CONTAINER BASED MULTI-ACCESS EDGE COMPUTING FOR 5G NETWORKS

BY

MOSUDI, Isiaka Olukayode
MENG/SEET/2017/7331

DEPARTMENT OF TELECOMMUNICATION ENGINEERING
FEDERAL UNIVERSITY OF TECHNOLOGY MINNA

AUGUST, 2021

CONTAINER BASED MULTI-ACCESS EDGE COMPUTING FOR 5G NETWORKS

BY

MOSUDI, Isiaka Olukayode

MENG/SEET/2017/7331

A THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGERIA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING IN COMMUNICATION ENGINEERING

AUGUST, 2021

ABSTRACT

The growth of the telecommunication industry is fast paced with ground-breaking engineering achievements. Despite this, portable mobile handheld devices have very low computational, storage and energy carrying capacity occasioned by the needs to satisfy portability, very small form factor, ergonomics, style and trends. Proposals such as cloudlets, cyber foraging, mobile cloud computing (MCC), and more recently but most applicable, multi-access edge computing (MEC) have been proffered. New and emerging use cases, especially the deployments of 5G will bring up a lot of latency-sensitive and resource-intensive applications. To address these challenges, this work introduced the use of secure containerization for MEC applications and location of MEC host at the 5G centralized unit within the radio access network (RAN) aiding offloading of computational, storage and analytics requirements close to UE at the fringe of the network where the data are being generated and results being applied. The major contribution of this thesis is the use of secured containerization technology to replace virtual machines, making it possible to use containers for MEC applications, reducing application overhead while satisfying the isolation of MEC infrastructure as required by the European Technology Standard Institute (ETSI) to ensure MEC application security. 5G end-to-end transport specifications were evaluated for the vantage location of MEC server within the radio access network and achieved theoretical values between 4.1ms and 14.1ms end-to-end latency. These figures satisfy requirements of VR/AR (7-12ms); tactile Internet (<10ms); Vehicle-to-Vehicle (< 10ms); Manufacturing & Robotic Control/Safety Systems (1-10ms). The results confirmed that edge computing has lower user plane latency figures and reduced backhaul traffic with lower application failure rate. Secured containerised Multi-access computing infrastructures have many advantages of mobile cloud computing for mobile wireless device computational power and energy carrying capacity deficiencies to cater asymmetric UE applications. Applications hosted within the RAN have better support for new and emerging application requirements in terms of high amount of computational, storage and analytics capabilities while at low latency figures. Edge deployments will reduce the pressure on network operators backhaul link saving end-to-end ecosystem from collapse due to heavy backhaul traffic that might result from billions of 5G UEs. No matter how fast 5G network will be, MEC will ensure the need not to transport huge data for processing in the cloud and returning the results to the UE. This will enhance privacy and security while also conserving bandwidth.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviations	Definition
5GC	5G Core Network
5GS	5G System
AF	Application Function
AAU	Active Antenna Unit
AMF	Access and Mobility Management Function
API	Application Programming Interface
AR	Augmented Reality
AUSF	Authentication Server Function
CAV	Connected Autonomous Vehicle
CCP	Common Compute Platform
CP	Control Plane
CPRI	Common Public Radio Interface
CUPS	Control/User Plane Separation
DN	Data Networks
DU	Distributed Unit
eCPRI	Evolved Common Public Radio Interface
eMBB	Enhanced Mobile Broadband
EPC	Evolved Packet Core
E-UTRAN	Evolved UMTS Terrestrial RAN
Gi LAN	GPRS Interface LAN
gNB	Next generation Node B (5G base station name)

gNB-CU	gNB-Centralised Unit
gNB-DU	gNB-Distributed Unit
HLFS	Higher Layer Functional Split
HTTP/2	Hypertext Transfer Protocol Version 2
I/O	Input/Output
IaC	Infrastructure as Code
ITS	Intelligent Transportation Systems
JSON	JavaScript Object Notation
LLFS	Lower Layer Functional Split
MCC	Mobile Cloud Computing
MEC	Multi-access Edge Computing
mMTC	Massive Machine Type Communication
NEF	Network Exposure Function
NF	Network Function
NG-RAN	NextGen RAN, Next Generation RAN
NGC	Next Generation Core
NGFI	Next Generation Fronthaul Iinterface
NR	New Radio
NRF	Network Repository Function
NSSF	Network Slice Selection Function
PCF	Policy Control Function
PDCP	Packet Data Convergence Protocol
QoE	Quality of User Experience
RAN	Radio Access Network
RF	Radio Frequency

RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Resource Management
RRU	Remote Radio Unit
SBA	Service Based Architecture
SDAP	Service Data Adaptation
SGi	Steering Gi
SMF	Session Management Function
UDM	Unified Data Management
UE	User Equipment
UMTS	Universal Mobile Telecommunication System
UPF	User Plane Function
uRLLC	Ultra-Reliable and Low Latency Communication
VCS	Version Control System
VNF	Virtualized Network Function
VR	Virtual Reality

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background to the Study

There are several constraints on mobile devices as well as other portable 5G user equipment (UE) devices. Computational resources, memory limitation, storage, network and energy carrying capacity are some of the constraints of cellular mobile communication UEs and these have a significant effect on the type of application software available and for how long a battery can hold a charge to support such applications. The major constraints include computational power, charge holding time, storage and memory limitations, especially for complex processes (Taleb *et al.*, 2017). There are latency-critical and resource-intensive services which are needed to be supported by 5G, these include: telepresence, robotics, factory automation, and intelligent transportation systems, Virtual Reality (VR), Augmented Reality (AR), medicals, smart grid, serious gaming, education and culture (Parvez *et al.*, 2018). More so, high definition images and multi-feed super high definition quality live video streaming demands for mobile users are constantly being escalated over the recent decade. 5G is projected to provide services that will support communication, computing, control and content delivery for high-intensity network traffic (Mao *et al.*, 2017).

The deployment of 5G cellular mobile telecommunication standard will herald explosive evolution of Information and Communication Technology (ICT) innovations for mobile devices, machine-to-machine, Internet of Things (IoT), emerging vehicle technologies - V2V and V2X, etc. There will be an enormous increase in the number of mobile devices, expectedly about 50 billion devices, but this number will be completely dwarfed by the exponential growth in the volume of data generated by resource-intensive and feature-

rich multimedia applications. These will create hype for mobile data traffic and compute requirements (Skarpness, 2017).

To resolve challenges posed by these constraints, the computational requirements of mobile applications were offloaded to be processed on tethered external infrastructures with adequate resources. These external infrastructures are usually commercial off-the-shelf or customized standardized IT infrastructures configured to process and return results for applications. Different interventions have been proposed, including cyber foraging, cloudlet, multi-access edge computing (MEC) and mobile cloud computing (MCC) (Mach and Becvar, 2017).

A cloudlet is a resource-rich computer or cluster of computers that is well connected to the Internet, trusted and is constantly available for use by untethered mobile devices within close proximity (Satyanarayanan *et al*., 2009). Cyber foraging is considered as dynamical augmentations of computing resources of mobile devices and opportunistically exploiting computing of tethered infrastructure in the nearby environment (Satyanarayanan, 2001). It is the capability of infrastructure to seamlessly undertake migration of computation from one node to another (Patil *et al.*, 2016). Cloud computing

(CC) is the abstraction of computing resources e.g. processor, RAM, storage and network services from separate hardware units while presenting as a pool of reusable on-demand shared computing infrastructure. This can be provisioned rapidly and released programmatically or manually involving minimal management effort while creating cost benefits and flexibility. The rate of cloud adoption is very high, majorly making it an economically viable option. MCC is the integration of CC to serve cloud-based web apps over the Internet for smartphones, tablets, and other portable devices. Cloud computing in mobile cellular networks, like every other technology, though a solution, has come with its fair share of challenges as cellular mobile communication technologies mature from

1G, 2G, 3G to 4G and looks forward to 5G. Ordinarily, cloud computing should provide enough resources for offloading of computational demands (Mach and Becvar, 2017). Cloud computing is predominantly application programming interface (API) driven and economically viable. Cloud infrastructure can be shared across many end users, developers, mobile network operators and corporations spanning over widely separated geographical locations, allowing the reduced cost of services compared to traditional legacy infrastructures. Despite all the potentials of cloud computing, it has been unable to fulfil mobile application end-to-end latency requirements due to long response times, due, in turn, to the centralized cloud architecture model resulting into high signal propagation delay, affecting the end-user quality of experience (QoE) (Taleb *et al*., 2017). Other concerns presented by the use of cloud computing include security and privacy, addressing, interoperability, bandwidth (Díaz *et al.*, 2016), as well as government policy (National Information Technology Development Agency (NITDA), 2019); (Okafor, 2017).

The advent of the so much anticipated 5G technologies emerging mobile applications such as Augmented Reality (AR), Virtual Reality (VR) (Alsafi and Westphal, 2016), face/voice detection and identification for surveillance, authentication and access control, connected autonomous vehicles (CAV), intelligent transportation systems (ITS) and highway traffic management systems, ultra-high-definition multi-feed live streaming. These are anticipated to be among the high resource demanding applications over wireless cellular networks. In particular, the newly emerging mobile Augmented Reality and Virtual Reality (AR/VR) applications are anticipated to be among the most demanding applications over cellular wireless networks (Erol-Kantarci and Sukhmani, 2018).

Field devices such as traffic signals, roadside sensors, face detection and identification, surveillance networks, location services, Intelligent Transportation Systems (ITS) and 3

highway traffic management systems. are gradually being connected to central monitoring systems for better traffic management. MEC seeks to address issues and challenges surrounding billions of field devices generating gigabits of low latency-sensitive data that require split seconds between data generation, data processing and eventual results being applied to certain control mechanisms. Advantages of edge computing include:

A. Access to real-time radio network information that creates opportunities that can be leveraged by applications (Shahzadi *et al*., 2017) giving rise to location-based services opportunities like location-aware advertising, asset tracking, connected autonomous vehicle, AR, ITS and highways traffic management systems.
B. Edge computing will improve Quality of User Experience (QoE) by leveraging on reduced latency and high throughput available when application service logics are computed on the edge servers within the RAN
C. Reducing data security breach and enhancing privacy by reducing the level of exposure through manipulation of data close to the source rather than transmitting via numerous routes to the cloud.
D. Edge computing creates a new and emerging market value chain in mobile networks thereby opening the network to third parties (Patel *et al*., 2014), who can develop and quickly create innovative applications, benefiting all parties (Huang *et al*., 2017).

1.2 Statement of the Research Problem

The motivation for this research work stems from the claim by Satyanarayanan *et al* (2009) that "resource poverty is a fundamental constraint that severely limits the class of applications that can run on mobile devices." It was also argued by Satyanarayanan *et al.* (2009) that "at any given cost and level of technology, considerations of weight, size, battery life, ergonomics, and heat dissipation exert a severe penalty on computational resources such as processor speed, memory size, and disk capacity." There are challenges of unacceptable latency figures in 4G deployments but in 5G which has higher traffic, it is feared that there might be an increase in latency. These have been perennial challenges to mobile communication UEs but the deployment of 5G is expected to further exacerbate the problems with the rise of new and emerging feature-rich mobile applications generating high-intensity network traffic. Besides the challenges posed to mobile UEs, this scenario will be exerting unprecedented pressure on backhaul and fronthaul networks.

Researches have been carried out to augment for computational resources as well as energy-carrying capacity constraints in the mobile wireless devices, but the shortcomings of the earlier proposed solutions include:

A. Unstable Infrastructure:

   The inadequate or total loss of stable augmenting infrastructure has been the bane of cyber foraging techniques to resolve the challenges of resource constraints in portable mobile UEs (Gordon *et al*., 2012); (Qing *et al*., 2013); (Dean and Ghemawat, 2004); (Huang *et al*., 2010); (ETSI, 2017). This usually creates a situation of low bandwidth that results in poor application behaviour. Whereas MEC is compatible with orchestration, software configuration management, IaC and VCS, It is also deployable on standard off-the-shelf or customized IT

infrastructure making it possible to have vertical and horizontal infrastructure scalability, the mechanisms required to achieve stable infrastructure.

B. Platform-specific:

Mobile Assistance Using Infrastructure (MAUI) in Cuervo *et al*. (2010), Code Offload by Migrating Execution Transparently (COMET) (Gordon *et al*., 2012) and Misco - a MapReduce framework for mobile systems in Dou *et al.* (2010) relied on Microsoft .Net Framework; the now discontinued, Android Dalvik Virtual Machine and Python respectively limiting the type of mobile applications making use of the solutions. MEC can support computer programming language or frameworks consistent with standard or customized IT platforms that require computational, memory, storage and power resources (ETSI, 2019).

C. High mobility interruption time or zero mobility of augmenting infrastructure.

High mobility interruption time of 30 milliseconds in 4G LTE whereas cloudlets (Satyanarayanan *et al.*, 2009) and MAUI in Cuervo *et al.* (2010) have zero support for mobility interruption. User mobility is not encouraged by these augmenting solutions. MEC on 5G will deliver mobility interruption time of zero millisecond (ITU, 2015); (Taleb *et al.*, 2017).

D. High user plane (UP) and control plane (CP) latency:

High data communication delay between the UE and Internet/cloud for cloudlets and mobile cloud computing-based strategies in Zhang *et al.* (2010); Chun *et al.* (2011); Kosta *et al.* (2012); (Gordon *et al.*, 2012). The total UP latency for MEC deployment in 4G Long Time Evolution Evolved Packet Core (LTE/EPC) is estimated, with reference to Parvez *et al.* (2018), as the summation of packet transmission delay between the UE via LTE, EPC to MEC host connected across

the Steering GPRS interface (SGi) for services sourced at MEC. Average container boot time is 1.87 seconds compared to Virtual Machines (VMs) average boot time of 94.90 seconds (Zhang *et al.*, 2018*),* containerized MEC can offer lower application control plane latency.

E. Inconsistencies between production and staging MEC environments:
   Containerized applications have the advantage of portability, running in the same environment during testing, staging and production deployments.

The issue of resource poverty has been a perennial challenge to the telecommunication industry. The scale of 5G will be enormous, both in terms of devices and data but the number of devices will be dwarfed by the volume of data will be generated (Skarpness, 2017). The huge amount of data, obviously, will require analytics as well as transport. There are already unacceptable latency figures in 4G LTE/EPC deployments but in 5G which has higher traffic, it is feared that there might be an increase in the latency.

1.3 Aim and Objectives of the study

The aim of this work is to demonstrate use cases for MEC deployments in 5G Networks by developing models of MEC infrastructure deployments.

The research objectives are to:

i. develop low latency models for MEC deployment for 5G networks through the evaluation of both 3GPP and non-3GPP components of 5G networks transport specifications.
ii. develop and deploy secure resource-intensive containerized mobile web application testbed implemented in Kata Containers (Kata Containers, 2019) for both edge and remote cloud servers.

iii. performance evaluation of the developed MEC models and determination of the best fit for 5G applications.

1.4 Justification of the study

The drawbacks of previous research works were explored. These include limitations posed by pre 5G wireless technologies on the maximization in meeting latency requirements of new and emerging applications consistent with International Telecommunication Union (ITU) specifications for 5G eMBB and uRLLC use cases.

1.5 Scope of the study

The research focused on only the Multi-access Edge Application part of Mobile Edge Host within the Multi-access Edge (ME) Host Level of ETSI MEC framework (ETSI, 2019a). This research work does not include any work on ME systems-level management, ME host-level management or network-level entities of MEC. This work also proposed the location of MEC host close to 5G centralized unit (CU), connected directly to packet data convergence protocol (PDCP). Disposable infrastructures required for mobile applications computation, storage and analytics deployed at the MEC are available as machine-readable definition files downloadable from synchronized but distributed repository with tracking and coordination of files modifications using Git.

In this work, the estimated user plane latency values were benchmarked with values of known low latency application use case requirements (Sutton, 2018b):

A. Virtual Reality & Augmented Reality: 7-12ms
B. Tactile Internet (Remote Surgery, Remote Diagnosis, Remote Sales): < 10ms
C. Vehicle-to-Vehicle (Co-operative Driving, Platooning, Collision Avoidance): < 10ms

D. Manufacturing & Robotic Control / Safety Systems: 1-10ms

Models for MEC based on infrastructure as code (IaC), containerization and version control system (VCS) deployment scenario for eMBB and uRLLC use cases were proposed for end-to-end 5G network, variously and concurrently serving multiple asymmetric mobile user computational requirements by leveraging on IaC, containerization and VCS to enable orchestrated provisioning, patching, freezing, caching, resuming and termination of infrastructure instances. This will enable the MEC server to continue to programmatically serve different mobile applications as at when required and free up resources when applications are not being served.

1.6 Thesis Outline

This thesis is composed of five chapters. In Chapter One, the work was introduced, motivation for the study of **Container-Based Multi-access Edge Computing for 5g Networks**, the aim and objectives were enumerated. Definition of terms and keywords, the research fundamentals, including the statement of the research problem were explained in this chapter.

The remaining part of this document is organized as follows: Chapter Two provides review of past research efforts to ameliorate the resources poverty challenges inherent in portable handheld mobile cellular user equipment. In Chapter Three - Methodology, both 3GPP and non-3GPP components of 5G networks transport specifications were evaluated. This chapter also contains the detail and description of the techniques used in the experiments. Results of the technical evaluation of 5G transport specifications are specified and discussed in Chapter Four. Results from the experiment comparing the behaviour of applications deployed at the MEC relative to mobile cloud computing

(MCC) deployments were equally presented in Chapter Four. Chapter Five contains the conclusions arrived as a result of this research work. Included in Chapter 5 are the research limitations and future works, contributions and recommendations.

CHAPTER TWO

2.0 LITERATURE REVIEW

2.1 Fundamentals

The continuous evolution of the overall end-to-end architecture into 5G requires technological advancements. The foundation of 5G is open source and the development of 5G is hinged on new sets of mature open source technology building blocks. These blocks are being applied in new use cases to provide platforms that will satisfy requirements of new and emerging information and communication technology innovations. Extending and distribution of computation, storage and analytics capabilities, available in the cloud infrastructure, to the RAN at the edge of the network close to the UE is, more like, a mini data centre at the network edge. Evolving to this architecture requires some enabling technologies to achieve speed, security, scalability and at a reduced cost (Haavisto *et al.*, 2019); (Yala *et al.*, 2019); (Dreibholz, 2020).

These “mini data centre” at network edges require virtualization technology that is built with the capacity to programmatically create and manage secure, lightweight, multi-tenant containers that provide services at the edge. This is specifically required in MEC platforms for user application lifecycle management involving various and concurrent multiple asymmetric mobile user storage, computational and analytic requirements. These open source technology building blocks include: Version Control System, Software Configuration Management, Infrastructure as Code (IaC), Containerization and Cloud Computing (Skarpness, 2017).

2.1.1 Cloud computing

Cloud computing is the abstraction of computing resources such as processor, RAM, storage and network services from separate hardware units while presenting as a pool of reusable on-demand shared computing infrastructure that can be rapidly programmatically provisioned and released or manually with minimal management inputs while creating cost benefits and flexibility. The rate of cloud adoption is very high majorly because of its economically viable, cloud infrastructure can be shared across many users and corporations over widely dispersed locations allowing for the reduced cost of services compared to traditional on-premise legacy infrastructure (Nkhoma and Dang, 2013).

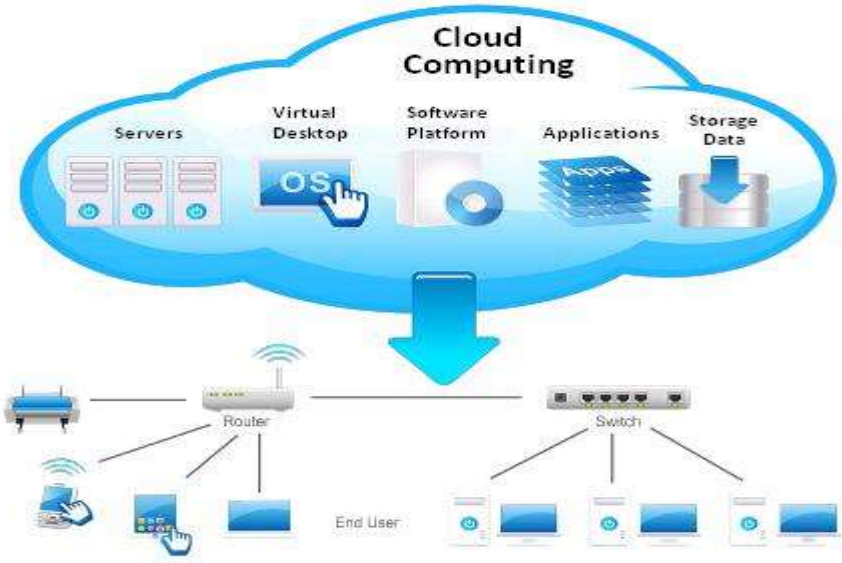

Figure 2.1: Cloud computing (Pritchard, 2010)

Cloud computing is achieved via complex automation system exploring optimization, orchestration, network storage area, virtualization, a variety of state of the art network infrastructure and management technologies indicated in Figure 2.1 are sewn together to create a conglomerate of solutions - networking; compute service; database as a service; object store; metering; data collection service; orchestration; big data processing framework provisioning; dashboard; bare metal provisioning service; image service; identity service; block storage that constitute cloud computing (Openstack, 2019). These

services offered by cloud computing are, to a large extent, limitless and available for mainframe, desktop environment, mobile platform and machine to machine communication. This has made cloud computing ubiquitous, servicing almost every sector of human endeavours from military to entertainment, education sector, commerce and industry, aviation, health, education, manufacturing, transportation, social services, government, research and development, espionage and intelligence community. It could be a private cloud setup by an enterprise solely for use by the enterprise, partners within its enterprise value chain. Cloud deployments solely for use by the third party as “pay as you use” service is a public cloud. Enterprise operating a private cloud but at one point or another depends on the use of the public cloud to augment shortfall of resources, expertise and/or manpower within the enterprise is a form of hybrid cloud deployment. There are situations whereby unused cloud resources are offloaded to be used outside the enterprise value chain, often public cloud is equally hybrid cloud.

