com/. Accessed: 2022-06-30.

[87] Open Community Rackspace Hosting NASA. *OpenStack*. https://www.openstack.org/. Accessed: 2022-06-30.

[88] Sage Weil et al. "Ceph: A Scalable, High-Performance Distributed File System." In: Nov. 2006, pp. 307–320.

[89] Forbes Guthrie, Scott Lowe, and Kendrick Coleman. *VMware vSphere design*. John Wiley & Sons, 2013.

[90] Jinho Hwang et al. "A component-based performance comparison of four hypervisors". In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE. 2013, pp. 269–276.

[91] Office for National Statistics. *Towns and cities in the UK - Office for National Statistics*. https://www.ons.gov.uk/aboutus/transparencyandgovernance/freedomofinformationfoi/townsandcitiesintheuk. Accessed: 2022-06-30.

[92] Inc Docker. *Docker Compose*. https://docs.docker.com/compose/. Accessed: 2022-06-30.

[93] Markus List. "Using Docker Compose for the Simple Deployment of an Integrated Drug Target Screening Platform". In: *Journal of integrative bioinformatics* 14 (June 2017).

[94] Nannan Zhao et al. "Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 32.4 (2021), pp. 918–930.

[95] Inc Docker. *Docker Hub*. https://www.hub.docker.com/. Accessed: 2022-06-30.

[96] Ian F Akyildiz, Xudong Wang, and Weilin Wang. "Wireless mesh networks: a survey". In: *Computer networks* 47.4 (2005), pp. 445–487.

[97] Dimitri Bertsekas and Robert Gallager. *Data Networks (2nd Ed.)* USA: Prentice-Hall, Inc., 1992.

[98] Aric A. Hagberg, Daniel A. Schult, and Pieter Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: 2008.

[99] W. R. Stevens. *TCP/IP Illustrated, Volume 1; The Protocols*. Addison-Wesley, Reading, MA, USA, 1994.

[100] Ashesh Jain et al. "Structural-rnn: Deep learning on spatio-temporal graphs". In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2016, pp. 5308–5317.

[101] Bing Yu, Haoteng Yin, and Zhanxing Zhu. "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting". In: *arXiv preprint arXiv:1709.04875* (2017).

[102] Amol Kapoor et al. "Examining covid-19 forecasting using spatio-temporal graph neural networks". In: *arXiv preprint arXiv:2007.03113* (2020).

[103] G. B. Lesovik et al. *Arrow of time and its reversal on IBM quantum computer*. 2017. URL: https://arxiv.org/abs/1712.10057.

[104] William Thomson. "IX.—On the Dynamical Theory of Heat. Part V. Thermo-electric Currents". In: *Transactions of the Royal Society of Edinburgh* 21.1 (1857), pp. 123–171.

[105] Douglas Comer. *Internetworking with TCP/IP / Douglas E. Comer*. eng. 3rd ed. Upper Saddle River, N.J.: Prentice Hall, 1995.

[106] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[107] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[108] François Chollet. *Keras*. https://github.com/fchollet/keras. 2015.

[109] RICHARD BELLMAN. "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684. URL: http://www.jstor.org/stable/24900506 (visited on 07/19/2022).

[110] European Commission. *Transforming Europe's Rail System - Shift2Rail*. https://rail-research.europa.eu/wp-content/uploads/2020/07/20200705_Partnership_High-Level-Paper.pdf. 2015.

[111] Office for National Statistics. *Towns and cities in the UK*. https://www.ons.gov.uk/aboutus/transparencyandgovernance/freedomofinformationfoi/townsandcitiesintheuk.

[112] A. Demers, S. Keshav, and S. Shenker. "Analysis and Simulation of a Fair Queueing Algorithm". In: *Symposium Proceedings on Communications Architectures amp; Protocols*. SIGCOMM '89. Austin, Texas, USA: Association for Computing Machinery, 1989, pp. 1–12. URL: https://doi.org/10.1145/75246.75248.

[113] Songwu Lu, V. Bharghavan, and R. Srikant. "Fair scheduling in wireless packet networks". In: *IEEE/ACM Transactions on Networking* 7.4 (1999), pp. 473–489.

[114] H.J. Chao et al. "Design of packet-fair queuing schedulers using a RAM-based searching engine". In: *IEEE Journal on Selected Areas in Communications* 17.6 (1999), pp. 1105–1126.

[115] J Bennett and H Zhang. *Worst-case fair packet fair queueing algorithms.* Tech. rep. tech. rep., Computer Science, Carnegie Mellon University, 1996.

[116] Zhuming Bi, Li Da Xu, and Chengen Wang. "Internet of things for enterprise systems of modern manufacturing". In: *IEEE Transactions on industrial informatics* 10.2 (2014), pp. 1537–1546.

[117] Microsoft. *HoloLens2.* 2022. URL: `https://www.microsoft.com/en-us/hololens/hardware.[Accessed:21-Jun-2022]`.

[118] Ariel Bleicher. "A surge in small cells [2013 tech to watch]". In: *IEEE Spectrum* 50.1 (2012), pp. 38–39.