# Resource Management in Multi-Access Edge Computing (MEC)

A THESIS SUBMITTED TO
LONDON SOUTH BANK UNIVERSITY
FOR THE PARTIAL FULFILMENT
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE FACULTY OF ENGINEERING

**2020**

By
Emeka Emmanuel Ugwuanyi
3331605
Department of Computer Science

RESEARCH TITLE

# Resource Management
# in
# Multi-Access Edge Computing
# (MEC)

## ABSTRACT

This PhD thesis investigates the effective ways of managing the resources of a Multi-Access Edge Computing Platform (MEC) in 5th Generation Mobile Communication (5G) networks. The main characteristics of MEC include distributed nature, proximity to users, and high availability. Based on these key features, solutions have been proposed for effective resource management. In this research, two aspects of resource management in MEC have been addressed. They are the computational resource and the caching resource which corresponds to the services provided by the MEC.

MEC is a new 5G enabling technology proposed to reduce latency by bringing cloud computing capability closer to end-user Internet of Things (IoT) and mobile devices. MEC would support latency-critical user applications such as driverless cars and e-health. These applications will depend on resources and services provided by the MEC. However, MEC has limited computational and storage resources compared to the cloud. Therefore, it is important to ensure a reliable MEC network communication during resource provisioning by eradicating the chances of deadlock. Deadlock may occur due to a huge number of devices contending for a limited amount of resources if adequate measures are not put in place. It is crucial to eradicate deadlock while scheduling and provisioning resources on MEC to achieve a highly reliable and readily available system to support latency-critical applications. In this research, a deadlock avoidance resource provisioning algorithm has been proposed for industrial IoT devices using MEC platforms to ensure higher reliability of network interactions. The proposed scheme incorporates Banker's resource-request algorithm using Software Defined Networking (SDN) to reduce communication overhead. Simulation and experimental results have shown that system deadlock can be prevented by applying the proposed algorithm which ultimately leads to a more reliable network interaction between mobile stations and MEC platforms.

Additionally, this research explores the use of MEC as a caching platform as it is proclaimed as a key technology for reducing service processing delays in 5G networks. Caching on MEC decreases service latency and improve data content access by allowing direct content delivery through the edge without fetching data from the remote server. Caching on MEC is also deemed as an effective approach that guarantees more reachability due to proximity to end-users. In this regard, a novel hybrid content caching algorithm has been proposed for MEC platforms to increase their caching efficiency. The proposed algorithm is a unification of a modified Belady's algorithm and a distributed cooperative caching algorithm to improve data access while reducing latency. A polynomial fit algorithm with Lagrange interpolation is employed to predict future request references for Belady's algorithm. Experimental results show that the proposed algorithm obtains 4% more cache hits due to its selective caching approach when compared with case study algorithms. Results also show that the use of a cooperative algorithm can improve the total cache hits up to 80%.

Furthermore, this thesis has also explored another predictive caching scheme to further improve caching efficiency. The motivation was to investigate another predictive caching approach as an improvement to the formal. A Predictive Collaborative Replacement (PCR) caching framework has been proposed as a result which consists of three schemes. Each of the schemes addresses a particular problem. The proactive predictive scheme has been proposed to address the problem of continuous change in cache popularity trends. The collaborative scheme addresses the problem of cache redundancy in the collaborative space. Finally, the replacement scheme is a solution to evict cold cache blocks and increase hit ratio. Simulation experiment has shown that the replacement scheme achieves 3% more cache hits than existing replacement algorithms such as Least Recently Used, Multi Queue and Frequency-based replacement. PCR algorithm has been tested using a real dataset (MovieLens20M dataset) and compared with an existing contemporary predictive algorithm. Results show that PCR performs better with a 25% increase in hit ratio and a 10% CPU utilization overhead.

## Acknowledgements

This research would not have been possible without the support I received throughout the journey. Firstly, I would like to thank my supervisors Dr. Muddesar Iqbal and Prof. Tasos Dagiuklas for their valuable advice and inspiration that have kept me motivated during my study. Their immense guidance, support and encouragement have helped me throughout my PhD and made me a better researcher. Their invaluable comments have helped in improving the quality of the work presented in this thesis. It has been a pleasure to have them on board in this research journey. Special thanks to Dr. Muddesar for introducing me to the field of Edge computing.

Secondly, I would like to thank my colleagues and members of the SuiteLab for their tremendous support, encouragement and suggestions.  spent most of our time locked in the lab sharing ideas, you all are like family to me. Special thanks to Mr. Saptarshi Ghosh for the useful feedback on my research outputs. You were my go-to when I need help debugging/improving my code or help with the improvement of my mathematical model. Your intellect is astounding, and I am deeply indebted to you.

Finally, I would like to thank my wife Mrs. Jessica Ugwuanyi for her patience and unmeasurable emotional support throughout this research. I would like to sincerely thank my parents Mr. and Mrs. Ugwuanyi and my siblings Mrs. Chineye Chukwu, Miss Ogechukwu Ugwuanyi, Miss Amuche Ugwuanyi, Miss Chioma Ugwuanyi, Mr. Chibuike Ugwuanyi and Mr. Favour Ugwuanyi for their support and encouragement on my path towards the doctoral degree and in my life in general. I would also like to thank my uncle Mr. James Eze for his financial support. Last but not least, I would like to thank my friends Mr. Oladapo Solomon and Mr. Ovomor Urumedji for their continuous support and encouragement.

# TABLE OF CONTENTS

## TABLE OF FIGURES

## TABLE OF TABLES

## TABLE OF ALGORITHMS

# LIST OF ACRONYMS

| Acronym | Full form |
|---|---|
| 1G | First Generation Cellular System |
| 2G | Second Generation Cellular System |
| 3G | Third Generation Cellular System |
| 4G | Fourth Generation Cellular System |
| 5G | Fifth Generation Cellular System |
| 6G | Sixth Generation Cellular System |
| 5GPPP | 5G Infrastructure Public Private Partnership |
| API | Application Programming Interface |
| BDP | Bandwidth Delay Product |
| CPU | Central Processing Unit |
| CSMA/CA | Carrier-Sense Multiple Access with Collision Avoidance |
| D2D | Device to Device |
| EDF | Earliest Deadline First |
| ETSI | European Telecommunications Standards Institute |
| FIFO | First In First Out |
| FiWi | Fiber-Wireless |
| GNS3 | Graphical Network Simulator-3 |
| IaaS | Infrastructure as a Service |
| ICN | Information Centric Network |
| ICT | Information and Communications Technology |
| IEEE | Institute of Electrical and Electronic Engineering |
| IIoT | Industrial Internet of Things |
| IoT | Internet of Things |
| IP | Internet Protocol |
| IPFS | Inter-Planetary File System |
| LFR | Least Frequently Used |
| LRU | Least Recently Used |
| LSTM | Long Short-Term Memory |
| LTE | Long Term Evolution |

| | |
|---|---|
| M2M | Machine-To-Machine |
| MCC | Mobile Cloud Computing |
| MD5 | Message-Digest |
| MEC | Multi-Access Mobile Edge Computing |
| MIMO | Multiple-Input Multiple-Output |
| MNO | Mobile Network Operators |
| MQTT | Message Queuing Telemetry Transport |
| NFV | Network Function Virtualization |
| NIC | Network Interface Controller |
| OFDM | Orthogonal Frequency Division Multiplexing |
| OPR | Optimal Page Replacement |
| OS | Operating System |
| QoE | Quality of Experience |
| QoS | Quality of Service |
| PaaS | Platform as a Service |
| RAM | Random Access Memory |
| RAN | Radio Access Networks |
| RAT | Radio Access Technology |
| RMS | Rate Monotonic Scheduling |
| RNN | Recurrent Neural Network |
| RTT | Round Trip Time |
| SDN | Software Defined Networks |
| SINR | Signal to Interference plus Noise Ratio |
| SaaS | Software as a Service |
| SON | Self-Organizing Networks |
| SQL | Structured Query Language |
| UE | User Equipment |
| URL | Uniform Resource Locator |
| uRLLC | Ultra-Reliable and Low Latency Communications |
| VM | Virtual Machine |
| VNF | Virtual Network Function |
| WFG | Wait For Graph |

# CHAPTER 1 INTRODUCTION

## *1.1 MOTIVATION AND OVERVIEW*

The Fourth Generation Network (4G) infrastructure is almost adequate for meeting the network needs of 2020. However, these needs are continuously increasing and 4G is not adequately equipped with features that can cope with this increase and hence would not be sufficient for future needs. This is due to the continuous increase of connected devices and the need for the network to support emerging latency-critical applications like Vehicle to Everything (V2X), Machine to Machine (M2M) communications, Augmented Reality, Virtual Reality, etc. Cisco predicts that there will be 12.3 billion mobile-connected devices by 2022 [1] and the overall mobile data traffic is expected to grow to 77 exabytes per month by 2022. The 4G network would not be able to accommodate this large-scale traffic without compromising the Quality of Service (QoS) which would directly impact the Quality of Experience (QoE) of the user. Therefore, the Fifth Generation Network (5G) has been designed to overcome the limitations of its predecessor 4G.

5G has been deployed in most cities and still in its early commercialization stage at the time of this writing. 5G [2] has been aimed to provide a wireless communication network that can support reliable ubiquitous connectivity of billions of devices and satisfy the increasing demand for higher data rates while ensuring low latency communications. This would be realized with the help of enabling technologies [3] like Millimeter Waves, small cells, massive Multiple Input Multiple Output (MIMO) and beamforming [4]. 5G also has an objective of reducing CapEX and OpEX through Softwarization using technologies like Software Defined Networking (SDN), Virtual Network Function (VNF), and Network Function Virtualization (NFV) [5]. The 5G uses case is categorized into 3 main parts which include Enhanced Mobile

Broadband (eMBB) [6], Ultra-Reliable Low Latency Communication (URLLC) [7], and Massive Machine Type Communication (mMTC) [6]. URLLC is likely the most talked-about 5G use case mainly because of the huge services it can support, the benefits, and also the challenges. Achieving URLLC is challenging due to the distant travel between the core network and the user equipment. URLLC would play a key role in the realization of the sixth generation mobile network (6G) [8] as the initial requirement of latency is set for less than 1 millisecond [9]. Multi-access Edge Computing (MEC) [10] would aid the realization of URLLC as it aims to bring network resources closer to the user.

MEC has changed the landscape of mobile computing from a centralized Mobile Cloud Computing to a decentralized Edge Computing platform. The key characteristic of MEC is to provide mobile computing, network control, and storage at the edge of the network. The sole benefit of this is to equip resource-limited mobile devices with the support for latency-critical and computation-intensive applications. This yield benefits for both the end-users and the network operators as it will drastically improve the QoS. This will also have an indirect effect on improving QoE. With research on 6G initialized, the edge is set to serve as an enabling technology by providing a readily available computational platform [9]. The envisioned benefits of MEC have motivated extensive efforts in industry and academia including the research on this thesis on contributing to solving the key challenges that it faces and develop the technology. The purpose of the work in this thesis is to improve the QoS of 5G by making contributions to the resource management of MECs with algorithms and techniques that improve the management of network resources. The resource management challenges investigated in this research include caching and resource provisioning.

Content caching on the edge is one of the ideas that have been explored extensively due to the benefits it offers. IP and mobile traffic have continued to grow in recent years, it is of no surprise that Cisco predicts that global mobile data traffic will increase seven-fold between 2017 and 2022. This massive growth could be linked to the rise in demand for mobile devices and the IoT. This excessive traffic causes back-haul congestion in the cellular network and put pressure on mobile network operators to provide measures to tackle network congestion. One of the unique characteristics of such traffic is that the content requests are highly concentrated and popular content will be repeatedly requested. Caching on the edge is one of the protuberant solutions for delivering popular content from the network edge to User Equipment (UE). The network congestion in the backhaul would be avoided if contents are delivered from the edge platform. The ultimate solution is to identify popular contents and cache them on the edge to relieve the heavy overhead burden of the network backhaul and reduce latency. However, the key problem is identifying which contents are popular at the right time as popularity changes with time. This has been one of the motivations of this research.

There have been increased popularity with the use of IoT and mobile devices that have limited computation capabilities compared to desktops and are not quite proficient in handling computation-intensive applications like smart video analytics, interactive augmented/virtual reality games, real-time navigation using wearables, etc. Therefore, computation offload to cloud servers has been widely adopted. Such applications are then offloaded to the central cloud server for execution which might be distant from the UE. However, for latency-critical applications offloading to the edge servers is a better option. Edge computing is designed to reduce the latency during offload due to the proximity placement of the edge node and the UE. The edge computing platform makes it possible for IoT applications to process data and access computational resources located closer to the endpoints. Thereby providing low response time to latency-sensitive applications that may operate on these platforms. There have been quite a few research works done in this area that explores different methods which include binary offloading, partial offloading, and stochastic task model offloading. One of the major focus of researchers in this area is on minimizing latency and energy consumption during offloading and resource provisioning. However, little attention is paid to deadlock handling during resource provisioning. Due to the finite number of resources available on the MEC nodes, there is a need to effectively manage resources to prevent over-provisioning of resources and deadlock. Deadlock may arise as a process may request resources that are held by another waiting resource thereby leading to a circular wait. Furthermore, deadlock may occur due to a huge number of devices contending for a limited number of resources if adequate measures are not put in place. It is crucial to eradicate deadlock while scheduling and provisioning resources on MEC to achieve a highly reliable and readily available system to support latency-critical applications. Addressing this problem has been one of the motivations of this research and therefore investigated in this thesis.

MEC has been an emerging ecosystem in the 5G paradigm, which focuses on the convergence of telecommunication and IT services and providing cloud computing at the edge of the radio access network. Adequate research is required in the domain to serve as reference points for vendors, service providers, and network operators to improve and develop network services. To the best of our knowledge, this research is one of the few in this area domain that proposes solutions in the form of algorithms to solve the highlighted problems and to help vendors, service providers, and network operators through this applied research. Most of the work carried out in this area of study is facilitated with comprehensive simulation platforms for researchers to test proposed algorithms promptly as the field is ever-growing. An applied approach using an emulator-based testbed has been adopted in this thesis as basic simulations may not be able to simulate the many factors of real-world scenarios. This helps to provide improvements to existing methods.

## *1.2 THESIS AIM*

The aim of this research is to design and develop a co-operative deadlock-aware resource provisioning scheduling scheme and a predictive co-operative caching scheme to contribute to more effective resource management in MEC. The architectural representation of the aim has been presented in Figure 1. As depicted in the figure, the realization of the aims marked 1 and 2, will lead to reduced traffic at the core which will ultimately lead to improved QoE and QoS.

### *1.3 RESEARCH CONTRIBUTIONS*

The following contributions have been made in the context of resource management of MEC and the improvement of the QoS of 5G during this research. The contribution has been divided into two main components as stated in the aim.

1. **MEC resource provisioning and deadlock prevention**
    a. A novel deadlock aware Resource Provisioning Algorithm (RPA) for MECs has been designed and developed. In this context, a distributed task model has been formulated for MECs that ensures reliability by avoiding deadlock. This model uses an adaptation of the Banker's algorithm [11] while testing and evaluating it in MEC and obtaining an optimized solution for distributed MEC systems. Extensive simulations conducted on the algorithm shows a reduced probability of deadlock occurrence.
    b. An algorithm for deadlock aware cooperative decision-making during resource provisioning has been proposed. The algorithm selects the optimum MEC candidate to offload a task, based on constraints such as bandwidth, Round Trip Time (RTT), CPU, memory, and deadlock constraint of possible candidates.
    c. A comparative analysis for MEC real-time systems has been carried out, to compare how deadlock avoidance and prevention mechanisms will perform in real-time scenarios using Rate Monotonic Scheduling (RMS) [12] or Earliest Deadline First (EDF) [12] in prioritizing workloads. In this analysis, different

metrics have been considered including RTT, queue waiting time, CPU utilization, and the ratio of local execution to cooperative MEC to cloud.

2. **MEC predictive caching scheme**
    a. A novel cooperative and predictive caching algorithm for MECs has been presented. The predictive algorithm is based on the Optimal Page Replacement Strategy (OPR) and uses polynomial regression to extrapolate incoming cache objects using historical data obtained from the relative frequency of the cached data. This also utilizes a modified Least Frequently Used (LFU) replacement algorithm which not only compares the frequency of the data in the cache but also compares the frequency of a newly obtained request as a backup algorithm when there is inadequate data for extrapolation. The cooperative caching strategy is used for sharing cache information among MECs.
    b. A novel content replacement caching algorithm for MECs has been developed. The algorithm aims at identifying cold cache blocks quickly and address the problem of temporal cache frequency associated with the LFU algorithm. The algorithm can find a victim based on frequency, recency, and network cost in constant time. It also uses a selective caching approach to reduce the overall miss ratio.
    c. A novel proactive caching algorithm for MECs based on association mining has been designed. The algorithm learns the association patterns of content requests in a MEC environment where the content popularity is time-varying and unknown. It leverages association sequential pattern mining to identify content requests with close relations and prefetch them when a user request is anticipated.

d. Finally, a greedy cooperative caching algorithm for MEC to efficiently manage the collaborative cache storage by reducing data redundancy and increasing the sharing of cache data between MECs has been presented.

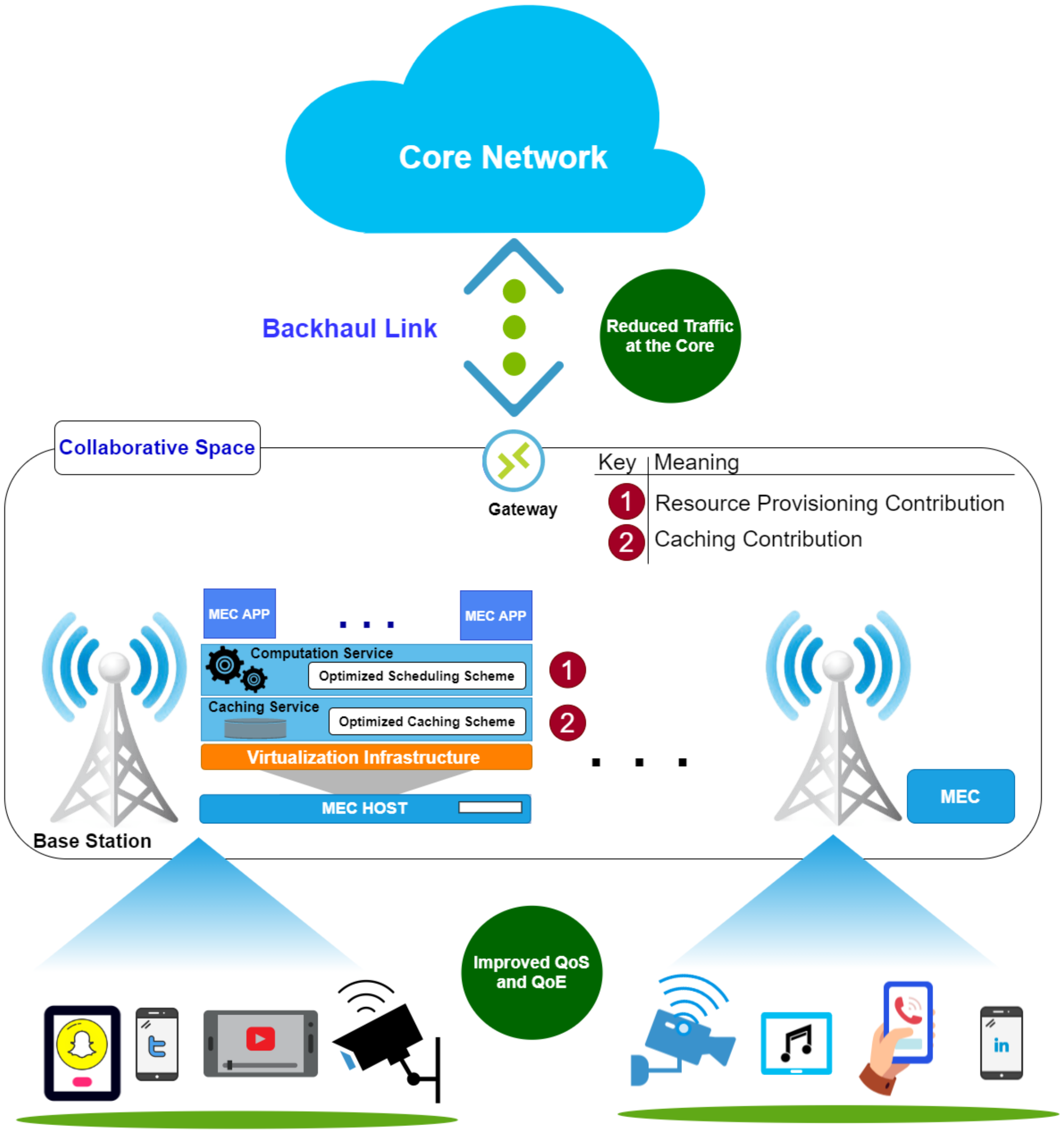

*Figure 1 Architectural Depiction of The Thesis Aim*

## 1.4 THESIS ORGANISATION

The thesis investigates the challenges associated with resource management in MEC and proposes solutions that have been tested comprehensibly by simulations and emulations. The structure of the thesis is as follows:

**Chapter 2: Literature Review**

Overview of 5G and enabling technologies. The enabling technologies reviewed include Multi-Access Mobile Edge Computing, Network Function Virtualization, Software Defined Network and Information-Centric Networks. The evolution of MEC is also discussed.

**Chapter 3: Deadlock Avoidance and Resource Provisioning in MEC**

The chapter is divided into two parts. In the first part, a deadlock aware algorithm is presented and discussed for scheduling resources for Industrial IoT (IIoT) devices onto a MEC platform which incorporates Banker's resource-request algorithm. The formulation of a distributed task model in a MEC has been presented that ensures reliability by avoiding deadlock. This has been preceded by an adaptation of the Banker's algorithm in the proposed solution and pushing its boundary by testing it in a new field (MEC) and obtaining an optimized solution for distributed systems. Finally, extensive simulations have been conducted on the algorithm to determine the effectiveness of the proposed solution.

In the second part, an evaluation and comparison of six algorithms that could be used to maintain a reliable and stable network have been carried out. Each of the six algorithms is a pair of a deadlock algorithm and a real-time algorithm. This research has been carried out using 3 deadlock strategies (Bankers algorithm, Wound Wait and Wait-die) and 2 real-time schemes (Earliest Deadline First and Rate Monotonic Scheduling). An experiment has been conducted to compare the CPU utilization, waiting time, round trip time, and offload handling performance. Each algorithm has been evaluated based on its key performance indicators and the algorithms that performed better based on the experimental constraints have been discussed.

**Chapter 4: Caching Resource Management in MEC**

This section is divided into two parts. In the first part, a hybrid cache replacement policy that supports cooperative caching for MEC platforms has been proposed. The hybrid algorithm merges a modified Belady's algorithm with a distributed cooperative caching algorithm. The prediction techniques used and a modified LFU algorithm has been presented.

The second part builds upon the contributions and findings of the previous section. Here, three novel schemes (proactive predictive scheme, a collaborative scheme, and a replacement scheme (PCR)) has been proposed to effectively manage the cache storage of MECs. The proposed algorithms have been tested using a real dataset (MovieLens20M dataset) and results have shown that the proposed algorithm performs better in terms of hit ratio and minimization of network access delay.

**Chapter 5: Conclusion and Future work**

This aims to draw the final remarks and conclusions of the presented work. Proposed optimizations and future research directions are also presented which includes a discussion of ideas for taking forward the research work presented in this thesis. Finally, the conclusion and the novel contributions of the work done in the framework of this thesis are summarized.

## *1.5 PUBLICATIONS*

The list of publications below arose from the work conducted during the course of the research described in this thesis.

### 1.5.1 JOURNALS

*Ugwuanyi, E., Ghosh, S., Iqbal, M. and Dagiuklas, T., 2018*. **Reliable Resource Provisioning Using Bankers' Deadlock Avoidance Algorithm in MEC for Industrial IoT**. *IEEE Access*, 6, pp.43327-43335.

*Ugwuanyi, E., Ghosh, S., Iqbal, M. and Dagiuklas, T., 2021.* **PCR: A Novel Predictive-Collaborative-Replacement Intelligent Caching Scheme for Multi-Access Edge Computing (MECs).** *IEEE Access*

Ghosh, S., Ugwuanyi, E., Dagiuklas, T. and Iqbal, M., 2019. **BlueArch–An Implementation of 5G Testbed**. JCM, *Journal of Communications*, pp.1110-1118.

### 1.5.2 CONFERENCE

*Ugwuanyi, E., Ghosh, S., Iqbal, M., Dagiuklas, T., Mumtaz, S. and Al-Dulaimi, A., 2019.* **Co-Operative and Hybrid Replacement Caching for Multi-Access Mobile Edge Computing**. *2019 European Conference on Networks and Communications (EuCNC)*,.

### 1.5.3 MANUSCRIPT AND REVIEW

*Ugwuanyi, E., Ghosh, S., Iqbal, M. and Dagiuklas, T., 2020.* **A Comparative Analysis of Deadlock Avoidance and Prevention Algorithms for Resource Provisioning in Distributed MEC**.
[Manuscript]

# CHAPTER 2 BACKGROUND

## *2.1 INTRODUCTION*

Mobile network communications systems have revolutionized the way people communicate by integrating communications with mobility. Since its creation, the mobile networks have been gradually morphing to deliver the ever-changing expectations of the users and to adapt to new mobile devices and the various ways users can access them. The change in usage of mobile networks and change in user expectations have led mobile network operators to sort for better ways for improving the existing infrastructure which has brought about the growth in the mobile networks from the first generation (1G) [13] to the fifth-generation (5G) [14]. 1G offered basic voice communication using analog data while 2G provided voice and limited data services. 3G offered the first mobile broadband-capable wireless network while 4G thrived to improve this by providing the first Internet broadband data transmission rates. 5G aims to deliver greater bandwidth and faster download speeds of up to 10 gigabits per second (Gbps) and the ability to deliver services such as driverless cars, virtual reality, augmented reality, etc. To achieve this, 5G will utilize technologies such as millimetre waves (mm waves) [13], Massive Multiple Input Multiple Output (MIMO) [15], beamforming [15], and small cells [13]. Other technologies that are among the driving forces behind 5G realization include, SDN [16] and NFV [17], ICN [18], C-RAN [19], network slicing[20], and edge computing [21].

In this chapter, the background of this thesis has been outlined and reviewed. This chapter discusses the overview of the current state of the art of mobile telecommunication networks. Here, the 5G concepts are introduced and the benefits are discussed. Thereafter, the evolution of mobile networks to 5G is discussed. Additionally, the changes that have been made to the network from the early beginnings of 1G to now the fifth increment have been discussed. The enabling technologies of 5G are also discussed in this section. There have been a lot of promises made about 5G and its benefits which include the improvement of spectral efficiency and delivery of mobile communication with massive connections, high reliability, and low latency. Therefore, the core technologies and innovations that would aid in delivering the promises made are outlined and discussed. The challenges in 5G and drawbacks along with proposed solutions are identified and why edge computing is crucial to achieving low latency in 5G. Thereafter discussion is made on the evolution of moving resources closer to the user (edge) from its early beginnings inform of CDN down to fog computing. Finally, the conclusion of the review is presented, highlight the key rationale of embarking on this research.

## 2.2 INTRODUCTION TO 5G

5G has been in the limelight for the past 5 years as the next generation of mobile wireless communication networks with visions of providing remarkably high data rates, extremely low latency, increase in manifold base station capacity, and improved QoS. The continuous growth and increase in the number of connected devices with the introduction of IoT have created an intricate network demand which has made it evident that the current 4G LTE/LTA networks do not have enough capacity to handle the incoming traffic of industry 4.0 emerging technologies. This is one of the problems that 5G is expected to resolve with the introduction of an edge platform by easing traffic in the core network. Other challenges that need to be addressed in the current network infrastructure include increased capacity, improved data rate, decreased latency, and better quality of service. The core enabling technologies for 5G include Millimeter Wave (mmWave), full-duplex, Software Defined Networks, Device 2 Device (D2D), edge computing and beamforming [13].

The fifth generation of radio networks i.e New-Radio (NR) is already deemed to be a true worldwide wireless web. This reputation comes as a result of its ability to seamlessly and ubiquitously connect everything up to 100 billion estimated connected wireless devices) and offer diversified use cases while meeting the quality-of-service requirements such as reliability, latency, data rates, security, coverage, and privacy.

Mobile wireless communications have come a long way from analog voice calls to current modern voice and video IP with end-to-end encryption, adept at providing high-quality mobile broadband services with end-user data rates of several megabits per second over wide areas and tens or even hundreds of megabits per second locally. However, there is still a lot to accomplish and a lot of improvements to be made due to the changing user behaviour and user demand in response to disruptive innovations and technologies. The future of networks will be to achieve an interconnected community with unbounded access to information and sharing of data, which will be accessible by everyone and everything, anywhere and anytime. This can be accomplished by evolving existing wireless-based technologies. Contemporary technological trends like IoT, Machine-Type-Communications, SDN, network automation and programmability, network slicing, and edge computing are already changing the landscape of traditional networks as we know it. Most of the changes that are being made in the existing infrastructure are to support trending mobile services such as video streaming, music streaming, social networking, online gaming, VR, and other interactive mobile applications. This thriving range of new and diverse services are becoming an integral part of

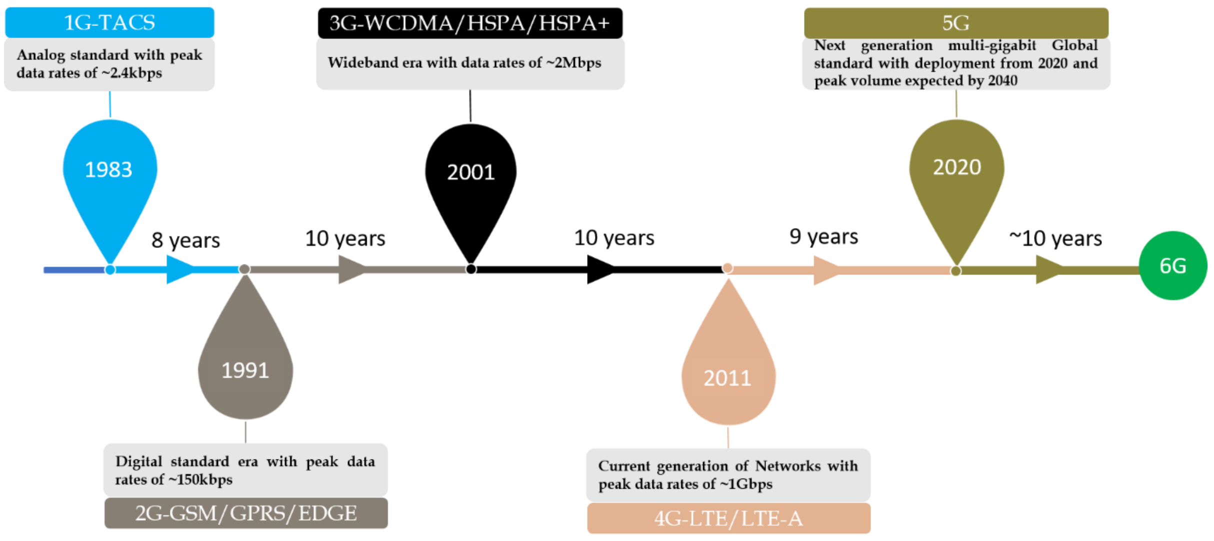
1G-TACS
Analog standard with peak data rates of ~2.4kbps
1983
8 years
1991
Digital standard era with peak data rates of ~150kbps
2G-GSM/GPRS/EDGE
10 years
3G-WCDMA/HSPA/HSPA+
Wideband era with data rates of ~2Mbps
2001
10 years
2011
Current generation of Networks with peak data rates of ~1Gbps
4G-LTE/LTE-A
9 years
5G
Next generation multi-gigabit Global standard with deployment from 2020 and peak volume expected by 2040
2020
~10 years
6G

*Figure 2 Evolution of Mobile Wireless Networks*

mobile user's entertainment and social life, therefore there is a need to adapt the existing infrastructure to societal trends.