2.1.2 Containerization

Containerization is the process by which an *Operating System* (OS) kernel allows running of isolated instances, called containers, within the user-space as depicted in Figure 2.2. A container is a standard unit of software environment that packages up a set of codes and

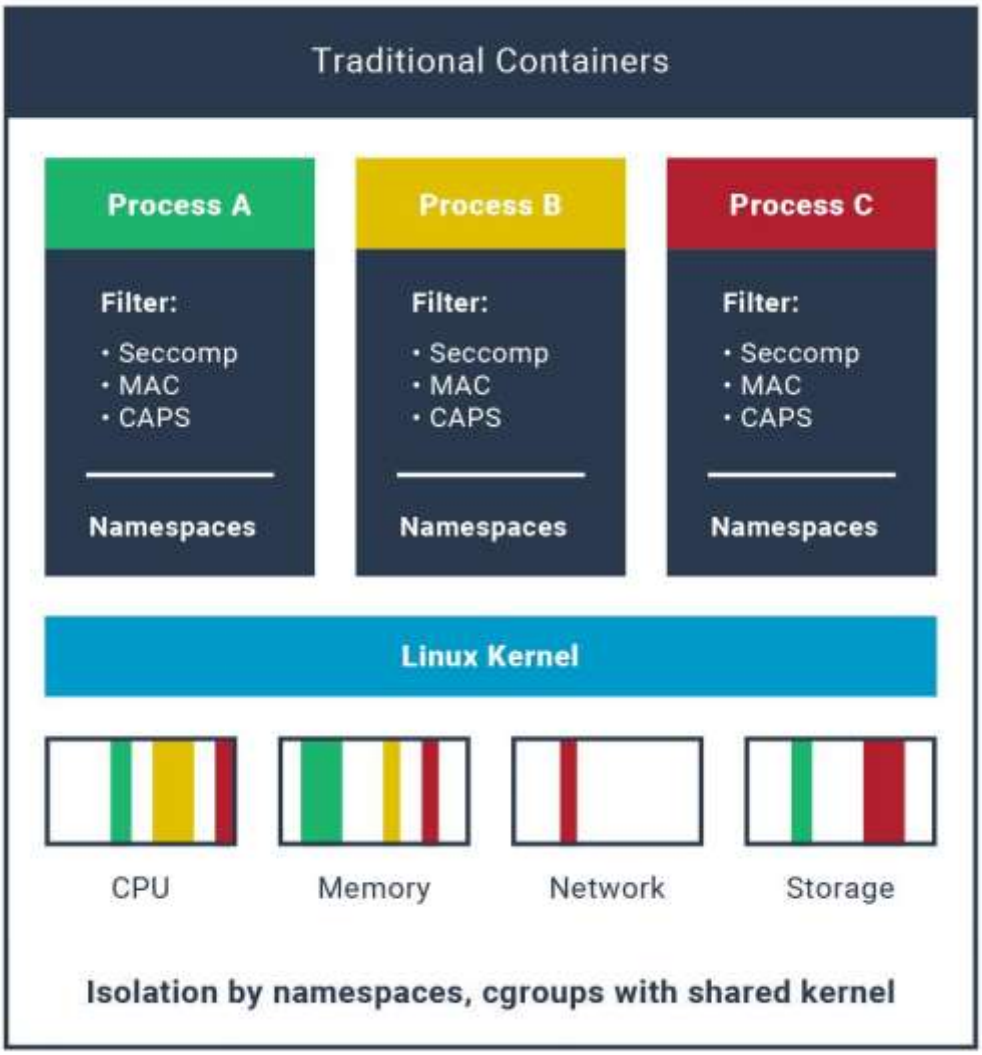

Figure 2.2: Containers running in different namespaces (Kata Containers, 2019c)

all its dependencies together as an abstraction at the OS application layer, therefore, making applications run quickly and more reliably from one computing environment to another facilitating efficient portability of applications. Multiple containers can run on the same IT hardware and share the underlying OS kernel, each running as isolated processes in its own separate userspace. Container images are usually in small sizes, about tens of

megabytes in size, allowing IT hardware handle more applications and require fewer

and Operating systems (Kumar Abhishek, 2020). Containers compared to VMs are more suitable for MEC for the sake of storage limitation and computing resources optimization handling more applications (Figure 2.3).

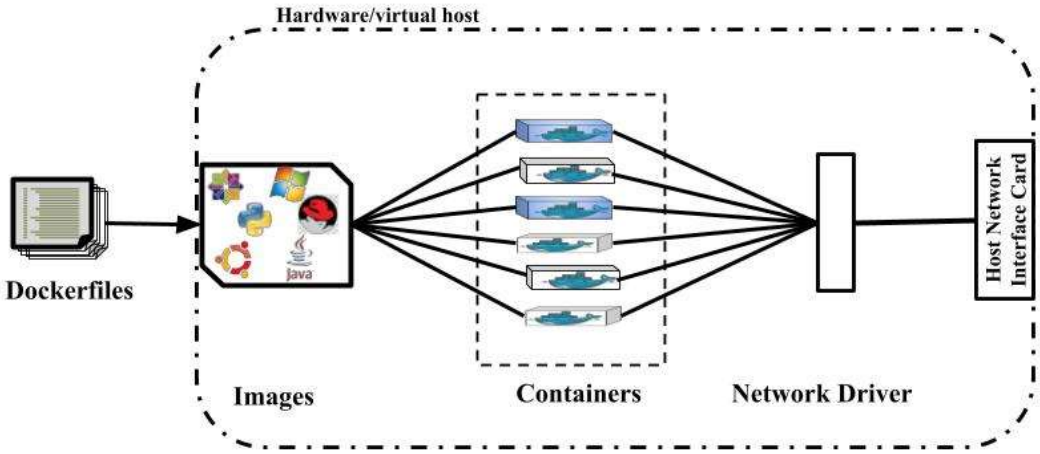

Figure 2.3: Depiction of containerization

Container images are lightweight, standalone, independently executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. A container image become *Containers* at runtime, abstracting an application from its environment and ensuring that it works seamlessly and uniformly despite differences, for instance between development and staging.

The use of Docker containerization, which is the industry standard for containers, is due to its portability allowing containers to run anywhere (Docker Inc., 2019). Docker containers are cross-platform and are capable of running on a variety of computing infrastructures. MEC resources can be allocated to containers for better isolation, performance and allowing for easy collaboration and deployment of applications across different mobile environments. Docker containerization was used in the course of this research effort. Industry standard for containers require that *Docker* containers run on Docker Engine as standard containers which is, so they could be portable anywhere (Erol-

Kantarci and Sukhmani, 2018). This allows for software to be decoupled from the underlying hardware resources, enabling packaged software to executable on multiple hardware architectures providing several benefits such as rapid provision, instantiation, and initialization of virtualized instances (Taleb *et al.*, 2017).

Containerization can provide infrastructure vehicles for migration of legacy applications into the 5G ecosystem. Infrastructure start-up time is short compared with VM; Reduced overhead cost in terms of compute, memory and storage. There is no need creating virtualized components like BIOS/UEFI, drives, separate kernels, RAM, storage, etc.; Heterogeneous Computing - Microservice could be applied to legacy applications like monolith e-commerce application migrated into several manageable containerized manageable microservices application units and deployed separately on different suitable specific compute platforms; field-programmable gate array (FPGA), CPU and graphics processing unit (GPU) but all unit working in a coordinated manner to achieve deliverables. MEC resources can be allocated to containers for better isolation, performance and allowing for easy collaboration and deployment of applications across different mobile environments (Nvidia, 2018). This will provide platforms for transformation of legacy monolithic applications into microservice based made available for use on the go.

Orchestrated containerized MEC will provide efficient infrastructures needed for migration of monolith legacy applications onto 5G service platform. This enables breaking down of large applications into microservice deployable on a large number of interconnected MEC platforms (Alam *et al.*, 2018). In other words, it is the process of decomposing a huge application into smaller manageable units with services, usually

exposed via representational state transfer (REST) application programming interface (API). The containerization ecosystem has become so mature, presenting a whole lot of its orchestrators; Docker Swan, Kubernetes (k8s), Marathon, Amazon container engine. Google container engine (GKE), and Azure service (Piparo *et al.*, 2018); (The Linux Foundation, n.d.); (Sanchez, 2014); (Google Cloud, 2019); (Hoque *et al.*, 2017); (Augustyn and Warchał, 2010). Advancements in Linux containers are making available more secure containers by taking advantage of underlying hardware virtualization capabilities to deliver security very similar to VM but lightweight and fast providing both speed and security for MEC platforms. With measures put in place to foil kernel exploits (blackMORE Ops, 2017) that could be used to compromise containers. Kata containers (Kata Containers, 2019a); (Kata Containers, 2019c) and firecracker (Firecracker, n.d.) are two leading open source containerization alternatives that are able to achieve isolation in containers by running separate dedicated kernel per container while tinkering with container runtime. The experiments in the research made use of Kata containers in effort to deploy a testbed.

2.1.3 Kata container

Kata container runs in a separate dedicated kernel (Figure 2.4), providing isolation of memory, network and I/O and can utilize the underlying hardware-enforced isolation relying on virtualization VT extensions but maintaining compatibility with industry standards like Kubernetes CRI interface, OCI container format, as well as legacy virtualization technologies while consistent standard Linux containers in performance. Kata is written in Rust programming language and it is based on KVM (Kernel Virtual Machine, 2016) hypervisor with option for QEMU/NEMU. NEMU is actually a stripped-down version of QEMU by removing emulation not required thereby reducing the attack

surface. It is more secure than a traditional container by replacing default container runtime, *runC* with Kata-runtime, (Kata Containers, 2019a). Relying on Kata-agent, *shim* for I/O while running *Kata-runtime* instead of *runC* container runtime as available in Docker. Kata containers are as light and fast as containers and integrate with the container management layers—including popular orchestration tools such as Docker swan and k8s —while also delivering the security capabilities of VMs (Kata Containers, 2019c).

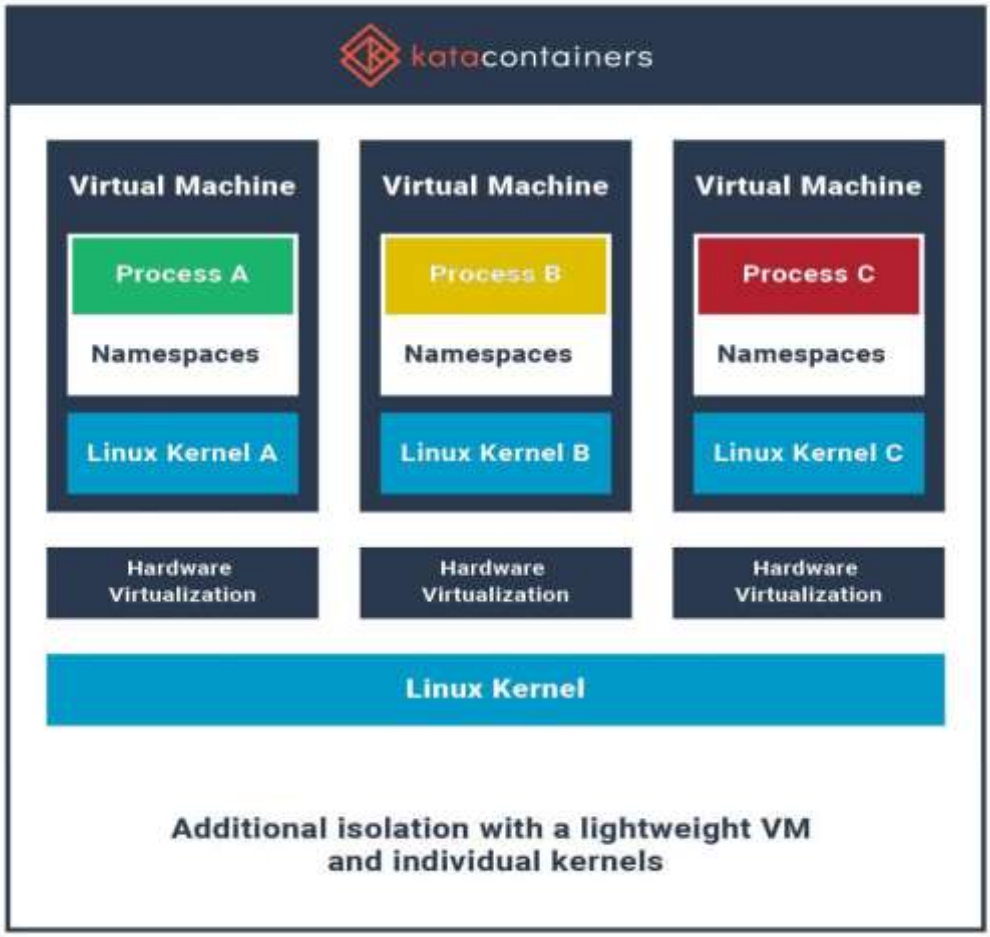

Figure 2.4: Containers running in isolated VM (Kata Containers, 2019c)

2.1.4 Infrastructure as Code (IaC)

Infrastructure as Code, Figure 2.6, is the management of data centres infrastructure life cycle - provisioning, monitoring, patching and termination, in a descriptive model using machine-readable descriptive definition files in conjunction with configuration management tools and source control system rather than manual or interactive

configuration mechanisms. Same IaC process can be engaged several times producing same result as the initial attempt, thus, increasing data centres operators' productivity and transparency facilitating for continuous integration, continuous delivery and continuous deployment. This property of IaC of achieving the same target configuration by deploying unmodified command or set of commands is referred to as idempotency. The implication of being applied several times without any difference in the result beyond the initial application emphasizes consistent, repeatable routines for provisioning and changing systems and their configurations. It is a key role in achieving target deliverables rapidly and at scale expected of deployed 5G MEC servicing billions of mobile devices. To enforce consistency, the target deliverables are represented as codes and any modifications of target deliverables are performed editing the source file rather than the target deliverables in the form of *Ansible-playbook* (Red Hat, Inc., n.d.) and Dockerfile. Changes to infrastructure are made at the source definition files instead of the target MEC applications. The changes could then apply automatically system-wide through a validated process.

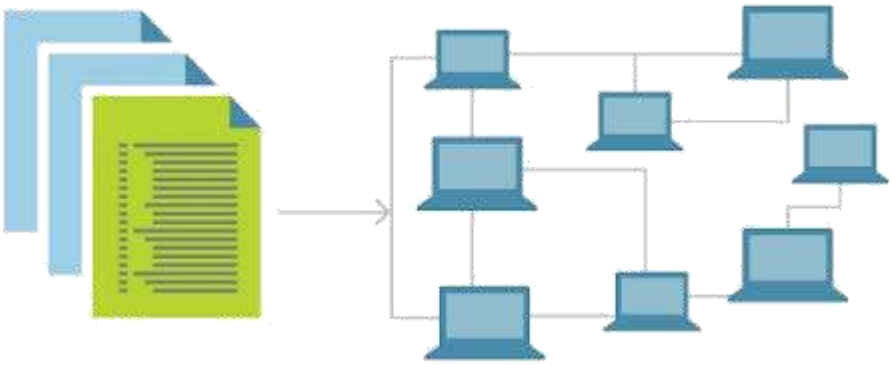

Figure 2.5: Infrastructure as code (Walker, 2018)

A. Software Configuration Management

Software configuration management, leaning towards the perspective of the Software Engineering Institute at Carnegie Mellon University, is an umbrella activity of software engineering practices developed to identify the structure of the product, their types and components, and making them unique and accessible in some form; controlling changes to it throughout the life cycle and the release of product; recording and reporting the status of components and change requests, and gathering vital statistics about components in the product; validating the completeness of a product and maintaining product consistency among the components; managing the construction and building of the product; ensuring the adequate and exact execution of the organization's procedures, policies, and product life-cycle model; while controlling the work and interactions between multiple developers on a product (Bamford and Deibler, 1995).

B. Version Control System (VCS)

It is a component of software configuration management, serving as a file repository/database (Figure 2.5) used to track changes and management by automatic backup and restore, synchronization, short-term/ long-term undo,

changes, track change in file ownership, application sandboxing, branching and merging (Better Explained, 2019). Git was used in the cause for this work. Git is a decentralized VCS for tracking files changes and teamwork coordination allowing for multiple personnel from diverse locations simultaneously working on the same project.

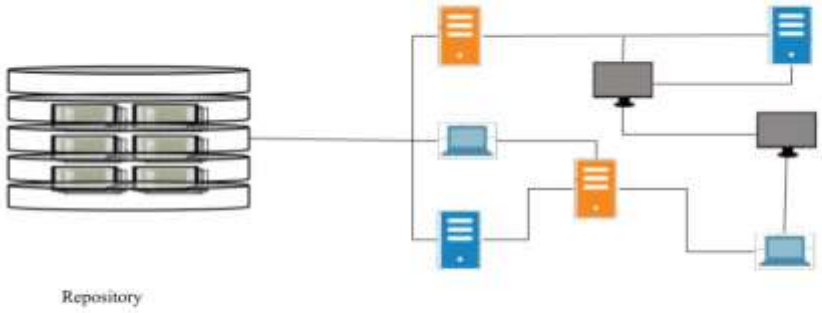

Figure 2.6: Version Control System

MEC infrastructures life cycles can be orchestrated using an Ansible configuration management tool. Ansible, owned by Redhat Corporation. Ansible is an open-source configuration management software which enables deployment and management of a large set of interconnected infrastructures - bare metals, VMs and containers. Ansible connection is based on the popular protocol, secure shell (SSH) and with the ability to fall back to *Paramiko* (Forcier, n.d.) python library. The choice of Ansible over Chef (Chef Software Inc., 2019), Puppet (Puppet, 2019) and Saltstack (SaltStack, Inc., 2019) is due to its scalability, easy installation process and it does not require installation of a specific agent or agents on target client machines. It is a push-based configuration tool. Ansible playbooks, written in YAML, are definition files describing set of specified different actions managed by Ansible. YAML is very close to natural language and easily human-readable making process of data descriptions of infrastructure very simple (both human-

readable and machine-parsable) (Pieplu, 2018). Ansible playbooks are more like instruction manuals for machines to work with, provisioning services on target infrastructures. Moreover, YAML based infrastructure description file organization simplifies the reusability of code.

CP latency is the time between UE idle state and the moment before the start of actual data packet transmission on the network. Having in mind that CP transition is not the transmission of a single message in one direction but messaging interchange consisting a number of messages back and forth to effect state changes. The major factors that reduce CP latency are a rapid deployment of computing infrastructures, reduced number of message interchanges and the location of control functions within the end-to-end network. Obviously, the use of hypertext transfer protocol 2 (HTTP2) (Belshe and Peon, 2015) in 5G will minimize the number of message interchanges. MEC servers are deployed close to CU or at centralised aggregation site with the CU. Orchestrated deployment of containers by using definition files downloadable from repositories close to CU will enable rapid deployment of computing infrastructures.

2.1.5 Multi-access edge computing

Edge computing is a distributed, open IT architecture that made up of decentralised processing power, enabling mobile computing and Internet of Things (IoT) technologies. In edge computing, as depicted in Figure 2.7, data is being processed by the UE device, where data is generated or by a nearby IT infrastructure, rather than being transmitted to a remote cloud location or data centre (Hewlett Packard Enterprise, n.d.). Multi-access edge computing, formerly named mobile edge computing, is defined by European

Telecommunications Standard Institute (ETSI) as a platform that provides IT and cloud computing capabilities within the radio access network (RAN) in close proximity to mobile cellular and non-cellular subscribers- It is network functionality that offers connected compute and storage resources at the fringe of network providing dramatic improvement of mobile network UE experience through near wireline latency (Patel *et al.*, 2014). MEC in 5G could be deployed at the cell sites, next generation NodeB (gNB) or within the data network (DN). The aim is to deliver compute, storage, and bandwidth much closer to UEs where data

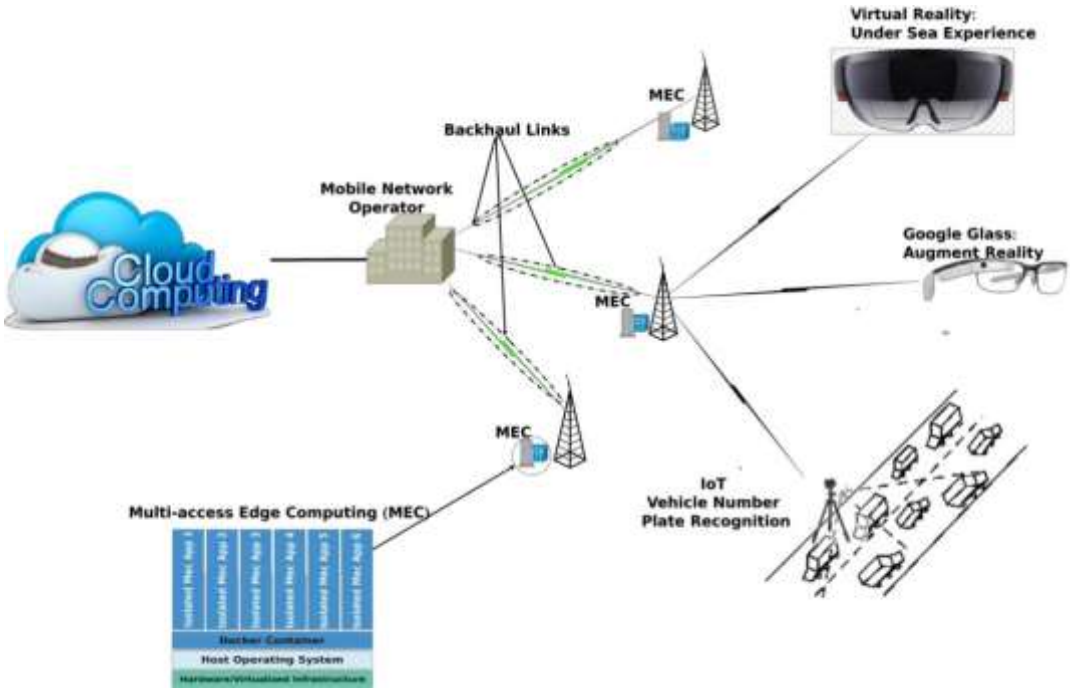

Figure 2.7: Edge Computing

being generated at network edge environment. The network edge environment is characterized by potentially high network latency with low and unreliable bandwidth—alongside distinctive service delivery and application functionality requirements that cannot be achieved relying on a pool of centralized cloud resources located in remote data centres. Satyanarayanan M. et al imagined a future in which cloudlet infrastructure is deployed much like Wi-Fi access points today (Satyanarayanan *et al.*, 2009), here comes MEC.

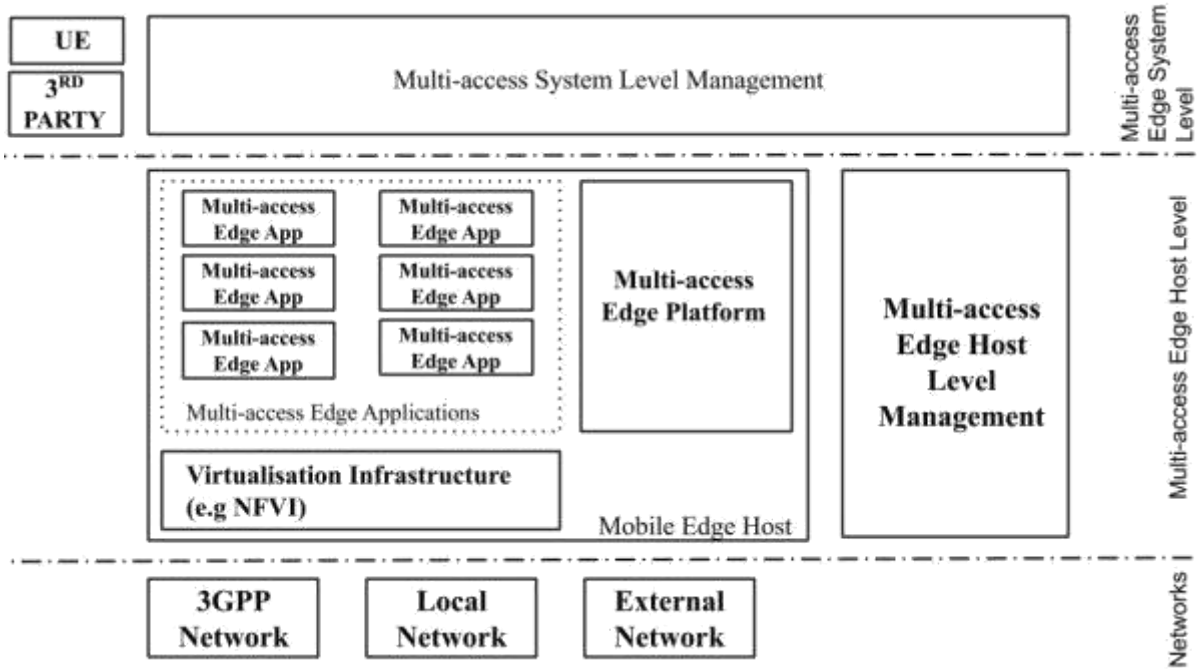

Figure 2.8: ETSI MEC Framework (ETSI, 2019a)

MEC, as indicated in Figure 2.8, is made up of three (3) software entities grouped into system level, host level and network-level entities (ETSI, 2019a). The ability of MEC to render compute, storage and analytics services at the edge while self-managed but fully integrated with 3GPP and external networks is derived from its functional elements which include: mobile edge (ME) host, ME platform, ME applications, and ME systems-level management (ME orchestrator, operation support systems (OSS) and user application lifecycle management proxy), ME host-level management (ME platform manager and virtualization infrastructure manager), user equipment application and customer-facing service portal (European Telecommunications Standards Institute(ETSI), 2019a).

2.1.6 5G Mobile Telecommunication Systems

5G is the 5th generation of cellular mobile communications, succeeding the 4G (LTE/WiMAX), 3G (UMTS) and 2G (GSM) systems. 5G performance targets include energy saving, cost reduction, higher system capacity, high data rate, reduced latency, massive device connectivity, with the capacity speeds up to 20 gigabits per second (Taleb *et al.*, 2017). The mission of 5G is to support the massive explosive evolution of information and communication technology (ICT) and the Internet. The four main functions of 5G are support for communication, computing, control and content delivery (4C) for high-intensity traffic applications like real-time online gaming, AR and VR, ultra-high-definition video streaming (Mao *et al.*, 2017). These applications are highly sensitive to latency and this can have an adverse effect on their performance, significantly degrading with non-uniform delay and throughput.

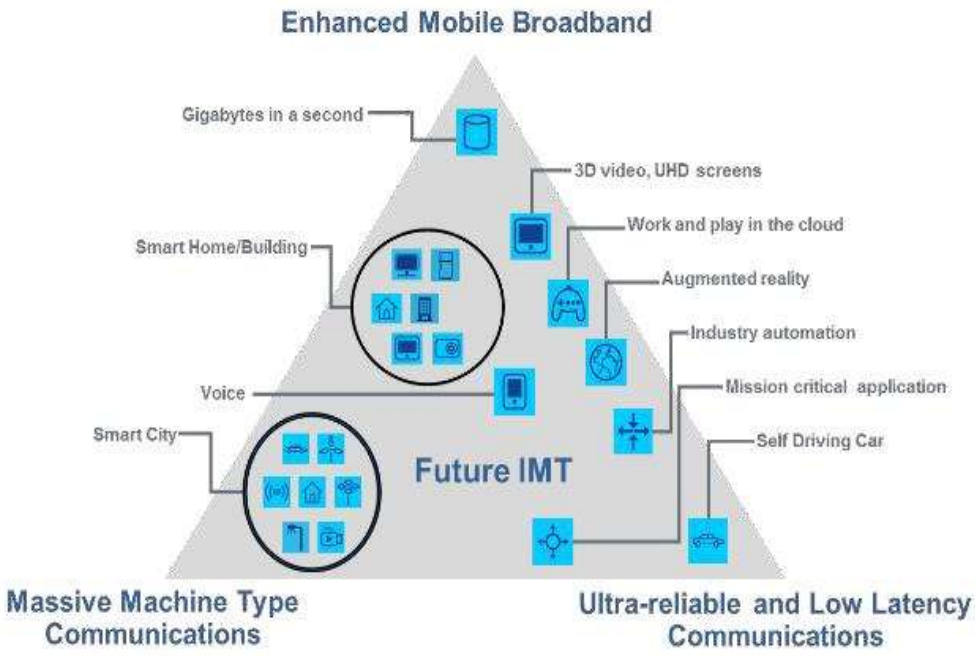

Figure 2.9: 5G use cases (International Telecommunication Union (ITU), 2015)

It is the proposed ITU IMT-2020 technology framework, Figure 2.9, with requirements to provision capabilities that will support enhanced mobile broadband (eMBB), massive 25

machine-type communications (mMTC) and ultra-reliable low latency communications (uRLLC) use cases (International Telecommunication Union (ITU), 2015) & (Mohyeldin, 2016). This paper focus is the delivery of ITU IMT-2020 minimum requirements for eMBB and uRLLC 5G use cases (Mohyeldin, 2016).

Table 2.1: ITU IMT-2020 Requirements (International Telecommunication Union (ITU), 2015) and (Mohyeldin, 2016)

	eMBB	**uRLLC**
User plane latency	4 ms	1 ms 0.5 ms *
Control plane latency	20 ms 10 ms *	20 ms 10 ms *
Peak data rate	20 Gbit/s - downlink 10 Gbit/s – uplink	
User experience data rate (Dense urban environment)	100 Mbits/s - downlink 50 Mbits/s – uplink	
Mobility interruption time	0 ms	0 ms

*3GPP target

Expectedly, 5G networks will be very fast, flexible, reliable and resilient with a one-way trip time of requests corresponding to 1ms for uRLLC use case, taking into account the growing mobile traffic. Successful working of technologies like device-to-device communications, millimetre wave and small cell densification can help to achieve the desired parameters for the 5G networks (Erol-Kantarci and Sukhmani, 2018); (Parvez *et al*, 2018). Depicted in Figure 2.11 is the 3rd Generation Partnership Project (3GPP) 5G reference point architecture.

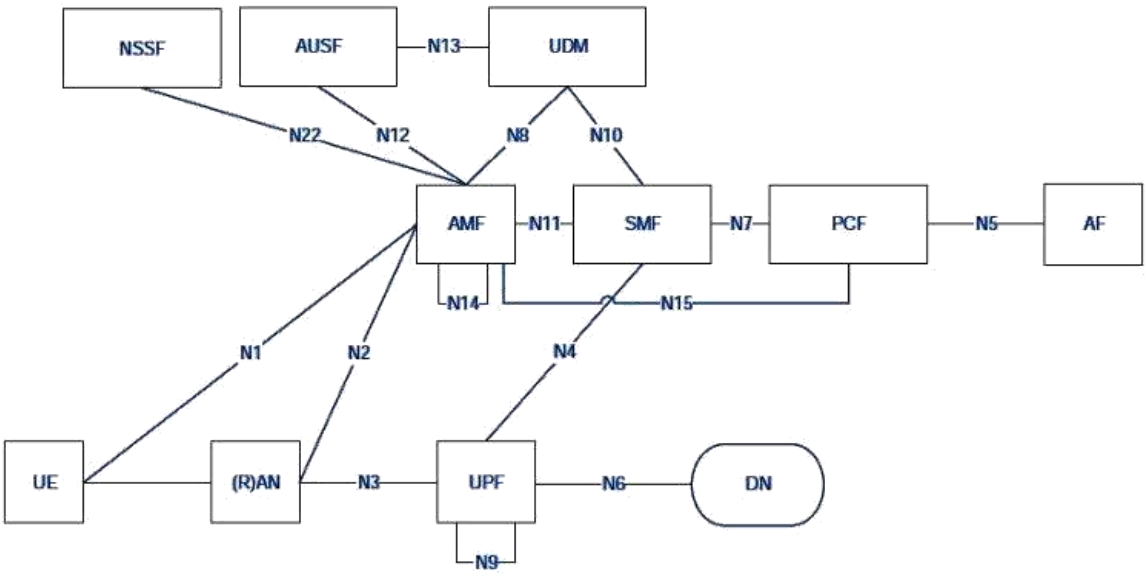

Figure 2.10: 3GPP 5G System Reference Point Architecture (ETSI, 2019b).

The fulcrum of 5G is control and user plane separation (CUPS), decoupling packet data network (PDN) gateway control plane (PGW-C) and PDN gateway user plane (PGW-U) functions of 4G/LTE evolved packet core (EPC) as part of 3GGP release 14 specifications (European Telecommunications Standards Institute (ETSI), 2018). CUPS allow for the separation of user plane function (UPF) from the 5GC out into between RAN and data network (DN) while session management function (SMF) is left within the 5GC network (European Telecommunications Standards Institute (ETSI), 2019b).

2.2 Review of Related Works

Several research works have been proposed to solve the issues of computational resources of mobile devices and energy constraints by offloading computational tasks to tethered infrastructures. Every suggested solution has been geared toward edge computing. Edge computing is the technology that brings together computing into the radio access network (RAN), providing computation, storage and analytics with very low latency while saving a lot of data traffic between network edges and the core network. Enormous network

traffic is expected with billions of devices coming with the rollout of 5G, which has stimulated research into edge computing.

Prominent among the past research efforts is the offloading of computation tasks based on the .NET framework to overcome the energy limitations of handhelds by leveraging nearby computing infrastructure, and Mobile Assistance Using Infrastructure (MAUI) (*Cuervo et al., 2010*). MAUI proposed a solution that enables energy-responsive offload of mobile code to the connected infrastructure but could not support multi-threaded applications and was only applicable to Microsoft .NET Common Language Runtime (CLR) based applications. Satyanarayanan *et al*. (2009) proposed hybrid solution making mobile devices function as thin clients, whereby all significant computation performed by VM in a nearby "cloudlet". It gracefully degrading mobile UE to a fall-back mode whereby significant computation occurring in a distant cloud, or solely its own resources, in the worst case. (Satyanarayanan *et al.*, 2009). Zhang *et al.* (2010) proposed a cloud computing model of a distributed framework that elastically extends application between mobile UE and the cloud. Huang *et al*, (2010) proposed a secure service-oriented mobile ad hoc networks (MANETs) communication framework named MobiCloud providing a platform for cloned image of UE as a virtualized component. Hyrax was proposed in Marinelli (2009), overlaying MapReduce (Dean and Ghemawat, 2004) on a cluster of mobile phones to provide infrastructure for mobile computing. Exploring the now-discontinued Android Dalvik Virtual Machine, a distributed runtime environment aimed at offloading workload from smartphones, Gordon *et al., (2012*) proposed Code Offload by Migrating Execution Transparently (COMET). In Dou *et al.* (2010), Misco, also a MapReduce framework was proposed but for devices with connectivity and supports for python programming language. Likewise, Cloudlet Aided Cooperative Terminals Service

Environment for Mobile Proximity Content Delivery (CACTSE) was proposed in Qing *et al.* (2013) by leveraging cooperating terminals to provide mobile internet content delivery service at the network edges. But this relied on resource-constrained mobile devices resources availability thereby challenging its scalability. In Chun *et al.* (2011) CloneCloud proposed the seamless transformation of mobile device computation into a distributed execution on cloud virtual machine (VM) and mobile devices computing resources. Kosta *et al.* (2012), proposed ThinkAir with capability to dynamically create, freeze, resume, and destroy VMs in the cloud as the need arises, thereby providing an efficient way to perform on-demand resource allocations as well as parallelism capable on-demand resource allocation critical to the management of asymmetric mobile users computational requirements (Kosta *et al.* 2012).

However, containerization is a lightweight alternative to full virtualization of VM. Virtual machines achieve isolation at the machine level and each virtual machine runs its own operating system, whereas containers are isolated within kernel level using individual namespaces. Claassen *et al.* (2016) concluded that containers have an inherent comparative advantage over VMs because of improved performance and reduced boot-up time which have a significant effect on control plane (CP) latency. CP latency is the transition time of a UE to switch from idle state to active state (Parvez *et al.* 2018). Containerization provides a lightweight option to virtualization by improving MEC services portability allowing more mobile user applications to be served by a single MEC Server. Containers have mechanisms for rapid application packaging and deployment to a large number of interconnected MEC platforms (Taleb *et al.*, 2017). The choice of VMs over containers was one of several proposed mechanisms employed to provide such isolation needs by the MEC platform to concurrently fulfil 3GPP-related security

requirements and satisfy concerns related to all the implications of 3GPP security, operator security policies and local regulatory rules (Patel *et al.*, 2014). In a qualitative comparison of VMs and containers in Taleb *et al.* (2017), of all the five indices; control plane latency (provisioning time), computation cost (light/heavy-weight), user plane (processes abstraction), scalability (overhead resource consumption), hardware abstraction (scale of virtualization) and security (isolation MEC applications), containers perform better in all except for security which actually relied on hardware abstraction to provide namespace isolation but not enough isolation as available in virtualization technologies. Docker-based containerization edge computing was introduced in Alam *et al.* (2018), in which a federated approach was proposed involving a layered and modular architecture that is running on cloud, fog, and edge devices representing sensing, mediation, and enterprise layers, respectively and offers containerized services and microservices. Container-based systems compared to VM-based systems are more efficient in reducing the overall for applications execution times; have multiple containers running in parallel as a result of better memory management (Adufu *et al.*, 2015), for creating a highly dynamic system while simplifying management and enables distributed deployments in Alam *et al.* (2018), therefore, more suitable for MEC for the sake of storage limitation and computing resources optimization, serving more applications on the same IT infrastructure. Recent advances in containerization technology has put to pay the security concerns bedevilling the containerization as a technology to improve the MEC performance in 5G deployments. In (Kata Containers, 2019c) containers run in separate dedicated kernels, providing isolation as required in (Patel *et al.*, 2014). Melike Erol-Kantarci and Sukhmani proposed caching strategies at the edge of evolved packet core (EPC) (Erol-Kantarci and Sukhmani, 2018). Caching within the RAN, at the CUs within the NG-RAN rather than at the DN or outside of 3GPP network via GPRS-

Interface/Steering Gi (Gi/SGi) LAN will reduce the pressure on the backhaul networks. Obviously the fronthaul link over the common public radio interface (CPRI) or evolve CPRI (eCPRI) will perform better since it is optic fibre link between DU and RRU/AAU whereas backhaul link i.e. between next-generation NodeB (gNB) could be a satellite, microwave link, or metro ethernet.

For the sake of brevity, the only pre 5G wireless technology reviewed was 4G LTE/EPC end to end network latency. Considering the 4G LTE architecture the transport network is divided into three segments; between the UE and E-UTRAN Node B, also known as Evolved Node B (eNB), eNB to EPC, and EPC to IP peering point.

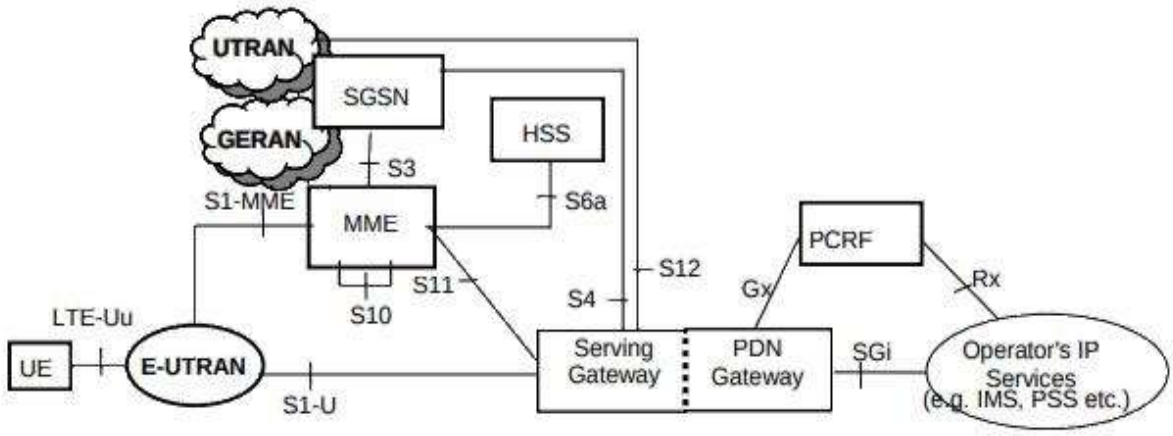

Figure 2.11: 3GPP 4G/LTE EPC reference architecture (*ETSI, 2017*)

The total one-way user plane latency considering 3GPP reference architecture (Figure 2.11) for application deployed on 4G/LTE (*Parvez et al., 2018*),

$$= \text{Radio} + \text{Backhaul} + \text{Core} + \text{Transport} \quad (2.1)$$

where:

T_{Radio} is the one-way packet propagation delay between UE and E-UTRAN

$T_{Backhaul}$ is the one-way packet propagation delay between E-UTRAN and 4G EPC.

T_{Core} is the one-way processing delay with the 4G EPC core network

$T_{Transport}$ is the one-way packet propagation between the EPC and packet data network (PDN). This might include propagation delay to the Internet if service requested by the UE is not within the operator network and has to be sourced from the Internet.

The earlier research efforts to compensate for resource constraint in mobile cellular devices were classified in terms of infrastructural setup and operational techniques as indicated in Table 2.1. From the literature review we came up with this comparative analysis of the augmentation infrastructure in Table 2.2.

Table 2.2: Classification of Augmenting Techniques

Infrastructure	Operational techniques
The research efforts could be classified, in terms of infrastructure setup, into the following categories:	Efforts proposed so far to compensate for resource constraint in mobile cellular devices involved numerous techniques, including:
Cyber foraging Cloudlet Mobile cloud computing (MCC) Multi-access edge computing (MEC)	Offloading of computation tasks to nearby computing infrastructure. Distributed execution frameworks. Mobile thin clients - a client-server arrangement. On-demand resource allocation. Middleware architecture. Mobile device cloning. Representational State Transfer.

Table 2.3: Augmenting Architecture

	Cyber Foraging	Cloudlet	Mobile Cloud Computing	Multi-access Edge Computing
State	Soft state only	Soft state only	Soft and hard state	Soft and hard state
Management	Self-managed: with little to no professional attention	Self-managed: with little to no professional attention	Requires 24X7 professionally operator input	Requires 24X7 professionally operator input
Environment	ad hoc computing facility	"Datacentre in a box" within business premises	Equipment room with power conditioning and climate control	"Datacentre in a box" within decentralised operator site/premises
Ownership	Owned by single or a few groups of users	Decentralized ownership by local businesses	Centralized ownership by Google, Digital Ocean, Amazon, IBM Oracle, Yahoo!, etc..	Decentralized/Centralized ownership by operator/third party
Network	LAN latency/ bandwidth	LAN latency/ bandwidth	Internet latency/bandwidth	LAN latency/bandwidth
Sharing	Numbered users at a time	Few concurrent users	100s-1000s of users at a time	Flexible capacity depending on the configuration
UE Mode	Offloading/middleware	Functions as a thin client	Distributed/Thin client/Cloning	API driven based on HTTP2/JSON

Table 2.4: Summary Literature review

Authors	Models	Methodology	Drawbacks
Satyanarayanan *et al.* (2009)	Cloudlet	A cluster of computers. Nearby mobile devices	High cloud ingress High latency
Marinelli (2009)	Hyrax Cloud Computing	Execution of MapReduce jobs on a cluster of mobile devices	High overhead for devices No developer support
Cuervo *et al.* (2010)	Mobile Assistance Using Infrastructure (MAUI)	Task Partitioning; Joint computation task offloading decision; Allocation of resources for mobile computing access point	Scaling of executions Platform specific
Dou *et al.* (2010)	Misco	Worker - server implementation of MapReduce Framework	Support only Python
Huang *et al.* (2010)	MobiCloud	Secure service-oriented mobile ad hoc networks (MANETs) communication framework	High latency Poor bandwidth utilization

Zhang *et al.* (2010)	Elastic Application Model	Distributed framework that extend mobile into cloud infrastructure	High latency High footprint
Chun *et al.* (2011)	CloneCloud	Offline analysis of the different running condition of process binary; Built database of precomputed partitions; Distributed execution	Required bootstrap for every application; Scalability; No developer support
Kosta, *et al.* (2012)	ThinkAir	Parallelizing using multiple VM; On demand resource allocation	High latency QoS, High overhead
Gordon *et al*. (2*012)*	Code Offload by Migrating Execution Transparently (COMET)	Distributed runtime environment aimed at offloading from smartphones	High execution time, High latency Low resource utilization
Qing *et al.* (2013)	Cloudlet Aided Cooperative Terminals Service Environment for Mobile Proximity Content Delivery (CACTSE)	Cooperating terminals for content delivery service at the network edge	No developer supports Low resource utilization
Alam *et al.* (2018)	Orchestration of Microservices for IoT Using Docker and Edge Computing	Scalable and modular architecture based on containerization Container orchestration	No Isolation No guarantee of idempotence;
Zhang *et al*. (2018)	A Comparative Study of Containers and Virtual Machines in Big Data Environment	Containers are more convenient than VMs, higher CPU and memory utilization, and better scalability	No Isolation

CHAPTER THREE

3.0 RESEARCH METHODOLOGY

In this chapter, 5G network 3rd Generation Partnership Project (3GPP) and non-3GPP transport components specifications were evaluated, and models for MEC deployment scenarios for 5G network were designed. These were carried out to provide the platform to compare MEC applications end to end transport latency in 5G deployment and 4G deployments. This research work leveraged on CUPS, lower layer splits, higher layer splits and 3GPP 5G service-based architecture (SBA) distributed common compute platform (CCP), which permits the location of virtualized network functions (VNFs) in different parts of the network to manage different capabilities. MEC hosts could be located at the centralized unit (CU) connected directly to the packet data convergence protocol (PDCP). This will affect the estimated total UP latency for MEC deployment in 5G.

The end-to-end transport latency has a significant effect on determining the value of UP latency, which in combination with control plane latency, determines the effective end-to-end latency. There was a need for a 5G capable integrated development environment in the quest to investigate deployment of MEC at the 5G CU, but a simulator was not available for this purpose. Instead, leaning on Docker containers, Kata-runtime, and Osbuilder - Kata containers guest OS building scripts, a sandbox application was built to gain insight into the advantages of computing at the network edge compared to computing at remote cloud servers. In order to compare MEC deployments versus MCC deployments of resource-intensive applications, a mobile web application was built, shipped in a secure container image, saved as a code and pushed to a repository. This combination of

servers. The chosen target was web application platforms because of its capabilities of execution on a wide range of devices and mobile environments without modification of application codebase.

The experimental environment (Figure 3.1) incorporated Ubuntu server, Ubuntu Docker image and Docker container engine with its default runtime, *runC,* replaced with Kata-Runtime. Ubuntu Kata container image was created using Osbuilder. On the strength of the evaluation the mobile web application was built and shipped in a secure container image infrastructure as a code (IaC) and pushed it to a repository. Python programming language was used for application logics, results dataset generation, cleaning and graphing; Python Flask for web application backend while the frontend was built using Cascading Style Sheets (CSS), JavaScript and HyperText Markup Language (HTML); Docker for application shipping; and Locust framework for load testing. The publicly available Atlassian Bitbucket and Docker Hub repositories were used for web application code base and container images respectively. The containerized mobile application was deployed using Docker, but Kata-runtime replaced *runC* to ensure app isolation at the kernel level. This ensured the deployment of MEC applications at the speed of containers while maintaining the security available in virtual machines.