## *2.3 EVOLUTION OF WIRELESS TECHNOLOGY TO 5G*

The mobile wireless communication system has gone through 4 major evolutional changes in the past few decades since its introduction in the late 1970s. These changes have been summarized in Figure 2. It is currently undergoing its 5th major evolutional change at the time of this writing. The huge demand for network connectivity worldwide and the introduction of mobile communication standards can be attributed to most of the changes seen in the networks today. The birth of wireless communication was in 1844 when Marconi successfully transmitted Morse code signals using radio waves wirelessly over a distance of 3.2 KMs with the help of electromagnetic waves [13]. This inception of wireless communication technology has now grown to be a very crucial technology in the modern-day community. In this section, the evolution of wireless communication networks is discussed, starting from the first generation of networks to the upcoming fifth wireless network generation. Following the trend, it can be predicted that the 6th Generation would be available by 2050.

### 2.3.1 1G

1G [13] was announced initially in the 1980s and was based on the analog transmission system for speech services. It was first introduced in 1979 by Nippon Telephone and Telegraph (NTT) in Tokyo with a data rate of up to 2.4kbps. The epoch reached Europe 2 years later and in 1982 the Advanced Mobile Phone System (AMPS) [22] was launched in the United States. Other major subscribers include Nordic Mobile Telephone (NMT)[23] and Total Access Communication System (TACS)[24]. AMPS was allocated a 40MHz bandwidth within the 800-900MHz frequency range by the Federal Communication Commission (FCC). AMPS offered 832 channels with a data rate of 10kps. AMPS initially used omnidirectional antennas and later changed to directional antennas for better frequency reuse. 1G has a lot of disadvantages including limited capacity and reckless handoff; inferior voice quality as analog signals are easily affected by interference; poor battery life and no data security as analog signals did not allow advanced encryption methods. As a result of this, there could be unwanted eavesdropping by third parties. Resource management in 1G was mainly focused on managing the channel unitization due to the availability of a handful of channels that had to be shared by all subscribers.

As of 2021, a limited NMT service in Russia remains the only 1G cellular network still in operation[1].

### 2.3.2 2G

At the end of the 1980s, the second generation of wireless technology [14] was announced but later released in the late 1990s. 2G was launched on the global system for mobile communications (GSM)[25] standard which used audio quality digital modulation to provide voice and limited data services. 2G was mainly used for voice communication and has a data rate of up to 64kbps. Compared to 1G systems, 2G mobile handset battery last longer because of low power in the radio signals. 2G systems also use digital multiple access technologies such as Code Division Multiple Access (CDMA)[26] and Time Division Multiple Access (TDMA) [26]. 2G provides services such as email and Short Message Services (SMS). Three primary advancements of the 2G technology over the prior were:

- Digitally encrypted voice conversations
- Spectrum efficiency
- Inclusion of mobile services like fax and paging.

[1] https://en.wikipedia.org/wiki/1G

Resource management on 2G was primarily focused on Radio Resource Management (RRM) which involves strategies and algorithms for controlling parameters such as transmit power, user allocation, data rates, handover criteria, modulation scheme, error coding scheme, etc. 2G is still in use at the time of this writing in 2021 in some parts of Europe, Africa, central and south America as a fallback, especially in rural areas. However, most carriers have made announcements of shutting down the 2G technology in countries like Japan, Australia, and the United States.

### 2.3.3 2.5G

2.5G [14] was released in the late 1990s. 2.5G generally uses a 2G framework but introduces packet switch along with circuit switching with improved data rates between 64kbps to 144kbps. 2.5G was developed between its predecessor and its successor. The main technologies that 2.5G introduced were GPRS (General Packet Radio Services)[27] and EDGE (Enhanced Data Rate for GSM Evolution)[27]. The launch of 2.75 also known as EDGE created a platform that allows improved data rates as a backwards-compatible extension of GSM [13].

### 2.3.4 3G

In the late 2000s, the 3rd generation of mobile wireless telecommunications networks [28] was introduced. 3G supports a data transmission transfer rate of at most 2Mbps. 3G systems merge high-speed mobile access to services based on the IP. 3G has an improved clarity characteristic from the 2.5G because it uses a wideband wireless network and therefore satisfies the International Mobile Telecommunications 2000 (IMT-2000) specifications. In 3G systems, an unconventional improvement was made to maintain the Quality of Service of the network. The additional feature of global roaming and the provision of mobile broadband access of several Mbps to smartphones and mobile modems in laptops and computers made 3G a staple at the time of its release. One of the major disadvantages of 3G over its predecessor is that 3G mobile phones consume more power. 3G was also more expensive to deploy than 2G. Since

3G involves the introduction and utilization of Wideband Code Division Multiple Access (WCDMA)[29], Universal Mobile Telecommunications Systems (UMTS)[30] and Code Division Multiple Access (CDMA) [26] 2000 technologies, the evolving technologies like High-Speed Uplink/Downlink Packet Access (HSUPA/HSDPA)[31] and Evolution-Data Optimized (EVDO)[32] has made an intermediate wireless generation between 3G and 4G named as 3.5G with an improved data rate of 5-30 Mbps [13]. 3G introduces a new type of radio architectures and concepts such as UMTS WCDMA Handovers [33].

3G phase-out started in 2020 as Vodafone[1] announced that it will switch off 3G across Europe[2]. Shortly after, T-Mobile[3] announced to be shutting down its 3G services in January 2021. In the US, Verizon[4] and AT&T[5] have made plans to switch off the 3G services by 2022.

### 2.3.5 3.75G

3.75G [15] system is an improved version of the 3G network with High-Speed Packet Access Plus (HSPA+) [28]. This evolved to be known as LTE (Long Term Evolution)[28]. LTE and Worldwide Interoperability for Microwave Access (WiMAX) [14] is the future of mobile data services as they can potentially provide a substantial number of users the facility to access a broad range of high-speed services like peer file sharing, on-demand video, and composite web services.

[1] https://www.vodafone.co.uk/
[2] https://en.wikipedia.org/wiki/3G
[3] https://www.t-mobile.com/
[4] https://www.verizon.com/
[5] https://www.att.com/

### 2.3.6 4G

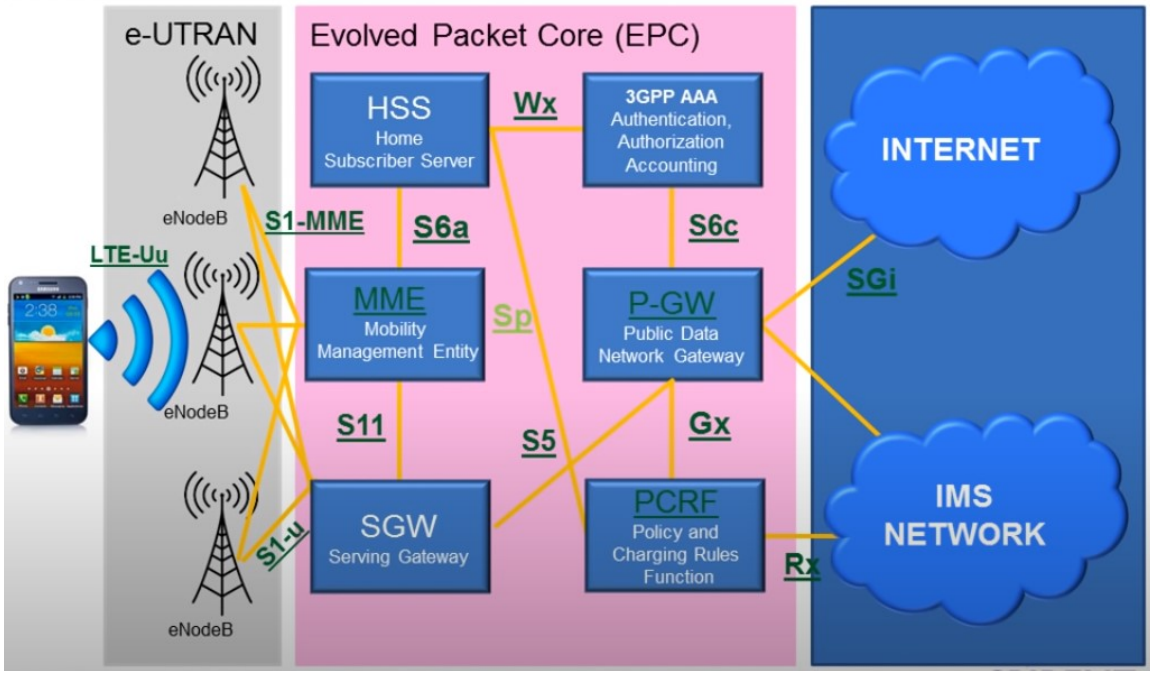
e-UTRAN
Evolved Packet Core (EPC)
eNodeB
eNodeB
eNodeB
LTE-Uu
S1-MME
S1-u
HSS
Home Subscriber Server
Wx
3GPP AAA Authentication, Authorization Accounting
S6a
S6c
MME
Mobility Management Entity
Sp
P-GW
Public Data Network Gateway
S11
S5
Gx
SGW
Serving Gateway
PCRF
Policy and Charging Rules Function
Rx
SGi
INTERNET
IMS NETWORK

*Figure 3 4G EPC Architecture [22]*

4G [34] was released in 2010 and referred to as the descendant of 2G and 3G standards which aims at providing mobile broadband internet access. 4G introduced 4G LTE Evolved Packet Core (EPC)[35] network which is a new architecture for the core network as seen in Figure 3. The EPC can handle both voice and data traffic compared to the 2G/3G architecture which has the packet switch network and the circuit switch network to handle data and voice traffic respectively. The 4G system improves the prevailing communication networks by imparting a complete and reliable solution, based on IP. The services that 4G aims to deliver are IP technology gaming, amended mobile web access, Multimedia Messaging Service, high definition mobile television, video conferencing, 3D TV, and cloud computing. 4G utilizes LTE, LTE advanced network systems, and WiMAX [14]. LTE uses Orthogonal Frequency Division Multiplexing (OFDM)[36] and OFDM Access, which divides a channel usually 5, 10, or 20MHz wide into smaller sub-channels or subcarriers each 15 kHz wide. The modulation schemes applied on each sub-channel are either Quadrature Amplitude Modulation (QAM)[37], 16QAM, or 64QAM. MIMO[15] is defined in 4G LTE, which is the ability to receive multiple inputs through the antennas and transmit multiple outputs through the transmitter. 4G LTE achieves a downstream data rate of up to 100Mpbs and an upstream data rate of up to 50Mbps under the best conditions due to the use of technologies such as OFDM and MIMO [38].

### 2.3.7 5G

The fifth generation of wireless networks [13], has been a long time coming since research began in 2016. The 5G overall architecture published by 5GPPP in June 2019 can be seen in Figure 4. 5G is the follow up major phase of wireless telecommunication standards since the 4G/IMT (International Mobile Telecommunications-Advanced) Advanced standards [35]. 5G systems aim to improve the 4G infrastructure with the inclusion of new advanced access technology like the Beam Division Multiple Access (BDMA) [39] and quasi-orthogonal or Filter Bank Multi-Carrier (FBMC) multiple access [13]. BDMA scheme would be used to increase the wireless communication system capacity and to handle the traffic surge from a large number of users in 5G systems. This technology is important in 5G systems as the number of connected devices continues to grow [1]. In using BDMA in the communication of base station and the mobile station, an orthogonal beam is allocated to each mobile station

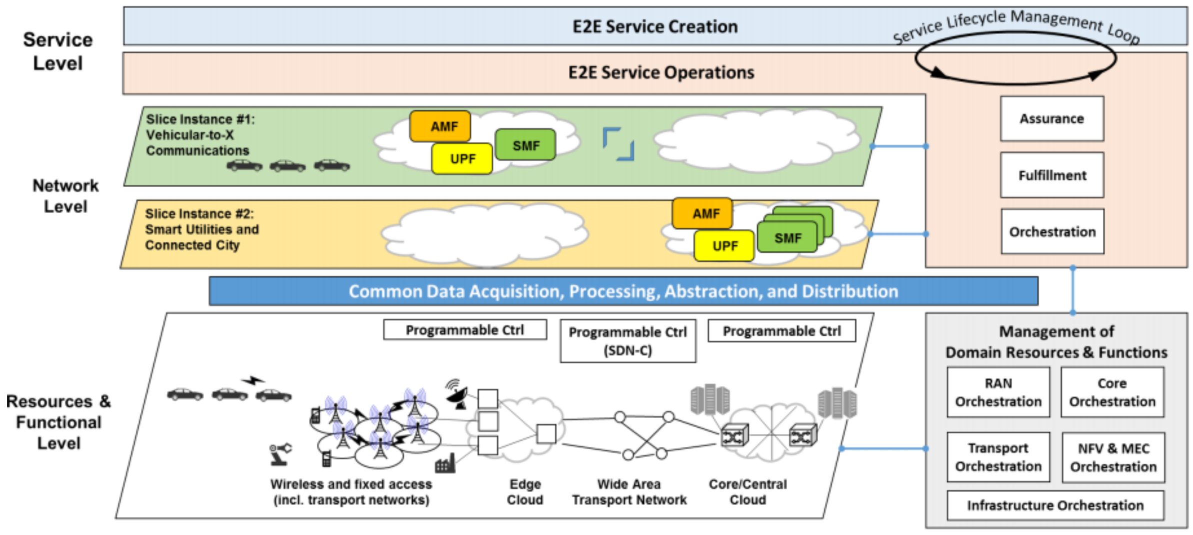
Service Level
E2E Service Creation
Service Lifecycle Management Loop
E2E Service Operations
Network Level
Slice Instance #1: Vehicular-to-X Communications
AMF
UPF
SMF
Slice Instance #2: Smart Utilities and Connected City
Assurance
Fulfillment
Orchestration
Common Data Acquisition, Processing, Abstraction, and Distribution
Resources & Functional Level
Programmable Ctrl
Programmable Ctrl (SDN-C)
Programmable Ctrl
Wireless and fixed access (incl. transport networks)
Edge Cloud
Wide Area Transport Network
Core/Central Cloud
Management of Domain Resources & Functions
RAN Orchestration
Core Orchestration
Transport Orchestration
NFV & MEC Orchestration
Infrastructure Orchestration

*Figure 4 5G Overall Architecture [2]*

and BDMA technique will divide that antenna beam according to locations of the mobile stations for giving multiple accesses to the mobile stations, which correspondingly increase the capacity of the system [13]. 5G targets at improving the exiting 4G system by introducing higher capacity, higher data rate, lower end to end latency, massive device connectivity, reduced cost, and consistent quality of Experience provisioning [14]. The ability to handle higher device connectivity and higher channel capacity would be improved by MU-MIMO (Multi-User MIMO) [37]. In comparison to single-user MIMO, MU-MIMO allows multiple wireless devices to simultaneously send or receive multiple data streams.

#### 2.3.7.1 Challenges Addressed by 5G

5G was set out to address 6 challenges moving forward from the fourth network generation. These challenges according to 5G-PPP include the following [40]:

1. **Higher System Capacity**: 5G aims at providing 1000 times higher wireless area capacity and more varied service capabilities compared to 2010. It also targets at achieving a 1000-fold system capacity per $km^2$.
2. **Higher Energy Efficiency:** 5G aims at saving up to 90% of energy per service provided. The focus will be on mobile communication networks where the dominating energy consumption comes from the radio access network. It also targets a 10 years battery life for low power machine-type devices.
3. **Lower service creation time**: 5G targets at reducing the average service creation time cycle from 90 hours to 90 minutes.
4. **Security and reliability:** The fifth network generation looks to increase the security and reliability of the existing 4G infrastructure by creating a more secure, reliable, and dependable Internet with a “zero perceived” downtime for service provision.
5. **Support Massive connectivity**: 5G would support more devices than the 4G infrastructure. Therefore, it will help to facilitate very dense deployments of wireless

communication links to connect over 7 trillion wireless devices serving over 7 billion people.

6. **Lower service cost**: Ensuring for everyone and everywhere the access to a wider panel of services and applications at a lower cost.

#### 2.3.7.2 5G Use Cases

Figure 5 shows an overview of the various applications of 5G use cases with different scenarios in different fields. The 3 main use cases are discussed in this section.

- **Enhanced Mobile Broadband (eMBB):** eMBB [6] is proposed to provide wide-area connectivity in both rural and urban areas that support stable data connections with very high peak data rates, as well as moderate rates for mobile devices. The key point is that users have access to a stable mobile broadband service anywhere and anytime and no two eMBB devices access the same resource simultaneously. This would be difficult to achieve due to the cost of deployment especially in rural areas with low population density distribution and low average revenue per user. eMBB will aim at maximizing data rate with a very low packet rate while guaranteeing moderate reliability. Application services that would benefit from eMBB require enhanced connectivity, higher user mobility, and higher connectivity. These applications include Augmented Reality /Virtual Reality (AR/VR) applications, 360 video streaming, etc.

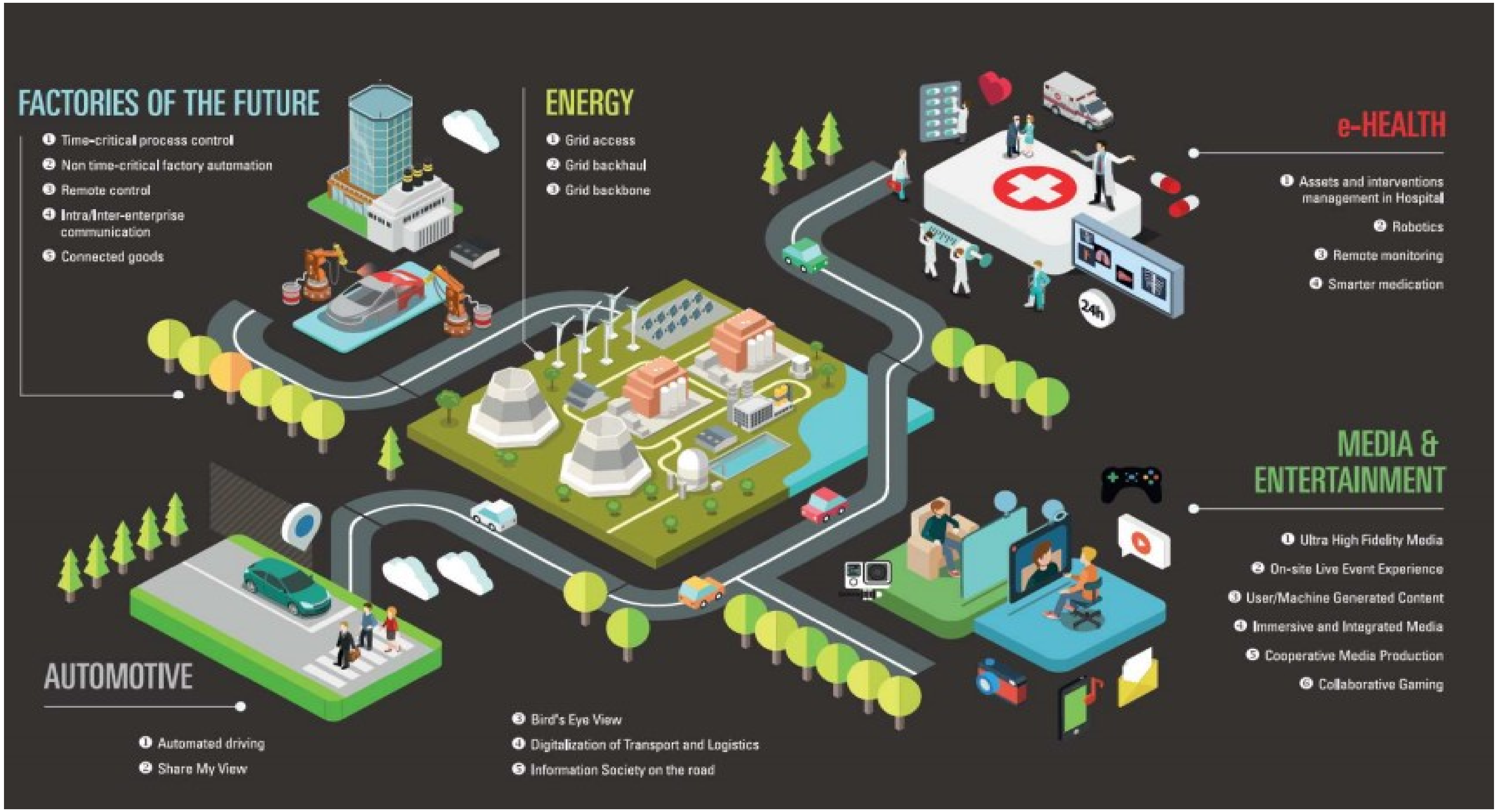
FACTORIES OF THE FUTURE
❶ Time-critical process control
❷ Non time-critical factory automation
❸ Remote control
❹ Intra/Inter-enterprise communication
❺ Connected goods
ENERGY
❶ Grid access
❷ Grid backhaul
❸ Grid backbone
e-HEALTH
❶ Assets and interventions management in Hospital
❷ Robotics
❸ Remote monitoring
❹ Smarter medication
24h
MEDIA & ENTERTAINMENT
❶ Ultra High Fidelity Media
❷ On-site Live Event Experience
❸ User/Machine Generated Content
❹ Immersive and Integrated Media
❺ Cooperative Media Production
❻ Collaborative Gaming
AUTOMOTIVE
❶ Automated driving
❷ Share My View
❸ Bird's Eye View
❹ Digitalization of Transport and Logistics
❺ Information Society on the road

*Figure 5 5G Use Cases Bird's Eye View [30]*

- **Ultra-Reliable Low-Latency Communications (URLLC):** URLLC [7] is likely the most talked-about 5G use case mainly because of the huge services it can support. URLLC aims to deliver a very reliable mobile wireless network with very low latency requirements. URLLC supports low-latency transmissions of small payloads with very high reliability from a limited set of terminals, which are active according to patterns typically specified by outside events, such as alarms. The URLLC requirements for the user plane as stated by [7] includes:
  - A reliability requirement of 99.999% with a user-plane radio latency of 1 ms for a single transmission of a 32-byte long packet.
  - An average user-plane radio latency of 0.5 ms for both uplink and downlink, without an associated reliability value.

Application services in different fields would benefit from URLLC development: examples are listed below.

  - Multimedia: URLLC can help improve use cases such as online gaming and video streaming where low latency is crucial.
  - Medical: URLLC can help deliver and support use cases with mission-critical communication links such as remote surgery with tactile internet. Other medical services that can be supported by URLLC include surgical robots, biosensors, and remote imaging.
  - Transport: URLLC would help support services like Autonomous driving and drone applications.

- **Massive Machine-Type Communications (mMTC)** [6]**:** With the rise of IoT and Machine to Machine (M2M) communication, mMTC was born. mMTC is a type of communication between machines using wired or wireless medium for actuation, information exchange, and data generation with little or no human intervention. In the context of 5G, mMTC would be particularly involved with wireless connectivity and communication of billions of devices and a key progression from IoT to the Internet of Everything (IoE). mMTC is dependent on eMBB to provide wide-area connectivity that can support the vast number of machine to machine interactions and also on URLLC to provide a reliable network with low latency for M2M communication. mMTC has a vast number of opportunity and use cases to offer and thus, has a lot of potentials to drastically change the day to day living of a substantial number of the population. This would be achieved in the part mMTC would play in the development of Smart cities with its contribution to IoE. Application services that would benefit from mMTC include drone applications, remote monitoring devices for health care, surveillance cameras, remote robotics, smart buildings, smart wearables, etc.

#### 2.3.7.3 Enabling 5G technologies

In this section, 5G enabling technologies are discussed. There are quite a few technologies that are contributing to the realization of 5G. Some of these technologies form the foundation on which 5G is built on and the others help contribute by solving challenges that the 5G system faces. In this section, some of the contributing technologies are discussed. Emphasis is made on the technologies that are related to this research.

##### 2.3.7.3.1 SDN

According to RFC 7426 [16], Software Defined Networking (SDN) refers to the ability of software applications to program individual network devices dynamically and therefore control the behaviour of the network as a whole. SDN can also be defined as a set of methods used to facilitate the delivery, design and operation of network services in a deterministic, dynamic and scalable manner. SDN introduces an abstraction and separates the forwarding and the control plane. This provides means for network control and manageability through network programmable applications. In Figure 6, the SDN architecture adapted from [16] is presented. The architecture summaries SDN abstractions in a detailed high-level diagram. The main components include the application plane, the control plane and the management plane. The application plane hosts applications and services that define the behaviour of the network. Applications can be implemented in small units and distributed across multiple planes as seen in Figure 6. The management plane provides a platform for configuring, monitoring and maintaining the network devices. Decisions regarding the state of the network are made in the management network. The management network focuses more on the operational plane than the forwarding plane. The control plane makes decisions on how packets are forwarded by one or more network devices and send the decisions down to the network device to follow. The control plane focuses more on the forwarding plane than on the operational plane. Communication between the control plane and the network device is done through the southbound interface with APIs like OpenFlow while communication with the control plane and the application plane is done with the northbound interface with APIs like the Representational State Transfer (REST). As computer networks become increasingly complex to manage, SDN would be increasingly important in reducing this complexity through its centralized interface. Thereby providing a platform to effectively manage network resources and improve network communications [41].

There are various research papers on SDN since its introduction. Liang and Qiu [42] have investigated suitable security architecture for SDN-based 5G network. They have argued that there is not a lot of work done on SDN security issues for mobile networks. Considering the mentioned problem, they proposed a software-defined security architecture for SDN based 5G network. The authors added a centralized security controller in their architecture which communicates with the SDN controller.

Ma et al [17] have studied SDN/NFV based frameworks for 5G service networks. The authors highlighted the issue with the management and deployment of network functions to provide customized services. To address this issue, the authors have proposed a management

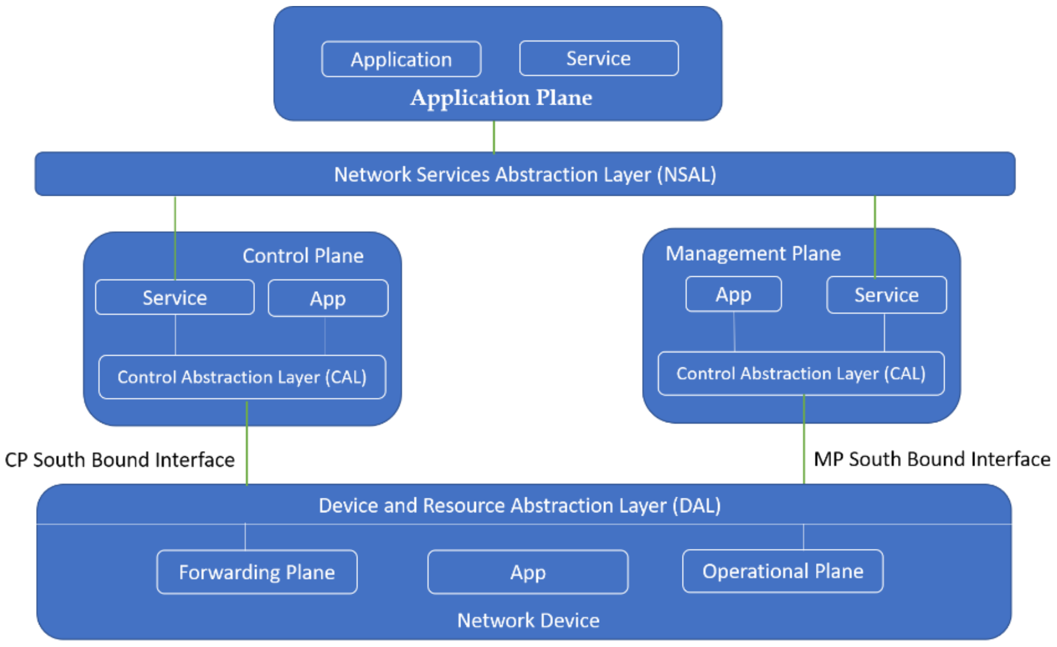

*Figure 6 SDN Architecture*

architecture for 5G service-based core network, based on SDN and NFV. The authors stated that combining their framework with SDN, NFV and edge computing could provide a distributed and on-demand deployment of network functions, network slicing, flexible orchestration, and resource provisioning. The authors concluded that simulation results depict that their framework is effective in operating costs.

#### 2.3.7.3.2 Network Function Virtualization (NFV)

Herrera et al [43] have defined NFV as a new network architecture framework where network functions that traditionally used dedicated hardware (middleboxes or network appliances) are now implemented in software that runs on top of general-purpose hardware such as high volume server. During physical network infrastructure updates in carrier networks, CapEX and OpEX often increase due to the need for specialized network hardware, referred to as hardware appliances or the middlebox. These hardware appliances are used by the carrier and network operators to meet their SLA's and they include QoS monitors, video transcoders, gateways, and proxies. The use of this hardware appliance introduces challenges associated with energy costs and CapEX [43]. NFV promises to resolve these issues by using virtualized technology to offer a new way of network design and eliminate the constant proliferation of hardware appliances. NFV helps realize hybrid scenarios of virtualized and physical networks thus leading to network cost reduction and network optimization. NFV is sometimes confused with VNF which has a different meaning. VNF [44] refers to the implementation of a network function using software that is decoupled from the underlying hardware. NFV would help in the full utilization of resources by having multiple network functions running on a single device. However, the challenge is effectively determining the optimal allocation of resources for a virtual network function [43].

Gero et al [45] have investigated orchestration in 5G using NFV. The authors propose a 5G orchestration tool for the multi-provider NFV framework. To validate the framework, the authors ran an industry control 5G use-case where a VNF offers a 3rd party solution as a VNFaaS.

Baldoni et al [46] have studied the use of the NFV framework for 5G neutral hosts. The authors have utilized NFV systems to address the pressing requirements regarding infrastructure management and multi-layer orchestration. They also address the challenging issue of merging the orchestration life-cycles of NFV and MEC. The authors propose a solution that combines extensions in the orchestration and Virtual Infrastructure Management Layers, along with solutions to the open issues for NFV MEC integration, to pave the way towards edge-aware NFV solutions for 5G neutral hosts.

Gharbaoui et al [47] have researched the latency impact on 5G SDN NFV infrastructures due to the self-adaptive service chaining. The authors demonstrated an orchestration system for 5G infrastructure supporting latency-minimized and self-adaptive service chaining over geographically distributed edge clouds interconnected through SDN.

#### 2.3.7.3.3 Information-Centric Networking (ICN)

There has been a rapid growth in IP traffic from 2014 to 2020 due to an increase in mobile traffic from social media and the popularity of IoT [1]. With its growth expected to increase in the upcoming years, there is a need to change the way data is accessed on the internet as the traditional infrastructure lacks adequate support for content distribution. Although there are application-specific solutions like CDN [48] and P2P [49], these come with issues such as a lack of incentives for peers to share their resources and security problems [18]. This is mainly because it is designed as a host-centric networking framework. Therefore, to obtain data from the internet, the IP address of the data must be specified. However, obtaining the data from the specified location might not be the best option as it might be distant, and the data could be obtained from a closer node. This issue has a significant impact on network performance as the end user's QoE, bandwidth, delay, and energy consumption could be compromised [50]. Data consumption on the Internet has slowly moved from location-dependent to information-dependent. Information-Centric Communication (ICN) is advocated to shift the communication focus from data location to the data itself by making the named data the priority in the network. Therefore, data can be sourced from the internet using the named data and not the data location or the IP address. ICN enables in-network caching and name-based packet forwarding, thus ICN nodes temporarily store cache contents. Therefore, one of the challenges associated with ICN is cache resource management.

There is a reasonable amount of research work done in the field of ICN which includes architectural design proposal, data naming and addressing, caching, data flow control, and mobility. Ueda et al [51] investigated the use of ICN for global content distribution. The authors presented a potential challenge of performance degradation in the global content distribution over ICN. To address this issue, the authors propose an ICN performance enhancing proxy (PEP) to mitigate degradation. According to the authors, this method can accelerate the growth of the end user's congestion window by prefetching future data packets.

The authors have conducted experimental simulations to validate their claim and concluded that their proposed method (ICN-PEP) can reduce startup time in global video streaming.