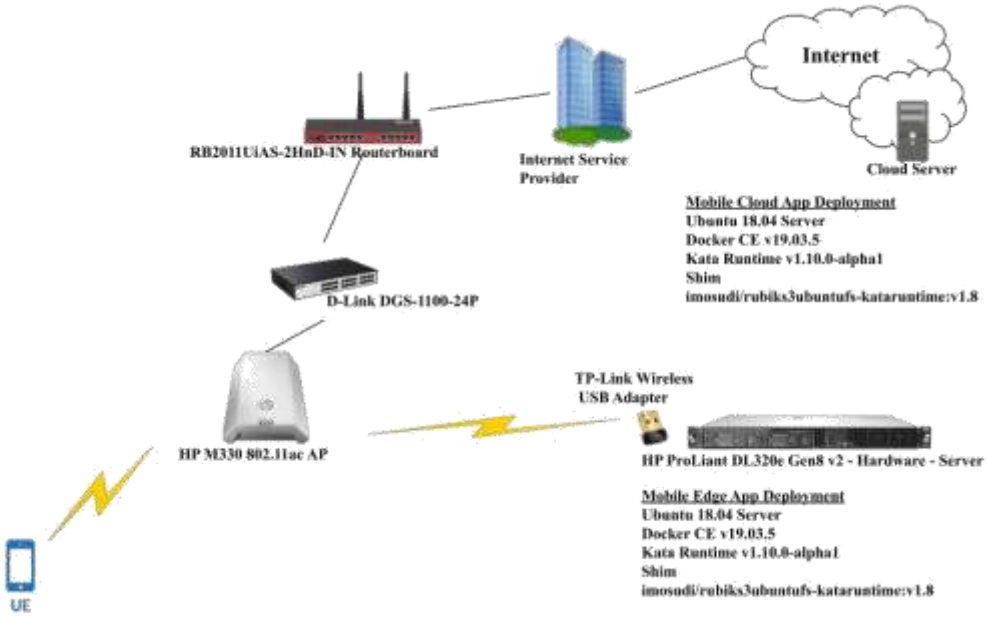

Figure 3.1: Experiment Configuration

The test mobile application was a memory and processor-intensive mobile web application that generated Rubik cubes images and provided breakdown of the cube details - total “cubelets”, faces cubelets and hidden cubelets. Holding the generated graphics in memory while rendering it on the end-user devices. This is comparable to graphics generation and rendering in mobile game applications. Figure 3.2 is a desktop screen display of the mobile web application while Figure 3.3a and b are iPhone X rendering of the mobile application.

Figure 3.2: Rubik’s cube app on desktop

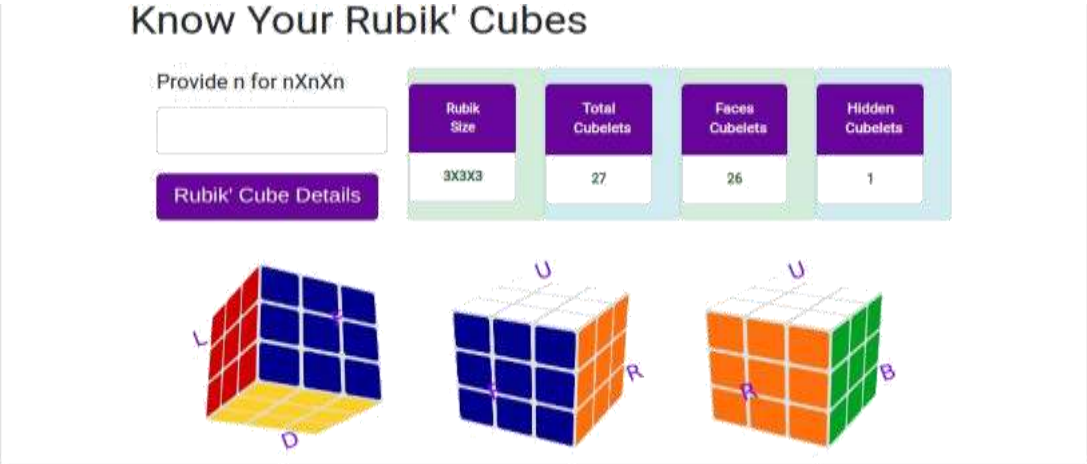

Figure 3.3a: Rubik's cube app on iPhone X (screen 1)

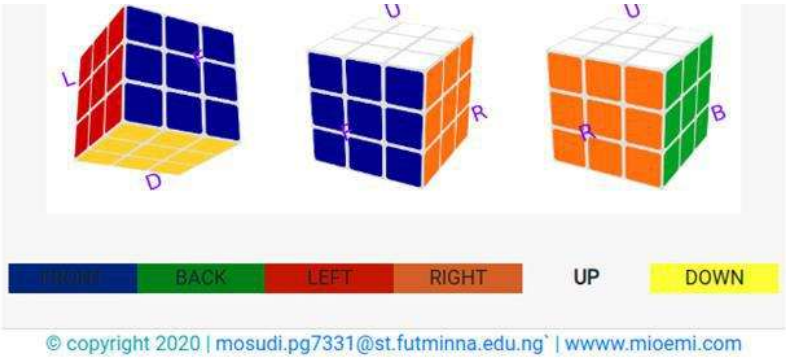

Figure 3.3b: Rubik's cube app on iPhone X (screen 2)

3.1 Transport Network

This focus is on 3GPP 5G service-based architecture (SBA) as presented in Figure 3.4 instead of the standard representation in reference point architecture (Figure 2.10) as MEC is brought into the RAN. 5G network functions interact directly, if required (ETSI, 2019b) by employing RESTful API principle over hypertext transfer protocol version 2 (HTTP/2) using data formatted in JavaScript Object Notation (JSON) and OpenAPI interface definition language. 3GPP 5G SBA core network functions (NF) interactions

occur over a common computer platform (CCP). CCP, mostly represented as a data bus, can actually be fully distributed permitting localization of virtualized network functions (VNFs) in different parts of the network to manage different capabilities.

5G software entities of concern to this research are network repository function (NRF) and the network exposure function (NEF). NEF allows access to shared data layers for MEC. It provides support for event exposure, packet flow description (PFD) management, provisioning information for an external party which can be used for the UE in 5GS, device triggering. It also provides support for transfer negotiation policies for the future background data transfer and provides the ability to influence traffic routing (ETSI, 2019b). NRF offers discovery functions allowing for software entities in the control plane, for example, can identify others and connect directly whenever there is a need to interact. It provides support for register, deregister as well as services updates to NF, NF services and notification to consumers for newly registered NF along with corresponding NF services. NRF provides capability which allows a particular NF service consuming component to discover set of NF instances with specific service or a target NF type. It also enables one NF service to discover specific NF services (ETSI, 2019b*)* while services available will be indexed via network exposure function (NEF) in the control plane (CP).

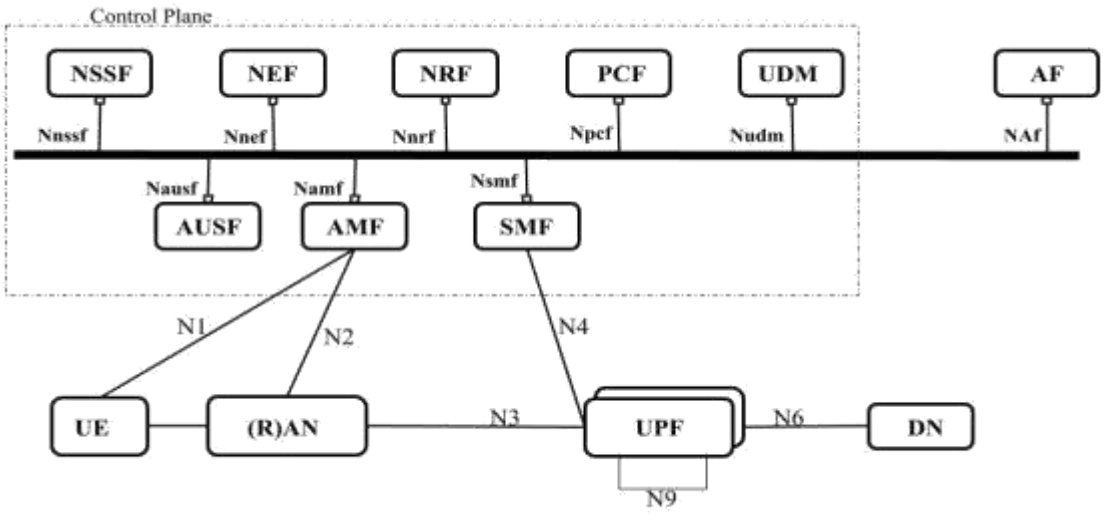

Figure 3.4: 3GPP 5G System Service-Based Architecture (ETSI, 2019b)

3.2 NG-RAN Decomposition

Considering the functional decomposition of NG-RAN achieved with gNB-CU - gNB-DU split connected together over F1 logical interface (Sutton, 2018a); (Shew, 2018); (ETSI, 2018c). F1 interface is the medium for interconnecting a gNB-CU and a gNB-DU of a gNB within an NG-RAN, or for interconnecting a gNB-CU and a gNB-DU of an en-gNB within an E-UTRAN. The F2 interface (eCPRI/CPRI/NGFI) is non-3GPP specified interface connects gNB-DU with active antenna unit (AAU), radio unit (RU) or remote radio unit (RRU) when deployed over distance (Ericsson AB, Huawei Technologies Co. Ltd., NEC Corporation and Nokia, 2019); (Smith *et al.*, 2018); (Knopp *et al.*, 2017). NG logical interface connects a set of gNB interconnected via Xn logical interface within an NG-RAN to the 5G core network (5GC). F1, F2 and NG interfaces constitute midhaul, fronthaul and backhaul networks, respectively.

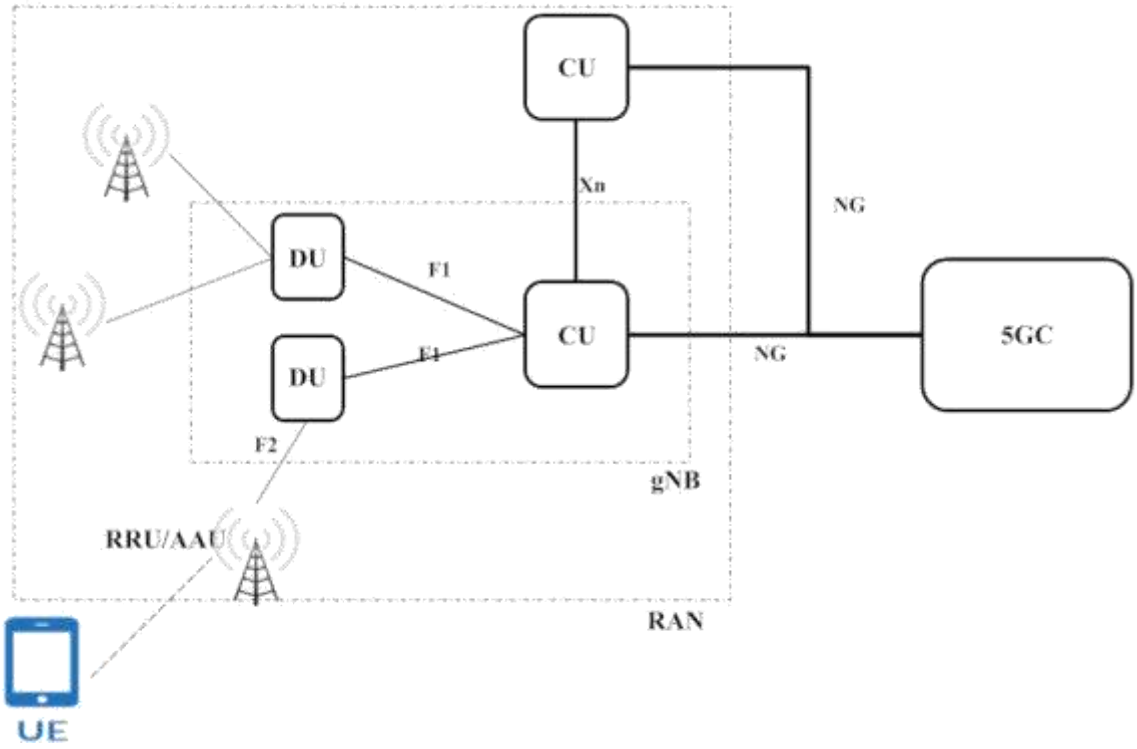

Figure 3.5: Functional Decomposition of NG RAN - Adapted from (Shew, 2018).

Considering eCPRI, this functional decomposition of RAN will permit the location of MEC server close to the CU sending and receiving user plane packet data traffic over F1 and eCPRI providing high bandwidth capacity at very low latency capable of supporting eMBB and URLLC use cases (International Telecommunication Union(ITU), 2015); (Mohyeldin, 2016). Decomposition of RAN permits distance separations between CU, DU and RRU/AAU while allowing for C (cloud, cooperative and centralized) - RAN configurations (Murphy, 2015); (Checko, 2016), (Huawei, 2017); (Kitindi, Fu, Jia, Kabir and Wang, 2017). The functional decomposition of RAN may not be as simple as depicted in Figure 3.5, but it depends on the level of applicable split options as available in Figure 3.6a and b, as this will determine the level of coordination capabilities that can be delivered by a C-RAN.

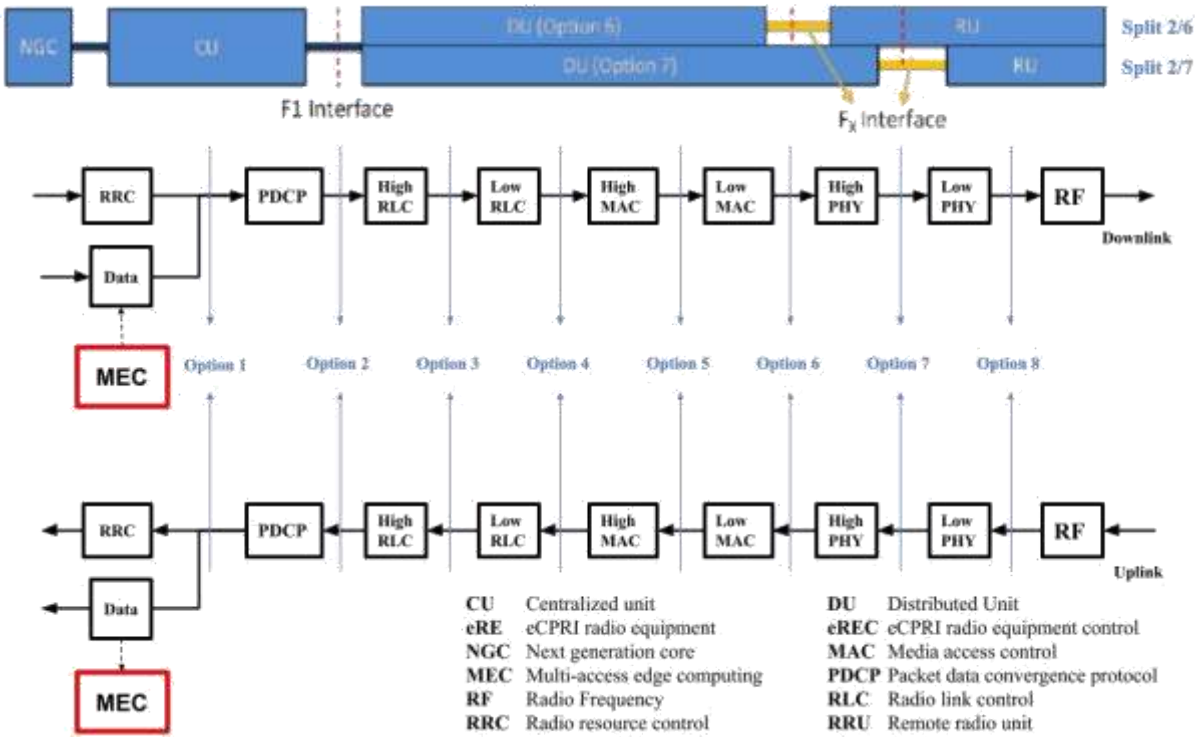

Figure 3.6a: RAN protocol split (Shew, 2018); (3GPP, 2017) with the addition of MEC

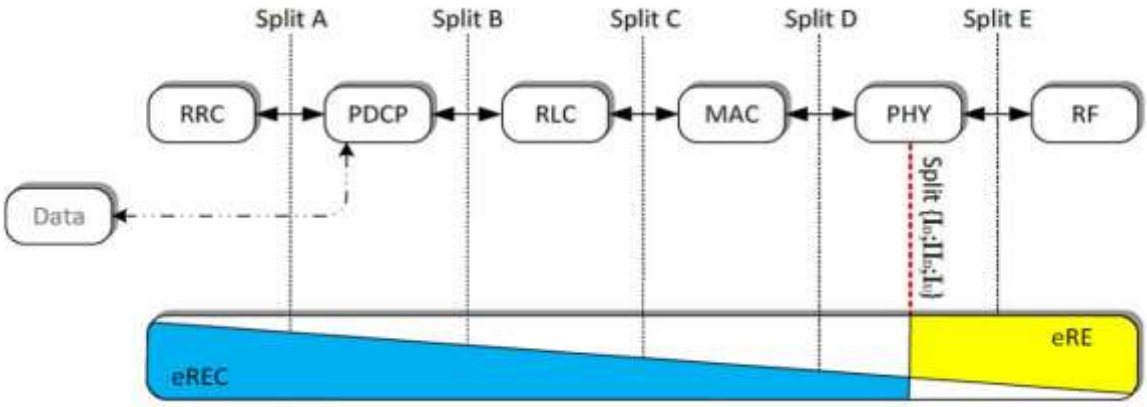

Figure 3.6b: eCPRI Functional decomposition on RAN layer level

3.3 5G MEC Models

Higher layer functional split (HLFS) option 2 for the midhaul and lower layer functional split (LLFS) option 7 for fronthaul will permit four RAN deployment scenarios *(Ericsson AB, Huawei* Technologies Co. Ltd., *NEC* Corporation and *Nokia,* 2019*)* and as a result four MEC deployment scenarios:

1. Independent RRU, DU and CU/MEC locations;
2. DU and CU/MEC co-located with distance separated RRU;
3. RRU and DU co-located with distance separated CU/MEC;
4. RRU, DU and CU/MEC integration within a single co-location.

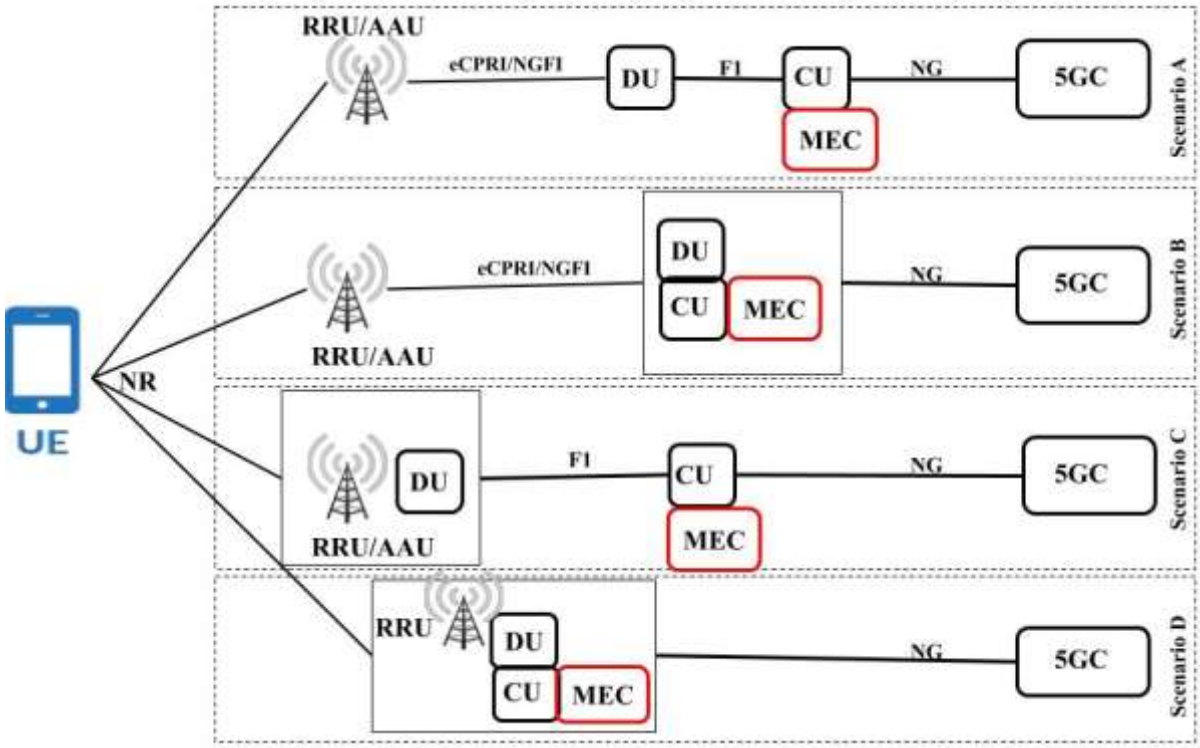

Figure 3.7: 5G MEC deployment models

3.4 Latency

Recall Equation 2.1

$$= \text{Radio} + \quad \text{Backhaul} + \text{Core} + \text{Transport}$$

Nothing is lost but everything is gained by modifying Equation 2.1 for LTE/EPC (Figure 3.8) producing Equation 3.1

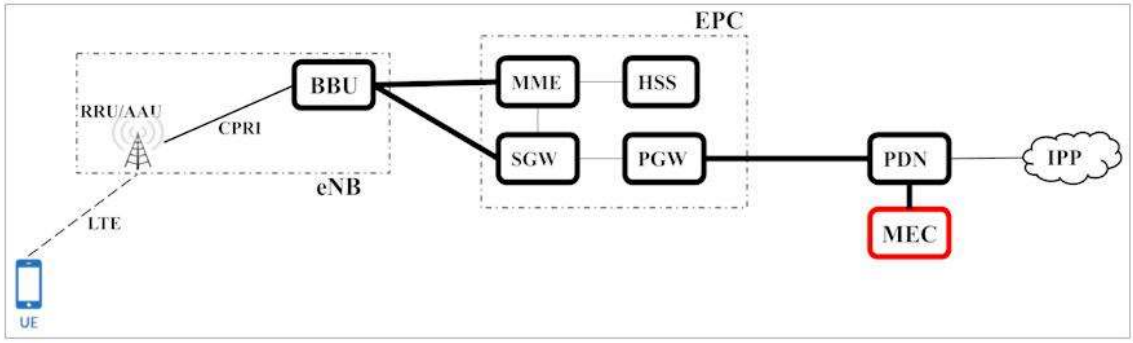

Figure 3.8: 4G Long-Term Evolution/Evolved Packet Core

$$= \text{LTE} + \text{EPC} + \text{Transport} \tag{3.1}$$

where:

T_{LTE} is the one-way packet propagation delay between UE and eNB, including packet processing time within UE and eNB.

T_{EPC} is the one-way packet propagation delay between eNB and EPC, including processing delay with the core network.

$T_{\text{Transport}}$ is the one-way packet propagation between the EPC and packet data network (PDN). This might include propagation delay to the Internet if service requested by the UE is not within the operator network and has to be sourced from the Internet.

Latency values will vary from one MEC deployment scenario, as presented above, to another and quantifying all the parameters is challenging due to differences in

performance of equipment along the end to end 5G network; from DU to CU, and all the way to MEC host. However, the assumed 1-way latency range between 5 and 8ms between CU and DU and in essence, 8ms network latency between CU and DU eases the co-location of the CU with MEC.

The total one-way user plane latency becomes:

$$= \text{NR} + \text{DU} + \text{CU} + \text{Transport} \quad (3.2)$$

where:

T_{NR} is the one-way packet propagation new radio (NR) delay between UE and DU, including packet processing time within the UE and DU.

T_{DU} is the one-way packet propagation delay between DU and CU, including processing delay with the CU.

T_{CU} is the one-way packet propagation delay between CU and 5GC is, including processing delay within the 5GC.

$T_{\text{Transport}}$ is the one-way packet propagation between the 5GC and data network (DN). This might include propagation delay to the Internet if service requested by the UE is not within the operator network and has to be sourced from the Internet.

Deployment of MEC in all the four scenarios in the model above provided the option direct connection between MEC and the CU. The total one-way user plane latency becomes:

$$= \text{NR} + \text{DU} + \text{CU} \quad (3.3)$$

The efforts to simulate 5G network transport latency to determine UP latency is being challenged by the fact that 5G next-generation (NG) core and RAN technologies are, to a large extent, still on white papers but enough specifications have been defined and written

about it. Therefore, 3GPP and non-3GPP specifications were the major sources of data for the research evaluation of UP latency for the proposed MEC deployment. The transport specifications for 5G interfaces were analysed and presented in Table 3.1. Using Equation 3.3, the transport specifications were applied on the four proposed network models in Figure 3.7 to arrive at the latency values presented in the results section of this thesis.

Table 3.1: 5G Interface specifications *(European Telecommunications Standards Institute* (ETSI), 2018c)

Network	Reach distance	Latency (1-way)	Capacity requirements
New radio (NR)		4 ms eMBB 2 ms URLLC	
Fronthaul (eCPRI)	1 ~ 20 km	< 100 μs	10Gb/s-825Gb/s
Midhaul (F1)	20 ~ 40 km	1.5 ~10 ms	25Gb/s-800Gb/s
Backhaul (NG)	5-80km (Aggregation) 20~300km (Core)		CU: 10Gb/s-25Gb/s CN: 100+Gb/s

3.5 Experiments

The goals of these experiments are to evaluate the increased mobile application responsive and overall Quality of User experience by the use of containerized application infrastructures deployed at the network edge in terms of latency. In an attempt to achieve the above stated mature open source technology tools were used to deploy new use cases for 5G networks.

3.5.1 Tools

The tools include:

1. Ubuntu server
2. Wi-Fi access point
3. Layer 3 network switch
4. Docker Container - community edition (Docker-CE)
5. Kata Container
6. Kata Runtime
7. Shims
8. Goland
9. Debootstrap
10. Python
11. Git
12. Osbuilder
13. Locust

Ubuntu 18.04.3 LTS server was installed on an HP ProLiant DL320e Gen8 v2 server, updated and upgraded to the most recently available updates. The followings were also installed were: Git, Wget, Curl, Snapd, Vim, Golang and Debootstrap. All these provided the mechanism to compare the estimated user plane latency values with those of known low latency use case requirements: VR/AR (7-12ms); tactile Internet (<10ms); Vehicle-to-Vehicle (< 10ms); Manufacturing & Robotic Control / Safety Systems (1-10ms).

Containerized web application was deployed for both cloud computing and network edge (Figure 3.9) scenarios. Load tests were conducted against defined performance metrics providing tons of results, which were further analysed and presented in graphs for easy comparative evaluations to validate the advantages of MEC over MCC.

network edges was made possible due to 5G network functional decomposition as well as Control Plane and User Plane Separation in same. Simulations were conducted for deployment labs for on-demand containerized applications based on IaC and version control systems. Using Locust, the application for the two deployments - edge and cloud, were programmatically load tested with requests to generate and render random size of “N by N by N”, with N ranges between three (3) and fourteen (14)

3.5.2 Experimental setup

The experiment setup processes included installation of tools and dependencies, initial container image build procedures and web application packaging procedures. Versions of application image for app deployment, code versions and shipping of the built images to repository for future use and the application container deployment were created. All of the processes were performed via BASH terminal, command language interpreter for the GNU operating systems. The setup involved running of the Linux commands in Appendix C.

3.5.3 The load tests

In the tests, scenario were created to provide the mechanism to compare the estimated user plane latency values with those of known low latency use case requirements: VR/AR (7-12ms); tactile Internet(<10ms); Vehicle-to-Vehicle (< 10ms); Manufacturing & Robotic Control / Safety Systems(1-10ms).

Using Locust, an open source event-based user load testing tool framework, a total of 3900 user requests by 30 unique users was simulated at the rate of 3 user requests per second (Appendix B). Rendering Rubik’s cubes of random sizes between three and fourteen while also in between, rendering other views of the mobile application.

Simulating a real-world scenario, where often, users are idle, figuring out what next to do with an application, the idling time was set at random values between 90 and 140 seconds.

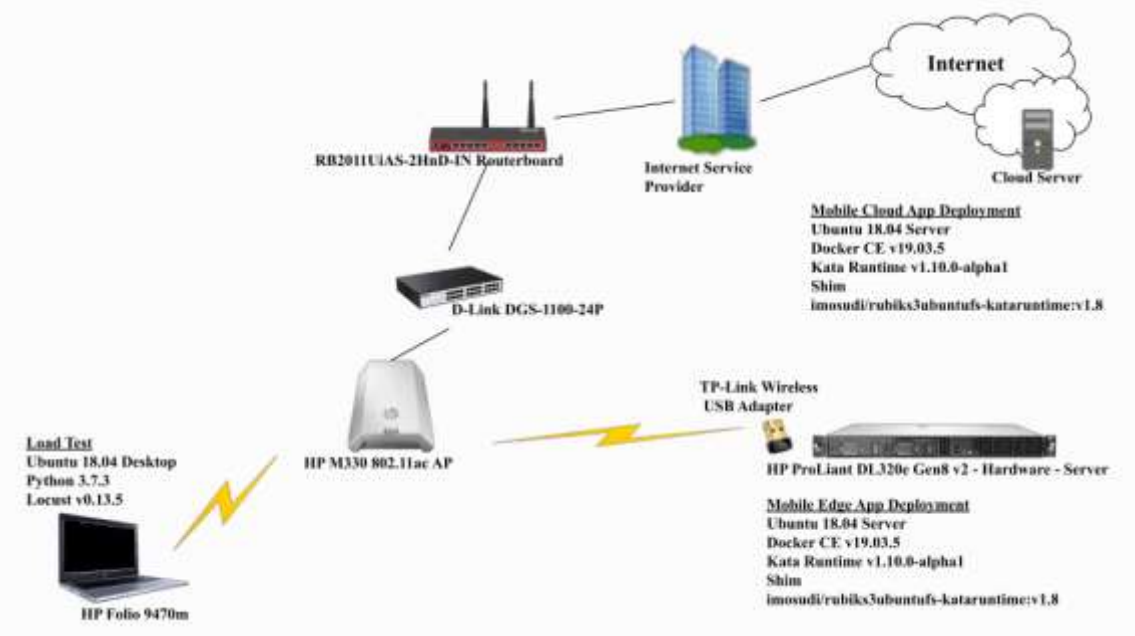

Figure 3.9: Experimental setup

The whole of the user behaviours was described in Python code and the experiments for each scenario lasted for roughly 120 minutes. Data sets from the experiment were retrieved in comma-separated values (CSV) file format. The relevant results from the experiments included the request rate, request failure rate, application download size, and minimum/maximum/median response time.

CHAPTER FOUR

4.0 RESULTS AND DISCUSSION

4.1.1 Low latency models developed for MEC deployment for 5G:

There were two sets of results being considered in this research work. The first set are the results retrieved from the application of 5G transport interface specifications on the four proposed MEC deployment models. Evaluating Equation 3.3 for eMBB for the proposed four deployment scenarios (Figure 3.7) by applying 5G specification values in Table 3.1:

(i) SCENARIO A

Scenario A features dual-functional splits in the RAN. The RRU/AAU at the cell site, DU at the aggregation site while CU and MEC deployed at the edge site (Figure 4.1a). Every interface interconnecting all 5G functional split contributed to the UP latency, optimally this prototype deployment produced an estimated round-trip time (RTT) value of 11.2ms.

$$= _{NR} + _{DU} + _{CU} \quad (3.3)$$

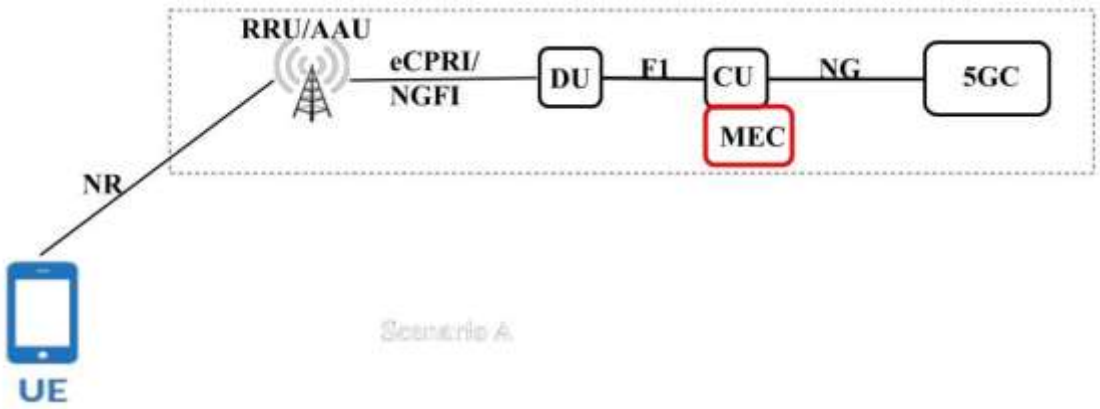

Figure 4.1a: Scenario A

Minimum

$$T = 4000 + 100 + 1500\ \mu sec$$

$$= 5600\ \mu sec$$

$$= 5.6 \times 10^{-3}\ s$$

Maximum

$$T = 4000 + 100 + 10000\ \mu sec$$
$$= 14100\ \mu sec$$
$$= 14.1 \times 10^{-3}\ s$$

(ii) SCENARIO B

Considering scenario B, this is a centralized RAN MEC deployment without F1 interface but employed only the lower layer functional split having DU, CU and MEC co-located at the edge site while RRU/AAU connected via eCPRI interface is deployed at a remote cell site (Figure 4.1b). Optimally, the RTT is 8.2ms.

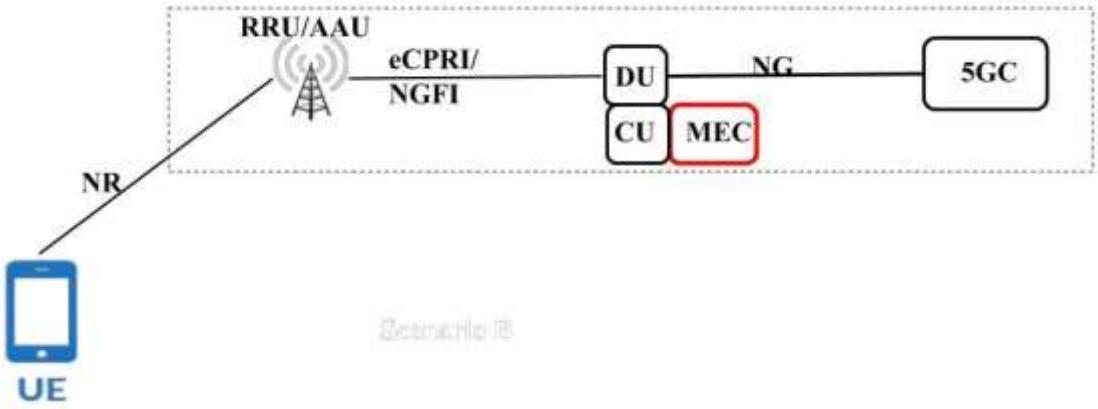

Figure 4.1b: Scenario B

$$T = 4000 + 100\ \mu sec$$
$$= 4100\ \mu s$$
$$= 4.1 \times 10^{-3}\ s$$

(iii) SCENARIO C

Scenario C features only 3GPP upper layer single functional split between DU deployed with RRU at the cell site and CU with MEC at the edge site (Figure 4.1c) with optimal RTT 11ms.

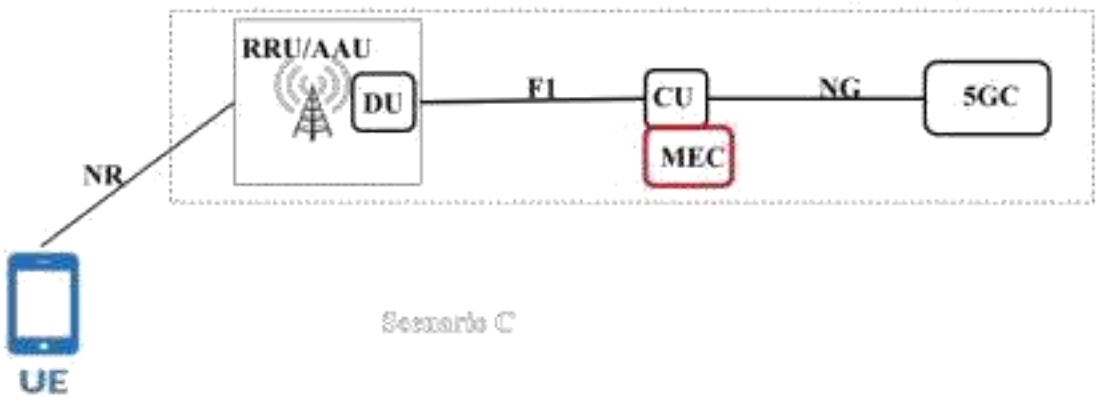

Figure 4.1c: Scenario C

Minimum

T =4000 + 1500 μs

= 5500 μs

= 5.5×10^{-3} s

Maximum

T =4000 +10000 μs

= 14000 μs

= 14×10^{-3} s

(iv) SCENARIO D

Scenario D is a monolithic RAN. This setup is most applicable to 5G MEC deployment for microcells, picocells and femtocells (Figure 4.1d) with RRT of 8ms.

$$= NR + DU + CU \quad (3.3)$$

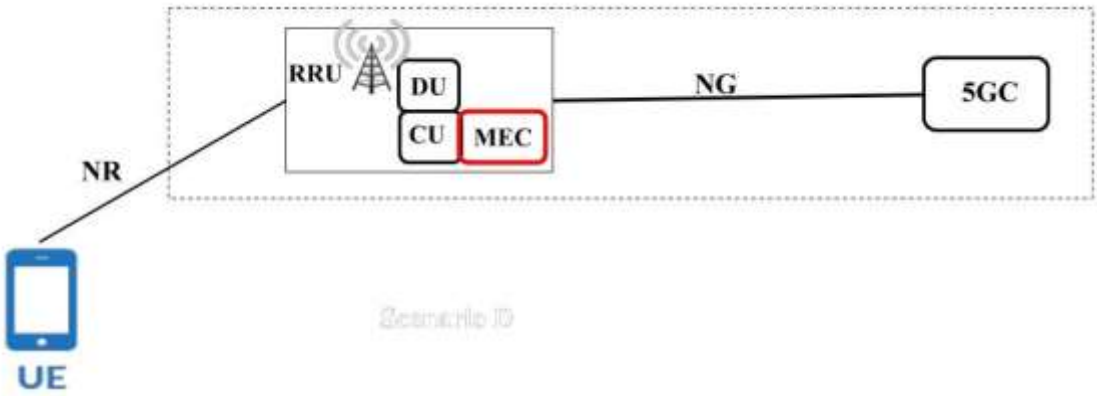

Figure 4.1d: Scenario D

T =4000 μs

= 4000 μs

= 4×10^{-3} s

4.1.2 Deployment of secured containerized mobile application for both edge and remote cloud servers:

The second set of results were retrieved from the experimental load test of the secured containerized mobile web application deployed on MCC and MEC as depicted in Figure 4.2a and Figure 4.2b respectively.

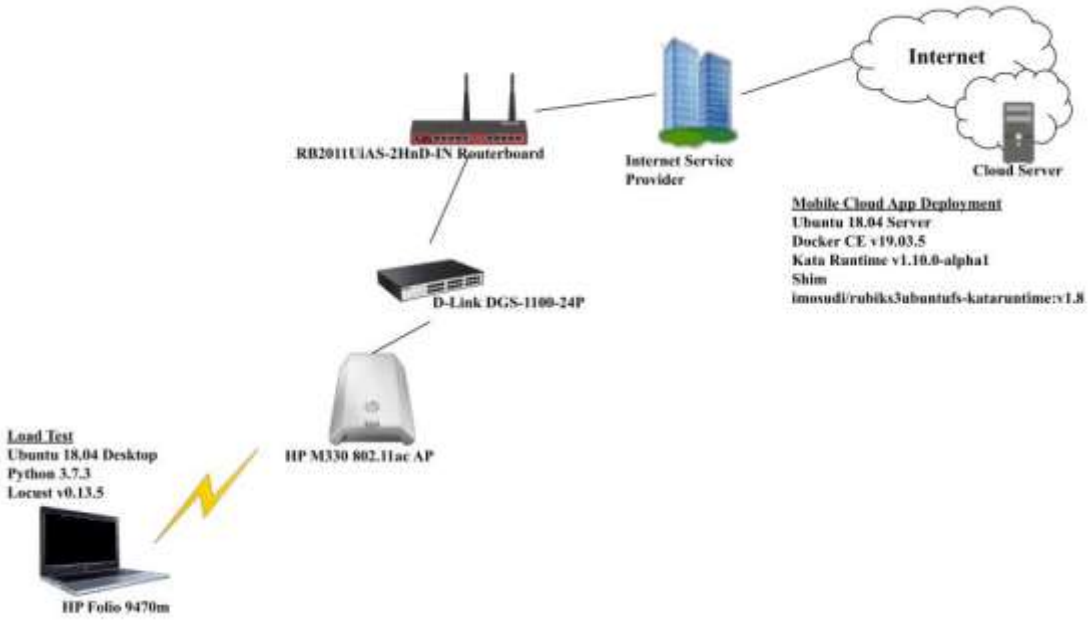

Figure 4.2a: Experiment - MCC test scenario

Figure 4.2b: Experiment - MEC test scenario

The results from the experiments included the total requests per second (req/s) made to the application deployed, request time stamps, the number of requests per second, requests failure per second (req/s), minimum, median and maximum application response time, average application data download size, 50 percentiles and 95 percentile application response time among other result parameters. All these for both edge site and cloud application deployments. Each time, the experiments lasted for about 2 hours. The results from the experiment span 3955 rows by 22 columns (Appendix D). It was observed that the application response time and the amount of downloaded application data follow the same pattern corresponding to the size of the Rubik's cube being rendered. Likewise, failures were more prevalent with the mobile cloud computing deployments compared with relatively stable edge computing deployments.

The rate of requests for both the edge site and cloud deployments were plotted using the procedures presented in Appendix A. The intention to simulate a random Rubik's cube size between values of three and fourteen did pay off, the request rate for both edge and cloud deployments were about the same for both experiments as indicated in Figure 4.3. This justified a fair comparison of both deployments.

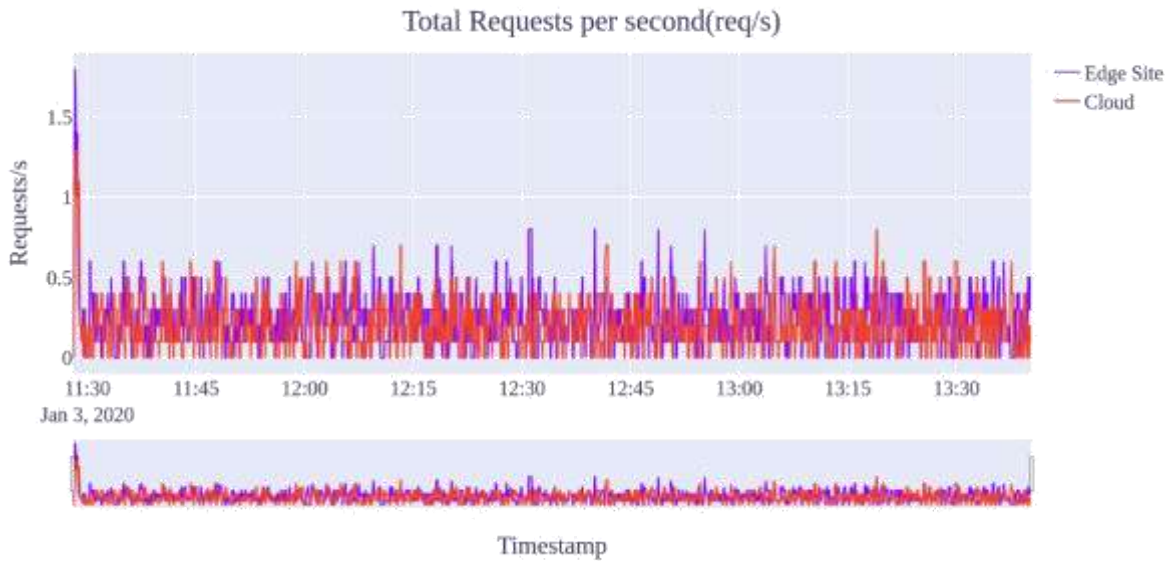

Figure 4.3: Total request rate/sec

This is followed by the request failure rate for both deployments. Figure 4.4 shows that failures were more prevalent in the cloud deployments. Application maximum and median response time in seconds for both edge and cloud deployments were presented in Figure 4.5.