Jung et al [52] have studied the utilization of on-path and off-path caching in ICN in-network storages. They argued that on-path caching had limitations because it bounds the caching to ICN nodes on the request path. To resolve this issue, they proposed a multiple hash routing mechanism that utilizes an off-path caching scheme for fast data retrieval. This works by hashing data-name into a geographical value which in turn helps in locating the nearest data storage.

Hasan et al [18] have investigated the problem of multimedia caching in ICN. The authors have presented a solution for an efficient content retrieval mechanism based on cluster-based caching in ICN. The proposed solution has utilized a cluster-based mechanism to solve the problem of on-path caching in ICN and increase the probability of content access and decrease the packet loss ratio. The simulation results obtained during the research shows reduced content transfer time compared to other methods.

###### 2.3.7.3.4 IoT

The Internet of Things (IoT) has been the buzzword when discussing the application of 5G. IoT encompasses everything connected to the internet. The term is recently being used to refer to sensors, smartphones, wearables, and devices which usually do not have network connectivity but now do and now can communicate with other devices on the internet. The combination of the data generated by these IoT devices and automation systems would aid in collecting information that can be analyzed to extract knowledge. IoT [53] can be defined as a system of interrelated computing devices, mechanical and digital machines or objects that are provided with Unique Identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. Due to the limited amount of resources available on IoT devices, resource management in IoT is crucial to fully utilize its benefits. There are several types of research on the application of IoT in 5G and the role of IoT in tactile internet. Hannaidh et al [54] have studied the thermoelectric energy harvesting devices to enable self-powered systems for IoT applications bringing sensors to places previously thought impractical or inaccessible. Kapassa et al [55] have investigated the use of IoT applications in 5G environments. The authors have studied the problem of high traffic demand in networks due to an increase in IoT applications and how network slicing would help solve this problem. They proposed an IoT-oriented architecture that supports network slicing for 5G enabled IoT services over the 5G core to meet requirements for establishing a network with high capacity while ensuring the maximum QoS to end-users. The proposed architecture has been developed using the MANO [56] framework and it aims at supporting the ultra-reliable 5G network infrastructure.

Redondi et al have examined the use of IoT in 5G networks using MECs [57]. The authors have investigated the benefits of using publish-subscribe IoT protocols with advanced caching and aggregation functionalities in a distributed system of MECs. The authors identify the underlying challenges in scaling up the pub-sub architecture due to the increase in IoT applications in 5G networks. Their proposed ideal solutions include automatic broker discovery, static broker bridging, and selective event routing. They concluded that information-centric networking could play a role in the efficient dissemination of information among brokers and many-to-many communication scenarios.

Wang et al [58] have investigated the role intelligent IoT will play in wireless communication systems and 5G technologies. The authors have studied the enormous data that would be generated by IoT sensors and the challenges that could be faced in data processing and data mining. To address the issue, the authors propose an IoT based intelligent algorithm for 5G technologies to process big data intelligently and optimize communication channels. To validate the efficiency of the proposed algorithm, the authors have provided an experiment for channel utilization and analyzed the changes in the key evaluation indicators.

#### 2.3.7.3.5 Edge Computing

Cloud computing has been widely adopted in the last decade. This surge in popularity is due to the benefits it provides to users. These benefits include a vast centralized resource for computing, storage, and network management. These resources are elastic so they can be scaled or minimized depending on the user's requirements. This model has seen the rise of businesses that offer central platforms (e.g. Amazon web service, Google Cloud and Dropbox etc.). However, in recent years, the traditional centralized cloud is no longer enough to keep up with new technological trends like IoT and latency-critical applications. This is mainly due to the centralized architecture of the cloud which cannot support latency-critical applications. To address this issue and decentralize the cloud platform, edge computing was born. Taleb et al [10] have described edge computing as an emerging ecosystem that aims at converging telecommunication and IT services, providing a cloud computing platform at the edge of the radio access network. Edge computing offers cloud services at the edge of the network thereby reducing end-to-end latency for UE, easing traffic in the core network, enabling computational-intensive applications and the use of resource-constrained devices.

Figure 8 [59] depicts the high-level architecture for edge computing. As shown in the diagram, the edge sits between the cloud and the end devices. The research community has been working on several types of Edge Computing (Fog, MEC, Cloudlet) while addressing several issues and challenges. These research challenges include edge orchestration, resource provisioning, offloading and computation, energy optimization, network slicing in the edge, mobility management, security and caching [60].

Wei et al [61] have investigated computation offloading in mobile edge computing. The authors stated that there is an increase in complexity in the development of mobile applications due to the limited resources of mobile devices and the demand for computation-intensive applications. To address this issue the authors suggested that computation offloading using mobile edge computing would be the most suitable fit. The authors proposed a computation offloading system model and architecture for mobile edge computing.

Li et al [62] have studied energy-aware server placement on edge computing. The authors looked to solve the problem of an energy-aware edge server placement by sorting after a placement scheme with low energy consumption. The authors formulated the problem as a multi-objective optimization problem and devised a particle swarm optimization-based energy-aware edge server placement algorithm for an optimal solution. The proposed algorithm was evaluated using a dataset from Shanghai Telecom and they concluded that the algorithm can reduce more than 10% of energy consumption and achieved a 15% improvement in computing resource utilization compared to other algorithms.

Xu et al [63] have conducted a study on resource allocation for edge computing. The authors stated that there are challenges in management and resource sharing to achieve a globally efficient resource allocation. To address this issue, they proposed a model which they named Zenith. The proposed model is responsible for the allocation of computational resources and allows service providers to establish resource sharing contracts with edge infrastructure providers. The contract helps to establish latency-aware scheduling and resource provisioning.

### 2.3.8 6G

The sixth generation of mobile networks (6G) research is still at the infancy stage and there is a lot of discussion on the vision, technologies, and standards. 6G would be required to address new requirements and support new technologies and applications that would be raised within the next decade. Some of the applications that require support from 6G have been discussed by [8]. This includes Holographic Telepresence (HTC), Extended Reality (ER), space and deep-sea tourism, and smart grid 2.0. These applications will require higher ultra-data rates and extreme ultra-low latency, paramount high availability and reliability that supersedes the capabilities of 5G. The feature difference between 5G and 6G has been outlined by [9]. The key summary of the comparison is detailed in TABLE 1.

*TABLE 1 COMPARISON OF FEATURES OF 5G AND 6G NETWORK*

| **Features** | **5G** | **6G** |
|---|---|---|
| **Peak data rates** | 20Gbps | ~1 Tbps |
| **Latency** | 1ms | < 1ms |
| **Area traffic capacity** | 10Mb/s/m$^2$ | 1Gb/s/m$^2$ |
| **Frequency bands** | Sub 6GHz mmWave (24- 52.6 GHz) | Sub 6GHz mmWave band Terahertz band (Visible light band) |
| **Connection density** | 1M devices/Km$^2$ | 10M devices/Km$^2$ |
| **Device services** | Reliable connectivity of devices. | Physical interaction in real-time scenarios. |
| **Network Type** | SDN, NFV, Slicing | SDN, NFV, Intelligent cloud, AI-based Slicing. Machine Learning, Deep Learning. |
| **Computing Technique** | Fog, Edge, cloud computing | Quantum and Edge computing |
| **Mobility** | 500 Km/h | > 700 Km/h |
| **Technology** | D2D communication, Ultra-dense Network, Relaying, Small Cell Access, NOMA | Visible Light Communication, Quantum Communication, Hybrid Access, Haptic technology, Adaptive Resource Allocation. |

5G usage scenarios include URLLC, eMBB and mMTC. However, the authors of [64] have proposed 3 more usage scenarios for 6G including Ultra-reliable Low-latency Broadband Communication (ULBC), ubiquitous MBB (uMBB) and massive Ultra-reliable Low-latency Communication (mULC). The authors have proposed ULBC to support use cases that require both eMBB and URLLC. This includes tactile internet, multi-sense experience, pervasive intelligence, ER and HTC. mULC would support use cases that require both URLLC and mMTC such as intelligent transport and logistics and global ubiquitous connectivity. Finally, uMBB would support use cases that require both eMBB and mMTC such as digital twin and enhanced on-board communications. The crossroad of the usage scenarios of 5G and 6G has been depicted in Figure 7.

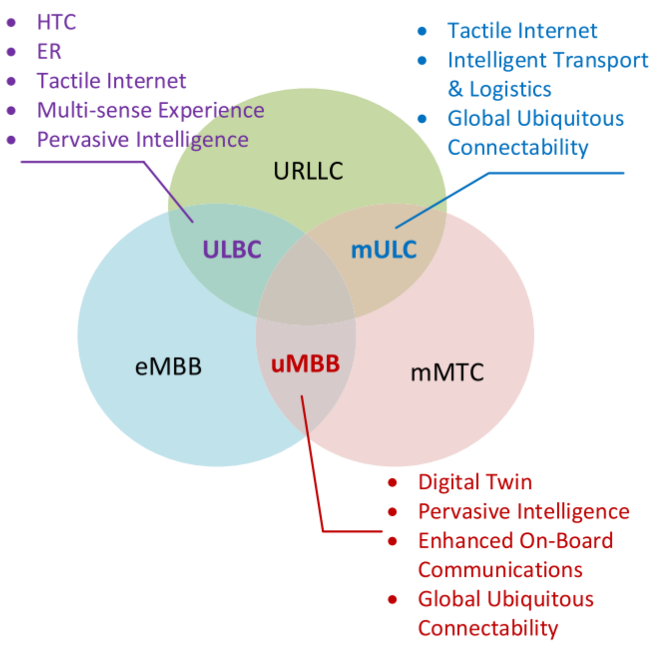

*Figure 7. 5G usage scenarios (eMBB, ULRRC, and mMTC) and the three enhanced scenarios (ULBC, UMBB and mULC) proposed by* [64]

### 2.4 *JOURNEY TO THE EDGE*

In the last decade, there has been a significant evolution of computing paradigms. However, in the last few years, Edge computing has gained a lot of popularity in both industry and academia due to the need for low latency support, the introduction of the Internet of things,

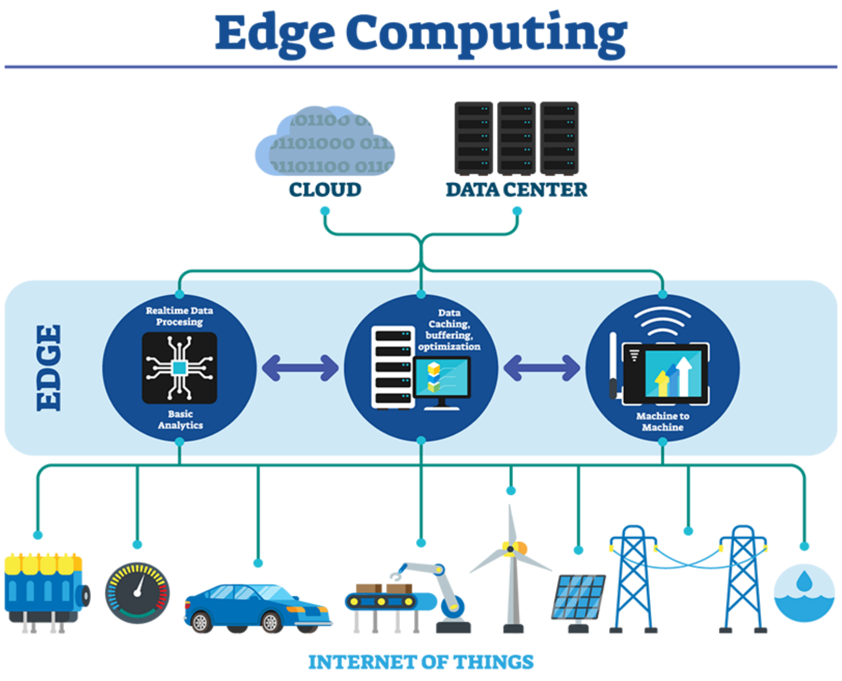

*Figure 8 High Level Edge Computing Diagram*

and the need to decentralize the cloud architecture. Cisco [65] defines Edge computing as a platform that brings processing close to the data source, and it does not need to be sent to a remote cloud or other centralized systems for processing. By eliminating the distance and time it takes to send data to centralized sources, the speed and performance of data transport are improved, as well as devices and applications on the edge. In this section, the evolution of edge computing is discussed and reviewed. Starting from the first pioneer of moving resources closer to the user, Content Delivery Networks (CDN) to Multi-Access Mobile Edge Computing.

### 2.4.1 CDN

The rise of the Internet, in turn, gave rise to the use and development of internet applications both on desktops and mobile phones. These applications need constant internet access for full functionality. However, given the limited bandwidth and sometimes unreliable networks, content delivery through the internet to these applications usually suffer from long delays. This long delay is usually attributed to the fact that there may be only one application content server delivering content to end-users. Therefore, the network latency varies depending on how geographically close the end-user is to the application server. To address this problem, the CDN [66] has been introduced. CDN is a system of distributed proxy servers and data centres that delivers content over the internet to provide low latency, high availability, and performance to end-users. The huge success of CDN started the evolution of moving resources closer to the user hence edge computing [66]. The biggest benefit of CDN is low latency which is achieved by providing a platform closer to the user where content can be cached.

Since the introduction of CDN, there have been many pieces of research done to improve the technology to offer better performance and lower latency. The first generation of CDN introduced in the 1990s has been designed to bring content geographically closer to users across different countries. The second wave of CDN has been designed to address the higher demand for audio, video, and streaming services to accelerate websites and support growing volume content. The third wave of CDN has been focused on cloud computing, energy awareness, and peer-to-peer production. It is expected that the fourth wave would be focused on autonomation and self-management. A review of some of the research work done in the CDN is discussed in the following paragraph.

There has been a lot of research done to improve the CDN by utilizing its platform for content caching thereby reducing users' content access time. Sung et al [48] proposed a cache placement strategy for wireless content delivery networks. The researchers investigated the problem of cache server placement to obtain an optimal placement strategy to reduce caching overhead cost and access time. The researchers proposed to use a star network topology for the origin server and the cache servers. This limits the hop metric from the cache server to the origin server to one, thereby reducing the access time. The researchers formulated the problem into an integer linear programming problem and designed an optimal cache placement strategy for a two-tier wireless CDN with an interference metric, which obtains better results as compared to an existing framework.

Roh et al [67] have researched an efficient content replacement strategy for CDN. Their research focus was to increase the total hit ratio of cache servers in CDN. The researchers have proposed a $p$-based LRFU-$k$ cache replacement framework which they claim solves the problems of exiting cache replacement algorithms like LRU, LRU-$k$, LFU, and LRFU. The $p$ in the name stands for the period span while $k$ stands for the number of time span. Using 30 minutes based LRFU-12, the proposed replacement framework is able to obtain a 6.5% increase in hit ratio during simulations compared to existing algorithms.

### 2.4.2 Cloud

Although the idea of cloud computing was invented in the 1960s by Joseph Carl Robnett Licklider on his project named ARPANET to connect people and data from anywhere at any time. It wasn’t until 2006 when Google CEO Eric Schmidt first used the term in an industry conference and the introduction of AWS S3 by amazon web services the same year that the adoption started to grow rapidly. Following subsequent efforts by CompuServe in 1983 which offered consumers disk space for file storage and AT&T in 1994 launched the first online storage platform for personal and business use. Cloud computing gained more adoption in personal use with the introduction of Dropbox in 2007 using AWS. It was not too long before other Tech Giants started to release their version of cloud environments. Google released its first version of cloud in 2008, Microsoft released Azure in 2010, also Rackspace was formed in 2010 and IBM released its cloud platform in 2011. Within a few years, cloud computing has slowly shifted from just online storage to also a platform that offers computation. This was popularized with the introduction of various cloud services like Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), etc. SaaS is a software licensing and delivery model where software is hosted centrally and paid for on a subscription basis. PaaS model provides customers with a platform to develop, run, and manage applications without the added complexity of building and maintaining the infrastructure of the platform. While IaaS offers computing architecture and infrastructure resources in a virtual environment to multiple users. From 2011 – 2015 businesses were being educated on the benefits of the cloud and how it could potentially help their businesses. From 2016 onwards cloud morphed from an option to an absolute necessity to businesses. Especially small and medium business with inadequate capital to maintain their own in-house IT infrastructure. In parallel to this cloud computing was also making an impact on computer networks and the internet. Most of the web content on the internet is stored in the cloud. There are only a handful of cloud providers to choose from, thereby creating a centralized architecture. Cloud computing provides a centralized platform for storage which is then supported by CDN. Cloud computing introduced a lot of benefits to its users including cost reduction, multitenancy, improved security, and performance.

There has been a wide range of research publications on cloud computing ranging from research on cloud adoption [68], cloud provisioning [69] and orchestration [70], cloud security [71], cloud architectures [72], and cloud applications [73]. In 2010, following the increased popularity of cloud computing, Khajeh-Hosseini et al wrote a case study [74] on how to migrate an enterprise IT system to IaaS. In 2012, Rodríguez-Silva et al [75] proposed the use of the cloud for video surveillance to improve smart surveillance. The researchers reported that the use of the cloud instead of the traditional system will help reduce costs and improve real-time video processing. In 2014, following the increased access to multimedia online, due to increased traffic from social media, Aazam et al [72] discuss the challenges faced by cloud computing and calls for a standardization of Cloud Media and Inter-Cloud computing for better provisioning services.

### 2.4.3 Mobile Cloud Computing

The rise in the use of mobile devices created a surge in mobile traffic. This trend also increased the demand for mobile applications. However, due to the limited resource specification of mobile devices, some heavy resource applications cannot run smoothly on mobile devices. This application includes natural language translators, image processing, virtual reality, etc. To address this problem, Mobile Cloud Computing (MCC) has been proposed. The Idea of MCC is to provide software level solutions for mitigating resource constraints in mobile devices [76]. MCC leverages the computational capabilities of the cloud and offloads tasks from mobile devices to be executed in the cloud. Offloading decisions are made mainly because of battery and resource constraints. Additionally, decisions are made in selecting the appropriate server for offloading. The offloading framework used is either partial or full offload. The need for mobile devices to share computational resources prompted a new type of MCC called Mobile Ad Hoc Cloud (MAC) [77]. The goal of MAC is to offload computational-intensive tasks from a mobile device to another available mobile device in the local vicinity. MAC leverages the heterogenous resource nature of mobile devices.

There were loads of research conducted on mobile cloud computing to be able to reap its benefits and reach its full potential. Farrugia S [78] has conducted research to determine the benefits of MCC and its feasibility. The research has highlighted the key limited resources from mobile devices to be the memory, battery, CPU time, and data usage. They explored MCC solutions and concluded that MCC is useful for extending the computational and storage resource of mobile devices.

AlShahwan F and Faisal M [79] have researched how MCC can help to provide support in delivering complex mobile web services. The research investigated the use of MCC in migrating, offloading, fragmenting, orchestrating, and federating mechanisms. In conclusion, they recommended the use of MCC to overcome mobile resource constraints. They also highlighted issues such as authentication and security of the distributed services and collaborative hosts as concerns to be explored.

### 2.4.4 Edge-Based Technologies

With the introduction of IoT and as the use of mobile devices continues to grow, there has been a demand for latency-critical applications for tactile internet [80]. The International Telecommunication Union (ITU) [81] has defined the Tactile Internet as a network that combines ultra-low latency with extremely high availability, reliability, and security. It is becoming apparent that the existing cloud infrastructure coupled with cloud and CDN could not support these latency-critical applications due to the centralized architecture. Therefore, there is a need to decentralize the cloud and bring its computation and storage services closer to the user. To achieve this, technologies like cloudlet, MEC, and fog have been proposed. They all share similar ideas of decentralizing the cloud infrastructure and bring resources closer to the user.

#### 2.4.4.1 Open Edge Computing/Cloudlet

Satyanarayanan et al [82] have proposed the cloudlet in the US to address the latency constraint of mobile cloud computing by extending the mobile device-cloud architecture. In their research paper, cloudlet has been defined as ***a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and available for use by nearby mobile devices.*** Cloudlets was later renamed Open Edge Computing (OEC). Cloudlets have been proposed to be deployed in places close to mobile devices such as the cellular base station or Wi-Fi access point. Figure 9 illustrates a high-level diagram for cloudlets [83]. The diagram consists of the cloud layer where most of the data and services are hosted. Then, the cloudlets are considered as the intermediary layer between the cloud and the end device. The term cloudlet has become the dominant terminology used to describe this kind of technology in the US. There has been numerous research done on cloudlets on both computational offloading

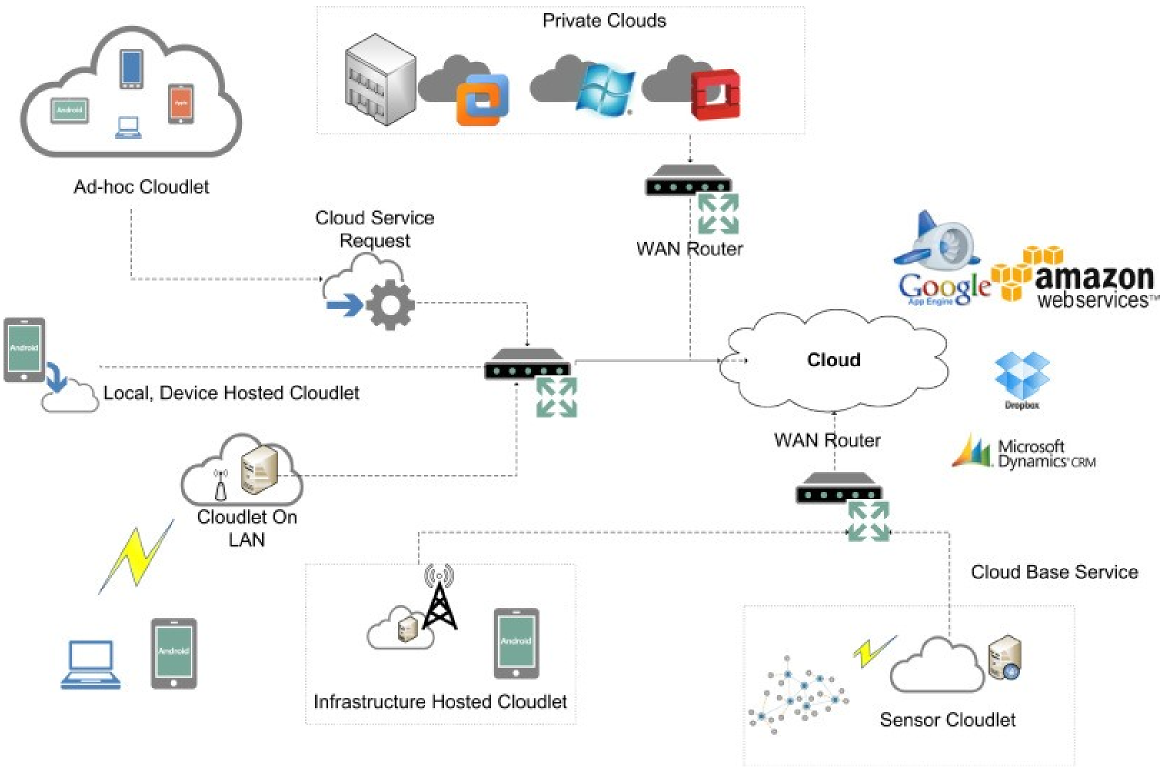

*Figure 9 High Level Open Edge Computing/Cloudlet Diagram*

and also data offloading on cloudlets [84]. One of the popular frameworks for computational offload with cloudlets has been the MAUI framework [85]. The framework has been proposed by researchers at UCL in collaboration with Microsoft and have used a face recognition application as a case study and obtained a significant increase in speed and energy performance during computational offload. Balasubramanian et al. have presented their research on data offloading by using cloudlets to cache data for delayed transfer. They have proposed a system to augment mobile 3G capacity by delay tolerance and fast switching. It utilizes predictions of Wi-Fi offload capacity to offload data as much as possible by delayed transfer [86].

#### 2.4.4.2 Fog Computing

The term Fog computing created by Cisco is defined as ***a standard that defines how edge computing should work. Fog facilitates the operation of computing, storage, and networking***

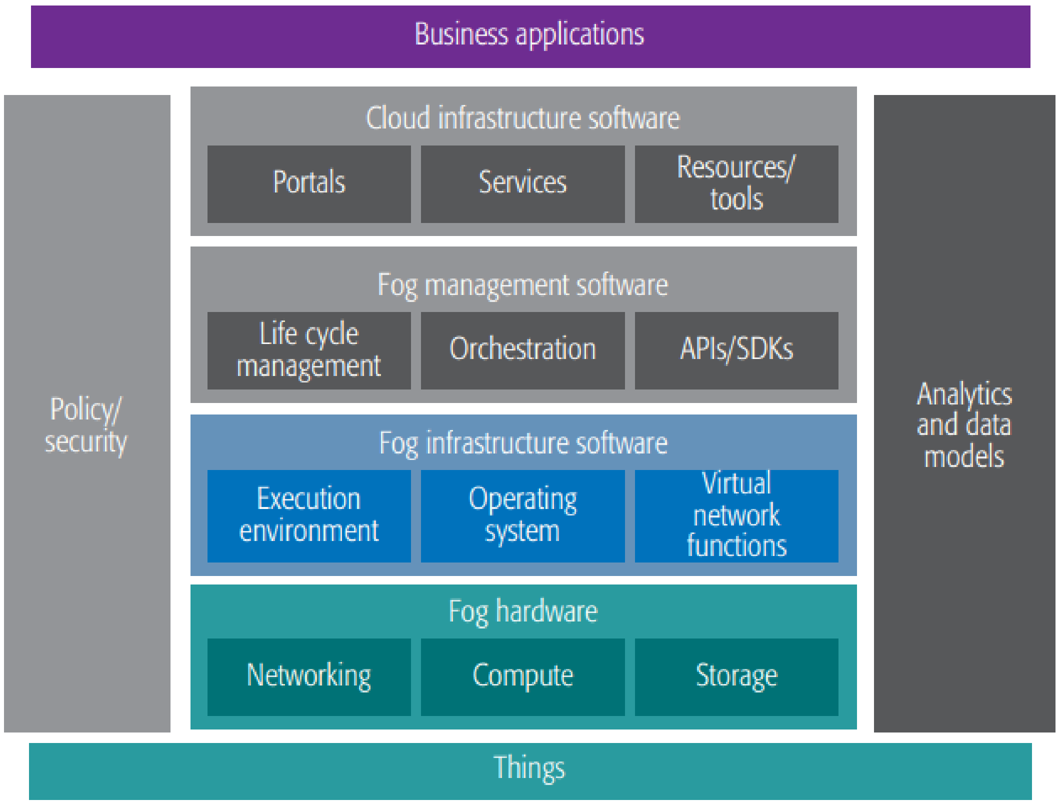

*Figure 10 Fog Software Architecture*

***services between end devices and cloud computing data centres*** [65]. Figure 10 [87] depicts a high-level software stack for fog computing which consists of sensors, actuators, and other things. The fog infrastructure area consists of a versatile execution environment that supports a wide range of operating systems and virtual network functions. The fog orchestration service sits in the fog management layer which supports rich APIs to permit the development of applications, well defined interfaces between the components. The stack also includes the policy/security and analytics and data models present at all levels of the stack. The goal of fog computing is similar to cloudlets in bringing cloud resources closer to mobile devices to reduce latency by decentralizing the cloud. The term fog has been used because fog is a cloud close to the ground. The OpenFog Consortium has been formed by Cisco and other companies to help push the research boundaries of fog computing. OpenFog Consortium defines fog computing [88] as *"a system-level horizontal architecture that distributes resources and services of computing, storage, control, and networking anywhere along the continuum from Cloud to Things"*. With the introduction of the OpenFog Consortium, there have been many research pieces on fog computing to help achieve its goal. Cech et al [89] have researched fog computing and proposed a fog computing architecture to share sensor data with blockchain. The blockchain functionality helped in securely collecting and sharing sensor data. The researchers utilized fog computing for the processing of data collected by the IoT sensors. Pfandzelter et al [90] have researched fog computing. They have highlighted the issues faced on data processing in the cloud which is mainly high latency. They have argued that fog computing is necessary to achieve low latency during data processing by moving cloud computational resources closer to the user.

#### 2.4.4.3 Multi-access Edge Computing (MEC)

MEC formally known as Mobile Edge computing is a term defined by European Telecommunication Standards Institute (ETSI). MEC can also be referred to as a small scale data centre with low to medium computational power and storage resources [91]. The idea is to design mini servers known as edge nodes that would handle storage and computation for mobile devices. These edge nodes are near the end-users providing a platform for caching and offloading to reduce the bandwidth consumption and latency of the network. The edge nodes complement the traditional cloud infrastructure by providing additional resources like cloudlet and fog computing. Figure 11 depicts the MEC framework [92] as defined by ETSI.

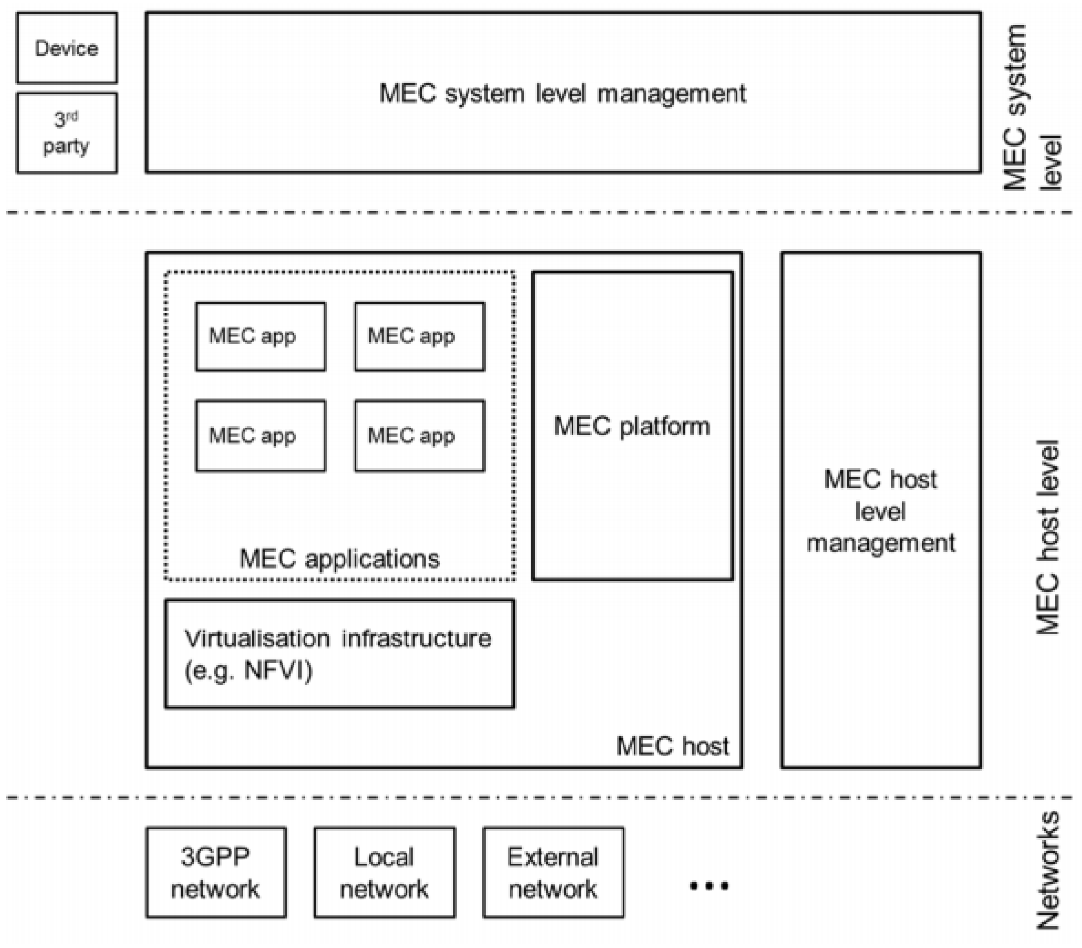

*Figure 11 MEC Framework*

The diagram shows the general entities involved in the MEC framework which are grouped into Network level, host level and system level. The framework defines the implementation of MEC applications as software-only entities that run on top of the virtualization infrastructure.