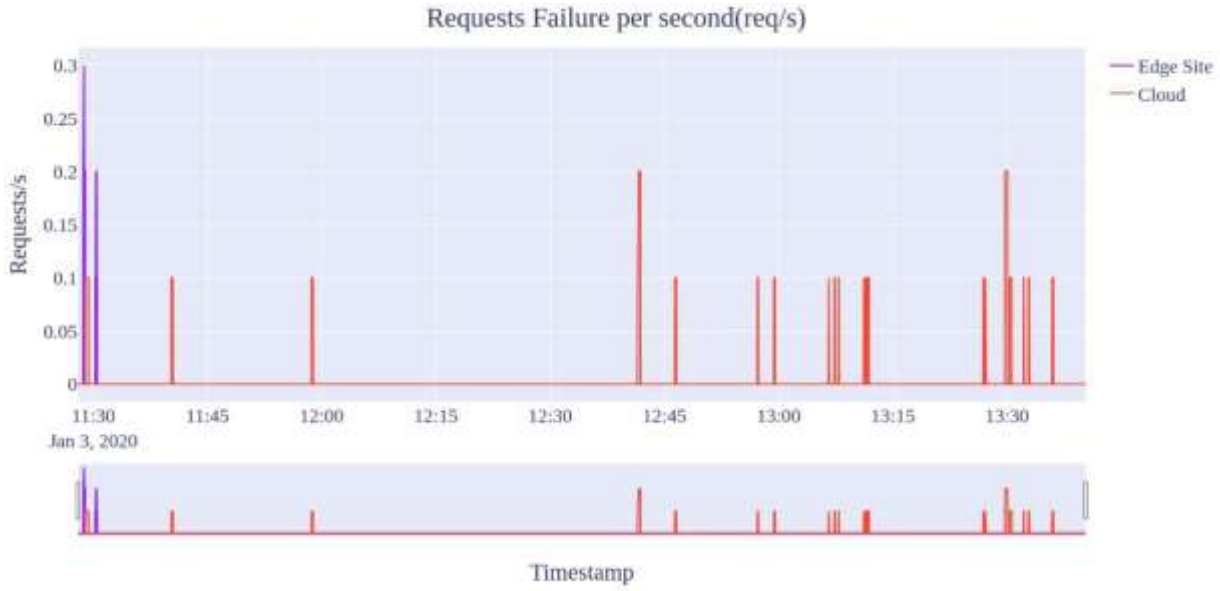

Figure 4.4: Total failed request rate

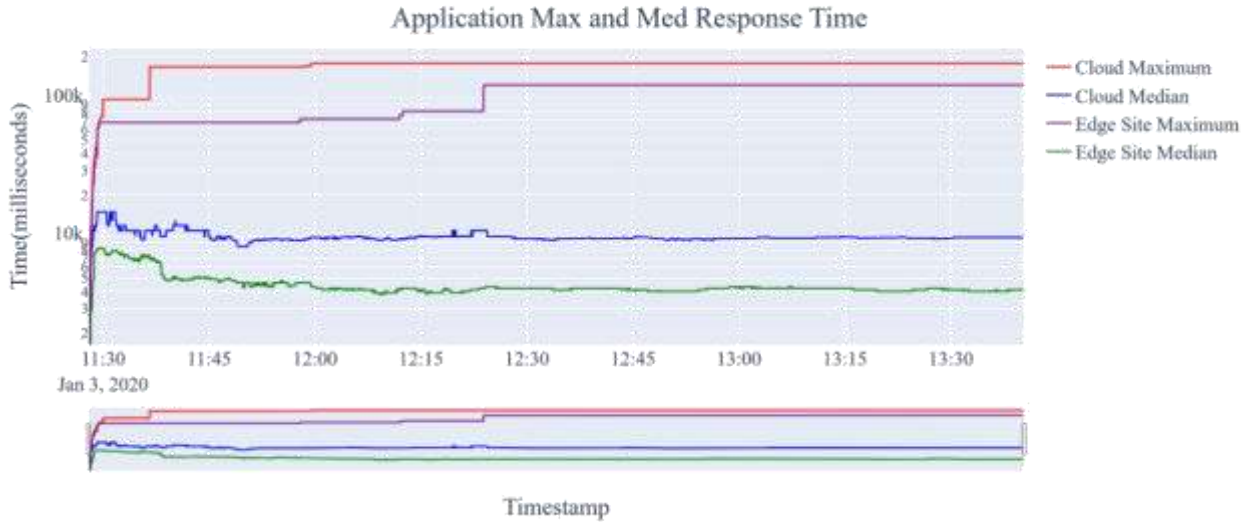

Figure 4.5: Application maximum and median response time

The minimum application response time in milliseconds during the experiment is presented in Figure 4.6 for both cloud and edge deployments. The average application data download for both deployments are presented in Figure 4.7. The same application version was deployed for both scenarios and with the convergence in the application request rates for both edge and cloud deployments as shown in Figure 4.3, these had a significant effect on the amount of application data downloaded for both scenarios during the experiments.

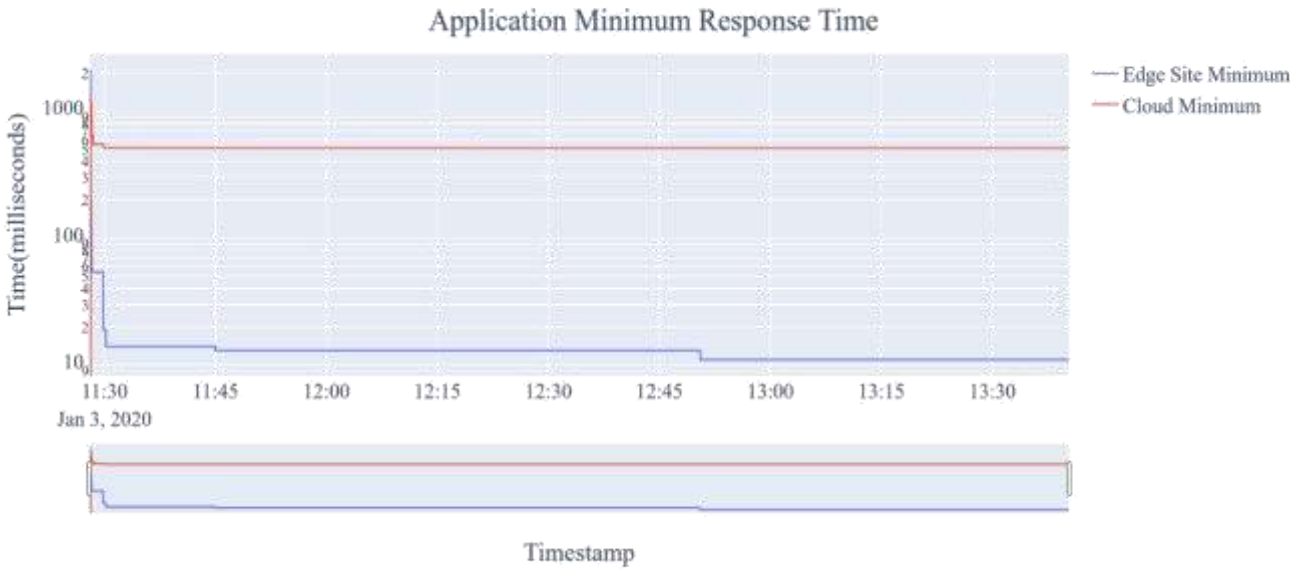

Figure 4.6: Application minimum response time

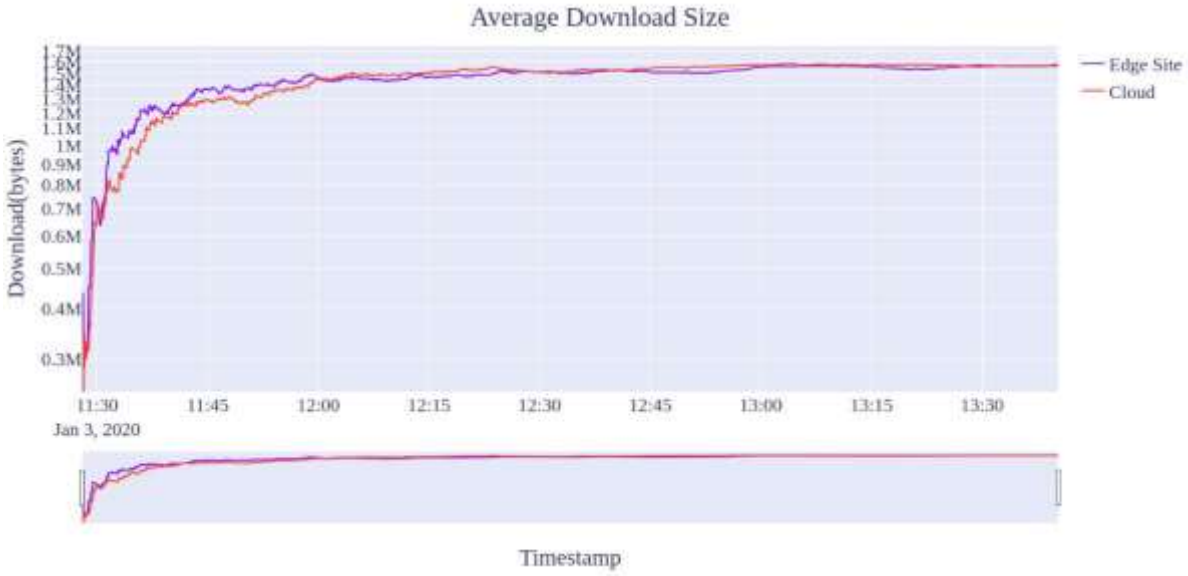

Figure 4.7: Average application content download in bytes

The application average data download also converged also validating fair comparison because both deployments experienced about equal amount of data download throughout the experiments

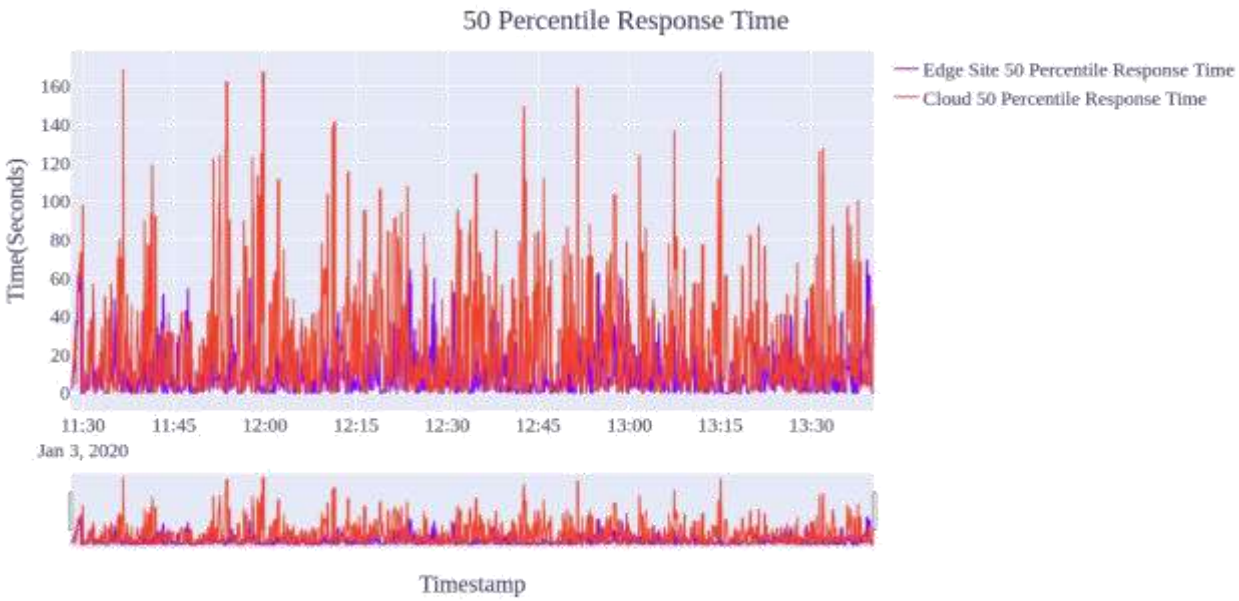

Figure 4.8: 50 Percentile application response time

The application response for half of the experiment duration, 50 percentile, second quartile or median response time is plotted in Figure 4.8. The general application response time over a period of two hours for 50 percentiles. The application response time for less than 95 percent of time span of the experiment is presented in Figure 4.8 and Figure 4.9

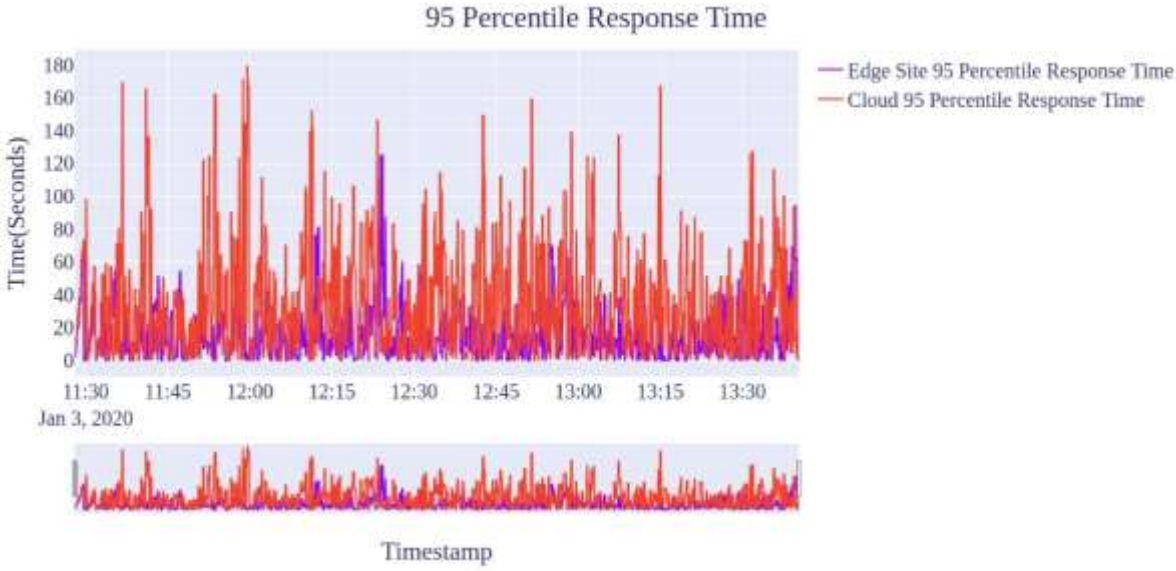

Figure 4.9: 95 Percentile of response time

4.2 Discussion

4.2.1 Evaluation of MEC models to determine the best fit for 5G applications:

The RTT values from the four MEC deployment prototypes are all within the latency requirements for Virtual Reality and Augmented Reality of 7-12ms, Tactile Internet less than 10ms, Vehicle-to-Vehicle less than 10ms and Manufacturing and Robotic Control/Safety Systems: 1-10ms. The prototype Scenario B and D completely fall within the mentioned applications latency requirements. Discussing the results extracted from the experiments, it is believed that new and emerging 5G mobile applications will run for short durations and this perfectly fit into containerization technologies, containers are built to run mostly for a short period of time. This fact together with the attendant save in compute resources are part of reasons for the choice of containers over VM. Analysis of deployments behaviours was considered for a period of thirty (30) minutes. Therefore, the parameters were closely observed and evaluated over a period of 30 minutes.

A close observation of the load test results shows that the total request rate for both edge site and cloud deployments showed similar characteristics in the number of requests, therefore, the experiment had a good ground to establish comparisons for the two scenarios as shown in Figure 4.10. This was corroborated by the average application

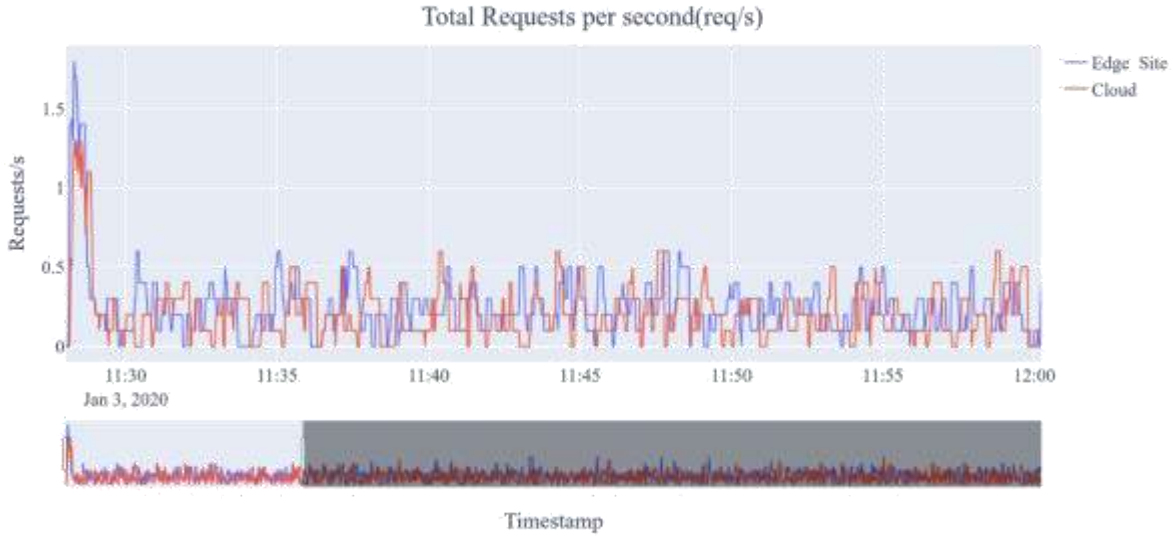

Figure 4.10: Total request rate/sec

data download for the two deployments scenarios as shown in Figure 4.7. Considering a production deployment scenario with several hundred or thousands geographically dispersed UEs connected to mobile applications hosted at a remote cloud data centre or within the operator DN, this will create high bandwidth traffic and exert serious penalty on the operator backhaul network. Whereas, in the proposed edge site deployment scenario UEs will take the advantage of running MEC applications hosted at the network edge, deployed at the CU thereby removing the issue of heavy traffic bottle neck on the backhaul networks.

Both median and maximum application response time were considered for both edge site and cloud deployments as indicated in Figure 4.12. It was observed the maximum latency figure for cloud deployment was too high for smooth running of mobile applications. There were initial failures reported for both edge and cloud deployment while the applications just starting up, but the initial failure finally disappeared. But there were evident application high failure rates for cloud deployments compared with the edge site

deployments as indicated in Figure 4.11. This will have adverse effects on adoption of

new and emerging latency sensitive applications. The maximum edge response time for edge deployment was a little above 60 seconds compared to about 160 seconds for cloud deployments in Figure 4.12. These latency figures are really unacceptable for but it is believed that deployment on a real 5G network with adequate MEC server resources can normalize the edge figures to acceptable values

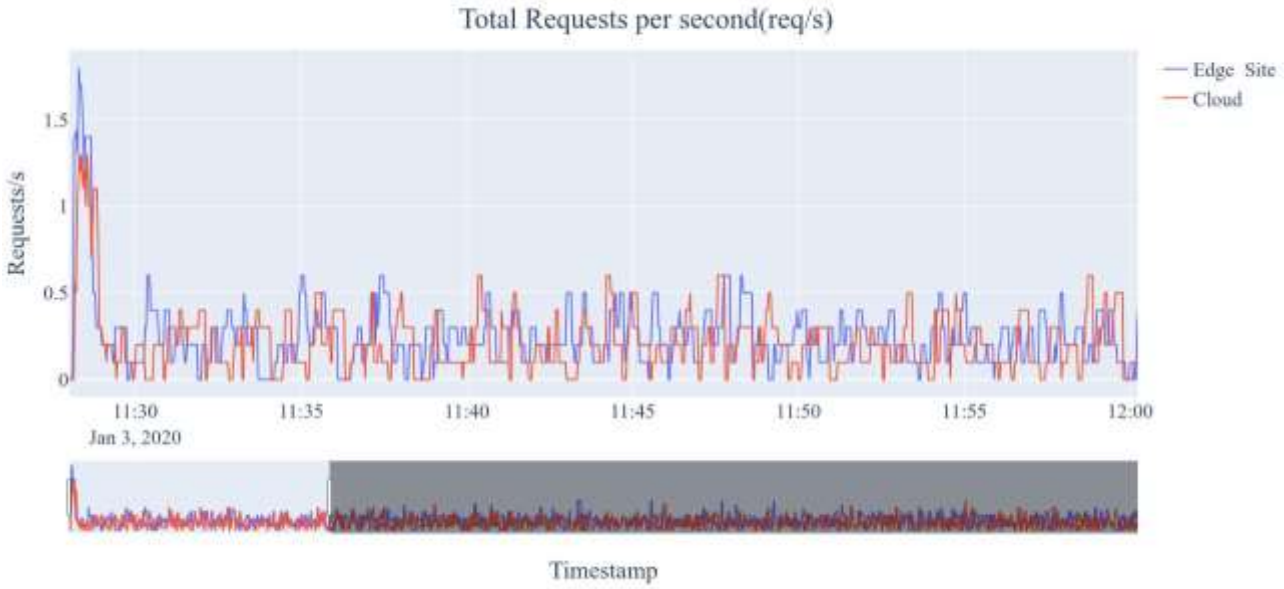

Figure 4.11: Total failed request rate (30 minutes)

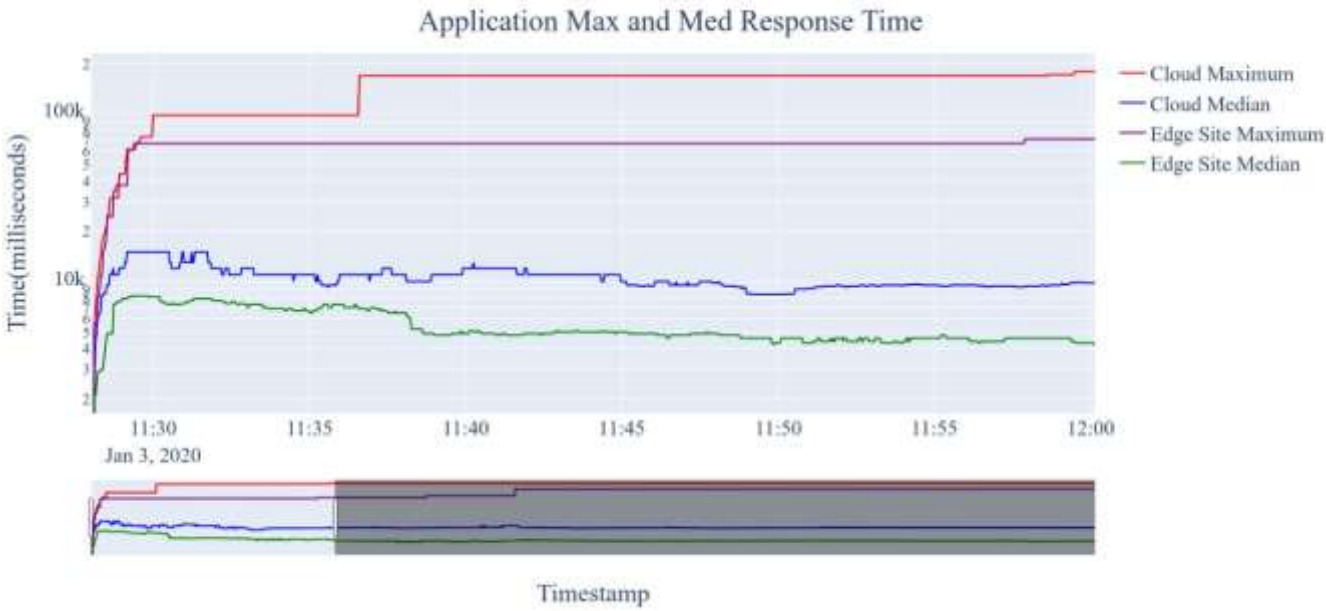

Figure 4.12: Application maximum and median response time (30 minutes)

The comparison of minimum application response time, Figure 4.13, confirmed the proposal for containerized applications deployment at the edge for 5G networks. The response time fell within acceptable latency requirements for new and emerging applications, comparing this with cloud deployment minimum values of around 500 milliseconds.

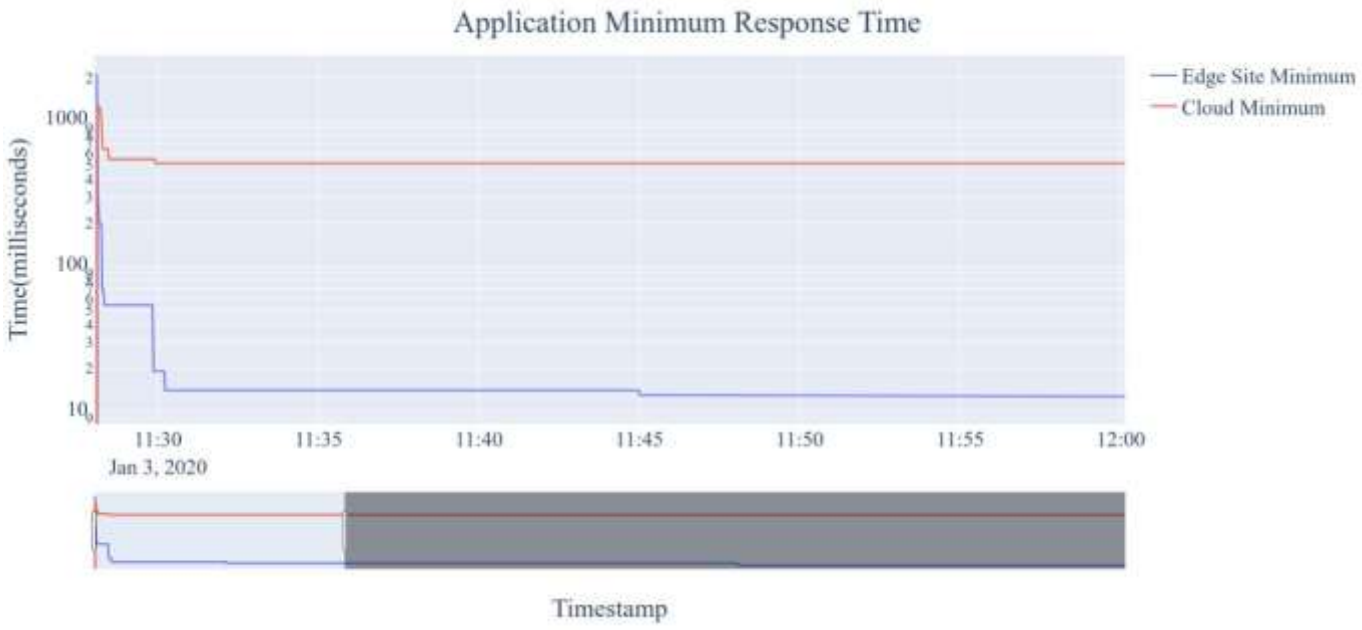

Figure .4.13: Application minimum response time (30 minutes)

Another set of relevant results were the 50th and 95th percentiles in Figure 4.14 and Figure 4.15 respectively. For about 50% of the application response time within 30 minutes the red colour of cloud response time dominates high figures as seen in the 30

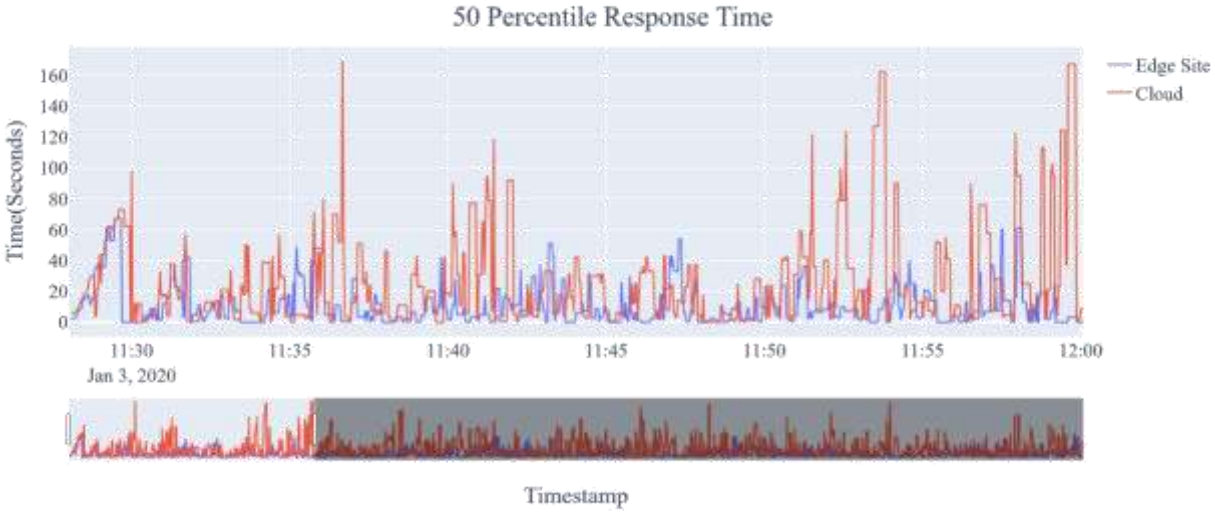

Figure 4.14: 50 Percentile of response time (30 minutes)

minutes graph. Of high interest is the 95th percentile which was used to evaluate the 95% regular and sustained application response time within 30 minutes (Figure 4.13). Clearly, edge deployment response time are more visible looking around low latency figures unlike the cloud deployment response time that dominates the graph skyline indicating consistent unacceptable high latency values

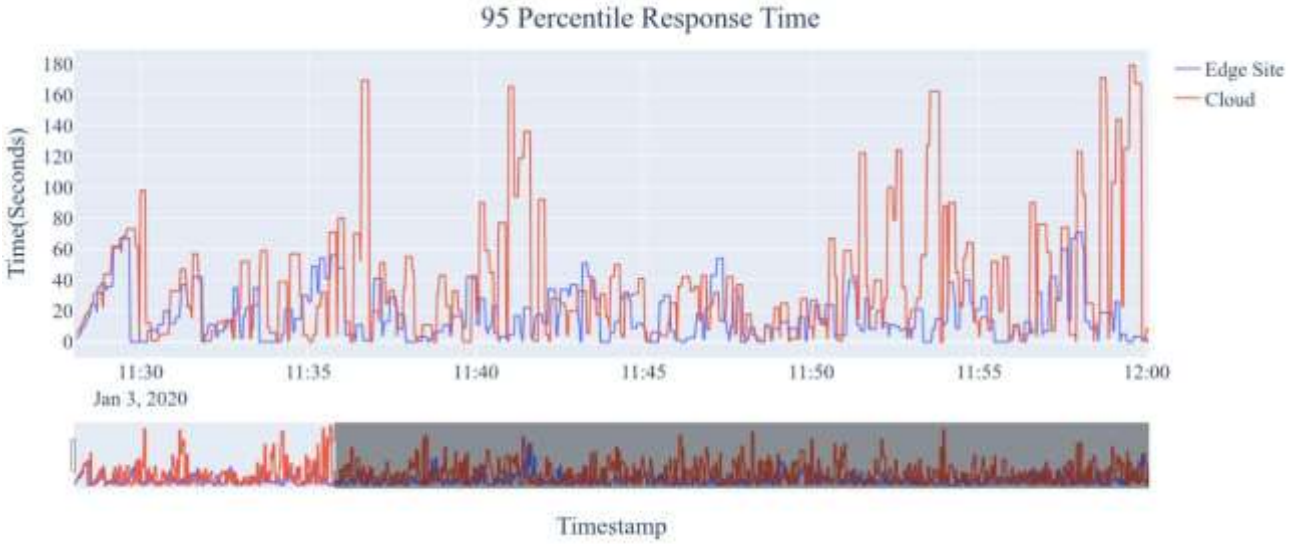

Figure 4.15: 95 Percentile of response time (30 minutes)

The fact that containers are secure and deployable for MEC infrastructures will increase the ability of enterprise developers by improving collaboration to quickly deliver scalable and reliable applications and services at pace required of 5G rollout while not jeopardizing the security of the end-to-end network. Containers provide the required DevOps ecosystem for developers and engineers to work across the entire application lifecycle, from designs, development, staging, to deployment operations for development and operations teams.

CHAPTER FIVE

5.0 CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Secured containerised Multi-access computing infrastructures have many advantages of mobile cloud computing for mobile wireless device computational power and energy carrying capacity deficiencies to cater asymmetric UE applications. Applications hosted within the RAN have better support for new and emerging application requirements. These requirements are in terms of high amount of computational, storage and analytics capabilities while at low latency figures. Secured containerization using Kata containers provide most of the essential features for multi-access edge computing infrastructures. This makes it suitable for resource-intensive mobile applications speed and isolation requirements to guarantee safety within the mobile ecosystem expected with massive deployment of 5G UEs. Secure containers will guarantee fast deployment of a huge number of applications due to its orchestration and automation capabilities. Considering the fact that no matter how fast 5G network will be, MEC will ensure the need not to transport huge data for processing in the cloud and returning the results to the UE, enhancing privacy and security concerns while also conserving bandwidth. There was still need for improvement on Kata containers to reduce performance overhead, especially for disk Input/output bound operations to significantly reduce application response time.

MEC for 5G Networks based on IaC, containerization and source control system providing very low user plane latency while at small footprint will enable high density utilization. This will provide the necessary mechanism to support communication, computing, control and content delivery for high throughput and highly latency sensitive 5G applications with consideration for mobile UE efficient energy management.

5.2 Recommendations

There is still a need for improvements on Kata Containers to reduce performance overhead, especially for disk Input/output bound operations to significantly reduce application response time. It is recommended that the experiment should include input/output performance test of Kata containers compared with *runC* runtime-based Docker containers to confirm the fears entertained of higher performance overhead in Kata containers.

5.3 Limitations and Future Works

One of the main limitations of the solution proposed in this thesis is not considering the integration of 5G virtualized network functions with the containerization technology. Despite the fact that 5G network functions are RESTful API compliant and communicate only over HTTP/2 using JSON while these features are also supported by containers. The experiment was not able to handle more than 3 requests per second by 30 unique users without significant failures for both cloud and edge test cases. Part of the reasons was the memory limitation on the physical server used for the edge scenario due to the high financial cost of memory upgrade whereas the financial cost of increasing the cloud server RAM capacity was marginal but there was a need to have the same resource configuration for both cloud and edge test cases to establish fair comparisons.

A future work could consider evaluation and improvement on input/output performance of kata containers relative to Docker containers without sacrificing the security advantages and also key into the HTTP/2 and JSON features that are also available in containerization to improve application response time that are consistent with new and emerging applications

5.4 Contributions

To the best of knowledge, every suggested solution has been geared toward edge computing. This research work proposed a stable augmentation infrastructure for mobile applications by making available a high amount of compute, storage and analytic resources within the RAN close to the end users thereby reducing the user plane latency while removing the challenges of platform-specific ecosystems due to the adoption of secured containerization technology. A stable augmentation infrastructure for mobile applications was presented. A paper titled "Multi-Access Edge Computing Deployments for 5G Networks" was presented at the 3rd International Engineering Conference (Mosudi *et al.*, 2019), Federal University of Technology, Minna, Nigeria. A full journal paper titled "A Performance Comparison of Docker and Kata Containers for Edge Computing" is currently being prepared for submission. These demonstrate the importance of **Container Based Multi-access Edge Computing** in meeting 5G enhanced mobile broadband (eMBB) and ultra-reliable and low latency communication (uRLLC) requirements

The test bed provided the opportunity to perform real world scenario load testing using Locust (Heyman, *et al.*, n.d). The experiments results showed that deployment at the edge produced low latency figures close to the requirements of emerging 5G applications. Low latency figures will enhance the end-user quality of experience (QoE). Edge deployments will reduce the pressure on network operators backhaul link saving end-to-end ecosystem from collapse due to heavy backhaul traffic that might result from billions of 5G UEs.

REFERENCES

3GPP. (2017). Technical Specification Group Radio Access Network; Study on new radio access technology: Radio access architecture and interfaces.

Adufu, T., Choi, J., & Kim, Y. (2015). Is container-based technology a winner for high performance scientific applications? *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, IEEE 2016, ISBN 978-4-88552-304-5,507-510.

Alam, M., Rufino, J., Ferreira, J., Ahmed, S. H., Shah, N., & Chen, Y. (2018). Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Communications Magazine*, *56*(9), 118–123.

Alsafi, R., & Westphal, C. (2016). Challenges in Networking to Support Augmented Reality and Virtual Reality. Retrieved October 30, 2019, from https://www.academia.edu/37963383/Challenges_in_Networking_to_Support_Augmented_Reality_and_Virtual_Reality

Augustyn, D. R., & Warchał, Ł. (2010). Cloud Service Solving N-Body Problem Based on Windows Azure Platform. *Computer Networks Communications in Computer and Information Science*, 84–95.

Bamford, R., & Deibler, W. J. (1995). Configuration management and 19 ISO 9001. Retrieved May 19, 2019, from http://www.ssqc.com/do25v6new.pdf

Belshe, M., & Peon, R. (2015). Hypertext Transfer Protocol Version 2 (HTTP/2). Retrieved May 19, 2019, from https://www.rfc-editor.org/info/rfc7540

Better Explained. (2019). A Visual Guide to Version Control. Retrieved May 19, 2019, from https://betterexplained.com/articles/a-visual-guide-to-version-control/

blackMORE Ops. (2017). "Find Linux Exploits by Kernel version. Retrieved July 3, 2019, from https://www.blackmoreops.com/2017/01/17/find-linux-exploits-by-kernel-version//

Canonical. (n.d.). Ubuntu Server - for scale out workloads. Retrieved January 30, 2020, from https://ubuntu.com/server

Checko, A. (2016). Cloud Radio Access Network architecture. Towards 5G mobile networks. Retrieved March 30, 2019, from https://orbit.dtu.dk/en/publications/cloud-radio-access-network-architecture-towards-5g-mobile-network

Chef Software Inc. (2019). Continuous Automation for the Continuous Enterprise. Retrieved May 20, 2019, from https://www.chef.io/why-chef/

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011). CloneCloud. *Proceedings of the Sixth Conference on Computer Systems - EuroSys 11*, 301–314.

Claassen, J., Koning, R., & Grosso, P. (2016). Linux containers networking: Performance and scalability of kernel modules. *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 713–171.

Cuervo, E., Balasubramanian, A., Cho, D.-K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui. *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services - MobiSys 10*, 49–62.

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*, *6*, 137–147.

Díaz, M., Martín, C., & Rubio, B. (2016). State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications*, *67*, 99–117.

Docker Inc. (2019). What is a Container? Retrieved May 19, 2019, from https://www.docker.com/resources/what-container

Dou, A., Kalogeraki, V., Gunopulos, D., Mielikainen, T., & Tuulos, V. H. (2010). Misco. *Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments - PETRA 10*, 1–8.

Dreibholz, T. (2020). Flexible 4G/5G Testbed Setup for Mobile Edge Computing Using OpenAirInterface and Open Source MANO. In AINA Workshops (pp. 1143-1153).

Ericsson AB, Huawei Technologies Co. Ltd., NEC Corporation, & Nokia. (2019). eCPRI Specification V2 - Common Public Radio Interface. Retrieved May 26, 2019, from http://www.cpri.info/downloads/eCPRI_v_2.0_2019_05_10c.pdf

Erol-Kantarci, M., & Sukhmani, S. (2018). Caching and Computing at the Edge for Mobile Augmented Reality and Virtual Reality (AR/VR) in 5G. *Ad Hoc Networks Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 169–177.

European Telecommunications Standards Institute (ETSI). (2017). TS 123 401 - V14.3.0 - LTE; General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access. - ETSI. Retrieved June 12, 2019, from https://www.etsi.org/deliver/etsi_ts/123400_123499/123401/14.03.00_60/ts_123401v140300p.pdf

European Telecommunications Standards Institute (ETSI). (2018a). TR 121 914 - V14.0.0 - Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G. - ETSI. Retrieved June 12, 2019, from https://www.etsi.org/deliver/etsi_tr/121900_121999/121914/14.00.00_60/tr_121914v140000p.pdf

European Telecommunications Standards Institute (ETSI). (2018b). TS 129 500 - V15.0.0 - 5G; 5G System; Technical Realization of Service Based Architecture. - ETSI. Retrieved May 16, 2019, from https://www.etsi.org/deliver/etsi_ts/129500_129599/129500/15.00.00_60/ts_129500v150000p.pdf

European Telecommunications Standards Institute (ETSI). (2018c). TS 138 473 - V15.2.1 - 5G; NG-RAN; F1 Application Protocol (F1AP). - ETSI.

European Telecommunications Standards Institute (ETSI). (2019a). ETSI GS MEC 003 V2.0. Retrieved June 12, 2019, from https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf

European Telecommunications Standards Institute (ETSI). (2019b). TS 123 501 - V15.5.0 - 5G; System Architecture for the 5G System. - ETSI. Retrieved April 16, 2019, from https://www.etsi.org/deliver/etsi_ts/138400_138499/138473/15.02.01_60/ts_138473v150201p.pdf

Firecracker. (n.d.). How It Works. Retrieved June 12, 2019, from https://firecracker-microvm.github.io/

Forcier, J. (n.d.). Welcome to Paramiko! Retrieved September 3, 2019, from http://www.paramiko.org/

Google Cloud. (2019) Google Kubernetes Engine Go to console(GKE). Retrieved June 12, 2019, from https://cloud.google.com/kubernetes-engine/

Gordon, M. S., Jamshidi, A. D., Mahlke, S., Mao, M. Z., & Chen, X. (2012). COMET: Code Offload by Migrating Execution Transparently. *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation October 2012, OSDI'12*, 93–106.

Haavisto, J., Arif, M., Lovén, L., Leppänen, T., & Riekki, J. (2019). Open-source RANs in Practice: an Over-the-air Deployment for 5G MEC. In 2019 European Conference on Networks and Communications (EuCNC) (pp. 495-500). IEEE.

Hewlett Packard Enterprise. (n.d.). What is Edge Computing? – Enterprise IT Definitions. Retrieved September 3, 2019, from https://www.hpe.com/dk/en/what-is/edge-computing.html

Heyman, J., Byström, C., Hamrén, J., & Heyman, H. (n.d.). What is Locust? — Locust 0.14.4 documentation. Retrieved November 4, 2019, from https://docs.locust.io/en/stable/what-is-locust.html

Hoque, S., Brito, M. S. D., Willner, A., Keil, O., & Magedanz, T. (2017). Towards Container Orchestration in Fog Computing Infrastructures. *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, 294– 299.

Huang, A., Nikaein, N., Stenbock, T., Ksentini, A., & Bonnet, C. (2017). Low latency MEC framework for SDN-based LTE/LTE-A networks. *2017 IEEE International Conference on Communications (ICC)*.

Huang, D., Zhang, X., Kang, M., & Luo, J. (2010). MobiCloud: Building Secure Cloud Framework for Mobile Computing and Communication. *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, 27–34.

Huawei technologies co, H. H. (2017, December). Front-haul networking for 5G: An analysis of technologies and standardization.

ITU. (2015). IMT Vision – Framework and overall objectives of the ...

ITU. (2017, November). Minimum requirements related to technical performance for IMT-2020 radio interface(s). Retrieved January 16, 2019, from https://www.itu.int/pub/R-REP-M.2410

Kata Containers. (2019). About Kata Containers: The speed of containers, the security of VMs. Retrieved May 17, 2019, from https://katacontainers.io/

Kata Containers. (2019). Kata containers OSbuilder. Retrieved November 3, 2019, from https://github.com/kata-containers/osbuilder

Kata Containers. (2019). Learn: An overview of the Kata Containers project. Retrieved May 17, 2019, from https://katacontainers.io/learn/

Kernel Virtual Machine. (2016). KVM. Retrieved May 5, 2019, from https://www.linux-kvm.org/page/Main_Page

Kitindi, E. J., Fu, S., Jia, Y., Kabir, A., & Wang, Y. (2017). Wireless Network Virtualization With SDN and C-RAN for 5G Networks: Requirements, Opportunities, and Challenges. *IEEE Access*, *5*, 19099–19115.

Knopp, R., Nikaein, N., Bonnet, C., Kaltenberger, F., Ksentini, A., & Gupta, R. (2017). Prototyping of Next Generation Fronthaul Interfaces (NGFI) using OpenAirInterface.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., & Zhang, X. (2012). ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. *2012 Proceedings IEEE INFOCOM*, 945–953.

Kumar Abhishek, M. (2020). High Performance Computing using Containers in Cloud. International Journal of Advanced Trends in Computer Science and Engineering, 9(4), 5686–5690. https://doi.org/10.30534/ijatcse/2020/220942020

Mach, P., & Becvar, Z. (2017). Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials*, *19*(3), 1628–1656.

Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*, *19*(4), 2322–2358.

Marinelli, E. E. (2009). Hyrax: Cloud Computing on Mobile Devices using MapReduce.

Mohyeldin, E. (2016). Minimum Technical Performance Requirements for IMT-2020 ... Retrieved January 16, 2019, from https://www.itu.int/en/ITU-R/study-groups/rsg5/rwp5d/imt-2020/Documents/S01-1_Requirements for IMT-2020_Rev.pdf

Mosudi,I., Zubair, S., Abolarinwa, J. (2019). Multi-Access Edge Computing Deployments for 5G Networks. 3rd International Engineering Conference (IEC 2019), Federal University of Technology, Minna.

Murphy, K. (2015). Centralized RAN and Fronthaul - ISEMAG. Retrieved March 30, 2019, from https://www.isemag.com/wp-content/uploads/2016/01/C-RAN_and_Fronthaul_White_Paper.pdf

National Information Technology Development Agency (NITDA). (2019). Nigeria Cloud Computing Policy Nigeria Cloud Computing Policy Release V1.2.

Nkhoma, M. Z., & Dang, D. P. T. (2013). CONTRIBUTING FACTORS OF CLOUD COMPUTING ADOPTION: A TECHNOLOGY-ORGANISATION-ENVIRONMENT FRAMEWORK APPROACH. International Journal of Information Systems and Engineering, 1(1), 30–41.

Nvidia. (2018). NVIDIA/nvidia-docker. Retrieved May 16, 2019, from https://www.nvidia.com/object/docker-container.html

Okafor, I. (2017). Legal Implications Of Offshore Cloud Computing: Data Sovereignty. Retrieved October 2019, from https://www.mondaq.com/Nigeria/Technology/561772/Legal-Implications-Of-Offshore-Cloud-Computing-Data-Sovereignty

Openstack. (2019). OpenStack Services. Retrieved October 3, 2019, from https://www.openstack.org/software/sample-configs#web-applications

Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A. I., & Dai, H. (2018). A Survey on Low Latency Towards 5G: RAN, Core Network and Caching Solutions. *IEEE Communications Surveys & Tutorials*, *20*(4), 3098–3130.

Patel, M., Hu, Y., Hédé, P., Joubert, J., Thornton, C., Naughton, B., … Klas, G. (2014). Mobile-Edge Computing – Introductory Technical White Paper - ETSI.

Patil, P., Hakiri, A., & Gokhale, A. (2016). Cyber Foraging and Offloading Framework for Internet of Things. *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*.

Pieplu, R. (2018). Ground Control Segment automated deployment and configuration with ANSIBLE and GIT. *15th International Conference on Space Operations*.

Piparo, D., Tejedor, E., Mato, P., Mascetti, L., Moscicki, J., & Lamanna, M. (2018). SWAN: A service for interactive analysis in the cloud. *Future Generation Computer Systems*, *78*, 1071–1078.

Pritchard (2010). What is cloud computing? (n.d.). Retrieved January 12, 2019, from https://www.itpro.co.uk/627952/what-is-cloud-computing

Puppet. (2019). Powerful infrastructure automation and delivery: Puppet. Retrieved March 20, 2019, from https://puppet.com/products/why-puppet

Qing, W., Zheng, H., Ming, W., & Haifeng, L. (2013). CACTSE: Cloudlet aided cooperative terminals service environment for mobile proximity content delivery. *China Communications*, *10*(6), 47–59.