Although there are many similarities between Cloudlet, Fog, and MEC, there are also some differences. These differences can be seen in the applications they support, the virtualization environment, and also Operational mode [88]. The table shows the differences between cloudlets, fog, and MEC as outlined by [88] and [93]. The frameworks proposed in this research are primarily based on the MEC architecture which includes a three-tier architecture of the cloud, the edge, and the end-user. Therefore, it could be adapted to work on the fog network as it also supports the 3-tier architecture but not cloudlets.

| FEATURES | CLOUDLETS | MEC | FOG |
|---|---|---|---|
| **APPLICATIONS** | Mobile Offloading | Supports applications for both mobile and non-mobile network edge | Supports a wider range of latency-sensitive applications for resource-constraint end devices |
| **VIRTUALIZATION** | Only depends on VMs | Can use other technologies apart from VMs | Other virtualization technologies can be used |
| **OPERATIONAL MODE** | Can work in standalone mode | Can work in standalone and can connect to the cloud | Cannot work in standalone, need support from cloud |
| **DEPLOYMENT COST** | Low | High | Low |

*Table 2 Comparison of Edge Technologies*

## 2.5 CHARACTERISTICS OF MEC

According to ETSI white paper [92], the key characteristics of MEC are listed below.

1. **On-Premises:** Using virtualization infrastructure, MEC would have access to local resources but can be isolated from the rest of the network. This is crucial for M2M and migration scenarios. This network separation also reduces security risks when dealing with safety systems that need high levels of resilience.
2. **Proximity**: Close proximity to the end-user and source of data is one of the key characteristics of MEC. This presents lots of opportunities including data gathering for information analytics. This is also beneficial for computationally intensive applications. The edge can also be leveraged for business-specific applications if it has direct access to the end devices.
3. **Lower latency:** The close proximity of MEC to end devices considerably reduces the network latency. This can be used to minimize the congestion in the core network and improve the QoE of the services delivered to the end-user.
4. **Location-awareness:** MEC can leverage low-level signalling in either WiFi or cellular network to determine the location of the devices connected to the edge network. This can be achieved through information sharing and it would create opportunities for location-based business-oriented use cases.
5. **Network context information:** Applications that provide network information and services of real-time network data can benefit businesses and events by leveraging MEC in their business model [94]. This can be achieved as real-time network data can be utilized to offer context-related services that can differentiate the mobile broadband experience.

## 2.6 NEED FOR EDGE COMPUTING

The trend of moving resources closer to the user is becoming inevitable due to a lot of factors. These factors have huge benefits for both the end-users and the network operators. In this section, the factors that lead to the realization of edge computing as outlined by [95] have been discussed.

6. **Real-Time QoS and Delay Sensitiveness:** The computational resources of mobile devices have gradually increased over the years. However, the computational resources are still not adequate for accomplishing most real-time use cases that have pre-defined QoS requirements. Cloud computing has been the key support for these devices by providing a pool of computation and storage infrastructure for devices with limited capacity. However, some of the emerging applications that run on these devices are very sensitive to delay. Therefore, the cloud environment alone would not be suitable for these latency-critical applications due to distant travel and real-time requirements. These case study applications include IoT sensors and robotics in the healthcare domain and V2X communication in autonomous vehicles. These applications are real-time sensitive and thus require high QoS. Therefore, by deploying servers closer to the devices that require real-time interaction, the overall latency can be decreased through high LAN bandwidth and a decreased number of hops [95].
7. **Battery Lifetime**: Energy consumption is a crucial element when discussing mobile devices. Due to its mobility feature, it cannot always be connected to an energy source like servers or desktops. Unlike the computation resources, the battery life of mobile devices has made very little improvement over the years. Task offloading has proved beneficial in the energy consumption reduction of mobile devices [76]. Task offloading is achieved either by utilizing the cloud server or by leveraging the edge servers.

However, it has been proved that offloading to the edge server achieves lower energy consumption than offloading to cloud servers [85] [96]. The execution of tasks locally in the device generates the highest energy consumption, unsurprisingly. Energy consumption is even more vital for IoT devices and wearables which have lower battery life than mobile phones. Offloading the workload from such devices to the edge would be more efficient in reducing the energy consumption and increasing the battery lifetime [97].

8. **Regulating Core Network Traffic:** With the continuous increase in the number of connected devices, there is a need to reduce network congestion at the core of the network. The congestion is caused by the limited bandwidth available at the core network. The traditional approach allows data generated by end devices to flow through the core network to access cloud servers for processing. However, processing this data at the edge would reduce the burden at the core network and optimize bandwidth. Additionally, data processing at the edge would also lead to reduced demand for computational resources from cloud servers.
9. **Scalability:** The number of end-user devices is expected to reach trillions in a few years and this evolution creates a significant scalability problem [98]. The cloud servers support dynamic scalability to meet user demands. However, congestion in the data centres may occur due to tremendous data volume being sent to cloud servers for processing. This problem occurs due to the centralised structure of the cloud environment. However, using a distributed architecture like the edge network, these services and applications can be replicated in the form of microservices and deployed over the edge servers. Therefore, if any of the edge servers becomes congested or fails, the corresponding microservice can be replicated and deployed in another edge server in the vicinity. Additionally, data pre-processing on the edge would reduce the burden in the cloud.

## *2.7 RESOURCE MANAGEMENT OF MEC*

MEC has been explored in this thesis rather than fog or OEC because MEC has been defined by the European Telecommunication Standards Institute. MEC facilitates enhancements to the existing applications and network infrastructure by providing two main services at the edge of the RAN. These include computational and caching services. These services contribute to the achievement of the factors highlighted in section 2.6. Therefore, to realize the full potential of MEC, it is important to effectively manage the computational resource and the caching resource.

### 2.7.1 COMPUTING IN MEC

Computation is one of the key services offered by MEC. This would be available as a large-scale distributed computing paradigm in which a pool of computing resources is available to users. This provides a platform for offloading and execution of tasks from devices with limited computational capacity. Computation offload in MEC would help to achieve some of the factors highlighted in section 2.6 including latency reduction, lower energy consumption, and reduced traffic in the core network. To guarantee ubiquitous services for all devices connected to the MEC, the computational resources should be evenly distributed across the network. Therefore, there is a need to design an optimal placement algorithm for MEC servers for effective resource management. With the deployment of edge servers, a huge load of the computational workload would be moved from the cloud servers to the MEC. The MEC servers have smaller computational resources compared to the cloud servers. Additionally, there would be a huge demand for these computational services due to the increasing amount of connected devices [99]. This excessive demand might cause overprovisioning of resources in MEC and may even lead to deadlock in the system if adequate measures are not in place [100]. Computational offloading and resource provisioning has been explored in the context of MEC [61] [101] [76]. However, the issue of deadlock in MEC during resource provisioning has not been addressed. Therefore, it is important to design an appropriate scheduling algorithm for optimal computing resource management to prevent the problem of over-provisioning of resources or deadlock to improve the QoE. This is one of the major problems explored in this thesis. This is discussed further in chapter 3.

### 2.7.2 CACHING IN MEC

Caching is another key service MEC offers. This will aid mobile end devices to achieve low latency for delay-sensitive applications. This would additionally ease the traffic in the core network and improve the user QoE. Caching in the MEC would help in improving the factors highlighted in section 2.6 by reducing the communication distance, reduction in access latency, reduction in energy consumption, and improvement of user QoE. There has been a continuous increase in the amount of traffic in wireless networks due to the increasing demand for mobile video streaming and the increasing popularity of social networking [102]. The MEC has a limited cache storage capacity so it is impossible to cache all contents on the edge. Therefore, the continuous growth of this data-intensive traffic will likely overwhelm the MEC. Hence, it is crucial to design efficient caching policies to maximize the benefits of local caching and sharing for future mobile networks. To address this problem, optimized cache resource management has been explored in recent years to enhance the caching efficiency, performance, resource management, and overall QoE of MEC [102] [18] [103].

Additionally, user request patterns are ever-changing and so is the popular content. Thus, the request pattern would often not follow the Zipf distribution [104]. Therefore, it is crucial to develop intelligent content caching at mobile network edges to dynamically adapt to the caching request pattern and improve content delivery efficiency. This is another problem explored in this thesis. This is discussed further in chapter 4.

## *2.8 CONCLUSION*

The future is only appreciated if the past is known. In this chapter, the background and the current state of the art of networks have been reviewed. This chapter has set the scene for the thesis by going through the decades of research effort made on mobile wireless networks. Section 1.3 has discussed the evolution of mobile networks from 1G to 5G. It also discussed the challenges, use cases, and enabling technologies of 5G. Additionally, it discussed the necessity of MEC in 5G to achieve the URLLC case study.

Section 1.4 has outlined the journey to the edge and reviewed the importance of technologies such as CDN, cloud, and MCC. The different edge platforms are then reviewed which includes fog, cloudlets, and MEC. The characteristics of MEC and the need for MEC have been discussed in sections 1.5 and 1.6 respectively.

Finally, in section 1.7, the core services that the MEC provide and why resource management in MEC is crucial to fully achieve the benefits of MEC has been discussed. Here, the problems explored in this thesis are stated and discussed. It is important to optimize the caching and computing service in MEC as they are equally dependent on each other on achieving the 5G URLLC case study and the improvement of the end-users' QoE.

In the next two chapters (chapter 3 and chapter 4), the highlighted problems have been explored and addressed with significant contributions to knowledge made in the process.

# CHAPTER 3 MEC COMPUTATIONAL RESOURCE MANAGEMENT

## *3.1 INTRODUCTION*

The previous chapter has discussed the recent advancements in computational-intensive and latency-critical applications and the need for edge cloud computing in meeting the latency requirement of these applications through computational offload. In this chapter, this problem has been further investigated. Here, the related literature has been extensively reviewed in section 3.2 where a research gap has been identified. This is due to the lack of research addressing the issue of deadlock during resource provisioning in MEC. Deadlock during resource provisioning in MEC can be described as a state where a process in a MEC node with partially provisioned resources is waiting for another process, including itself to release resources or release a lock to resources causing a circular wait. This problem can also be distributed if the waiting process and the process with the resource are in different MEC nodes which could occur during re-offloading. This chapter addresses the problem of deadlock in real-time MEC services. Therefore, in section 3.2.1 the current research on resource provisioning in MEC has been reviewed. This validates the assumption that very little research on deadlock aware provisioning frameworks exists. Hence, various deadlock and real-time scheduling algorithms have been reviewed in sections 3.2.2 and 3.2.3 respectively. Section 3.3 focuses on the design of a deadlock-aware framework for resource provisioning in MEC environments. Thereafter, section 3.4 presents a comparative analysis with the experimental results of deadlock-aware algorithms based on the proposed framework. Finally, the chapter is concluded in section 3.5.

## *3.2 LITERATURE REVIEW*

Considering the decentralized architecture of MEC as opposed to the traditional centralized cloud infrastructure, it is important to investigate an efficient mechanism to offload and execute mobile and IoT applications on the edge of a network.

There have been several proposals for resource provisioning techniques to offload mobile application workloads on MEC [105] [106] [101]. Nevertheless, none of the previous works on MEC considers deadlock during offloading and resource provisioning which is a concern for distributed systems. There are four major strategies for handling deadlock in distributed systems. These include (i) ignore, (ii) detect and recover, (iii) prevention, and (iv) avoidance. The first two are commonly used because the last two are difficult to implement [107]. Few researchers have opted for detection and recovery strategies as shown in TABLE 3. This is not always ideal because, in a scenario where the system needs to be readily available, any amount of downtime can be very costly. Deadlock avoidance strategy is said to be the most effective, but it is difficult to implement in distributed systems because of communication overheads and therefore labelled impractical [108].

Researchers have previously used load balancing algorithms to level out the workload between servers in MEC and avoid resource over-provisioning [109] [110]. Tham and Chattopadhyay [109] have proposed a load-balancing scheme for distributed computing on the edge of a network based on heuristics. They have used an edge model of a group of nodes connected over a wireless ad-hoc network formulating a convex optimization problem. The simulation results have shown near-optimal performance in most cases. Load balancing schemes reduce the chances of deadlock but do not eliminate it from the system. Deadlock prevention and/or avoidance scheme is a more suitable approach as it eliminates the chances of deadlock in the system [111].

With the advancement of 5G and SDN, the communication overheads can now be reduced thereby making it practical to implement the deadlock avoidance algorithms in a distributed system. The idea of separating the control plane from the data plane means there would be less communication between the routers and switches because they share a centralized control plane [112].

*TABLE 3 DEADLOCK STRATEGIES*

| | |
|---|---|
| Detection algorithms | Lamport's algorithm [113] |
| | Chandy-Misra-Haas algorithm [114] |
| | Parallel Deadlock Detection Algorithm [115] |
| | Detection in heterogeneous systems [116] |
| | Unstructured deadlock detection [117] |
| | Aida et al [118] |
| | Farajzadeh et al [119] |
| | Akikazu et al [120] |
| Prevention algorithms | Load balancing methods [121] [122] |
| | Deadlock Prevention Algorithm in Grid Environment [123] |
| | Wound-wait [124] |
| | Wait-die deadlock [124] |
| | Lin Lou et al [125] |
| | Kamta [126] |
| | Ding et al [127] |
| | Chuanfu et al [128] |
| Avoidance algorithms | Banker's algorithm [11] |
| | Resource allocation graph [129] |

The current state of the art shows that researchers have previously used load balancing to avoid overprovisioning and deadlock in MEC. However, to the best of our knowledge deadlock avoidance has not been addressed in a MEC context. Therefore, in this study, a novel resource provisioning algorithm for deadlock avoidance on multi-access edge computing has been proposed. The proposed algorithm is different from load balancing because in load balancing there is a load balancer that first accepts the request and uses a mechanism to distribute it to servers. As opposed to this, deadlock avoidance is the focus of the proposed method. Here, the task goes directly to the MEC servers for execution and only gets redirected if the time and resource constraints of the task cannot be satisfied while ensuring the system is in a safe state.

The widely used deadlock avoidance algorithm due to its efficiency is the Banker's algorithm proposed by Dijkstra [11]. Banker's algorithm is a resource allocation algorithm that simulates a system using predefined variables and predetermines the safeness of a system before granting a task allocation request [11]. It is mainly used in operating systems where it runs on a single machine. In this study, it has been used in a distributed environment where resource information is shared by systems within the environment.

### 3.2.1 REVIEW ON RESOURCE PROVISIONING IN MEC

Resource provisioning in MEC is a challenging problem for Internet Service Providers due to the impact it has on the efficiency of the system and the QoS. Researchers have previously addressed this resource provisioning problem in cloud computing. However, resource provisioning in MEC is more challenging mainly because the edge servers have more resource constraints than the cloud servers and the edge servers would be deployed as a distributed environment compared to the centralized cloud. Enabling distributed computing and storage capabilities at the edge of the network will benefit delay-sensitive and computation-intensive mobile applications.

There has been a considerable amount of work done in the area of resource provisioning in MEC. Badri et al [130] have proposed a risk-based optimization for resource provisioning in MEC. In their work, they have considered that the resource requirements of mobile applications are stochastic. Therefore, they have formulated a chance-constrained stochastic program problem. They have resolved this using the Sample Average Approximation method.

Kherraf et al [131] have studied resource provisioning and workload assignment in MEC and formulated the problem as a mixed-integer program to jointly decide on the number of nodes, the location of MECs, and applications to deploy. They have solved this by decomposing it into two problems, a delay aware load assignment sub-problem and dimensioning edge servers sub-problem. They have proposed optimized provisioning of edge computing resources with a heterogeneous workload in IoT networks. They concluded that the proposed tool could be used by network operators to develop cost-effective strategies for edge network planning and design.

Chang et al [132] have studied resource provisioning in MEC in the area of minimizing energy consumption of cellular networks. In their research, they have investigated both the communication and computation aspects of resource provisioning to improve energy efficiency. They have modelled the system as tandem queues and studied the trade-off between the subsystems on energy consumption and service latency. Based on this, they have proposed an algorithm to determine the optimal provision of both communication and computation resources to minimize the overall energy consumption without sacrificing the performance on service latency.

Yu et al [133] have proposed a collaborative computation offloading framework for MEC. The authors have considered an offloading scenario where multiple mobile users offload duplicated computation tasks to the edge servers. Hence, creating an opportunity for edge servers to share computational results. The aim is to develop an optimal collaborative offloading strategy with data caching enhancements to reduce end-user latency. The problem has been formulated as a multi-label classification in which a Deep Supervised Learning approach has been employed to address the issue. Numerical results have shown that the proposed scheme achieves reduced delay and energy consumption compared to other schemes.

Zhou et al [134] have proposed a resource provisioning scheme for heterogeneous IoT applications on cloud-edge platforms. The scheme has been aimed at minimizing long-term operational costs while guaranteeing both hard and soft deadlines for heterogeneous IoT applications. The proposed framework employs a Lyapunov optimization technique to make online resource provisioning greedy decisions without prior knowledge of the statistics of the edge system. The authors have evaluated the efficiency of the proposed approach using realistic traffic and cost traces.

Ma et al [135] have proposed a mobility-aware and delay-sensitive service provisioning scheme for mobile edge cloud networks. The authors have formulated two novel optimization problems of user service request admissions with the focus of maximizing the accumulative network utility and throughput. The authors have utilized a constant approximation algorithm and an online algorithm to address the formulated problems. The authors have demonstrated the efficiency of the proposed scheme using experimental simulations.

There have been other proposals for resource provisioning techniques to offload mobile application workloads on MEC [136] and [137]. Nevertheless, none of the previous works on MEC considers deadlock during offloading and resource provisioning which is a concern for distributed systems such as MEC. It is quite unexplanatory why this research area in MEC is not a priority given the consequences. This has created motivation for this research to fill this void.

### 3.2.2 DEADLOCK HANDLING STRATEGIES

The previous section has reviewed resource provisioning frameworks on MEC, and it has been concluded that there is little research on deadlock aware resource provisioning frameworks in MEC. Therefore, this section reviews the various strategies for handling deadlock.

In a trusted computing scenario using edge nodes and IoT devices, where high availability and reliability are crucial factors for a good user experience, deadlock-free operations are

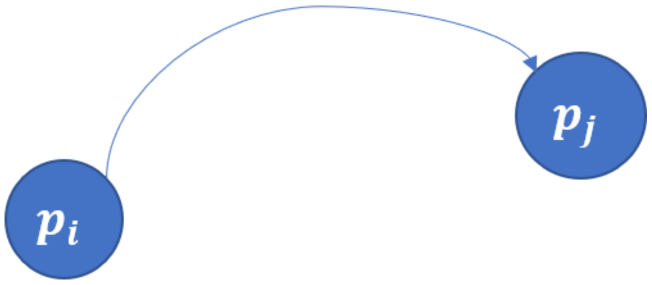

*Figure 12 Wait For Graph (WFG)*

important in achieving this goal. The absence of deadlock strategies to detect, recover or eradicate deadlock on such a system might cause deterioration of the system's performance and ineffective use of energy as deadlock might occur but the system has no way of recognizing what has happened. The standard toolset for deadlock detection is the Wait for Graph (WFG). This model's relationship between the processes and the resources involved. Here, each node represents a process ($p_i$) and an arc is originated from a process waiting for a resource to a process ($p_j$) holding the resource as represented in Figure 12. In this section, deadlock handling strategies have been reviewed. There are four conditions necessary for a deadlock to occur and each of the strategies tries to eliminate one of them.

- Conditions necessary for a deadlock to occur include the following:
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait

#### 3.2.2.1 Deadlock detection

In the design and development of a multi-threaded system, deadlock detection could be chosen as a way of handling deadlock in that system. If the algorithm employed detects a deadlock, the next step would be to recover the system from the deadlock it suffered. Therefore, deadlock detection and recovery go hand in hand. Here, an algorithm is employed to examine the state of the system to determine if a deadlock has occurred. If this is the case, another algorithm is used to recover the system from deadlock. To detect the presence of deadlock, a resource allocation graph, and a corresponding WFG is used for a single instance for each resource type. Note that in this strategy, deadlock may occur, after which the system then detects the occurred event and then tries to recover itself. This causes an overhead of the run-time costs of maintaining the necessary information and executing the detection algorithm. Additionally, there might be potential losses inherent in recovering from a deadlock [138].

To detect deadlock for a single instance of each resource type using the wait-for graph, an edge from $E_i \; to \; E_j$ in a wait-for graph implies that process $E_i$ is waiting for the process $E_j$ to release a resource that $E_i$ needs. The edge $E_i \rightarrow E_j$ only exists in a WFG if the corresponding resource allocation graph contains two edges $E_i \rightarrow R_q \; and \; R_q \rightarrow P_j$ for some resource $R_q$. A deadlock exists in the system if there is a cycle in the WFG. Using this, the detection algorithm requires a runtime order of $n^2$ operations where $n$ is the number of vertices in the graph. For several instances of a resource type, the runtime order to detect a deadlock would be $m * n^2$ where $m$ is a vector that indicates the number of available resources of each type [138].

There is extensive research on deadlock detection schemes. Aida et al [118] have proposed a novel scheduling strategy for efficient deadlock detection. They have championed the deadlock detection strategy as a more optimistic and feasible solution to resolve a deadlock.

They have tested the efficiency of their algorithm based on its scalability in the number of resources and processes. They concluded that the performance of a deadlock handling algorithm depends fundamentally upon deadlock detection scheduling and the rate of deadlock formation. Farajzadeh et al [119], have proposed a distributed deadlock detection algorithm based on history-based edge chasing which resolves the deadlock as soon as it detects it. According to their research, this action reduces the average persistence time of the deadlock compared to other detection algorithms. Akikazu et al [120] have proposed a deadlock detection algorithm for distributed processes. In their research, they have formulated a deadlock detection scheduling problem with the presence of system failures and derived a deadlock detection time that minimizes long-run average cost per unit time. They have concluded that the number of distributed processes and the system failure probability give a great effect to the long-run average message-complexity per unit time, but not the deadlock scheduling time. Other research on deadlock detection includes Lamport's algorithm [113] and Chandy-Misra-Hass algorithm [114].

#### 3.2.2.2 Deadlock Prevention

Deadlock prevention algorithms handle deadlock in a system by trying to prevent one of the necessary four conditions required for a deadlock to occur from happening. In a typical distributed system, there is at least one non-sharable resource. Therefore, the mutual exclusion condition must hold. Due to this, deadlock cannot be prevented by denying the mutual exclusion principle. To prevent deadlock by eliminating hold and wait, two possible protocols could be used.

- The first one requires that all resources a process needs are allocated to the process before the start of execution. This will eradicate hold and wait but might lead to the underutilization of the system.

- Another method could be to allow a process to request new resources only after releasing the current set of resources. This may lead to starvation.

Deadlock can also be prevented by preempting resources from a process if the resources are required by a higher priority process. This strategy of process termination during execution is inappropriate for real-time systems in which the elapsed execution time of the process must be predictable [139]. The final method of preventing deadlock is done by eliminating the circular wait condition. To ensure that this condition never holds, a protocol can be used to impose a total ordering of all resource types and require that each process requests resources in increasing order of enumeration. As an example, if $R = \{R_1, R_2 .. R_z\}$ is a set of resource types that have been assigned unique integer numbers from $1\ to\ z$. A process can only request resources in increasing order of enumeration. Therefore, if a process request $R_i$ then it can only request for another resource $R_j \Leftrightarrow F(R_j) > F(R_i)$ [138].

There have been many deadlock prevention strategies proposed by different researchers in different computer science fields to eradicate deadlock by preventing one of the necessary four conditions required for a deadlock to occur. In the field of Service-Oriented Architecture infrastructure, Lin Lou et al [125] have proposed a deadlock prevention strategy to eradicate the possibility of a deadlock that may be caused by resource locking based two-phase commit protocol. This requires that each transaction obtains all needed locks before the second commit phase. To solve this, they have utilized a timestamp-based restart policy for global resource allocation. Kamta [126] have also investigated the use of deadlock prevention algorithms in distributed systems. In the research, a voting and priority mutual exclusion-based approach has been utilized. To eradicate possible ties that may be caused by this approach, constraints such as Shortest Job Scheduling First and non-mask-able interrupts have been employed. The research has concluded that the proposed approach prevents processors from entering an idle state and reduces problems like resource starvation and unfairness. As a result of this, throughput may be increased in the system.

In the context of web Service-Oriented systems where the competition of resources by web services could lead to deadlock. Ding et al [127] have proposed a method to analyze and verify the deadlock prevention solutions using trace semantics of Communication Sequential Processes. The proposed formal modelling approach proved useful in the verification of deadlock solutions analyzed in the paper. Furthermore, in the context of Grid systems with resource sharing capabilities, simultaneous requests of co-allocation of resources by multiple applications could lead to deadlock. To address this problem, Chuanfu et al [128] have proposed a deadlock prevention method for the fast allocation of grid resources based on an atomic transaction. Utilizing this method, all resources required by a process at the time of the request are specified. The request succeeds if all the resources required are available. Otherwise, the request fails and none of the resources requested is granted. The proposed algorithm achieves a lesser average waiting time when compared to an existing algorithm (Order-based deadlock prevention protocol).

In this research, the two preventive algorithms that have been explored are wound-wait and wait-die. Both algorithms use timestamp-based techniques and they favour the older processes with an older timestamp. These algorithms have been reviewed because they are fit for purpose with appropriate time complexity. They have also been used in a practical environment like database systems compared to other algorithms which are mostly theoretical. This is important as this research is mostly experimental.

###### 3.2.2.2.1 Wound-wait algorithm

The wound-wait deadlock [124] prevention algorithm is a non-preemptive technique. Here, when an older process requests a resource that is currently held by a younger process, the younger process is rolled back. However, when a younger process requests a resource that is held by an older process, the younger process waits. If $P_i$ and $P_j$ are both processes and $P_i$ requests for a resource held by $P_j$. Then $P_i$ is rolled back if $t(P_i) > t(P_j)$ . $i.e\ P_i\ is\ younger$ else $P_i$ can wait. Here, $t(P_i)$ and $t(P_j)$ are timestamps.

#### 3.2.2.2.2 Wait-die algorithm

Wait-die deadlock [124] prevention algorithm is a preemptive technique. Here, when an older process requests a resource that is held by a younger process, the older process waits. However, when a younger process requests a process that is held by an older process, it dies. If $P_i$ and $P_j$ are both processes and $P_i$ requests for a resource held by $P_j$. Then $P_j$ is rolled back if $t(P_i) < t(P_j)$ $i.e\ P_i$ is older. Where $t(P_i)$ and $t(P_j)$ are timestamps, else $P_i$ can wait.

### 3.2.2.3 Deadlock Avoidance

Deadlock prevention has been discussed in the last section. The drawbacks of using a prevention method are low device utilization and reduced system throughput. Alternatively, a deadlock avoidance mechanism could be used. In contrast to the prevention method, this works by requiring additional information on the complete sequence about how resources are to be requested. With this prior information of the requests and resources, the system can decide if a process must either run or wait with the motive of avoiding a possible deadlock. For each request made, the system decides by considering the available resources, the resources currently allocated to each process, and future requests and release of resources by processes. For the avoidance method to work, the simplest model requires that the maximum amount of resources for each resource type is declared. With these data, the model ensures that a circular-wait condition never exists during the dynamic resource allocation. Any potentially unsafe resource request is denied. The system is said to be in a safe state if the maximum number of resources requested for each process can be allocated and there exists no possible sequence of future requests in deadlock. If a safe sequence exists, then the system is said to be in a safe state. There is a safe sequence of process $[P_1, P_2, P_3 \dots P_n]$, if the resource request that would be made by each process $P_i$ can be satisfied by the currently available resources including resources held by all $P_j$ $with\ j < i$ [138]. A system is said to be in an unsafe state if it is not guaranteed that all possible sequences of future requests will not produce a deadlock. Not all unsafe states is a definite deadlock. However, it may lead to a potential deadlock in a system. There are two well-known deadlock avoidance algorithms which are the resource allocation graph algorithm and Banker's algorithm.

#### 3.2.2.3.1 Resource Allocation Graph

The resource allocation graph [129] is only used if the resource allocation system has only one instance of each resource type. It is one of the deadlock avoidance algorithms. While applying the resource allocation graph for deadlock avoidance, if a process $P_i$ requests for a resource $R_j$, $(P_i \rightarrow R_j)$, the request is only granted if $R_j \rightarrow P_i$ does not lead to a cycle in the resource-allocation graph. The safeness of the system is checked by using a cycle-detection algorithm which requires an order of $n^2$ operations where $n$ is the number of processes in the system.

#### 3.2.2.3.2 Banker's Algorithm

For a resource allocation system with multiple instances of each resource type, the Banker's algorithm [129] could be used because a resource-allocation graph is not appropriate to such a system. Banker's algorithm is another algorithm for deadlock avoidance. If the Banker's algorithm is used, then each process must declare the maximum amount of resources for each resource type that it will require to complete execution. This declared number must not exceed the total amount for each resource type in the system. If a process requests a resource, the system determines if allocating the resource will leave the system in a safe state. If true, the resources are allocated, otherwise, the process waits until some other process releases enough resources. Four data structures must be maintained while using the Banker's algorithm.

- $\boldsymbol{Available}$: This is a vector of the number of available resources for each resource type. The length of the vector is $m$, where $m$ refers to the number of available resources.

- $\boldsymbol{Max}$: This is a matrix of the maximum resource demand for each process. The matrix size is $mn$ where $m$ is the number of available resources and $n$ is the number of processes.
- $\boldsymbol{Allocation}$: This is a matrix of the number of resources currently allocated for each process. The matrix size is $mn$ where $m$ is the number of available resources and $n$ is the number of processes.
- $\boldsymbol{Need}$: This is a matrix of the remaining amount of resources that each process needs. The matrix size is $mn$ where $m$ is the number of available resources and $n$ is the number of processes. $Need = Max + Allocation$

The time complexity to determine if a state is safe or not is $mn^2$. There has been a lot of research carried out in the improvement of Banker's algorithm over the years. In each case, the algorithm is extended, improved, or applied in a different area in computer science. The most notable adjustments were made in 1999 [140], 2000 [141] and 2006 [142]. Sheau-Dong Lang [140] has assumed that the control flow of the resource-related calls of processes forms rooted trees. Based on this, he has proposed a quadratic-time algorithm that decomposes these trees into regions and computes the associated maximum resource claims before process execution. The information collected is used at runtime to verify the safeness of the algorithm using the original Banker's algorithm.

Tricas et al in 2000 [141] have applied Banker's algorithm in the field of flexible manufacturing systems. They have modelled the problem using Petri nets and proposed two improvements based on the knowledge of process structure. Their research has proven that the improved algorithm has much more concurrency than the original Banker's algorithm.

Lee et al in 2006 [142] have also applied Banker's algorithm in the design of system-on-a-chip. In the research, they have proposed a parallelized version of the Banker's algorithm they termed PBA (Parallel Banker's Algorithm). The proposed approach is fully hardware-oriented and exploits the maximum parallelism available in hardware. According to their research, this has introduced complexity at the expense of reducing the runtime complexity of the algorithm to $O(n)$ in a worst-case scenario.

Other deadlock avoidance algorithms have been developed. This includes the graphical deadlock avoidance algorithm [143] proposed by El-Kafrawy. To solve the deadlock avoidance problem for sequential resource allocation systems, a polynomial graphical solution has been employed. The graph updates dynamically each time a new resource is requested. Another example is the deadlock avoidance algorithm for streaming applications proposed by Li et al [144] using both a propagating algorithm and a non-propagating algorithm.

### 3.2.3 Real-Time Scheduling

In the previous section, different ways of handling deadlock have been reviewed with regards to resource management. Another important aspect of computational resource management, especially during computation resource provisioning is scheduling. Scheduling is important to ensure that each offloaded task is allocated enough time to complete execution. Therefore, minimizing resource starvation and ensuring fairness amongst the processes utilizing the resources. Therefore, various scheduling schemes have been reviewed in this section.

In a trusted computing environment where high availability and reliability are important factors, often there is a specific response deadline time constraint the system must meet. If this is the case, the system is said to be a real-time system. The system may or may not meet this time constraint. This depends mainly on the capacity of the system to perform computations at a given time. In a real-time environment, there are multiple tasks with different criticality levels. The tasks could be either soft real-time, hard-real-time, or firm real-time.

Assume a given set of tasks $T = \{t_1, t_2, t_3 \dots t_n\}$. Task $t_i$ is said to be a hard real-time task if the execution of $t_i$ must be completed by a given deadline $D_i$ and $W_i \leq D_i$ where $W_i$ is the worst-case execution time of $t_i$. Task $t_i$ is said to be a soft real-time task if the penalty it pays increases as the $r_i$ increases. In this scenario, $r_i$ is the time elapsed between the deadline of $t_i$ and the actual completion time. The penalty function $P(t_i) = 0$ if $W_i \leq D_i$ else $P(t_i) > 0$. Task $t_i$ is said to be a firm real-time task if an increase in reward depends on how early $t_i$ finishes its computation before the given deadline $D_i$. The Reward function $R(t_i) = 0$ if $W_i \geq D_i$ else $R(t_i) > 0$. In this research, two optimal Real-time scheduling algorithms which are Rate Monotonic Scheduling Algorithm (RMS) and the Earliest Deadline First algorithm (EDF) have been explored. These algorithms were selected because they are well-known baseline scheduling algorithms for real-time systems [145] and they also have a competitive time complexity.

#### 3.2.3.1 Rate Monotonic Scheduling Algorithm (RMS)

The RMS Algorithm is a priority-driven algorithm with priorities well known before the arrival of the task. These priorities are determined by the period of each task and are the same for all instances of the same task. RMS is the most widely used real-time algorithm [145]. Some assumptions have been considered including the following: (i) The tasks have no precedence constraints and all tasks are independent. (ii) It is assumed that only processing requirements are significant. (iii) It is assumed that the tasks have no non-preemptable section and the cost of preemption is negligible. (iv) It is also assumed that the tasks are periodic and that $t_i$ has a higher priority than $t_j \iff i > j$. The shorter the period, the higher the priority. If a lower priority task $t_j$ is running and a higher priority task $t_i$ is waiting to run, $t_i$ will preempt $t_j$. RMS assigns higher priority to tasks that uses the CPU more often. RMS is referred to as optimal real time algorithm because if a given set of processes cannot be scheduled by RMS, then it cannot be scheduled by any other algorithm that uses static priorities [138].

#### 3.2.3.2 Earliest Deadline First Algorithm (EDF)

The EDF algorithm is also a priority-driven algorithm that assigns priorities according to tasks deadline. EDF gives a higher priority to a task $t_i$ that has an earlier deadline $d_i$. Time $t_i$ will always preempt a task with a lower priority $t_j$ which have a higher deadline $d_j$. EDF uses a dynamic priority assignment. The priority of the tasks is assigned as the tasks arrive based on the tasks' deadline requirements. The priorities of other tasks may have to be adjusted to reflect the deadline of the newly runnable process. The following assumptions are made when using the EDF scheduling algorithm: (i) The tasks have no precedence constraints and all tasks are independent. (ii) It is assumed that only processing requirements are significant. (iii) It is assumed that the tasks have no non-preemptable section and the cost of preemption is negligible. The EDF algorithm has a worst-case runtime of $O((N+\propto)^2)$ where $\propto$ is the number of aperiodic tasks and $N$ is the total number of requests in each hyper-period of $n$ periodic tasks in the system [145]. EDF is referred to as an optimal uniprocessor real-time scheduling algorithm because it schedules tasks so that they meet their deadline requirement with 100% CPU utilization and if EDF cannot feasibly schedule a set of tasks on a uniprocessor then no other algorithm can. This is proved by using the time slice swapping technique [145].

## *3.3 Deadlock Aware Resource Provisioning in MEC using Bankers' Algorithm.*

The previous section has reviewed the current resource provisioning frameworks proposed for MEC platforms, none of which are deadlock aware. Furthermore, the various ways of handling deadlock have been reviewed in a step to address the problem and bridge the gap. This research focuses on real-time systems. Hence, real-time scheduling algorithms have also been reviewed. Therefore, in this section, a deadlock aware resource provisioning framework has been proposed. The proposed algorithm utilizes Banker's resource request deadlock avoidance algorithm.

Resource provisioning in MEC depicts a multiprogramming environment where several resources may compete for reusable resources. In the context of resource provision in MEC, several devices compete for limited reusable resources provided by the MEC platform. The idea is to schedule application tasks from mobile devices to MEC nodes for execution. Since there is a finite amount of resources in MEC, these must be managed effectively to prevent scheduling a task to an edge node that does not have adequate available resources to execute the offloaded task. This environment is usually prone to deadlock because a process may request resources that are held by another waiting process thereby leading to a circular wait [138]. Deadlock is an undesirable problem that has been studied extensively in operating systems [138], resource allocation systems [146], and manufacturing systems [147] [148]. MEC is a distributed system [21] and such systems are susceptible to deadlock. Therefore, it is crucial to employ adequate deadlock measures to ensure the reliability of the system [149].

Deadlock-free operation is a key characteristic for industrial sites that require high reliability and availability from their infrastructure to achieve the daily goal of the industry. The standard toolset for deadlock detection is WFG [138]. In the absence of algorithms to detect and recover systems from deadlocks, a situation may occur where the system is in a deadlock state, and yet there is no way of recognizing what has happened. In this section, a deadlock aware algorithm for scheduling resources for IoT devices onto a MEC platform which incorporates Banker's resource-request algorithm is presented. Bankers have been used as it guarantees deadlock-free operations as reviewed earlier.

Banker's algorithm works by simulating and using specified resources to predetermine deadlock conditions for all pending activities and deciding if allocation should be allowed to continue. Banker's algorithm requires three important inputs for execution which are the NEED matrix, MAX matrix, and available vector (AVAIL vector) [11]. Communication overhead is usually incurred while maintaining these required inputs. The proposed

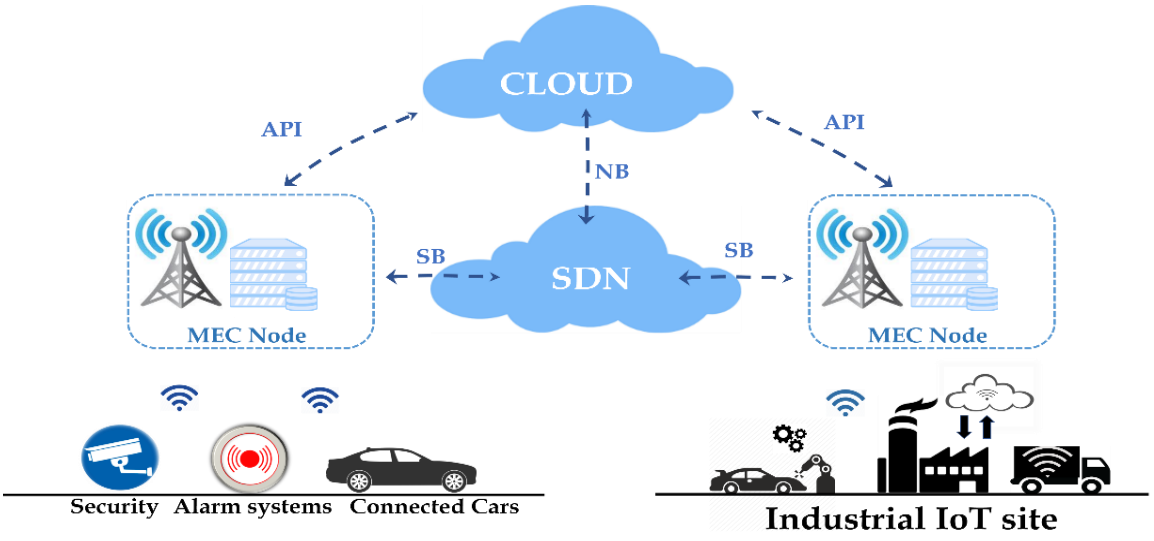

*Figure 13 MEC Case Study Architecture for Industrial IoT*

algorithm is more effective if implemented using Software Defined Networking (SDN) to reduce the impact of the communication overhead that would be generated by the resource-request algorithm. The idea of separating the control plane from the data plane in SDN [16] means there would be less communication between the routers and switches because they share a centralized control plane.

### 3.3.1 System Architecture

Figure 13 shows a high-level view of the MEC topology adopted in this study. In this scenario, due to the resource and computation limitation of the IoT devices, they heavily depend on MEC nodes to execute their workload. Therefore, tasks are offloaded from the IoT devices to be executed on a MEC platform. The distributed edge nodes communicate with each other through SDN. Requests that cannot be processed on the edge node would be forwarded to the cloud through the API. To reduce latency, traffic to the cloud is generally avoided. The SDN controller uses its North Bound Interface (NBI) to communicate with the cloud and communication with the edge nodes is done using the South Bound Interface (SBI). Each edge node comprises of a monitoring tool that calculates the resource utilization of the MEC node (CPU, RAM, and Storage). This information is shared between the edge nodes as metadata. Therefore, each MEC node that forms a part of the network is assumed to keep the resource utilization information of its neighbouring nodes. This helps the edge nodes decide the most suitable edge if re-offloading is required. This process has been explained further, later in this chapter.

Each edge node in the network sends updated metadata after each event. This metadata describes the resource utilization of the edge node after the event. The term network is used here loosely to describe the Multi-access edge architecture.

### 3.3.2 Deadlock in Distributed MEC

To describe the deadlock condition in distributed MEC, let's assume a set of processes $P = \{p_1, p_2 .. p_n\}$ and a set of resources $R = \{r_1, r_2 .., r_m\}$, where $n$ and $m$ are the number of processes and resources respectively. These resources and processes are present in the collaborative MEC space. However, they might not reside in the same MEC. Deadlock occurs if a process $p_i$ is waiting for a resource $r_{\boldsymbol{a}}$ that is currently held by another process $p_{\boldsymbol{j}}$. Additionally, $p_{\boldsymbol{j}}$ is waiting for a resource $r_{\boldsymbol{b}}$ that is currently held by $p_i$. If either $p_i$ nor $p_j$ can be preempted while in a waiting state, the system would be in a deadlock.

### 3.3.3 System Model

Let us consider a distributed architecture that consists of a pool of MEC nodes that is used as a platform for resource provisioning. The tasks seeking to be offloaded will utilize the MEC resources through a request-response mechanism. Hence the problem can be modelled as a

| System Model Parameters Section 3.4.3 | |
|---|---|
| **Notation** | **Meaning** |
| $P$ | Set of processes |
| $p_i$ | Process with index $i$ |
| $R$ | Set of resources |
| $r_i$ | Resource with index $i$ |
| $CL$ | Set of edge nodes |
| $Cl_i$ | Edge node |
| $M$ | Set of IoT devices |
| $ms$ | IoT device |
| $ch$ | Channels |
| $k$ | CSMA/CA back-off time out order |
| $\tau_{dl}$ | Real-time deadline |
| $W$ | Set of tasks |
| $T_i$ | Task to execute |
| $[c_i, m_i, n_i, d_i]$ | CPU, memory, network, and data size respectively |
| $\alpha_i$ | Fraction of $d_i$ to be executed |
| $l_i$ | Offloadable data size |
| $t_i$ | Transmission time |
| $r_i$ | Transmission rate |
| $P_i$ | Power |
| $g_i$ | Gain |
| $B$ | Bandwidth |
| $N_0$ | Noise density |
| $E_{i,off}$ | Offload Energy |
| $\tau_{local}$ | time spent to calculate if the task is offloaded |
| $\tau_{route}$ | Routing time |
| $\tau_{wait}$ | Task waiting time on the edge |
| $Size_r$ | Size of the ready queue |
| $MAX$ | Max amount of resource |
| $AVAIL$ | Amount of available resources |
| $N_{Total}$ | Total task submitted to the edge node |
| $t_{rms}$ | Time to schedule task by RMS |
| $N_{oc}$ | Number of tasks offloaded |
| $t_{ba}$ | Time for bankers to find safe seq. |
| $TT$ | Expected turnaround time |
| $N_{reoff}$ | number of tasks to be re-offloaded |

Directed Regular Graph. Due to the high volume of offloading traffic from the underlying scalable network, the target scenario stands out to be a soft real-time system. Let's consider a mesh network of a finite non-empty set of edge nodes $CL$ ={$Cl_1, Cl_2 \dots Cl_n$} and a finite non-empty set of IoT devices $M$ ={$ms_1, ms_2 \dots ms_n$} connected to the edge network such that $ms_i \in M$ and $Cl_j \in CL$ maintains a disjoint many-to-one cardinality. Here an edge node is connected to many IoT devices, but no IoT device is connected to multiple edge nodes. Communication between $CL$ and $M$ happens over a wireless band with a fixed number of channels

$\{ch_i | 1 \leq i \leq k\}$ and collision is managed by the CSMA/CA protocol [150, p. 12]. As the number of channels is fixed, the optimal multi-access mechanism would be CSMA/CA [151]. The CSMA/CA maintains a back-off time, less than the real-time deadline $\tau_{dl}$ making the system scalable and dynamic. The system model comprises the communication model and the computation model. The communication model deals with the optimization of communication parameters for better energy savings while the computation model deals with the optimization of the execution time with deadlock immunity.

#### 3.3.3.1 Communication Model

Let's consider a workload $W = \{T_1, T_2 \dots T_n\}$ which contain a set of tasks to be offloaded by a mobile station. Each task $T_i$ is characterized by $[c_i, m_i, dl_i, d_i]$ which denotes CPU, memory, deadline, and data size respectively. During the IoT application development, the developer specifies which part of the total workload can be offloaded (*Remotable Object*) and which part must be executed locally. $W$ is the partition marked to be offloaded. A similar method is used by Microsoft's MAUI [85] to partition .Net applications to be partially offloaded based on the device condition. The transmission time $t_i = \left(\frac{d_i}{r_i}\right)$ where $r_i$ is the transmission rate. Let's assume that the channel is Additive White Gaussian Noise [152] (AWGN) in nature. In an AWGN channel $C_i$ between 2 antennas with $R$ distance apart and transmission power $P_R$ *and* $P_T$ with gains $G_R$ and $G_T$ respectively. According to Friis transmission equation,

$$P_R = \frac{P_T G_T G_R c^2}{(4\pi R f)^2} \propto P_T G^2 \ (\because G_t = G_r = g) \tag{1}$$

where $f$ is the frequency and $c$ is the speed of light. Therefore, $r_i$ can also be expressed as the following using Shannon theorem for the same channel $C_i$ as

$$r_i = B \log_2\left(1 + \frac{P_i g_i^2}{N_0 B}\right) \tag{2}$$

Here, $P_i$ is the received power ($P_R$) measured in $mJ$, $B$ is the bandwidth, $g$ is the channel gain and $P$ is the transmission power. The equation can be rewritten for $P_i$ as equation (3)

$$P_i = \frac{N_0 B \left(2^{\frac{r_i}{B}} - 1\right)}{g_i^2} = \frac{1}{g_i^2} h\left(\frac{d_i}{t_i}\right) \tag{3}$$

where

$$h(x) = N_0 B \left(2^{\frac{x}{B}} - 1\right) \tag{4}$$

which is monotonically increasing with $x$. Hence the energy consumption for the offloading task is equation (5)

$$E_{i,off} = \frac{d_i P_i}{r_i} = t_i P_i = \frac{t_i}{g_i^2} h\left(\frac{d_i}{t_i}\right) \tag{5}$$

Therefore, $E_{i,off} = O(t_i)$ . Energy optimization can be obtained by the following model.

$$maximise\ E_{saved} = \sum_{i \in W} \left(E_{i,local} - E_{i,offload}\right) \tag{6}$$

subject to,

$$\tau_{i,local} + \sum_{i \in W} \tau_{i,route} + \sum_{i \in W} \tau_{i,wait} \leq \tau_{i,dl_i} \tag{7}$$

where $\tau_{i,local}$ is the time spent to calculate if the task is offloaded, $\tau_{i,route}$ is the time spent in routing the task from the local device to the edge for execution while $\tau_{i,wait}$ includes all other time delays the task waits before being executed. $\tau_{i,dl_i}$ is the task deadline. The edge nodes are assumed to be in a mesh topology therefore, the maximum hop distance is 1. Hence, $\tau_{i,route}$ can be assumed as $O(1)$. The $\tau_{i,wait}$ includes CSMA/CA binary exponential back-off, encoding delay, multiplexing delay, propagation delay, service etc. However, the binary backoff time is exponential while the rest is polynomial. Therefore, we assume the overall wait time $\tau_{i,wait} = O(n^k)$ . As deadlock freezes the system, the waiting time keeps increasing by $2^k$

until it reaches the maximum *k* value and times out due to CSMA/CA *binary exponential back-off* characteristics [150].

#### 3.3.3.2 Computation Model

Computation starts after the offloaded data stream is received by a MEC node. Here a decision is made whether the requested task either gets executed on the subjected edge node or re-offloaded to another one. The decision is made based on the resource request WFG of each edge node and the availability of the other nodes in the mesh. Hence the system is a mesh of interconnected priority queues. Note that the WFG is made for each MEC node and not distributed across all nodes. The priority is based on a safe sequence from the Banker's algorithm which guarantees no deadlock using a preventive and avoidance measure. The precomputing delay contributes to $\tau_{i,wait}$ and ensures it is below the deadline.

The edge node maintains two queues. First, a prioritized job queue whose priority is maintained by the real-time scheduler (Rate Monotonic Scheduler (RMS)). To achieve real-time criteria, RMS suggests that frequently occurring tasks should be given higher priority [153]. Tasks get popped out in the job queue in FIFO order and then checked if the requested resource can be accommodated by the subjected edge node $Cl_s$. If not, it finds another edge node $Cl_d$ that is most eligible and offloads. If $Cl_d$ executes the task on time, then $Cl_s$ increases the $d^{th}$ index on its $Affinity_s$ vector that it maintains, otherwise it decreases. This affinity vector is initialized with 0 and used to maintain reliability record and tie-breaker purpose. A *Request* ≤ *Available* is said to be valid and put into the Ready Queue which is finite with size $Size_r$ and prioritized with Banker's generated safe sequence.

$$Size_r = \left\lceil \frac{BDP}{mean(l)} \right\rceil = \left\lceil \frac{nB\ RTT}{2\sum_{i=1}^{n} d_i} \right\rceil \tag{8}$$

BDP shows the number of bits the channel can accommodate, hence the ratio of BDP and the average task is the number of tasks that can be queued ensuring mutual exclusion property.

When a task is inserted into a ready queue, it gets an index based on its resource requirement. Starvation is handled with ageing. If a task $T_i$ gets placed into a ready queue with index $v$, then the expected turnaround time $TT(T_i) = v * awt_s$ where $awt_s$ is the average waiting time of the edge node $Cl_s$.

In the worst-case scenario, for $n$ processes and $m$ resources *Banker's algorithm* takes $O(n^2m)$ time. Since the number of resources is fixed ($K$), which are the CPU, memory, and data size. Hence the time complexity is $O(n^2K) = O(n^2)$. Since the algorithm is applied to the ready queue, the maximum task it can retain is $size_r$. Hence, the Banker's algorithm takes $O(size_r^2)$ to generate a safe sequence.

**Lemma 1:** *The consumed energy for offloading and the transmission time, shares a linear relationship.*

*Proof.* From equation 3 & 4 it can be inferred that, the partial relationship between $E_{i,off}$ & $t_i$ for a given gain ($g_i$) and offload size ($d_i$) is,

$$E_{i,off} = \frac{t_i}{g_i^2} h\left(\frac{d_i}{t_i}\right) = t_i 2^{\frac{1}{t_i}} \tag{9}$$

Using asymptotic analysis of the given function,

$$O(E_{i,off}) = O(t_i) \times O\left(2^{\frac{1}{t_i}}\right) \tag{10}$$

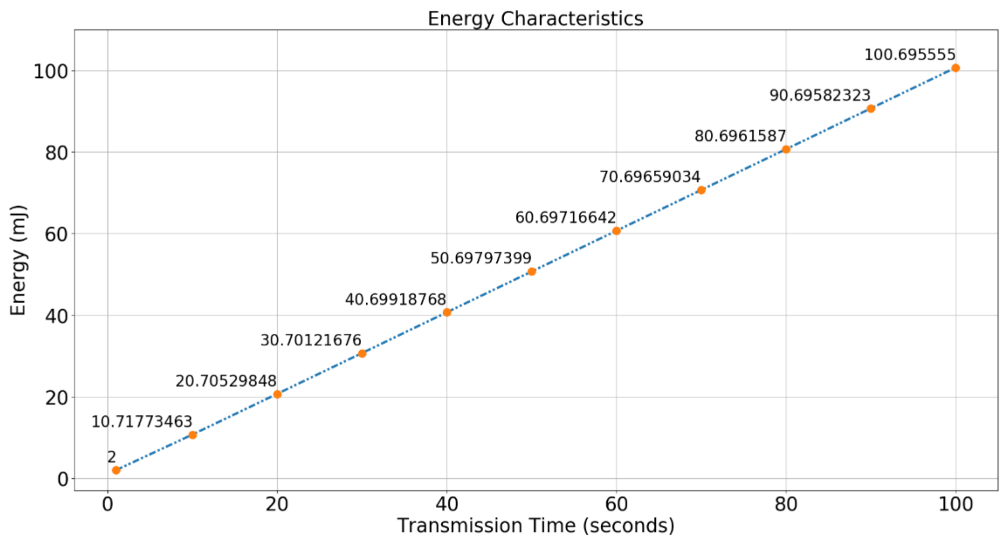

*Figure 14 Energy Characteristics Obtained from equ.8 using the following $t_i$ range [1,10,20,30,40,50,60,70,80,90,100]*

Now the second element is a monotonically decreasing sequence with a lower bound $0$. Hence, it has a constant asymptotic upper bound $c \in R$, therefore $O(1)$.

Hence,

$$O(E_{i,off}) = O(t_i) \times O(1) = O(T_i) \tag{11}$$

This can be verified by plotting equation 8. (Figure 14)

### 3.3.4 Proposed Resource Provisioning Algorithm (RPA)

In this section, the design and analysis of the proposed resource provisioning algorithm (RPA) have been discussed. The algorithm fetches tasks from the task queue, which is RMS scheduled, therefore most frequently used tasks get higher priority. Tasks from the Job queue then migrates to the ready queue. The proposed algorithm alters the order in which the tasks leave the job queue and stays in the ready queue. The following are the criteria used for the ordering.

**Algorithm 1: Resource Provisioning Algorithm (RPA)**

```
Input: $W\ [c_i, m_i, dl_i, d_i]$
Output: $Resource\ Provision\ Plan\ for\ t_i$
Steps
    1.  Do
    2.  $Job.insert(t_i)$
    3.  $k \leftarrow 0\ ; max_k = input("Max\ retry\ attempt")$
    4.  While $(Ready.isfree(\quad) = true)$ do
    5.      $Ready.insert(Job.delete(t_i))$
    6.      $J_{cur} \leftarrow Ready.delete(t_i)$
    7.      $J_{cur}.status = Assigned$
    8.      if $J_{cur}.MAX < node.AVAIL$:
    9.          $ind = bankers(J_{cur})$
    10.         $Time = (AWT) * (ind)$
    11.         if $Time < dl_i$:
    12.             $Assign$
    13.         Else
    14.             $goto\ step$ 17
    15.         End if
    16.     Else
    17.         $Find\ Cl_d\ from\ CL[nodes]$:
    18.         $Max(Cl_d.AVAIL - J_{cur}.MAX)$
    19.         $send(J_{cur})$
    20.         $Wait\ until\ response$
    21.         if $response = success$:
    22.             Return result
    23.         Else $wait(2^{k++})\ where\ k\ is\ iteration\ count$
```

```
24.             $if\ (Timeout\ or\ k = max_k)$:
25.                 $\mathbf{Return}\ Fail$
26.             $\mathbf{End\ if}$
27.         $\mathbf{End\ if}$
28.     $\mathbf{End\ if}$
29. $\mathbf{End\ Loop}$
```

- **Case 1 – Over Provisioning**: Each task comes with its maximum resource need, recorded in the MAX vector. If the maximum need exceeds the total available resources, then it searches for a MEC node that satisfies the constraint. If no such MEC node is found the task waits for a certain amount of time which increases in a binary exponential order with each iteration of the request before it times out.
- **Case 2 - Safe Request**: if the MAX is less than the current node's AVAIL then the task enters Banker's safe state algorithm and be given a safe sequence index at which the task gets executed. Banker's algorithm guarantees a safe sequence that never causes deadlock.
- **Case 3 - Time feasibility**: A resource-intensive task in a resource constraint MEC may suffer from starvation by waiting. Ageing is used here to improve waiting time, although it requires the process to stay waiting to age. Hence the algorithm calculates waiting time by the product of the average waiting time of the current node and the index of the task. If the waiting time exceeds the soft deadline of the task, it finds an alternative node to meet the criteria.

A task is said to be feasible if it doesn't overdemand and the generated waiting time is less than its latency constraint. The waiting time of a task is the product of the average waiting time of the executing node $Cl_i$ and the safe index Bankers' algorithm produces. The algorithm allows a feasible task to execute locally else it gets executed remotely. A task that demands resources that are not available on the local MEC or a task with unsuccessful execution by a remote MEC must be kept on waiting until timeout. The waiting period increases with a binary exponential order with each attempt. A registry is also maintained to keep track of the tasks submitted for remote execution and their status. Figure 15 depicts the complete workflow of RPA.

**Lemma 2:** *RPA is not suitable for hard real-time but soft real-time tasks.*

The response time of the algorithm depends on various timing factors such as

i. *Queuing Delay:* Takes place due to processing overhead, context switching, etc. of other processes rather than the subjected one. It also depends on the system state and load.

ii. *Transmission Delay:* The total time taken for an offloaded task to return to the UE includes the transmission delay which varies with the size of the offloaded task and the available bandwidth.

The given uncertainty conditions make a hard deadline infeasible as opposed to a soft deadline (lemma 3), hence the statement.

**Lemma 3:** *If there exists a feasible MEC node for a task, RPA handles the task within a finite time.*

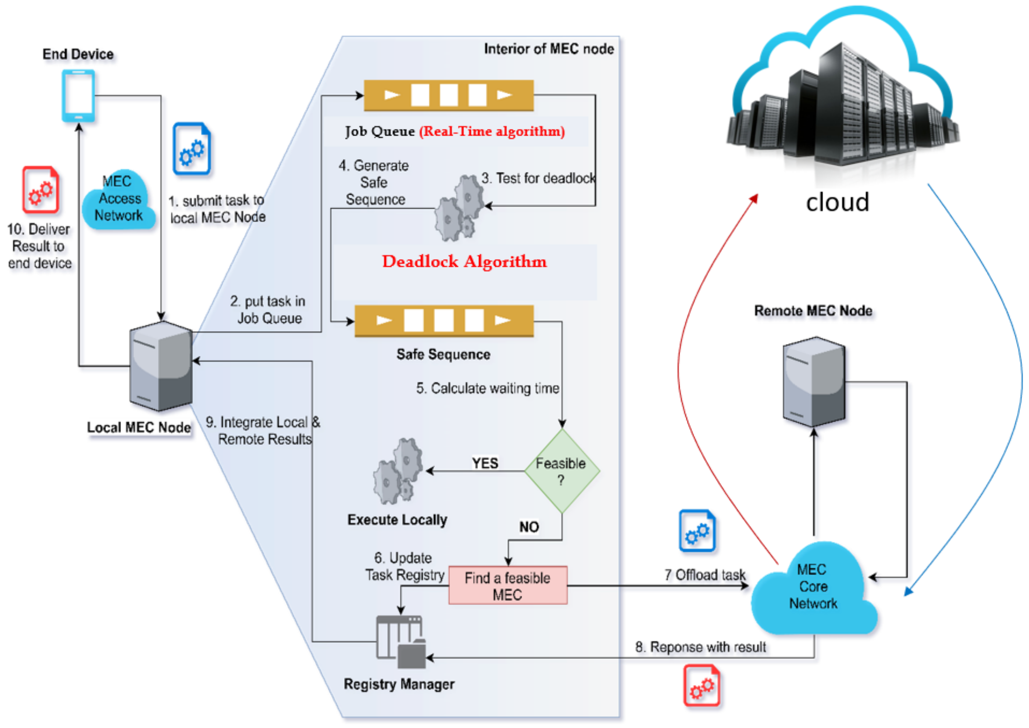

*Figure 15 RPA Workflow: Algorithm Workflow Structure that has been adapted for modelling the task flow from the end device to the MEC node for processing*

To prove the lemma, each of the three feasibility cases discussed earlier as a task waits a finite amount of time under RPA are studied.

- **Case 1:** If the task over demands-resources to its original MEC node and a remote node failed to execute, it must wait twice the time for resubmission hence the timeout occurs in $\log_2 timeout$ iteration.
- **Case 2**: if the task makes an unsafe request, it looks for a remote node to get offloaded. Since all the $AVAIL$ information is reactively shared and the decision is made based on the global map of $AVAIL$s. Therefore, the task gets offloaded only once and onto the optimal remote MEC node. This prevents node hopping and total execution time can be $T_{total} = T_l + T_r + 2C_{lr}$ where $T_l, T_r$ & $C_{ij}$ are local execution, remote execution, and transmission time respectively.
- **Case 3**: If a task makes a safe request but has a large *NEED*, it must wait for the resources to be available. If a remote node can execute it in less time, it is offloaded $(T_l < T_r)$. Therefore, this guarantees the optimal remote node selection.

### 3.3.5 Simulation

Simulations have been carried out to demonstrate the validity of the proposed technique. The simulations are based on the complexity analysis of the algorithm and energy optimization as discussed in the previous section. The energy, $E_{i,off}$ is required by an edge node $Cl_s$ to offload a task of $d_i$ $size$ for $t_i$ unit time through a channel link of $B_i$ bandwidth using an antenna of $g_i$ and noise density $N_0$ is equation (12).

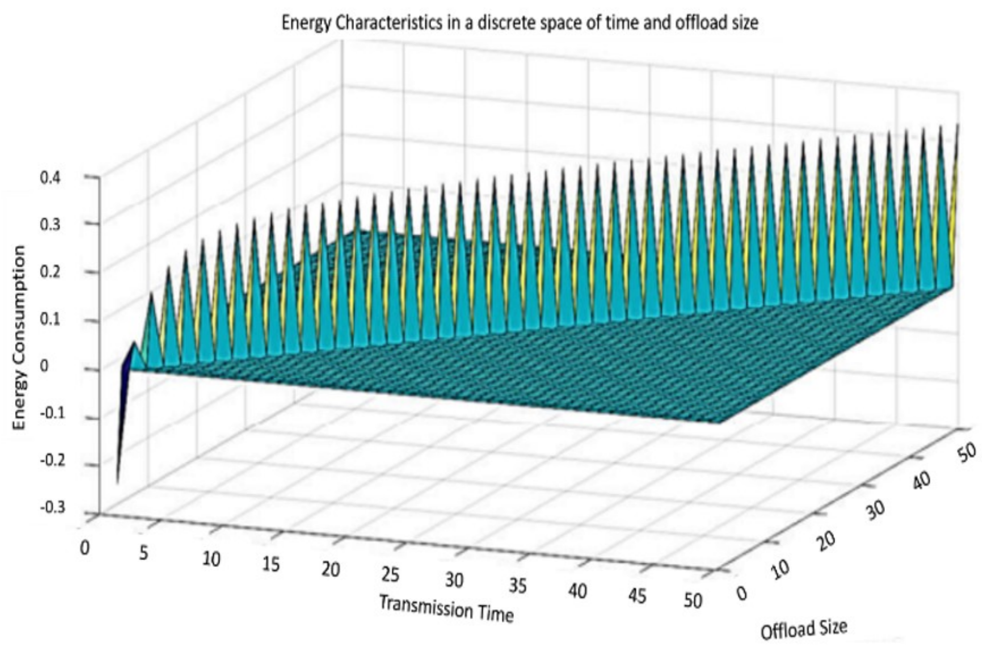

*Figure 16 Offload Energy Characteristics using equ.13 with varying range of transmission time and offload size [0-50]*

$$E_{i,off} = \frac{t_i}{g_i^2} N_0 B \left( 2^{\frac{d_i}{Bt_i}} - 1 \right) \tag{12}$$

Since gain, bandwidth, data size, and noise density are predetermined by the communication system, hence the relation can be reformulated into an asymptotic upper bound form as equation (13).

$$E_{i,0ff} = O\left( t_i^l . 2^{\frac{d_i}{t_i^l}} \right) \tag{13}$$

The graph in Figure 16 shows a critical value of transmission time and payload length as the energy consumption of the antenna rises exponentially. The context suggests that if there's a deadlock then the waiting time component will increase indefinitely resulting in significantly large energy consumption. Since the transmission time is a function of the data length and a constant data rate, therefore the transmission time is a random variable distributed over a Bernoulli's PDF. To find the expectation (E) this can be shown that the surface integral in equation (14) cannot be expressed in a closed-form.

$$E = \int_0^{\frac{BDP}{n}} \int_0^{\tau_{i,dl_i}} t_i 2^{\frac{d_i}{t_i}} dt_i dx_i. \tag{14}$$

Equation (14) states the growth rate of $E_{i,off}$ where $x_i = d_i$. Plotting this growth characteristic within a close range of [0, 50], the response characteristic surface in Figure 16 is obtained. Each spike on the graph depicts the exponential growth of energy discussed earlier. With an increase in transmission time and length, the peak energy consumption grows at a constant rate of $ln\ 2$. The growth characteristics of the $E_{i,off}$ in equation (13) can also be shown by the partial derivatives with respect to transmission time $(t_i)$ and offload length$(l_i)$

$$\frac{\partial^2 E_{i,off}}{\partial l_i \partial t_i} = -\left(\ln 2 \frac{2^{\frac{d_i}{t_i}}}{t_i}\right). \tag{15}$$

The surface plot of the equation (15) is depicted in Figure 18.