Red Hat, Inc. (n.d.). Working With Playbooks¶. Retrieved September 3, 2019, from https://docs.ansible.com/ansible/latest/user_guide/playbooks.html

SaltStack, Inc. (2019). SaltStack Documentation. Retrieved May 20, 2019, from https://docs.saltstack.com/en/latest/

Sanchez, C. (2014). Scaling Docker with Kubernetes. Retrieved June 12, 2019, from https://www.infoq.com/articles/scaling-docker-with-kubernetes/

Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications*, *8*(4), pg10–17.

Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, *8*(4), 14– 23.

Shahzadi, S., Iqbal, M., Dagiuklas, T., & Qayyum, Z. U. (2017). Multi-access edge computing: open issues, challenges and future perspectives. *Journal of Cloud Computing*, *6*(1).

Shew, S. (Ed.). (2018). ITU-T Technical Report. Retrieved May 16, 2019, from https://www.itu.int/dms_pub/itu-t/opb/tut/T-TUT-HOME-2018-PDF-E.pdf

Skarpness, M. (2017). Keynote: Beyond the Cloud: Edge Computing - Mark Skarpness ... Retrieved December 18, 2019, from https://www.youtube.com/watch?v=mCDXnls6pQk

Smith, P., Erfanian, J., Goyette, D., Gilson, M., MacKenzie, R., Sutton, A., … Soghomaonian, M. (2018). NGMN Overview on 5G RAN Functional Decomposition.

Sutton, A. (2018a). 5G Network Architecture. *The ITP (Institute of Telecommunications Professionals) Journal, 12*(1), 9–15. Retrieved from http://www.engagingwithcommunications.com/publications/ITP_Papers/5G_network_architecture/ITP_Journal_Vol12_Part1_Pages9-15_5G_Network_Architecture.pdf

Sutton, A. (2018b). 5G Network Architecture, Design and Optimisation. Retrieved January 2019, from https://tv.theiet.org/?videoid=11548

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., & Sabella, D. (2017). On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys & Tutorials, 19*(3), 1657–1681.

The Linux Foundation. (n.d.). Production-Grade Container Orchestration. Retrieved June 12, 2019, from https://kubernetes.io/

Walker, T. E. (2018). Productivity with continuous integration & delivery. Retrieved November 3, 2019, from http://files.informatandm.com/uploads/2018/10/Introducing_the_CI.CD_Workflows_in_the_Cloud_TaraWalker.pdf

Yala, L., Iordache, M., Bousselmi, A., & Imadali, S. (2019). 5G Mobile Network Orchestration and Management Using Open-Source. In 2019 IEEE 2nd 5G World Forum (5GWF) (pp. 421-426). IEEE.

Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, (2018). "A Comparative Study of Containers and Virtual Machines in Big Data Environment," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018.

Zhang, X., Jeong, S., Kunjithapatham, A., & Gibbs, S. (2010). Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. *Lecture Notes of the Institute for Computer Sciences, Social*

Informatics and Telecommunications Engineering Mobile Wireless Middleware, Operating Systems, and Applications, 161–174.

APPENDIX A

```
#plot.py
#This was used to plot all our graphs to manage the presentation of
#the CSV dataset from our results
#

import pandas as pd

#import matplotlib.pyplot as plt

import plotly.express as px
import plotly.graph_objs as go
import plotly.offline as ply

from datetime import datetime

dt =
pd.read_csv('graphcsv2/edge_cloud_log_03_01_2020_stats_history.csv')

dt['Timestamp_edge'] = pd.to_datetime(dt['Timestamp_edge'], unit='s')
dt['Timestamp_cloud']= pd.to_datetime(dt['Timestamp_cloud'],unit='s')

#Selecting and manipulating columns of interest

timestamp_edge                      = dt['Timestamp_edge']
total_requests_edge                 = dt['# requests_edge']
failures_edge                       = dt['# failures_edge']
total_requests_rate_edge            = dt['Requests/s_edge']
failed_requests_rate_edge           = dt['Requests Failed/s_edge']
med_response_time_edge              = dt['Median response time_edge']
min_response_time_edge              = dt['Min response time_edge']
max_response_time_edge              = dt['Max response time_edge']
average_content_Size_edge           = dt['Average Content Size_edge']
fifty_response_time_edge            = dt['50%_edge']
ninety_5_response_time_edge         = dt['95%_edge']

timestamp_cloud                     = dt['Timestamp_cloud']
total_request_cloud                 = dt['# requests_cloud']
failures_cloud                      = dt['# failures_cloud']
total_requests_rate_cloud           = dt['Requests/s_cloud']
failed_requests_rate_cloud          = dt['Requests Failed/s_cloud']
med_response_time_cloud             = dt['Median response time_cloud']
min_response_time_cloud             = dt['Min response time_cloud']
max_response_time_cloud             = dt['Max response time_cloud']
average_content_Size_cloud          = dt['Average Content Size_cloud']
fifty_response_time_cloud           = dt['50%_cloud']
ninety_5_response_time_cloud        = dt['95%_cloud']
```

```
trace_a1          =     go.Scatter(x = timestamp_edge,       y =
total_requests_rate_edge,     name = 'Edge Site')
trace_a2          =     go.Scatter(x = timestamp_cloud,      y =
total_requests_rate_cloud,    name = 'Cloud')

layout_a          = dict       (
                        font = dict(family = 'Times New Roman',
                                    size = 24
                                          ) ,
                  title = dict(text = 'Total Requests per
            second(req/s)',
                  #xanchor = 'center',
                  x = 0.5,
                  yanchor = 'top',
                  font = dict(
                  size = 34)
                  ),
                  xaxis = dict(title = "Timestamp"),
                  yaxis = dict(title = 'Requests/s'),
                  xaxis_rangeslider_visible=True
                  )

trace_b1          =     go.Scatter(x = timestamp_edge,       y =
med_response_time_edge/1000,  name = 'Edge Site Median Response Time')
trace_b2          = go.Scatter(x = timestamp_cloud,          y =
med_response_time_cloud/1000, name =  'Cloud Median Response Time')
trace_b3          = go.Scatter(x = timestamp_edge,           y =
max_response_time_edge/1000, name = 'Edge Site Maximum Response Time')
trace_b4 = go.Scatter(x = timestamp_cloud, y =
max_response_time_cloud/1000, name = 'Cloud Maximum Response Time')

layout_b          =     dict (
                        font = dict(family = 'Times New Roman',
                        size = 24
                        ) ,
                        title = dict(text = 'Application Response
                        Time',
                        #xanchor = 'center',
                        x = 0.5,
                        yanchor = 'top',
                        font = dict(
                        size = 34)
                        ),
                        xaxis = dict(title = "Timestamp"),
                        yaxis = dict(title = 'Time(seconds)'),
                        xaxis_rangeslider_visible=True
                  )
```

```
trace_c1          =     go.Scatter(x = timestamp_edge,      y =
failed_requests_rate_edge,    name = 'Edge Site')
trace_c2          =     go.Scatter(x = timestamp_cloud,     y =
failed_requests_rate_cloud,   name = 'Cloud')

layout_c          =     dict (
                        font = dict(family = 'Times New Roman',
                        size = 24
                        ) ,
                  title = dict(text = 'Failed Request per
                  second(req/s)',
                  #xanchor = 'center',
                  x = 0.5,
                  yanchor = 'top',
                  font = dict(
                              size =  34)
                              ),
                  xaxis = dict(title = "Timestamp"),
                  yaxis = dict(title = 'Requests/s'),
                  xaxis_rangeslider_visible=True
                        )

trace_d1          =     go.Scatter(x = timestamp_edge,      y =
min_response_time_edge, name = 'Edge Site Minimum Response Time')
trace_d2          =     go.Scatter(x = timestamp_cloud,     y =
min_response_time_cloud, name = 'Cloud Minimum Response Time')
#trace_d3         =     go.Scatter(x = timestamp_edge,      y =
med_response_time_edge, name = 'Edge Site Median Response Time')
#trace_d4         =     go.Scatter(x = timestamp_cloud,     y =
med_response_time_cloud, name = 'Cloud Median Response Time')

layout_d          = dict (
                        font = dict(family = 'Times New Roman',
                                    size = 24
                                     ) ,
                  title = dict(text = 'Application Minimum Response
                  Time',
                  #xanchor = 'center',
                  x = 0.5,
                  yanchor = 'top',
                  font  = dict(
                              size =  34
                              )
                              ),
                  xaxis = dict(title = "Timestamp"),
                  yaxis = dict(title = 'Time(milliseconds)'),
                  xaxis_rangeslider_visible=True
                              )
```

APPENDIX B

#locustfile.py

#These were used to define user behaviour to load test the deployed web applications #and for the generation of CSV result dataset for further analysis #

```
#Cloud locustfile
#locust --csv=locust_log_cloud --host /67.172.223.157 --no-web -c 30 -r
3 from locust import HttpLocust, TaskSet, task, between import random

from random import randrange
#import test_random_integer

#Importing Logging
import logging
import os.path
from datetime import datetime

#Define target host
#HOST = "http://167.172.223.157"
#PORT = ":9082/"



#Creating log filename
locust_current_time=datetime.now()
logfilename = str(locust_current_time) + '.log'
logfile =
os.path.join('/home/mosudi/Documents/mio_drive/mengloadtest/locust_log_d
i
r/',
logfilename)
print(logfile)
logging.basicConfig(filename="logfile.log", level=logging.INFO) #,
# format='%(asctime)s %(levelname)s %(name)s %(threadName)s
: %(message)s')
#logging.basicConfig(filename=logfile, level=logging.DEBUG)
```

```
class UserBehaviour(TaskSet):
      def on_start(self):
            """ on_start is called when a Locust start before any task is
scheduled """
            #http://167.172.223.157:9082/
            self.client.get("http://167.172.223.157:9082/"
            ) pass

      @task(3)
      def index(self):
            #self.client.get("/")
            n = randrange(3, 14)
            print(n)
            self.client.post("http://167.172.223.157:9082/", {"n": n })
            pass

      @task(2)
      def contact(self):
            self.client.get("http://167.172.223.157:9082/contact")
            pass

      @task(1)
      def about(self):
            self.client.get("http://167.172.223.157:9082/about")
            pass

class WebsiteUser(HttpLocust):
      task_set = UserBehaviour
      wait_time = between(90, 140)
"""class WebsiteUser(HttpLocust):
task_set = UserBehaviour
wait_time = lambda self: random.expovariate(1)*1000"""
```

```
#Edge locustfile
#locust --csv=locust_log_edge --host 192.168.5.155 --no-web -c 30 -r
3 from locust import HttpLocust, TaskSet, task, between import random

from random import randrange
#import test_random_integer

#Importing Logging
import logging
import os.path
from datetime import datetime

#Define target host
#HOST = "http://192.168.5.155"
#PORT = ":9082/"



#Creating log filename
locust_current_time=datetime.now()
logfilename = str(locust_current_time) + '.log'
logfile =
os.path.join('/home/mosudi/Documents/mio_drive/mengloadtest/locust_log_d
i
r/',
logfilename)
print(logfile)
logging.basicConfig(filename="logfile.log", level=logging.INFO) #,
# format='%(asctime)s %(levelname)s %(name)s %(threadName)s
: %(message)s')
#logging.basicConfig(filename=logfile, level=logging.DEBUG)
```

```
class UserBehaviour(TaskSet):
      def on_start(self):
            """ on_start is called when a Locust start before any task is
scheduled """
            #self.client.get("http:/167.172.223.157:9082/")
            self.client.get("http://192.168.5.155:9082/"
            ) pass

      @task(3)
      def index(self):
            #self.client.get("/")
            n = randrange(3, 14)
            print(n)
            #self.client.post("http://167.172.223.157:9082/", {"n": n })
            self.client.post("http://192.168.5.155:9082/", {"n": n })
            pass

      @task(2)
      def contact(self):
            #self.client.get("http://167.172.223.157:9082/contact")
            self.client.get("http://192.168.5.155:9082/contact")
            pass

      @task(1)
      def about(self):
            #self.client.get("http://167.172.223.157:9082/about")
            self.client.get("http://192.168.5.155:9082/about")
            pass
class WebsiteUser(HttpLocust):
      task_set = UserBehaviour
      wait_time = between(90, 140)

"""class WebsiteUser(HttpLocust):
      task_set = UserBehaviour
      wait_time = lambda self: random.expovariate(1)*1000"""
```

APPENDIX C

Bash Commands

```
#Install requirements
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get install -y debootstrap curl git wget snapd vi

#Installing Go
sudo snap install go --classic

#Installing kata-runtime, kata-proxy and kata-shim
ARCH=$(arch)
BRANCH="${BRANCH:-master}"
sudo sh -c "echo 'deb
http://download.opensuse.org/repositories/home:/katacontainers:/releases:/${ARCH}:/${BRAN
CH}/xUbuntu_$(lsb_release -rs)/ /' > /etc/apt/sources.list.d/kata-containers.list"
curl -sL
http://download.opensuse.org/repositories/home:/katacontainers:/releases:/${ARCH}:/${BRAN

CH}/xUbuntu_$(lsb_release -rs)/Release.key | sudo apt-key add - sudo -E apt-get update

sudo -E apt-get -y install kata-runtime kata-proxy kata-shim

#Installing Docker CE
sudo -E apt-get -y install apt-transport-https ca-certificates software-properties-common
curl -sL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
arch=$(dpkg --print-architecture)
sudo -E add-apt-repository "deb [arch=${arch}] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
sudo -E apt-get update
sudo -E apt-get -y install docker-ce
sudo usermod -aG docker ${USER}

#Image build procedures
git clone https://github.com/imosudi/osbuilder.git

# Creating Rootfs with systemd as init
export USE_DOCKER=true

# Building guest O/S rootfs based on Ubuntu
sudo image-builder/image_builder.sh -r "${PWD}/rootfs-builder/rootfs-bionic/"
```

```
# Building Ubuntu Docker image based rootfs created above
sudo -E PATH=$PATH make USE_DOCKER=true rootfs

#Application packaging procedures #Clone Mobile web application from git
repo git clone https://imosudi@bitbucket.org/imosudi/rubiks3.git

#Prepare Ubuntu Docker image for app
deployment docker run -it ubuntu-rootfs-osbuilder
/bin/bash 2baf2236122d# apt update && apt
upgrade -y 2baf2236122d# apt install python3-pip

#Create image for app deployment
docker commit 2baf2236122d imosudi/ubuntu-rootfs-osbuilder:v10.0

#Edit application image Dockerfile
cd rubiks3/
#Dockerfile
FROM imosudi/ubuntu-rootfs-osbuilder:v10.0

#Dockerfile Author / Maintainer
MAINTAINER mosudi.pg7331@st.futminna.edu.ng
|
|
|
ENV PORT 9082
CMD ["/usr/bin/python3", "app.py"]
#Building a versioned Docker application container
 docker build -t imosudi/rubiks3ubuntufs-kataruntime:v1.5 .

#Ship the built images to repository for future use
 docker push imosudi/ubuntu-rootfs-osbuilder:v10.0
 docker push imosudi/rubiks3ubuntufs-kataruntime:v1.5

#Deploy application
docker run -it -p 9082:9082 imosudi/rubiks3ubuntufs-kataruntime:v1.5

#Web application available via Docker host IP address
http://HOST-IP-ADDRESS:9082
```

APPENDIX D

Table 4.1: Experiment Data

	Timestamp_edge	Timestamp_cloud	# requests_edge	# requests_cloud	...	50%_edge	50%_cloud	95%_edge	95%_cloud
0	2020-01-03 11:28:04	2020-01-03 11:28:01	0.0	0	...	NaN	NaN	NaN	NaN
1	2020-01-03 11:28:06	2020-01-03 11:28:03	2.0	0	...	2100.0	NaN	2100.0	NaN
2	2020-01-03 11:28:08	2020-01-03 11:28:05	8.0	0	...	2100.0	NaN	3000.0	NaN
3	2020-01-03 11:28:10	2020-01-03 11:28:07	12.0	3	...	2500.0	5300.0	4600.0	5600.0
4	2020-01-03 11:28:12	2020-01-03 11:28:09	13.0	5	...	2500.0	5600.0	4900.0	6700.0
5	2020-01-03 11:28:14	2020-01-03 11:28:11	16.0	8	...	3000.0	6400.0	8100.0	8500.0
6	2020-01-03 11:28:16	2020-01-03 11:28:13	21.0	10	...	4100.0	6400.0	8200.0	9500.0
7	2020-01-03 11:28:18	2020-01-03 11:28:15	24.0	13	...	7300.0	6700.0	9100.0	11000.0
8	2020-01-03 11:28:20	2020-01-03 11:28:17	29.0	17	...	7600.0	8200.0	11000.0	13000.0
9	2020-01-03 11:28:22	2020-01-03 11:28:19	29.0	19	...	7900.0	9500.0	11000.0	15000.0
10	2020-01-03 11:28:24	2020-01-03 11:28:21	30.0	21	...	8700.0	11000.0	15000.0	16000.0
11	2020-01-03 11:28:26	2020-01-03 11:28:23	35.0	22	...	10000.0	13000.0	16000.0	18000.0
12	2020-01-03 11:28:28	2020-01-03 11:28:25	39.0	25	...	15000.0	15000.0	16000.0	19000.0
13	2020-01-03 11:28:30	2020-01-03 11:28:27	41.0	27	...	15000.0	15000.0	18000.0	19000.0
14	2020-01-03 11:28:32	2020-01-03 11:28:29	47.0	32	...	16000.0	18000.0	24000.0	21000.0
15	2020-01-03 11:28:34	2020-01-03 11:28:31	49.0	33	...	16000.0	18000.0	24000.0	22000.0
16	2020-01-03 11:28:36	2020-01-03 11:28:33	49.0	34	...	16000.0	19000.0	24000.0	25000.0
17	2020-01-03 11:28:38	2020-01-03 11:28:35	49.0	35	...	18000.0	20000.0	24000.0	25000.0
18	2020-01-03 11:28:40	2020-01-03 11:28:37	49.0	37	...	14000.0	21000.0	24000.0	31000.0
19	2020-01-03 11:28:42	2020-01-03 11:28:39	50.0	39	0...	11000.0	27000.0	19000.0	31000.0
20	2020-01-03 11:28:44	2020-01-03 11:28:41	52.0	440	...	14000.0	29000.0	35000.0	32000.0
21	2020-01-03 11:28:46	2020-01-03 11:28:43	52.0	45	...	15000.0	30000.0	35000.0	32000.0
22	2020-01-03 11:28:48	2020-01-03 11:28:45	52.0	47	...	15000.0	30000.0	35000.0	32000.0
23	2020-01-03 11:28:50	2020-01-03 11:28:47	54.0	47	...	34000.0	30000.0	38000.0	32000.0
24	2020-01-03 11:28:52	2020-01-03 11:28:49	54.0	47	...	35000.0	30000.0	38000.0	32000.0
25	2020-01-03 11:28:54	2020-01-03 11:28:51	54.0	47	...	35000.0	4100.0	38000.0	32000.0
26	2020-01-03 11:28:56	2020-01-03 11:28:53	54.0	48	...	38000.0	4100.0	38000.0	22000.0

27	2020-01-03 11:28:58	2020-01-03 11:28:55	54.0	49	...	34000.0	22000.0	34000.0	44000.0
28	2020-01-03 11:29:00	2020-01-03 11:28:57	55.0	49	...	34000.0	22000.0	34000.0	44000.0
29	2020-01-03 11:29:02	2020-01-03 11:28:59	56.0	49	...	34000.0	44000.0	36000.0	44000.0
...	...	...	...	...	...	...	...	...	.. .
3925	2020-01-03 13:39:37	2020-01-03 13:39:19	1939.0	1743	...	61000.0	12000.0	94000.0	12000.0
3926	2020-01-03 13:39:39	2020-01-03 13:39:21	1939.0	1743	...	61000.0	12000.0	94000.0	12000.0
3927	2020-01-03 13:39:41	2020-01-03 13:39:23	1940.0	1747	...	26000.0	39000.0	94000.0	93000.0
3928	2020-01-03 13:39:43	2020-01-03 13:39:25	1940.0	1747	...	26000.0	39000.0	94000.0	93000.0
3929	2020-01-03 13:39:45	2020-01-03 13:39:27	1941.0	1748	...	3600.0	39000.0	94000.0	93000.0
3930	2020-01-03 13:39:47	2020-01-03 13:39:29	1942.0	1748	...	3600.0	39000.0	94000.0	93000.0
3931	2020-01-03 13:39:49	2020-01-03 13:39:31	1944.0	1748	...	3600.0	39000.0	60000.0	93000.0
3932	2020-01-03 13:39:51	2020-01-03 13:39:33	1945.0	1748	...	7700.0	49000.0	62000.0	93000.0
3933	2020-01-03 13:39:53	2020-01-03 13:39:35	1945.0	1748	...	7700.0	16000.0	62000.0	16000.0
3934	2020-01-03 13:39:55	2020-01-03 13:39:37	1946.0	1748	...	7700.0	16000.0	62000.0	16000.0
3935	2020-01-03 13:39:57	2020-01-03 13:39:39	1946.0	1749	...	7700.0	16000.0	62000.0	16000.0
3936	2020-01-03 13:39:59	2020-01-03 13:39:41	1946.0	1750	...	7700.0	9800.0	62000.0	16000.0
3937	2020-01-03 13:40:01	2020-01-03 13:39:43	1948.0	1750	...	120.0	9800.0	62000.0	9800.0
3938	2020-01-03 13:40:03	2020-01-03 13:39:45	1949.0	1750	...	120.0	9800.0	12000.0	9800.0
3939	2020-01-03 13:40:05	2020-01-03 13:39:47	1949.0	1750	...	120.0	9800.0	12000.0	9800.0
3940	2020-01-03 13:40:07	2020-01-03 13:39:49	1951.0	1752	...	120.0	5200.0	20000.0	9800.0
3941	2020-01-03 13:40:09	2020-01-03 13:39:51	1951.0	1752	...	120.0	1200.0	20000.0	5200.0
3942	2020-01-03 13:40:11	2020-01-03 13:39:53	1953.0	1752	...	17.0	1200.0	20000.0	5200.0
3943	2020-01-03 13:40:13	2020-01-03 13:39:55	1954.0	1752	...	37.0	5200.0	20000.0	5200.0
3944	2020-01-03 13:40:15	2020-01-03 13:39:57	1954.0	1752	...	17.0	5200.0	20000.0	5200.0
3945	2020-01-03 13:40:17	2020-01-03 13:39:59	1954.0	1752	...	17.0	1200.0	37.0	1200.0
3946	2020-01-03 13:40:19	2020-01-03 13:40:01	1954.0	1752	...	17.0	1200.0	37.0	1200.0
3947	NaT	2020-01-03 13:40:03	NaN	1753	...	NaN	6500.0	NaN	6500.0
3948	NaT	2020-01-03 13:40:05	NaN	1753	...	NaN	6500.0	NaN	6500.0
3949	NaT	2020-01-03 13:40:07	NaN	1754	...	NaN	45000.0	NaN	45000.0
3950	NaT	2020-01-03 13:40:09	NaN	1754	...	NaN	45000.0	NaN	45000.0

3951	NaT 2020-01-03 13:40:11	NaN	1754 ...	NaN	45000.0	NaN	45000.0
3952	NaT 2020-01-03 13:40:13	NaN	1754 ...	NaN	45000.0	NaN	45000.0
3953	NaT 2020-01-03 13:40:15	NaN	1755 ...	NaN	45000.0	NaN	45000.0
3954	NaT 2020-01-03 13:40:17	NaN	1756 ...	NaN	24000.0	NaN	45000.0

[3955 rows x 22 columns]