Analytically, equation (15) signifies the rate of change of energy consumption with respect to varying offload length and latency. The value of $(t_i, d_i)$ is taken in a close range. It can be shown in Table 4 that the growth gets steeper as the range increases. It can be observed that the plot saturates after the range [1,40].

*Table 4 Energy Growth in discrete-time and size from equ 14*

| $(t_i, d_i)$ range | Rate of change of $E_{i,off}$ |
|---|---|
| [1,10) | $\mathbf{3.5 \times 10^2}$ |
| [1,20) | $\mathbf{3.5 \times 10^5}$ |
| [1,30) | $\mathbf{3.5 \times 10^8}$ |
| [1,40] | $\mathbf{3.5 \times 10^9}$ |
| [1,100] | $\mathbf{3.5 \times 10^9}$ |

#### 3.3.5.1 Simulation Results

Since there exist no related experiments with MEC context in the literature, experiments have been performed by comparing the results of a system with and without using RPA.

Figure 17 shows time comparison graphs between a system with no deadlock prevention measures and a system running the proposed algorithm. The graphs have been plotted with their corresponding time complexities for $n$ number of tasks subject to a constant $k$ (timeout order: this value is application dependent). It can be seen in Figure 17 that as $k$ increases, the time consumption of the system with no deadlock measures surpasses the system running the proposed algorithm. Since time is directly proportional to energy, it can be deduced that the algorithm optimizes the energy of a system by eliminating deadlock.

### 3.3.6 Complexity Analysis

If $N_{Total}$ tasks are submitted to an edge node, the job queue will hold them in priority as

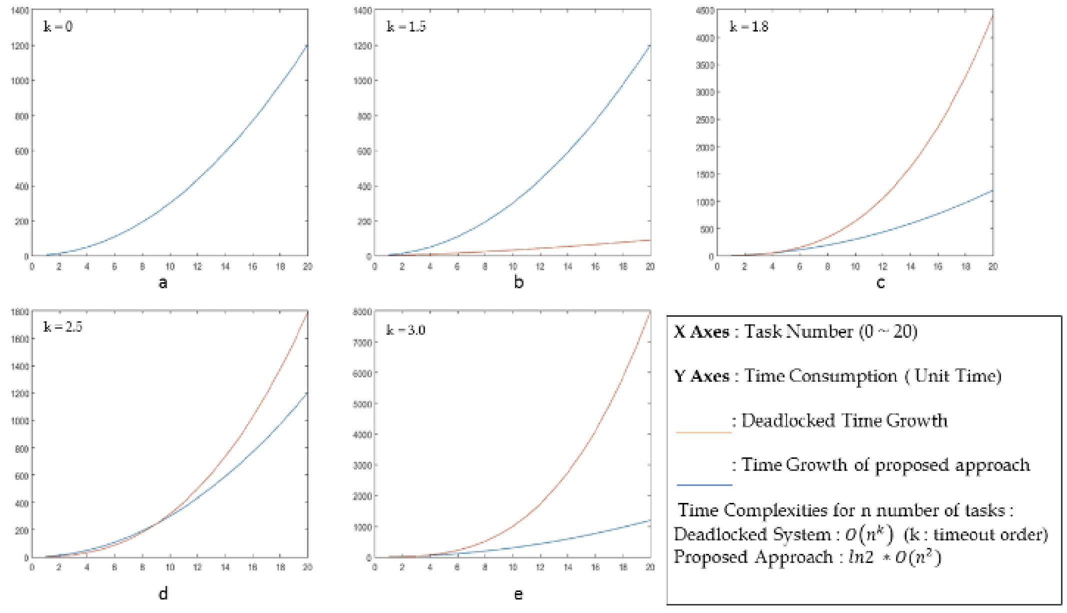

*Figure 17 Comparison of Time Consumption of System With & Without Using RPA*

generated by the RMS algorithm which takes $t_{rms}$ time. Based on tasks' request and the

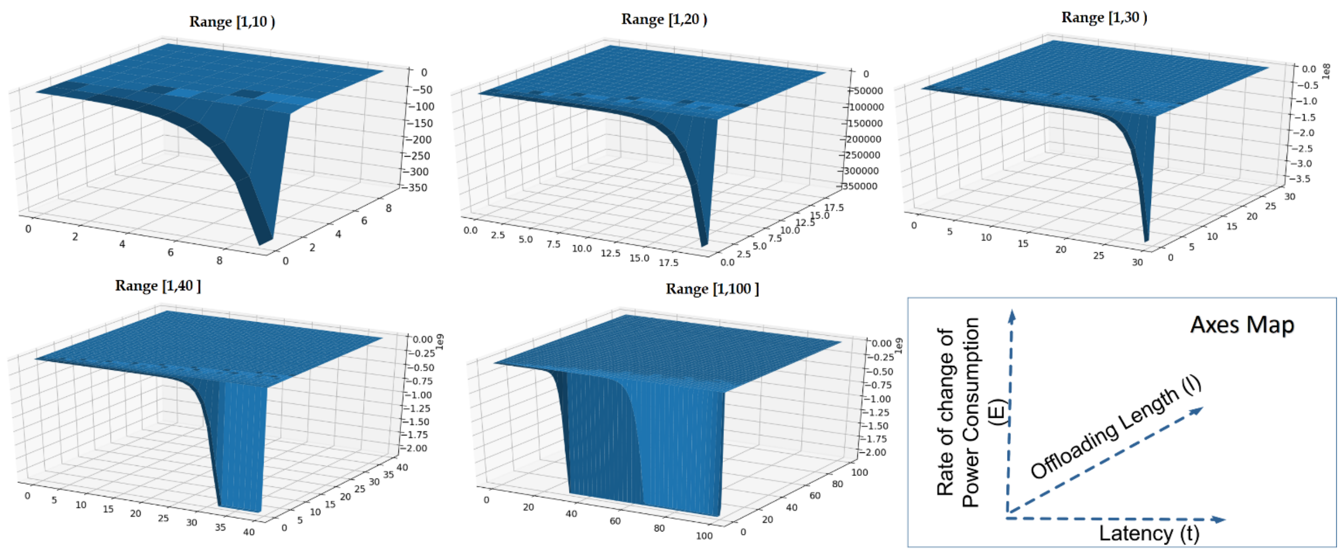

*FIGURE 18 Growth Characteristics of The Rate Of Change In Offload Energy With Varying Latency And Offload Length In A Close Range using equ14. Slope Gets Steeper With Increasing Range, Saturates At [1,40]*

subjected edge node's availability or resources, $N_{oc}$ tasks are offloaded to an eligible edge node $Cl_j$ as an overcommitted task. An efficient binary search implementation can find such $Cl_j$ in $\log_2 c - 1$ time. The remaining $N_{Total} - N_{oc}$ tasks will be put into Banker's algorithm that takes $t_{ba}$ time to find the safe sequence in a worst-case scenario. If a task $T_i$ gets a safe index *v*, and the $TT(T_i) > Q(dl_i)$ then, the task will be offloaded to another MEC node that can perform the execution within the deadline. The function queue calculates the probability of executing the task and maintaining the deadline after all the communication and queuing. This is done by maintaining the affinity matrix. Hence, the time complexity of $Q$ is $N_{reoff} \log_2 N_{reoff}$, where $N_{reoff}$ is the number of tasks to be re-offloaded. Therefore, the maximum time a task can take to be executed if it got offloaded twice and being the lengthiest task can be expressed as

$$T_{max} = 2[ln2 + \log_2(c-1) + 3(N_{Total} - N_{oc})^2 + N_{reoff} \log_2 N_{reoff} + rtt] + t_{exec} \quad (16)$$

The worst-case complexity of RMS and Bankers algorithm can be deduced to $ln2$ and $3n^2$ respectively.

*TABLE 5 ALGORITHMS USED DURING EXPERIMENT*

| | |
|---|---|
| **Deadlock Prevention Schemes** | Wait-die Algorithm |
| | Wound Wait algorithm |
| **Deadlock Avoidance Scheme** | Bankers resource request algorithm |
| **Realtime scheduling Schemes** | Rate monotonic scheduling algorithm |
| | Earliest Deadline First Algorithm |

## *3.4 Comparative Analysis for Deadlock Avoidance and Prevention for MEC*

In this section, a comparative study is done on deadlock avoidance and deadlock prevention algorithms for MEC environments. Here 6 case study algorithms have been considered based on the framework proposed in Figure 15 [100]. Each compared algorithm is composed of a deadlock algorithm and a real-time scheduling algorithm. The algorithms used for this design can be seen in Table 5.

Six compared algorithms have been considered as shown in Table 6. The algorithm workflow for each of the six algorithms is the same structure as is in Figure 15. Tasks are sent from the end device to the local edge node for resource provisioning. In the MEC node, tasks are put into a job queue and the queue is prioritized using a real-time scheduling algorithm. Then a deadlock algorithm is employed to reduce or eradicate the chances of deadlock. Thereafter, waiting time is calculated for each task received and an assumed finishing time $P_{ti}$ for each task $t_i$ is estimated. If $P_{ti} < D_{ti}$ where $D_{ti}$ is the deadline for task $t_i$, then a MEC $M_i$ is identified that meets the deadline requirement using a cooperative decision algorithm. This algorithm is detailed in section 3.4.5.7. If such MEC is not found, then the tasks are sent to the central cloud to be executed and the MEC node acts as a proxy. For each of the compared algorithm, this structure remains the same but appropriate real-time algorithm and deadlock algorithm is utilized. Further details about the algorithm structure are detailed in the previous section [100].

*TABLE 6 COMPARED ALGORITHMS*

| Alias | Compared Algorithms |
|---|---|
| $ALG_1$ | RMS and Banker algorithm |
| $ALG_2$ | EDF and Banker algorithm |
| $ALG_3$ | RMS and Wound Wait |
| $ALG_4$ | RMS and Wait die |
| $ALG_5$ | EDF and Wound Wait |
| $ALG_6$ | EDF and Wait Die |

### 3.4.1 System Model for Comparative Analysis

In this section, a distributed architecture has been considered that consists of a pool of MEC nodes. A finite non-empty set of MEC servers in the same cluster is denoted as $\mathcal{M} = \{M_1, M_2 .., M_n\}$. Let's assume that a finite non-empty set of end devices $\mathcal{U} = \{u_1, u_2 .., u_n\}$ are connected to the edge network such that $u_i \in \mathcal{U}$ *and* $M_j \in \mathcal{M}$ maintains a disjoint many-to-one cardinality. Hence, an edge server is connected to many end devices, but no end device is connected to multiple edge servers. Each $u_i$ has a workload $W_{u_i} = [T_1^{u_i}, T_2^{u_i}, .. T_n^{u_i}]$ which contains an array of tasks to be executed. For each $T_j^{u_i}$ in $W_{u_i}$ the $u_i$ computes an offloading decision $a_j \in \{0,1\}$, where $a_j = 0, a_j = 1$ represents "execute locally" and "offload", respectively. It is assumed that the end device makes an offloading decision based on its battery life and computational resources. Let's assume that each $u_i$ is connected to the closest $M_j$ and hence offloads all $T_j^{u_i} \in W_{u_i}\ s.t\ a_j = 1$. For each $T_j^{u_i}$ that is offloaded the $u_i$ also sends a requirement vector $REQ = \{c_i m_i, l_i, s_i\}$ that is characterized by CPU cycles, memory, maximum latency, and data size respectively.

### 3.4.2 Computational Model for Comparative Analysis

Let's denote the computation capacity of each MEC $M_j$ in $\mathcal{M}$ as $\mathcal{S}_{M_j}$. This is the CPU frequency.

Let's assume that each $M_j$ maintains a queue $Q_{M_j} = [T_1^u, T_2^u \dots T_n^u]$ of tasks offloaded to $M_j$. The execution time of a task $T_i^u$ offloaded to $M_j$ *is*

$$f_i = \frac{c_i}{\mathcal{S}_{M_j}} \tag{17}$$

The waiting time for a newly added task $T_{n+1}^u$ *is*

$$WT_n = \sum_{i=1}^{n} f_i \tag{18}$$

Therefore, the total processing delay $\mathcal{K}_i$ for $T_i^u$ is

$$\mathcal{K}_i = WT_n + f_i \tag{19}$$

TABLE 7 MEANING OF PARAMETERS FOR COMPARATIVE ANALYSIS SYSTEM MODEL

| Notation | Meaning |
|---|---|
| $\mathcal{M}$ | Denotes a cluster of MEC nodes |
| $M_j$ | Denotes a MEC node |
| $\mathcal{U}$ | Set of end devices |
| $u_i$ | Denotes end device |
| $W_{u_i}$ | The total workload for an end device $u_i$ |
| $T_j^{u_i}$ | The task for an end device $u_i$ |
| $REQ$ | Requirement vector |
| $\mathcal{S}_{M_j}$ | CPU frequency |
| $f_i$ | The execution time of the task $T_i^u$ |
| $WT$ | Waiting time |
| $\mathcal{K}_i$ | Processing delay |
| $\mathcal{H}_i$ | Communication cost |
| $tr_{M_j}$ | Transmission rate of $M_j$ |
| $RES$ | Required resource type |
| $\lvert RT \rvert$ | number of resource-type |
| $P$ | set of processes |
| $MAX$ | max resource for each $RES$ |
| $\tau_i$ | The time period for EDF |
| $D_i$ | Deadline |
| $E$ | Constraints for EDF |
| $W_w$ | Constraints for wound-wait |
| $W_d$ | Constraints for wait-die |
| $BA(x)$ | Bankers algorithm function |
| $TC$ | Total Time cost $(\mathcal{K}_i + \mathcal{H}_i)$ |
| $M_j^s$ | Edge status vector |
| $Cl_{M_j}^{status}$ | Set of all $M_j^s$ in the edge cluster |

### 3.4.3 Communication Model for Comparative Analysis

For a task $T_i^u$ offloaded to an edge node $M_j$, the communication cost $\mathcal{H}_i$ of offloading the task can be expressed as the following,

$$\mathcal{H}_i = \tau_{i,local} + \tau_{i,route} + \tau_{i,wait} \tag{20}$$

where $\tau_{i,local}$ is the time spent to calculate if the task is offloaded, $\tau_{i,route}$ is the time spent in routing the task from the local device to the edge for execution while $\tau_{i,wait}$ includes CSMA/CA binary exponential back-off, encoding delay, multiplexing delay, transmission delay, propagation delay, service delay etc. This has been explained in more detail in section 3.3.3.1.

### *3.4.4* Modelling of Algorithms in Table 5

While using any of the algorithms in Table 6, an identification $id$ is required to uniquely identify each process. To define the requirements for a set of processes $P$, for the compared algorithms, the variable constraints for the algorithm components are first defined.

#### 3.4.4.1 Bankers Algorithm

In using the Banker's algorithm, for each process sent to the MEC for resource provisioning, two vectors are required. The resource type required $RES$ by the process and the maximum resource for each resource type $MAX$. Therefore, for each process, the resource type constraint can be expressed as

$$RES = \{\, res_i \in \{0,1\} |\ i\ \in \{0 \dots (|RT| - 1)\,\}\,\} \tag{21}$$

where, $|RT|$ is the number of resource-type. When $r_i = 0$, the resource type at the $i^{th}$ position is not required otherwise $r_i = 1$ specifies the resource type is required. It is assumed there are three resource types (CPU, Memory, and Storage) that can be claimed by each process. Likewise, the maximum resource for each resource type is defined as

$$MAX = \{max_i \mid \forall\, i \in RES_r\} \tag{22}$$

Where,

$$RES_r = \{res_i \in RES \mid res_i = 1\} \subseteq RES \tag{23}$$

### 3.4.4.2 RMS

While using the RMS algorithm, for each process that is sent to the MEC, the computation time or capacity $C_i$ and the time period $\tau_i$ will be required.

$$\tau_i = \left\{ \frac{1}{f_i} \middle| \forall\, i \in \{1 \dots |P|\} \right\} \tag{24}$$

Therefore, variable constraints needed for RMS

$$REQ_i = \{ [C_i, \tau_i] \mid \forall\, i \in \{1 \dots |P|\} \} \tag{25}$$

### 3.4.4.3 EDF

While using the EDF algorithm, for each process sent to the MEC for resource provisioning the computation time or capacity $C_i$, time period $\tau_i$ and the deadline $D_i$ will be required. Therefore, the variable constraints needed for EDF are considered

$$E = \{ [C_i, \tau_i, D_i] \mid \forall\, i \in \{1 \dots |P| \} \} \tag{26}$$

### 3.4.4.4 Wound-wait /Wait die

For each process sent to the MEC, while using the wound-wait or wait-die, the resource type required $RES$ and the time stamps $Ts$ for each process are required. Here, $RES$ is obtained the same way as in *(21)* and $Ts = \{t_i \mid i \,\forall \{1 \dots |P|\} \}$

Therefore, the variable constraints for Wound wait $W_w$ and Wait-die $W_d$

$$W_w = W_d = \{RES_i, Ts_i \mid i \,\forall \{1 \dots |P|\} \} \tag{27}$$

### 3.4.5 Deadlock Constraint of Algorithms

#### 3.4.5.1 Rate Monotonic Scheduling and Banker algorithm

In this algorithm for a set of processes $P$, combining *(21)* and *(23)*

$$P = \{P_i \mid \forall P_i \in (id, RES, MAX, REQ)\} \quad (28)$$

Let $RMS(x)$ and $BA(x)$ be functions of RMS and Banker's algorithm respectively. Then,

$$RMS(P) \rightarrow P_{rt} \subseteq P \quad (29)$$

Where, $P_{rt}$ is a set of tasks that can be executed in real-time. Then putting $P_{rt}$ in the Banker's function,

$$BA(P_{rt}) \rightarrow P_{safe} \quad (30)$$

Where, $P_{safe}$ is the deadlock-free safe sequence.

#### 3.4.5.2 Earliest Deadline First and Banker algorithm

In this algorithm for a set of processes $P$, combining *(21)*, *(23)* and *(26)*

$$P = \{P_i \mid \forall P_i \in (id, RES, MAX, E)\} \quad (31)$$

Let $EDF(x)$ and $BA(x)$ be a function of the EDF algorithm and Banker's algorithm respectively. Then,

$$EDF(P) \rightarrow P_{rt} \subseteq P \quad (32)$$

Where, $P_{rt}$ is a set of tasks that can be executed in real-time.

$$BA(P_{rt}) \rightarrow P_{safe} \quad (33)$$

Where, $P_{safe}$ is the deadlock-free safe sequence.

#### 3.4.5.3 Rate Monotonic Scheduling and Wound Wait

In this algorithm for a set of processes $P$, combining equations (27) and (30)

$$P = \{P_i \mid \forall P_i \in (id, W_w, REQ)\} \tag{34}$$

Let $RMS(x)$ and $W_W(x)$ be a function of the RMS algorithm and Wound wait algorithm respectively. Then,

$$RMS(P) \rightarrow P_{rt} \subseteq P \tag{35}$$

Where, $P_{rt}$ is a set of tasks that can be executed in real-time.

$$W_w(P_{rt}) \rightarrow P_{safe} \tag{36}$$

Where, $P_{safe}$ is the deadlock-free safe sequence.

#### 3.4.5.4 Rate Monotonic Scheduling and Wait-die

In this algorithm for a set of processes $P$, combining equations (27) and (30)

$$P = \{P_i \mid \forall P_i \in (id, W_d, REQ)\} \tag{37}$$

Let $RMS(x)$ and $W_d(x)$ be a function of the RMS algorithm and Wound die algorithm respectively. Then,

$$RMS(P) \rightarrow P_{rt} \subseteq P \tag{38}$$

Where, $P_{rt}$ is a set of tasks that can be executed in real-time.

$$W_d(P_{rt}) \rightarrow P_{safe} \tag{39}$$

Where, $P_{safe}$ is the deadlock-free safe sequence.

#### 3.4.5.5 Earliest Deadline First and Wound Wait

In this algorithm for a set of processes P, combining equations (27) and (33)

$$P = \{P_i \mid \forall P_i \in (id, W_w, E)\} \tag{40}$$

Let $f: EDF(x)$ and $f: W_w(x)$ be a function of the EDF algorithm and Wound wait algorithm respectively. Then,

$$EDF(P) \rightarrow P_{rt} \subseteq P \tag{41}$$

Where, $P_{rt}$ is a set of tasks that can be executed in real-time.

$$W_w(P_{rt}) \rightarrow P_{safe} \tag{42}$$

Where, $P_{safe}$ is the deadlock-free safe sequence.

#### 3.4.5.6 Earliest deadline First and Wait Die

In this algorithm for a set of processes P, combining equations (27) and (33)

$$P = \{P_i \mid \forall\, P_i \in (id, W_d, E)\} \tag{43}$$

Let $f: EDF(x)$ and $f: W_d(x)$ be a function of the EDF algorithm and Wait-die algorithm respectively. Then,

$$EDF(P) \rightarrow P_{rt} \subseteq P \tag{44}$$

Where, $P_{rt}$ is a set of tasks that can be executed in real-time.

$$W_d(P_{rt}) \rightarrow P_{safe} \tag{45}$$

Where, $P_{safe}$ is the deadlock-free safe sequence.

#### 3.4.5.7 Cooperative offloading decision

In this sub-section, the cooperative offloading decision making for the proposed algorithm has been presented. In the proposed algorithm, the offloading decision is made by considering the time constraint of a task $T_i^u$ and the deadlock constraint of the MEC resource. For the time constraint, the following must be satisfied for execution,

$$TC < l_i \tag{46}$$

Where $TC$ is the time cost and is a summation of the computational cost and communication cost.

$$\mathcal{K}_i + \mathcal{H}_i < l_i \tag{47}$$

Substituting (*19*) and (20) in (47)

$$WT_n + f_i + \tau_{i,local} + \tau_{i,route} + \tau_{i,wait} < l_i \tag{48}$$

The deadlock constraint is determined using one of the algorithm models in the previous section. For simplicity let's assume that the constraint is determined by the $BA(x)$ in *(30)*. $BA(x)$ returns a safe sequence or false if the system is not in a safe state. $M_j$ makes an offloading decision $a_i$ for each newly added task $T_i^u$. The range of decision $a_i \in \{0,1,2\}$, where $a_i = 0$, means execute locally, $a_i = 1$, means send to another edge node and $a_i = 2$ means offload to the central cloud.

$$TC < l_i \; and \; BA(x) \rightarrow Safe \tag{49}$$

If (*49*) is $True$, then $a_i = 0$. Otherwise, an MEC that fits the description is sort after. If such MEC exists, then $T_i^u$ is offloaded to it else it is offloaded to the cloud.

To ensure that an edge node $M_j$ can calculate (*49*) for another edge node $M_k$, each $M_j$ multicasts its status $M_j^s$ to the MEC cluster after each update to $WT_n$. To reduce the communication overhead, the number of MECs in a cluster is minimized.

$$M_j^s = \left\{\mathcal{S}_{M_j}, WT_n, Mem_{M_j},\right\} \tag{50}$$

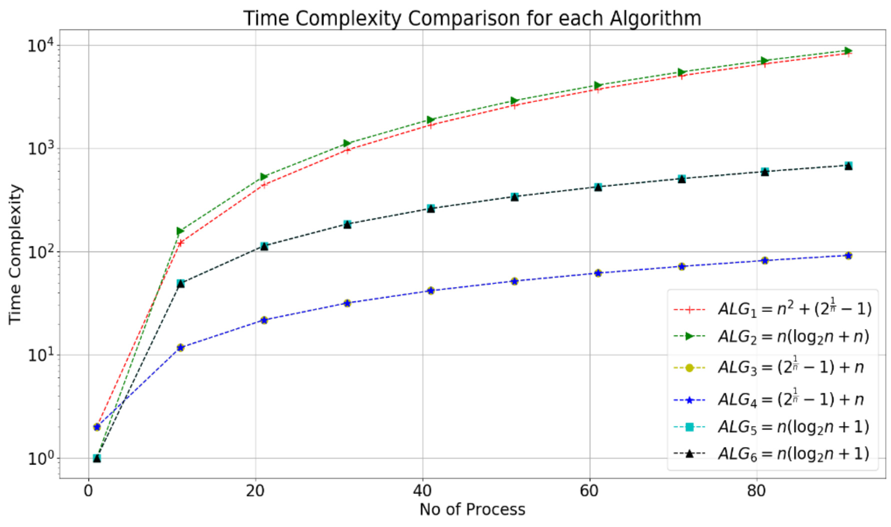

*Figure 19 Time Complexity Analysis for each Algorithm assuming that the number of resources is constant.*

where $Mem_{M_j}$ is the memory utilization of $M_j$. Therefore, each $M_j$ maintains the following

$$Cl_{M_j}^{status} = \{M_1^s, M_2^s, ..M_n^s\} \tag{51}$$

The cooperative algorithm is presented in algorithm 2.

**Algorithm 2: Cooperative Offloading Decision Algorithm**

```
Input: $T_i^u\{c_i m_i, l_i, s_i\}$, $M_j^s\left\{S_{M_j}, WT_n, Mem_{M_j},\right\}$
Output: $a_i$
Steps
    1.  do
    2.  $tk = equ(35)$
    3.  Substitute inputs and determine tk
    4.  if $tk = True$ do
    5.      $a_i \leftarrow 0$
    6.      return $a_i$
    7.  else do
    8.      for $M_j$ in $Cl_{M_j}^{status}$ do
    9.          Substitute $M_j^S$ and determine tk
    10.         if $tk = True$ do
    11.             $a_i \leftarrow 1$
    12.             return $a_i$
    13.             [End Loop]
    14.     $a_i \leftarrow 2$
    15.     return $a_i$
    16.     [End else]
    17. end
```

### 3.4.5.8 Time Complexity

The differences in the time complexity of the algorithms used in this research to design each algorithm can be seen in Table 8. Comparing the deadlock algorithms, the Banker's algorithm has the highest order of time complexity. However, comparing the scheduling algorithms, the EDF algorithm has the highest order of complexity. The time complexity graph in Figure 19 compares the time complexity of each of the compared algorithms. The graphs show the scalability of each of the compared algorithms with an increase in the number of processes and resources. The graph illustrates that $ALG_1$ and $ALG_2$ are the most scalable case study

*Table 8 Time Complexity Comparison of Each Component Algorithm*

| **Algorithms** | | **Complexity** |
|---|---|---|
| Deadlock | Banker's algorithm | $O(mn^2)$ |
| | Wound wait | $O(mn)$ |
| | Wait-die | $O(mn)$ |
| Scheduling | RMS | $n\left(2^{\frac{1}{n}} - 1\right)$ |
| | EDF | $n \log_2 n$ |

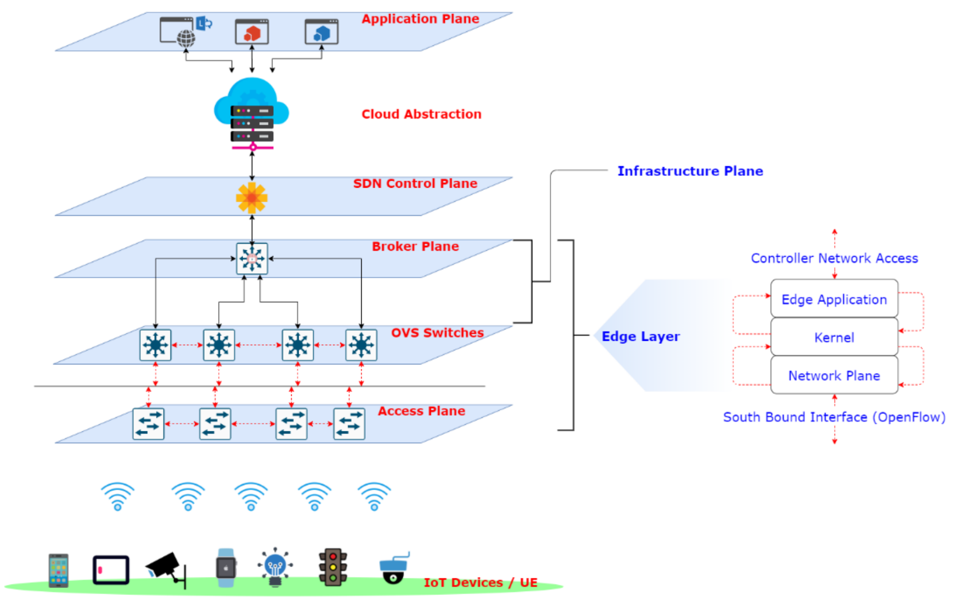

*Figure 20 High Level Deployment Architecture that has been adapted for experimentation. The figure consists of the edge layer, the SDN control plane and the cloud abstraction.*

algorithms with increase in the number of processes and no of resource types. On the other hand, $ALG_3$ and $ALG_4$ is the least scalable algorithm out of the six algorithms.

### 3.4.6 Experimental Setup

In this section, the experimental setup is presented and how the compared algorithms are tested is outlined. The components that make up the system and the associated tools are discussed. This section is broken down into two subsections as two different experimental setups were used to evaluate and compare the case study algorithms. Both experiments have been carried out using Graphical Network Simulator-3 (GNS3) platform [154]. GNS3 is a network software emulator first released in 2008 that can be used to emulate complex networks with a combination of virtual and real devices. More detail of the experimental setup is available in appendix 1.

#### 3.4.6.1 Deployment Architecture

Figure 20 illustrates the high-level diagram of the experimental deployment. The Edge layer consists of the access plane and the infrastructure plane. The Edge layer extends the conventional infrastructure by providing compute and storage capacities to the IoT devices/UE for resource provisioning. Compute and storage decisions are made in the Edge layer through the edge application. The edge servers in the MEC layer collaborate among themselves using the network plane to support the demand from the IoT devices/UE. Routing and forwarding decisions are made by the SDN controller in the control plane. The broker plane provides a publish-subscribe network infrastructure that supports IoT protocols like MQTT for transport messages between the edge and IoT devices. While using the MQTT protocol, two QoS modes have been used for communication, at-*least-once,* and *exactly-once* [155]. The at-*least-once* (best-effort delivery) is used if the process sent cannot be scheduled in real-time while the *exactly-once* (highest level of service) is used in the process that can be scheduled in real-time. The end devices reach the edge layer through a cellular

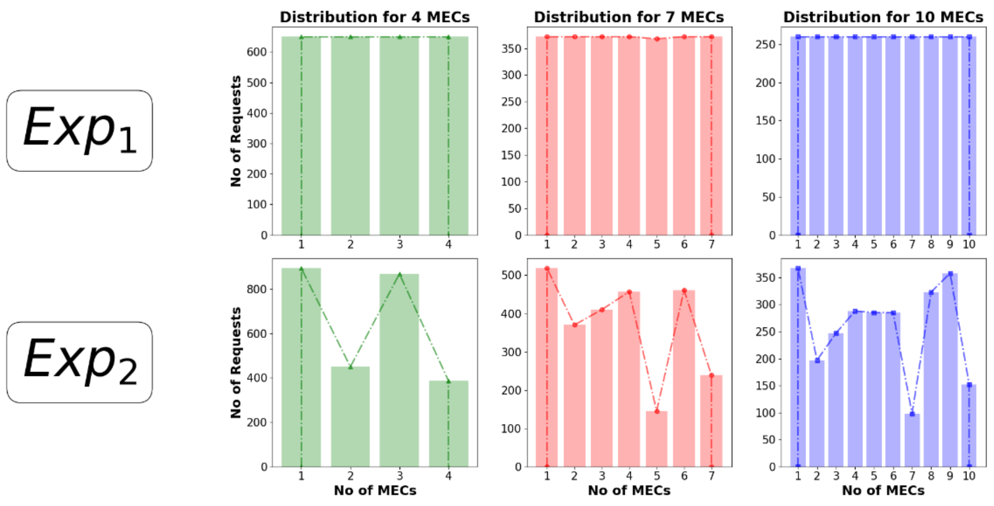

*Figure 21 Task Distribution Adapted for Exp1 And Exp 2*

communication link. Processes are sent to the cloud if they cannot be scheduled in the edge layer.

In the experimental setup in the GNS3 platform [156], the MEC is deployed as Linux hosts with SDN capabilities running docker engine as the virtualization infrastructure. Each of the algorithms is implemented using python [156] and is deployed as an MEC service on each MEC node. During the experiment, each MEC node in the MEC layer runs one of the algorithms listed in Table 6. The end devices are emulated using Ubuntu Linux containers. The device in the user layer generates tasks based on predefined experimental task profiles and sent to the MEC for processing. The MEC then applies the scheduling and deadlock algorithm for processing. The task profile includes the CPU, memory, data size, and latency constraints of each task. The task is either executed locally, re-offloaded to another MEC, or sent to the cloud. The cloud is simulated as a group of Linux hosts running the task processing services application.

Two experimental setups have been used to evaluate the algorithms. For each of the setup, the experiment has been conducted using 4 MECs, 7 MECs, and 10 MECs. For each of the experiments conducted, each MEC receives 2600 requests. Each request contains $|T_i|$ number of tasks where, $|T_i| \in \{1,2,..n\}$. The higher the value of $n$, the more load on the MEC and the more difficult it is to meet the deadline. $n$ is set to 3 in the following experiments. The task arrival at each MEC node is assumed to follow the Poisson arrival process with a varying

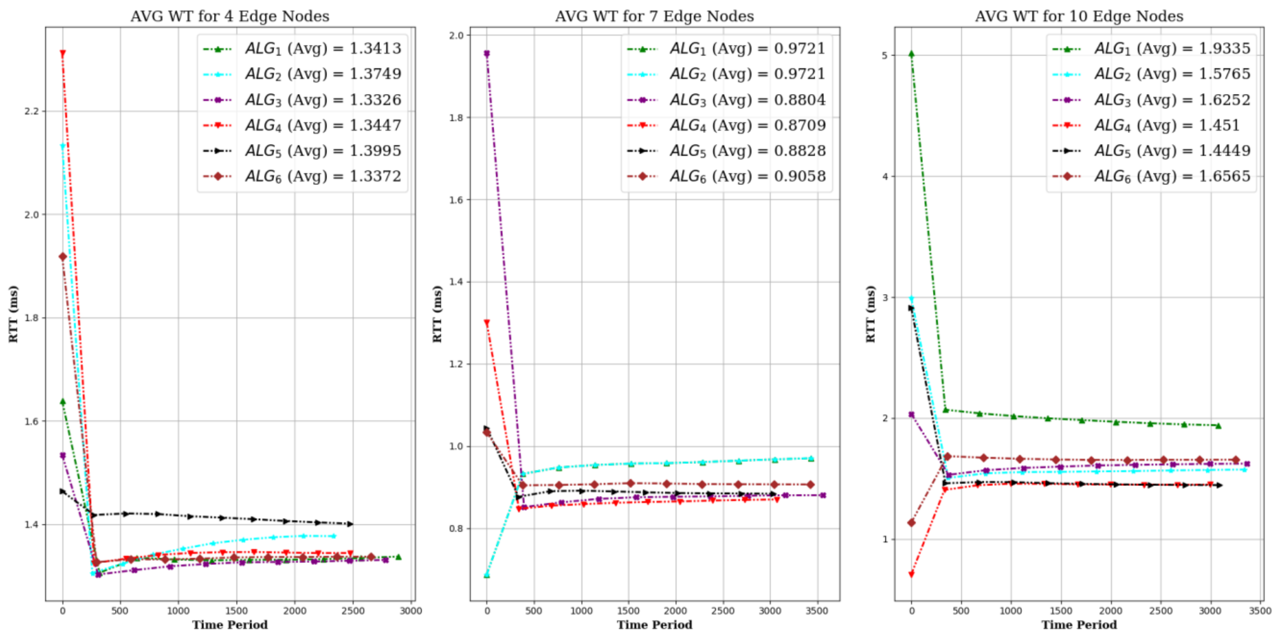

*Figure 22 The waiting time obtained for each of the algorithm during the experiments in milliseconds and the average waiting time for Exp1 is displayed.*

arrival rate $\lambda_t (t \in \{1,2..n\})$. The difference between the two experimental setups is the client request distribution.

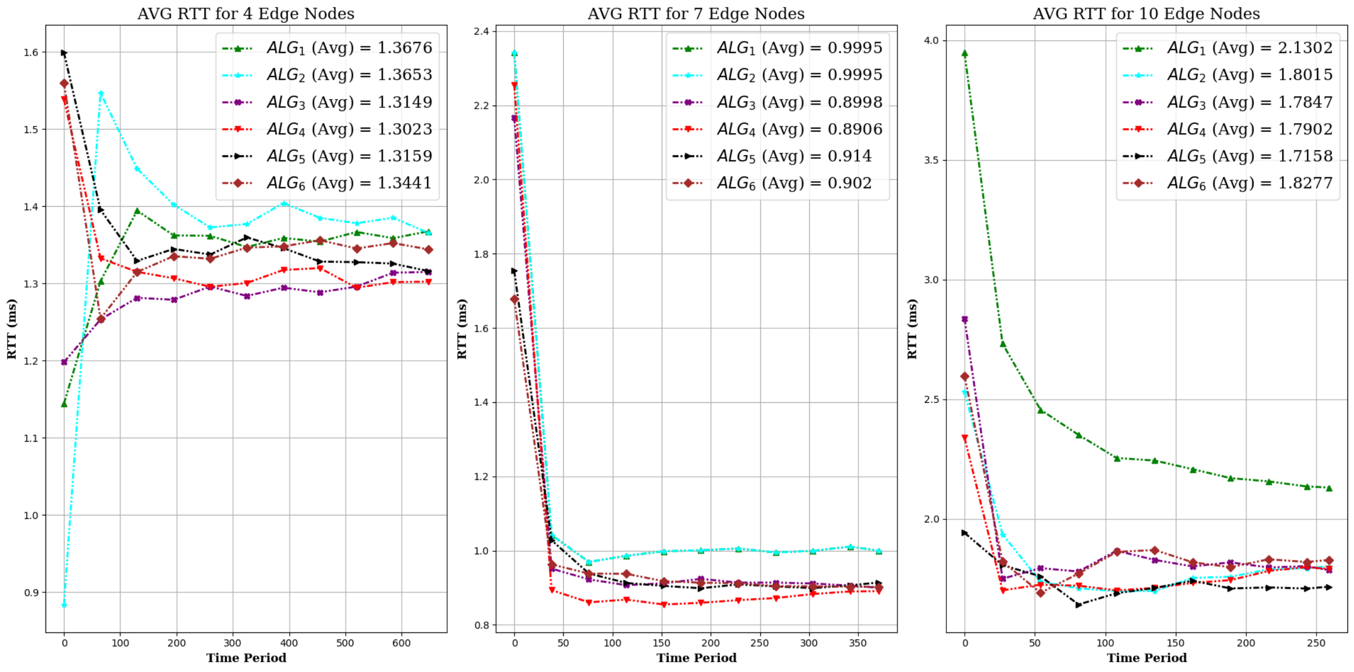

*Figure 23 RTT Comparison for Experimental Setup 1 The RTT obtained for each of the algorithm during the experiments in milliseconds and the average RTT for Exp1 is displayed*

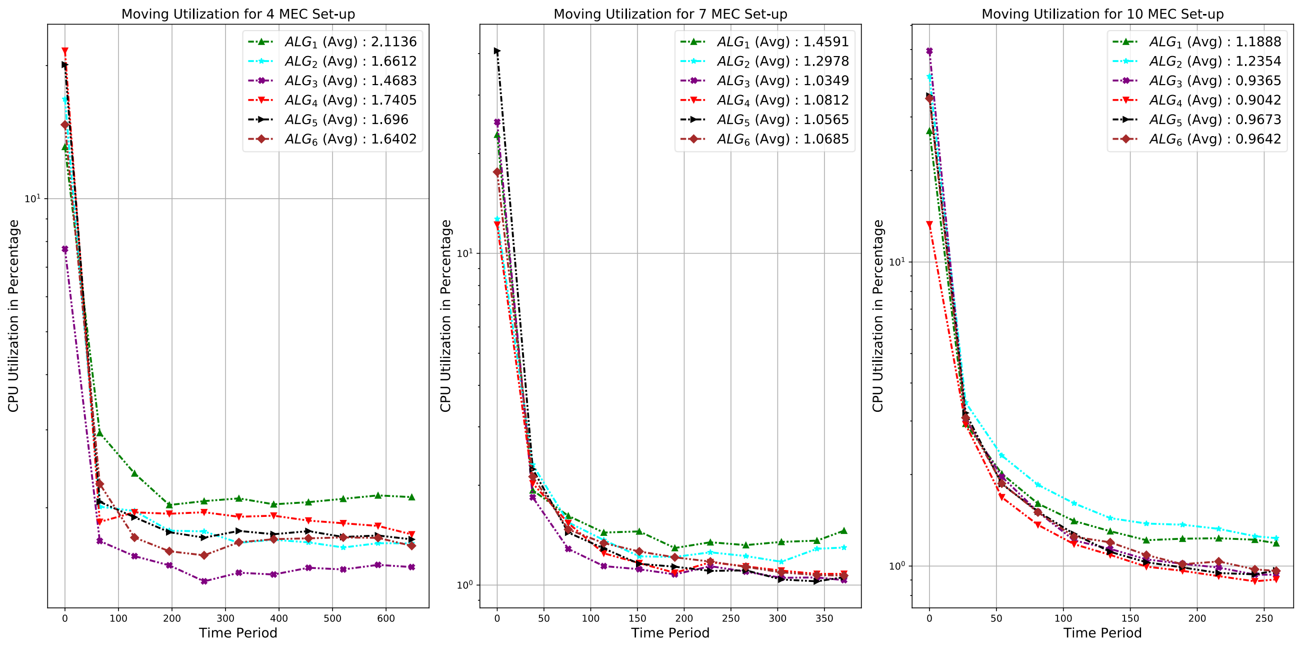

*Figure 24 The CPU utilization for each of the algorithm during the experiments in percentage and the average CPU utilization for Exp1 is displayed*

#### 3.4.6.2 Experimental Setup 1 (Exp1)

In this experiment, every MEC node receives the same number of total requests in each run. Figure 21 $Exp_1$ depicts the request distribution used for experiment setup 1. This setup simulates a scenario during co-operative offload where MECs are equally busy.

#### 3.4.6.3 Experimental Setup 2 (Exp2)

In this experiment, every MEC node receives is the same number of total requests in each run. Figure 21 $Exp_2$ depicts the request distribution used for experiment setup 2. This setup simulates a scenario during co-operative offload where MECs are unequally busy.

### 3.4.7 PERFORMANCE COMPARISON RESULTS

In this section, the performance of the compared algorithms obtained during the experiment has been evaluated. The metrics used for these comparisons are listed below.

- CPU Utilization of the MEC node
- Round Trip time
- The waiting time
- The ratio of tasks re-offloaded or executed locally on the MEC

#### 3.4.7.1 Experimental Setup 1 Results

This section contains experimental results obtained during experimentation using the experiment setup 1.

#### 3.4.7.1.1 CPU Comparison for $Exp_1$

The diagram in Figure 24 shows the experimental results obtained when the CPU utilization for each of the algorithm is compared. From the results obtained it can be deduced that the CPU utilization of the MEC platform decreases gradually with an increase in MEC nodes. This occurs because of a reduction in the number of tasks processed per MEC node. The $ALG_4$ provides the best CPU utilization convergence as depicted in Figure 24. The least convergence

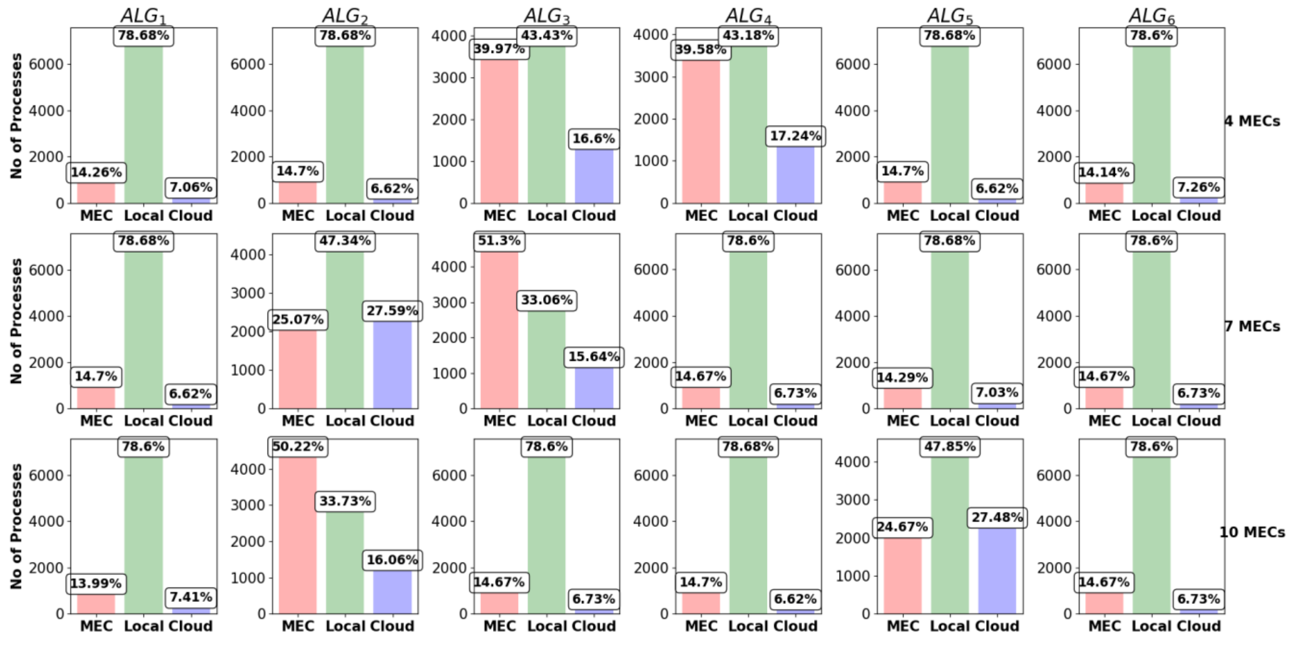

Figure 25 The figure shows a comparison of the amount of tasks that have been executed in the local MEC, re-offloaded to another MEC or cloud in Exp1

CPU utilization is the $ALG_1$.

#### 3.4.7.1.2 RTT Comparison for $Exp_1$

The RTT depicted in Figure 23 is obtained from the perspective of a MEC $M_i$. $M_i$ monitors periodically the RTT to reach each of its neighbouring MECs. Therefore, each MEC would

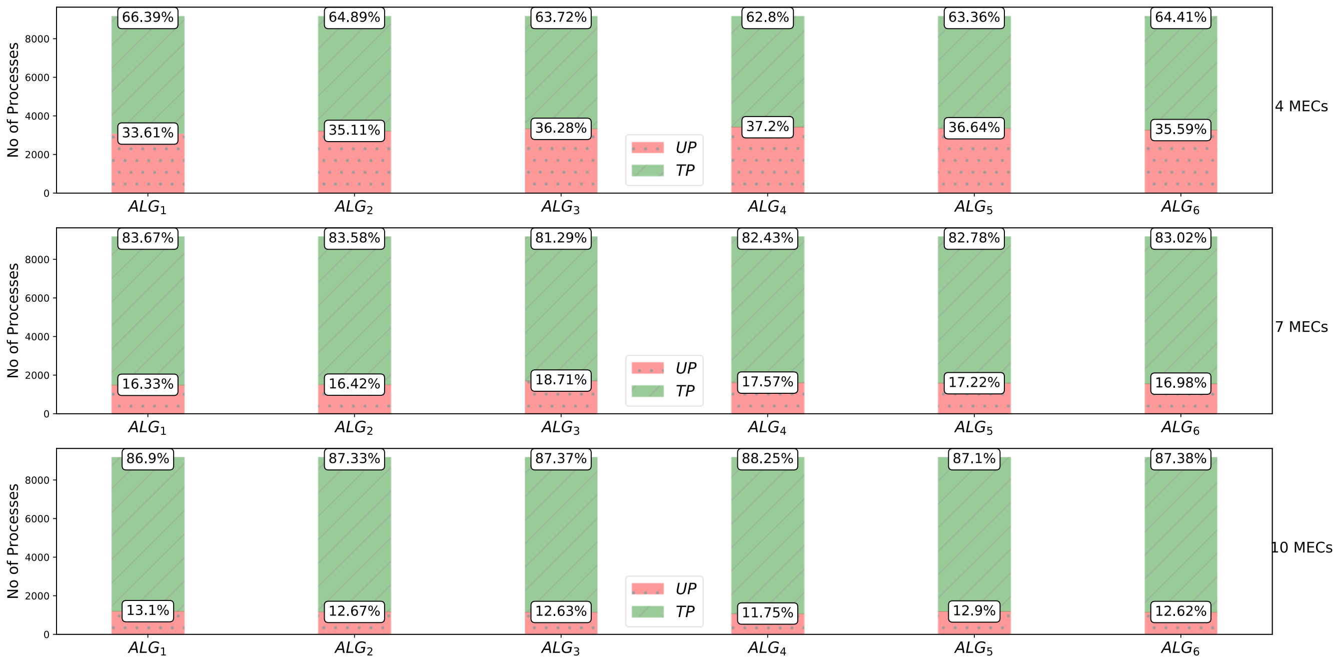

*Figure 26 Comparison of The Ratio Of Processes That Missed Their Deadline To The Processes That Failed To Meet Deadline. TP (Timely Process) Is Used To Denote Processes That Meet Execution Deadline While UP (Untimely Process) Is Used To Denote Tasks Processes that missed deadline for Exp1*

monitor $N - 1$ MEC nodes during the experimentation, where N is the total number of MECs in the cluster. The RTT is used to approximate the communication delay between the MECs. The RTT is ranged from an average of 0.89 to 2.13 milliseconds.

#### 3.4.7.1.3 Waiting Time Comparison for $Exp_1$

The waiting time is crucial during the algorithm run time because it is one of the factors used by each MEC to determine which of the neighbouring MEC is suitable for task re-offload if need be. The waiting time here is obtained for each of the MEC periodically, similar to how the RTT is obtained. The waiting time $WT_{m_i \to m_j} = rtt_{m_i \to m_j} + Q_{mj}$, where $rtt_{m_i \to m_j}$ is the RTT from $M_i \ to \ M_j$ and $Q_{m_j}$ is the queue waiting time for $M_j$. During re-offloading, a MEC with the lowest waiting time is always selected to evenly balance the load across the MEC platform. The waiting time for all six algorithms is shown in Figure 22. $ALG_3$ maintain a lower average in most of the experimental runs. It can be seen that the waiting time decreases initially and then reaches a stable state. This is because of the load balancing effect of the tasks on the MEC cluster.

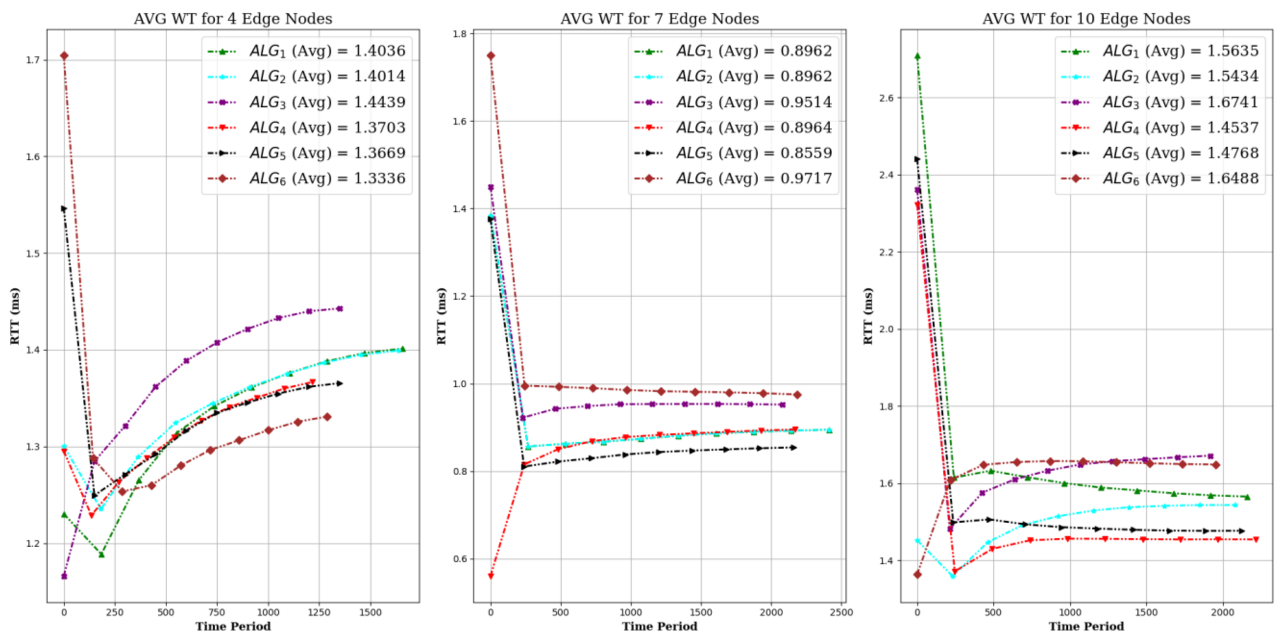

*Figure 28 The waiting time obtained for each of the algorithm during the experiments in milliseconds and the average waiting time for Exp2 is displayed.*

#### 3.4.7.1.4 Offload vs Local Comparison for $Exp_1$

In this section, a comparison is made between the ratio of tasks re-offloaded, tasks that are executed locally on the MEC, and tasks executed on the cloud. This is shown in Figure 25. According to the figure, the maximum percentage of tasks executed locally on the MEC is 78%. The outcome displayed on the graph depends on the scheduling algorithm employed. There are some noticeable similarities and differences among the six algorithms. An increase

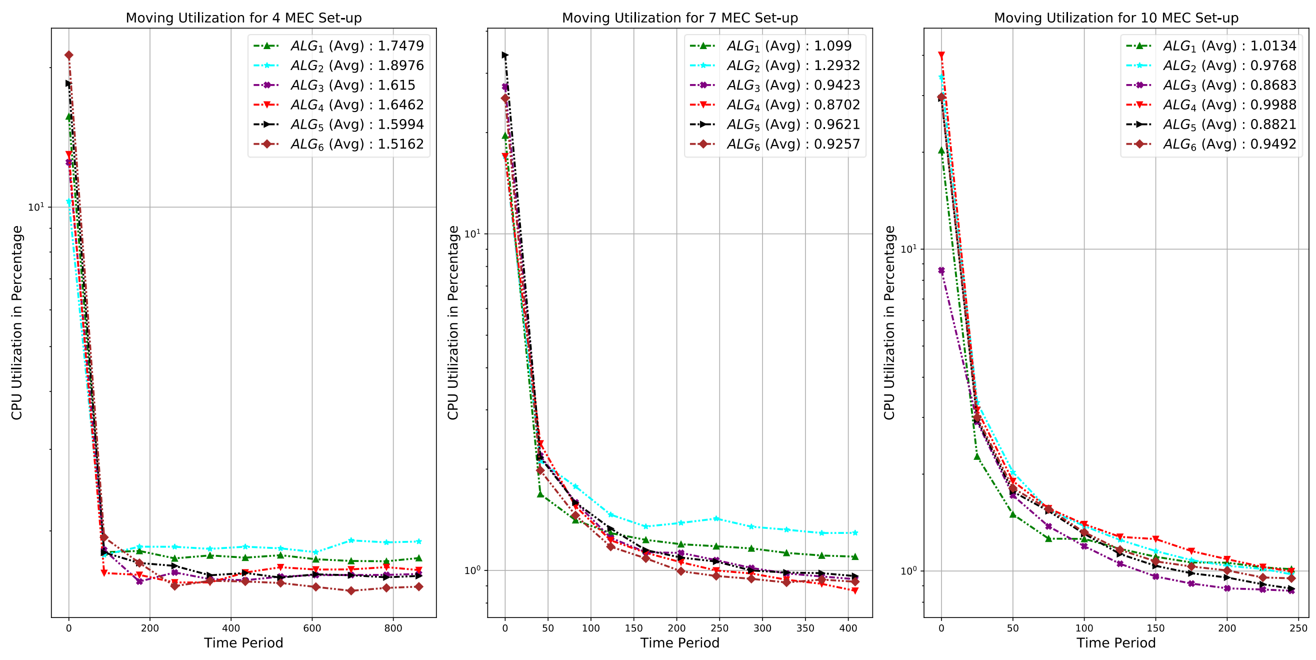

*Figure 27 The CPU utilization for each of the algorithm during the experiments in percentage and the average CPU utilization for Exp2 is displayed*

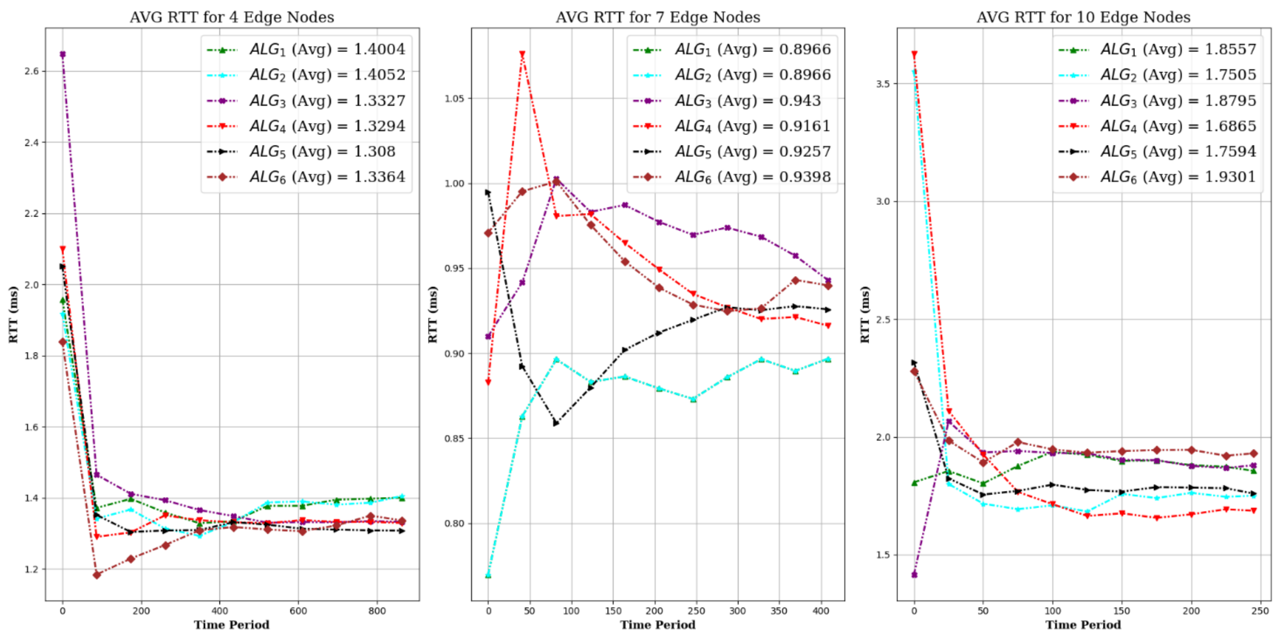

*Figure 29 The RTT obtained for each of the algorithm during the experiments in milliseconds and the average RTT for Exp2 is displayed.*

in the number of nodes has very little effect on $ALG_1$ and $ALG_6$. However, for $ALG_2$ and $ALG_5$ (both uses EDF), an increase in the number of nodes increases the number of tasks re-offloaded to MEC and cloud. This has an opposite effect on $ALG_3$ and $ALG_4$ which both uses RMS for scheduling.

#### 3.4.7.1.5 Comparison of the ratio of processes that meet Execution Deadline ($Exp_1$)

In this section, the ratio of tasks that meet their execution deadline during the experiment for $Exp1$ is compared for all six compared algorithms. These results are obtained from the client's perspective. Each task that is sent out by the client to the MEC node has a deadline constraint and is monitored to make sure that the deadline constraint is met as the task travels through the MEC platform and back to the client node. The percentage of tasks that meet the deadline constraint is labelled here as TP (Timely Process) while the percentage of tasks that did not meet the deadline constraint is labelled here as UP (Untimely Process). It can be seen in Figure 26 that more tasks meet the deadline as the number of MECs increases. It can also be seen that more tasks meet their deadline while using $ALG_4$ for $Exp_1$ compared to the other algorithms because it maintains a considerably lower RTT than others.

### 3.4.7.2 Experimental Results for setup2 ($Exp_2$)

This section contains experimental results obtained during experimentation using the Experimental setup 2.

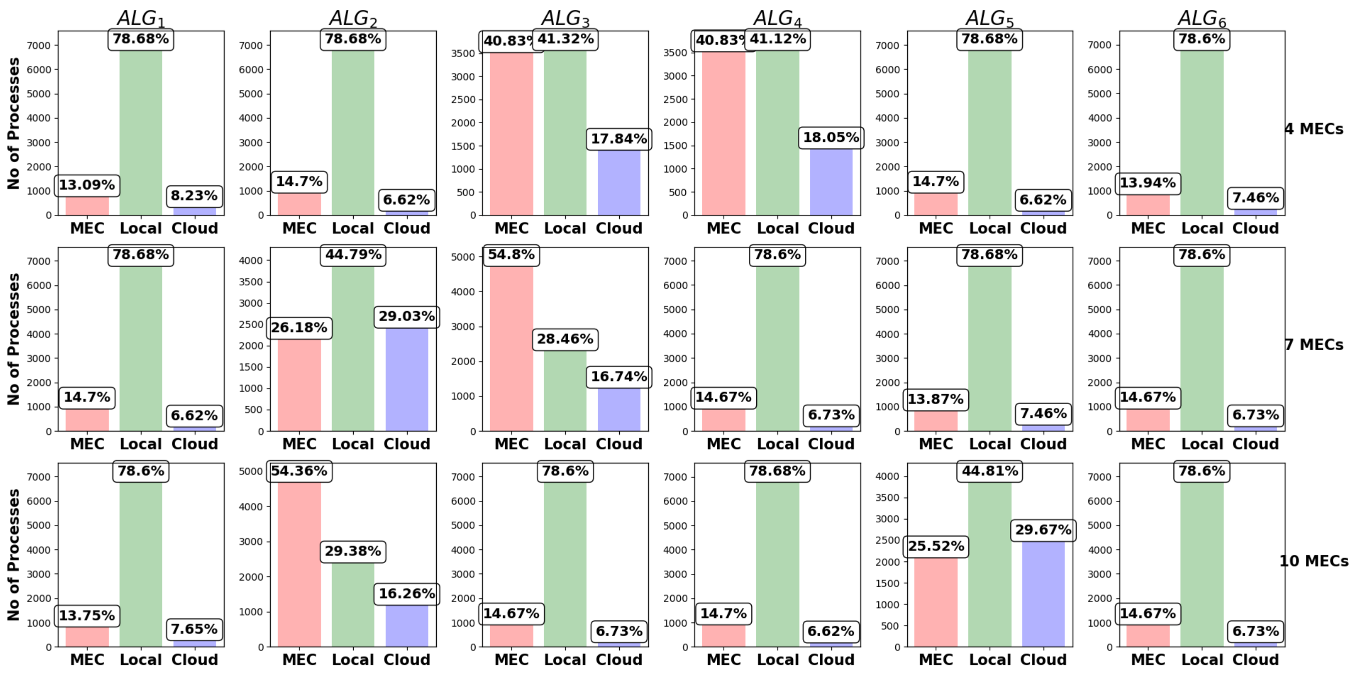

Figure 30 Comparison Between The Ratio of Processes That Was Executed Locally in The MEC To Processes That Were Re-Offloaded for Exp2

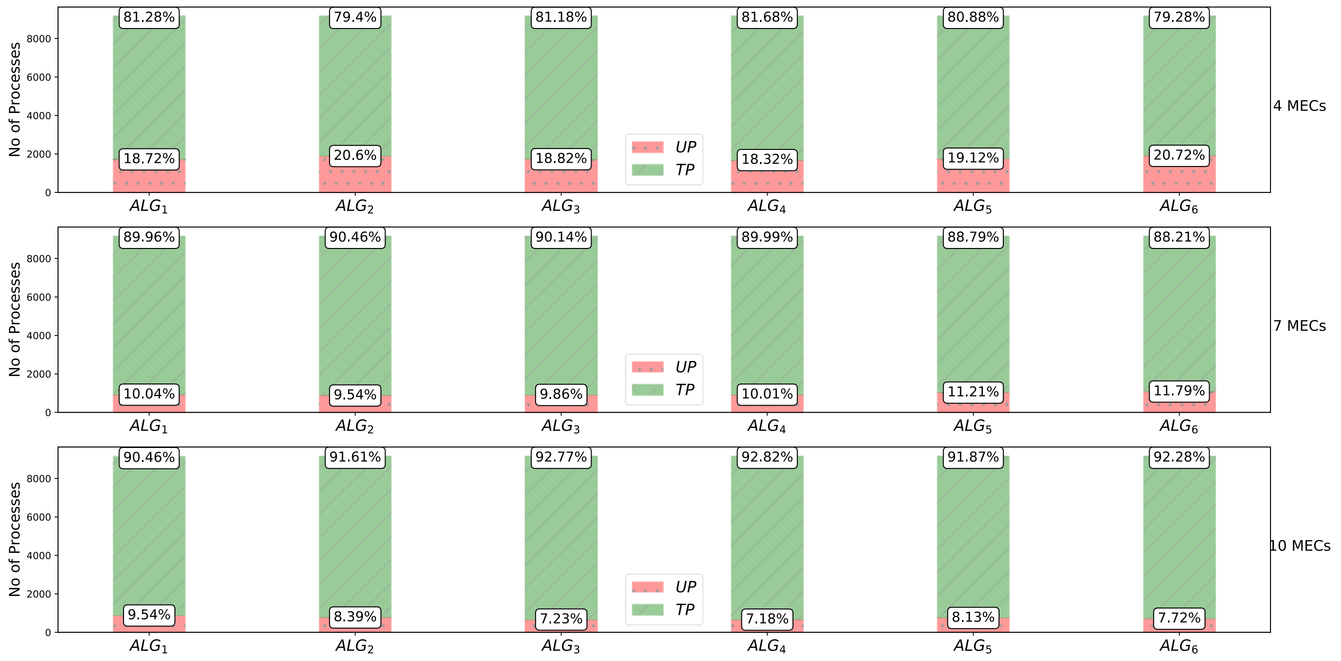

Figure 31 Comparison Of The Ratio Of Processes That Missed Their Deadline To The Processes That Failed To Meet Deadline. TP (Timely Process) Is Used To Denote Processes That Meet Execution Deadline While UP (Untimely Process) Is Used To Denote Tasks Processes that missed deadline for Exp2

#### 3.4.7.2.1 CPU Comparison for $Exp_2$

Figure 27 shows the CPU utilization results obtained for the algorithms during the $Exp2$. As depicted in the figure, the CPU utilization for each of the compared algorithms decreases with an increase in the MEC node. This can be attributed to the sharing of the total workload sent by clients among the MEC nodes. It can also be seen that the CPU utilization of the case study algorithms for $Exp2$ is lower than the $Exp1$. This observation is expected as in $Exp1$, the MECs were equally busy throughout the experiment. Averaging the results of the 3 sub experiments with 4, 7, and 10 MECs, $ALG_2$ obtains the highest CPU utilization while $ALG_6$ achieves the slowest CPU utilization during the experiments. $ALG_2$ uses a deadlock avoidance algorithm while $ALG_6$ uses a deadlock prevention algorithm.

#### 3.4.7.2.2 RTT for $Exp_2$

The RTT obtained for $Exp2$ can be seen in Figure 29. Each participating MEC records the RTT for each of the MEC in the platform to be used for offloading decisions. The RTT is obtained here in the heterogeneous setup similar to how it is obtained in the homogeneous setup. The round-trip time for the $Exp2$ ranged between 0.89 to 1.93 milliseconds . Averaging the results of the 3 sub experiments with 4, 7, and 10 MECs, ALG4 obtains the lowest overall RTT while $ALG_6$ achieves the highest RTT. $ALG_4$ and $ALG_6$ are both deadlock prevention algorithms. However, $ALG_4$ uses RMS for task scheduling while $ALG_6$ uses EDF.

#### 3.4.7.2.3 Waiting Time for $Exp_2$

The waiting time convergence for the $Exp2$ can be seen in Figure 28. $Exp2$ setup seems to have a more predictable convergence than the $Exp1$ as the waiting time converges between 0.89 to 1.63 milliseconds for each of the experimental runs from 4 to 10 MECs. The load balancing effect is also witnessed here as the nodes reaches a stable state. On average, $ALG_4$ attains lower waiting time while $ALG_1$ acquires higher waiting time during the 3 sub-experiments from 4 to

10 MECs. $ALG_4$ utilises a deadlock prevention algorithm while $ALG_1$ employs a deadlock avoidance algorithm.

#### 3.4.7.2.4 Offload vs Local Comparison for $Exp_2$

In this section, comparisons are made between the ratio of tasks executed locally, re-offloaded to the cloud, or re-offloaded to another MEC. It can be seen in Figure 30 that an increase in the number of MECs leads to an increase in the percentage of tasks re-offloaded to a neighbouring MEC for all compared algorithms. The overall behaviour here is similar to what is shown in $Exp_1$. $ALG_6$ and $ALG_1$ algorithm had the best performance result with the number of tasks executed locally for each run above 78%. $ALG_2$ achieves the highest increase rate in tasks re-offloaded to be executed in a neighbouring MEC as the number of MECs increases.

#### 3.4.7.2.5 Comparison of the ratio of processes that meet Execution Deadline

In this section, the ratio of tasks that meet their execution deadline during the experiment for $Exp2$ is compared for all six compared algorithms. These results are obtained from the client's perspective. Each task that is sent out by the client to the MEC node has a deadline constraint and is monitored to make sure that the deadline constraint is met as the task travels through the MEC platform and back to the client node. The percentage of tasks that meet the deadline constraint is labelled here as TP (*Timely Process*) while the percentage of tasks that did not meet the deadline constraint is labelled here as UP (*Untimely Process*). It can be seen in Figure 31 that more tasks meet the deadline as the number of MECs increases. It can also be seen that more tasks meet their deadline while using the $ALG_2$ for each experimental run from 4 MECs to 10 MECs compared to the other algorithms. $ALG_1$ obtains the lowest $TP$ for 10 MECS while $ALG_4$ achieve the highest $TP$. $ALG_4$ utilises a deadlock prevention algorithm while $ALG_1$ employs a deadlock avoidance algorithm.

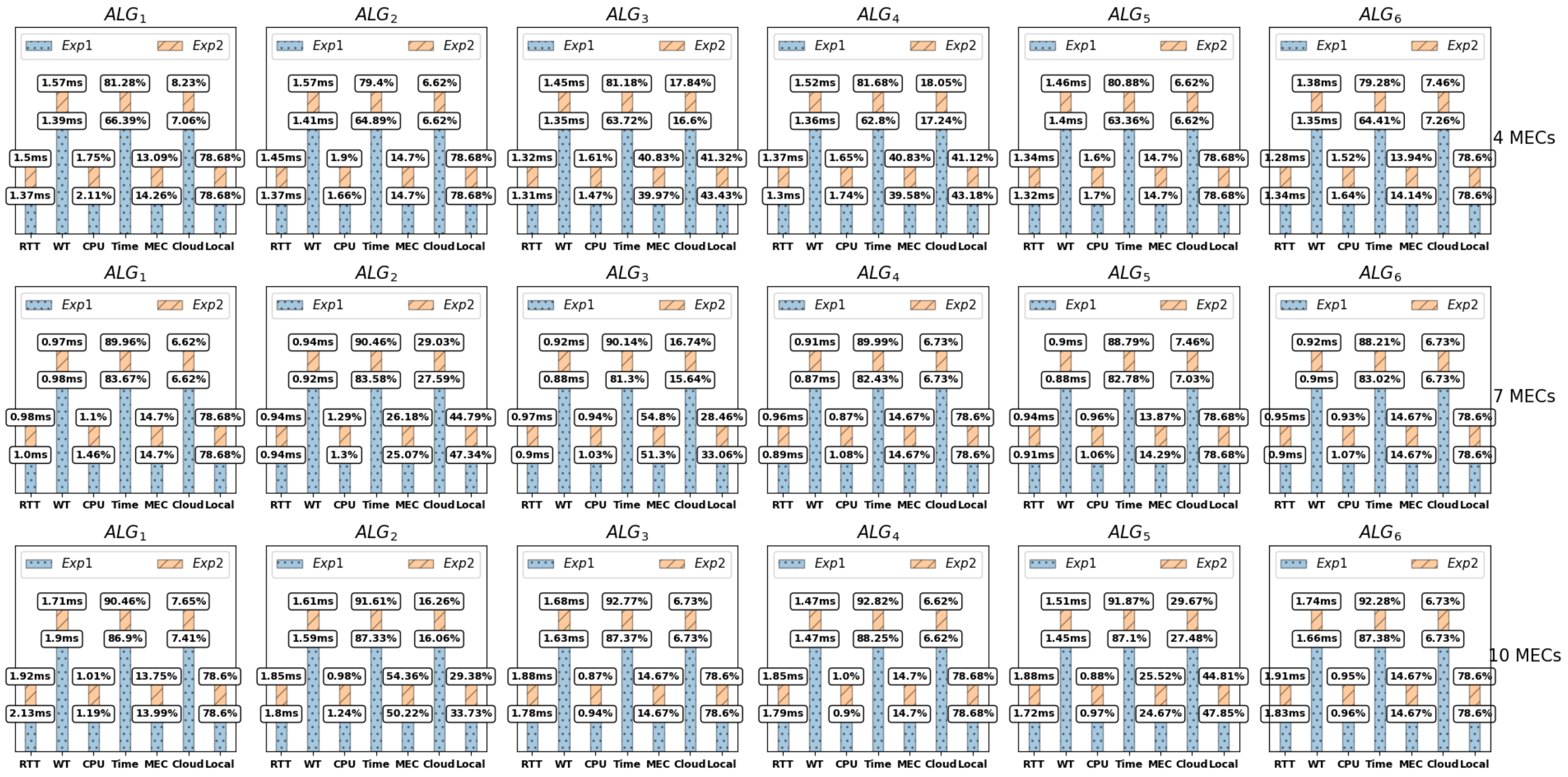

*Figure 32 Summary Comparison of the results obtained in Exp1 and Exp2 (Graph is not drawn to scale) (Graph is Not Drawn To Scale)*

#### 3.4.7.3 $Exp1$ *and* $Exp2$ Comparison

Figure 32 summarizes the difference in the experimental outcomes of $Exp1$ and $Exp2$. The figure shows the average outputs of each of the experimental runs (4, 7, and 10). The figure shows that $ALG_3$ obtains better CPU utilization compared to other algorithms. $ALG_6$ *and* $ALG_1$ obtains better percentage of tasks executed locally compared to other algorithms in $Exp1$ *and* $Exp2$. Comparing algorithms that obtains better overall percentage of tasks executed on time, $ALG_4$ provides the best performance among all the algorithms under study. $ALG_1$ and $ALG_2$ uses a deadlock avoidance algorithm while $ALG_3$ uses a deadlock prevention algorithm. It is difficult to generalize these results as it can be seen that none of the algorithms is superior over the other in all comparing metrics. Each algorithm performs well in a particular metric but not all. The optimum difference between these algorithms is the ability to keep the system in a safe state. However, it is difficult to emulate this in a test environment.

## *3.5 Conclusion*

In this chapter, MEC computing has been explored. Through an extensive state-of-the-art literature review, deadlock during resource provisioning in MEC has been defined as a research gap that needs addressing. This has formed the basis of the problem addressed in this chapter. It is crucial to investigate this problem as reliable communication is very vital in achieving 5G uses cases such as URLLC. This chapter has contributed to the design and development of a reliable MEC environment by eliminating the chances of deadlock during resource provisioning. Additionally, sub-problems such as task load balancing among MECs, task re-offloading and MEC co-operation during resource provisioning has been addressed.

The deadlock problem has been explored using a case study for IoT devices. This case study has been used because it supports two of the 5G use cases which include URLLC and mMTC. Additionally, IoT devices have inadequate computation and storage resources therefore they offload a majority of their workload. Assuming these IoT devices offload their workload to the edge platform and considering that the edge nodes have a finite amount of resources. A continuous increase in the workload and IoT devices dependent on the edge resources might lead to over-provisioning which may result in a system deadlock because of many devices contending for limited and shared resources. The simulation results confirm this behaviour. This issue has been fully modelled in this chapter considering both the communication and computational model. To address the problem, a resource provisioning framework for deadlock avoidance for MEC has been proposed to maintain a more reliable network system for IoT devices. The proposed framework has incorporated Bankers' resource request deadlock avoidance algorithm. An avoidance algorithm has been chosen to ensure the eradication of deadlock in the system. The framework also includes a cooperative mechanism for selecting an appropriate MEC node if task re-offloading is required. This ensures that the latency constraints are satisfied during offloading and the MEC with adequate resources is selected for re-offloading. Extensive simulation tests confirm deadlock if the system is in an unsafe state and there is a continuous increase of IoT applications dependent on the edge node. Results have also confirmed the hypothesis that the proposed algorithm can eradicate deadlock in the system and thus ensuring system reliability and availability.

Additionally, comparisons have been made between different deadlock algorithms for MEC using the proposed framework. The study has been aimed to investigate the difference if a deadlock prevention algorithm has been used in the proposed framework rather than a deadlock avoidance algorithm. The study has been carried out in a step on building a reliable and readily available MEC platform that can be used to deliver low latency requirements for 5G case study scenarios. Using a case study for real-time systems, comparisons have been made on how each deadlock mechanism will perform in real-time scenarios employing popular baseline real-time algorithms such as RMS or EDF for prioritizing workloads. Two experimental setups have been designed on the GNS3 platform for evaluating the compared algorithms. The metrics used in the comparison include RTT, waiting time, CPU utilization, the ratio of tasks that meet the deadline, and the ratio of local execution to cooperative MEC to cloud. Results have shown that the avoidance algorithm does better in the percentage of tasks executed locally and the overall percentage of tasks executed on time while prevention algorithms obtain better CPU utilization. It has been concluded that the optimum difference between these two algorithms with regards to the framework is the ability to keep the system in a safe state and thus, eradicate deadlock. Thereby, improving the overall QoS of the system

# CHAPTER 4 MEC CACHE RESOURCE MANAGEMENT

## *4.1 INTRODUCTION*

The previous chapter has explored the improvement of QoS in 5G networks by optimization of computational resources in MEC with a deadlock-aware scheduling algorithm. This chapter also contributes to the improvement of the QoS in 5G by exploring an efficient caching scheme to optimize the cache resource and reduce latency.

Content caching has evolved over the years considering technology trends and policies enforced for efficient caching. There are limited locations for caching unit deployment at mobile networks. Before the advancement of MEC, the main places caching units are deployed are in Radio Access Network (RAN), core network, and the user devices [107]. The core happens to be mostly used for cache deployment [157] because the traffic can be reduced by two-thirds. However, deploying cache units in MEC is bound to yield greater benefits because the nodes are close to the users [107].

One of the promising functionalities of MEC is the ability to provide caching capabilities to mobile devices in 5G, enabling fast popular content delivery for delay-sensitive applications. MEC is still in its infant stage, therefore there are few studies on MEC about content caching. In this chapter, efficient caching schemes for MEC has been explored. Firstly, an extensive literature review has been carried out on caching in MEC. The need to develop a caching scheme that can adapt to dynamic cache popularity changes due to the ever-changing user request pattern has been identified. Therefore, a predictive caching scheme that can adapt to the caching user request pattern has been presented. The prediction of user request has been obtained using Lagrange extrapolation. Additionally, a different cache prediction approach has been explored to increase the user request prediction accuracy, increase the hit ratio, and improve the QoE. The two proposed approach supports co-operative caching for MEC to improve the collective MEC cache-store.

## *4.2 LITERATURE REVIEW*

Caching algorithms have been studied widely by the research community in different research fields including MEC. This ranges from operating systems to network caching and in-between. In this section, a review is carried out on the caching algorithms that are relevant to this study.

### 4.2.1 Conventional Replacement Algorithms

In this section, the conventional algorithms that are still commonly used have been reviewed. Conventional here refers to standard, and well-accepted popular algorithms. The main priority of algorithms reviewed in this section is to increase the hit ratio.

- **First In First Out (FIFO):** This is one of the simplest replacement policies in terms of time complexity and implementation. In a FIFO queue, cache objects are placed in the tail of the queue. If there is a need for replacement, cache objects are removed from the head until there is enough space for the incoming request. The time complexity of the algorithm is $O(1)$.
- **Least Recently Used (LRU):** LFU [129] is still one of the commonly used algorithms. When the cache is full, the policy replaces the cache object which has not been referenced for the longest of time. The strategy is based on the observation that blocks that have been recently referenced are likely to be used again in the future. LRU works well with workloads that exhibit strong temporal locality. However, according to [158], LRU does not work well with file server caches. The time complexity of the algorithm is $O(1)$.
- **Least Frequently Used (LFU):** LFU is another classic cache replacement algorithm. LFU maintains a reference count for all cached objects. Therefore, when the cache is full it replaces the cache object with the lowest reference count. The rationale of the algorithm is that some cached blocks are more frequently accessed than others.

Therefore, the frequency count could be a good estimate of the probability of a cached object being requested. LFU has two main drawbacks. Firstly, there may be a tie if two cached objects have the same frequency. Secondly, a cache object may accumulate a large reference count and never replaced even if the cache object is no longer active. There have been many improvements proposed to address the drawback of LFU. One of these improved versions is the aged LFU. This policy gives different weight to recent and old references. Aged LFU performs better than the original LFU [159]. The time complexity of LFU is $O(\log(n))$

- **Least Recently Used K(LRU-K):** This algorithm combines both LFU and LRU schemes. It was first introduced in database disk buffering. The basic idea of LRU-k [160] is to keep track of the times of the last K references to popular cache objects. This information is then used to estimate statistically the interarrival times of references on a request-by-request basis. The replacement decision is based on the reference density observed during the past K references. When K is small, cold cache objects are identified quicker as such cache objects have a wider span between the current time and the $k_{th}$-to-last reference time. The time complexity of LRU-K is $O(\log(n))$.
- **Least Recently Frequently Used (LRFU):** LFRU [161] is another algorithm that combines LFU and LRU. The strategy of this algorithm is to replace cache objects that are least frequently used and not recently used. Each cached object is associated with a Combined Recency and Frequency value (CRF). The cache object with the lowest CRF value is replaced. Each request of a cache object contributes to its CRF. LRFU has a time complexity of between $O(1)$ and $O(\log(n))$.
- **Frequency Based Replacement (FBR):** FBR [162] algorithm is a hybrid replacement scheme that combines both LRU and LFU to capture the benefits of both algorithms. FBR has been initially proposed for managing caches for file systems, management

systems, or disk control units [162]. FBR maintains LRU queues of cache objects with the same frequency count. To address the problem of cache objects, accumulating large reference counts, the algorithm maintains 3 sections. These include $F_{new}$, $F_{middle}$ and $F_{old}$. These sections are used to bound the frequency count of certain cache objects and they are required algorithm parameters. FBR also requires 2 additional parameters, $A_{max}$ and $C_{max}$. $A_{max}$ refers to the maximum average of frequency counts to be maintained while $C_{max}$ is the maximum chain count. More details can be found in [162]. The replacement decision is primarily based on the frequency count. According to [158], FBR is the best algorithm compared to LRU and LFU. The time complexity of FBR ranges from $O(1)$ to $O(\log(n))$.

- **Two Queue (2Q):** 2Q algorithm[163] has been proposed as an improvement to LRU-k. The motivation is to reduce the access overhead and remove cold cache objects quickly. The 2Q uses two LRU queues $A1_{out}$ and $A_m$ and an additional FIFO queue $A1_{in}$. The cache objects are initially stored in the $A1_{in}$ when first accessed. When the cache is evicted from $A1_{in}$ it is then added to $A1_{out}$. If a cache object in $A1_{out}$ is accessed, it is moved to $A_m$. The authors have proposed a scheme to select the efficient sizes of $A1_{in}$ and $A1_{out}$. The 2Q performs better than FBR, LRU and LFU for second level buffer caches [159]. The time complexity of 2Q is $O(1)$.
- **Multi-Queue (MQ):** MQ[159] has a comparable technique to 2Q. The motivation is to create an algorithm that supports minimal lifetime for cache objects, has a frequency-based priority, and supports the temporal frequency. MQ uses $m$ number of LRU queues, where $m$ is a parameter. Cache objects in certain queues have a longer lifetime than others depending on the queue the object lies. MQ also uses a history FIFO queue $Q_{out}$ of limited size to store recently evicted cache objects. MQ evicts the cache object

in the tail of the LRU queue with the least frequency. MQ [159] performs better than FBR, Q2, LRU, and LFU. The time complexity is $O(1)$.

### 4.2.2 Network-Aware Replacement Algorithms

Many factors affect the performance of the replacement algorithm used in network caching. These include the requested object size, latency, bandwidth, miss penalty, temporal locality, and long-term access frequency. Successful application of these algorithms can reduce network traffic, response time, and server load. The algorithms reviewed in this section take at least one of these parameters into consideration during cache replacement.

- **GreedyDual (GD) Algorithms:** GD Algorithm has several variations, but the key objective is to replace the cache object with the lowest cost value based on a specified cost function. These variations have different cost functions. The original GreedyDual has been proposed by Young [164]. The motivation of the algorithm has been to deal with cache objects that have the same size but incur a different cost in bringing them to the cache-store. When a cache object is retrieved, a value $H$ is assigned to it. This is the cost of bringing the cache object to the cache-store. The algorithm replaces the cache object with the $min\ H$ and then all cache objects reduce their $H\ value$ by $min\ H$. The time complexity for this is $O((n-1)*\log(n))$. Two other variations of GD are listed below:
    - **GD-Size:** GD-size [165] extends the original GD by adding the size of the case to the cost function. Therefore, $H = cost/size$ where $size$ is the cache size and $cost$ could vary depending on the cache priority. The cost could be set to 1 if the goal is to maximize the hit ratio. It could be set to the downloading latency if the goal is to minimize average latency. Finally, it could be set to the network cost if the goal is to minimize the total cost. The time complexity of GD-Size $O(\log(n))$

    - **GreedyDual-Size with Frequency (GDSF):** GDSF [166] has been proposed as an extension of GD-Size. The limitation of GD-Size is that it does not consider the popularity of the cache objects during cache replacement. GDSF has a $H = F * (cost/size) + L$, where F is the frequency count and L is a running age factor. $L$ starts at 0 and is updated for each replaced object if the priority key of this object is in the priority queue.

- **Least Unified-Value (LUV):** LUV [167] allocates a calculated value to each cached object. When the cache is full, the cache object with the lowest value is replaced. The value is calculated by $weight * H$, where weight is the retrieval cost $(cost/size)$ and $H$ is the probability that the object is going to be re-referenced in the future. The time complexity of LUV is $O(\log(n))$.

- **Lowest-Latency-First (LLF):** LLF ranks the cache objects based on their download latency. When the cache is full it replaces the cache object with the lowest latency. The motivation of this scheme is to minimize the total latency in the system. The time complexity of the algorithm is $O(\log(n))$.

### 4.2.3 Size Aware Algorithms

In this section, algorithms that predominately make cache replacement decisions based on the requested object size are reviewed.

**Size:** The Size algorithm [166] replaces the largest cache object when the cache is full. The strategy is to increase the cache hits by increasing the number of cache objects in the queue. Therefore, to minimize the miss ratio one large object is replaced rather than many smaller ones. The limitation of this approach is that the smallest cache objects which are rarely accessed are never replaced.

- **Size-Adjusted LRU:** The size-adjusted LRU [166] associates a cost-to-size ratio to each of the cached objects. The cost value is a function of the size and access time of the cache object. The cost-to-size ratio is $1/(Size * \Delta T)$. $\Delta T$ is the elapsed time from last access time to current time. The time complexity of the algorithm is $O(\log(n))$.
- **Least Recently Used – Size adjusted and Popularity aware (LRU-SP):** LRU-SP [168] uses two extensions of the LRU algorithm, namely Size-adjusted LRU and segmented LRU. The cost to size function of LRU-SP is $(\frac{nref}{size * \Delta T})$. Cache objects are put into a limited number of groups according to [log $(size/nref)$]. When the cache is full only the last 20 cache objects are considered for replacement. Therefore, the cache object with the lowest cache-to-ratio value is replaced. The time complexity of the algorithm is $O(1)$.
- **Log(Size)+LRU:** Log(Size)+LRU [166] evicts the document that has the largest log(size) and is the least recently used among all documents with the same log(size)
- **Pitkow/Recker:** Pitkow/Recker [166] removes the least recently used document, except if all documents are accessed within a given time interval, in which case the largest one is removed.

### 4.2.4 Edge Caching Algorithms

There has been a considerable amount of research carried out on MEC to enhance its cache performance. To accomplish this, the caching algorithm must be designed to support cache sharing among MEC nodes, reduce latency and bandwidth, and increase network robustness and reliability. In this section, such relevant algorithms have been reviewed.

Wu et al [169] have proposed a collaborative edge caching mechanism for ICN. The scheme advocates cache redundancy by replicating cache object to the next hop whenever a cache hit occurs. Ndikumana et al [170] have proposed a collaborative scheme for edge computing with a collaborative space defined by the network administrator based on hop count distance between edge nodes. In the proposed scheme, the edge nodes periodically exchange resource and cache updates. In [171], an inter and intra tier collaborative hierarchical caching mechanism over 5G edge computing has been proposed. The scheme makes caching decisions to minimize the number of wireless hops in obtaining a cache object while maximizing the hit ratio. Liu et al [172] have proposed a collaborative online edge caching algorithm. The scheme uses a Bayesian clustering technique to group users based on their request preferences. Popular preferences are then cached to improve the global cache hits. Saputra et al [173] have proposed two proactive and cooperative caching frameworks for mobile edge networks. In the first approach, the edge nodes send data to a central server which creates a deep learning model based on the data popularity and sends it to the edge servers. The second approach allows each edge node to create a local model and then send it to the central server for model aggregation. The aggregated global model is then sent back to the edge nodes.

Few researchers have proposed edge caching strategies for 5G that combines both computation and data caching. Markakis et al [174] have proposed a proactive extreme edge caching strategy that predicts and prefetch popular contents based on big data analysis. On the other hand, Sungwook K. [175] has leveraged a holistic caching structure for caching in small base stations using game theory. The developed hybrid algorithm uses split caching where one part caches popular content for communication and the other part caches computation offloading services.

Caching cooperation among MEC servers can increase overall caching efficiency. Many researchers have used cooperative caching in MEC. In [176] a Mix-Cooperative (MixCo) caching strategy has been developed for MEC servers in a Fiber-Wireless (FiWi) access network to reduces latency and increase cache performance. Huang et al in [177] have proposed a CMAC(Cooperative Multicast-Aware Caching) strategy to reduce the average latency of delivering content and Yang et al in [178] have explored ways of avoiding pollution attacks on cooperative MEC caching. Cooperative caching can be divided into two categories, centralized cooperative caching and distributed cooperative caching. In this chapter, distributed cooperative caching has been explored.

Chen et al [179], have proposed a neural collaborative filtering caching strategy for edge computing. The proposed method incorporates a greedy algorithm, a popularity prediction algorithm, and a content cache replacement algorithm. Simulation results show that the proposed algorithm can outperform baseline algorithms with regards to hit rate, transmission delay, and content cache space utilization.

Xu et al [180] have proposed a hybrid edge caching scheme for tactile internet in 5G. The proposed scheme has been aimed at energy efficiency improvement in proactive in-network caching. The cache replacement policy proposed assumes that the cache files follow Zipf distribution. Simulation results have shown that the proposed method achieves better latency compared to conventional caching algorithms.

### 4.2.5 Predictive Caching Algorithms

The optimal caching algorithm is an algorithm that can accurately predict the cache request pattern for $t + 1$, where $t$ is the current time, and use this to make appropriate caching decisions. For such algorithms, the prediction cost is usually expensive, and many at-times fail to be consistently accurate. Therefore, there are only a handful of such caching algorithms. These algorithms are reviewed in the following paragraphs.

Qi et al [181] have proposed a proactive caching scheme for the wireless edge. The proposed scheme applies federated learning for cache popularity prediction. The authors highlighted a security vulnerability with centralised learning as it needs to collect information from users during the learning process which can be personal. Utilizing the proposed approach, each user uploads a weighted sum of preference and file popularity to the base station where models are aggregated. Simulation results show that the proposed scheme achieves a close cache-hit ratio to a centralised learning approach.

Tan et al [182] have proposed a reinforcement learning-based optimal computing and caching scheme for edge networks. The problem has been formulated as an infinite-horizon average-cost Markov Decision Process (MDP). The authors have aimed at maximizing bandwidth utilization and decreasing the quantity of data transmitted. The scheme considers long-term file popularity and short-term temporal correlations of user requests to fully utilize bandwidth. Simulation results show that the proposed policy scheme can predict content popularity and user future demands.

Sajeev et al [183] have proposed a machine learning algorithm for web caching. The algorithm utilizes the Multinomial Logistic Regression (MLR) to classify web cache "object's worthiness". The variable, "object worthiness" is a polytomous (discrete) variable that depends on 6 parameters. These 6 parameters include popularity, recency, size, popularity consistency, delay, and object type. If the cache is full, the least worthy cache object is replaced. The algorithm requires parameter tuning and the simulation result shows that the algorithm performs better than LRU, LFU, and GDSF in hit rate and byte rate.

Dutta et al [184] have proposed a caching framework for mobile wireless networks. The proposed technique uses a predictive scheme for both replacement and prefetch. The problem has been formulated as a QoE optimization problem and solved using Markov Predictive Control and Markov Decision Process. Prediction is done using the FP-Growth association rule-based algorithm. Empirical results have shown that the proposed algorithm performs better than LFU, LRU, and FIFO in terms of hit ratio.

Chan et al [185] have proposed a big data-driven predictive caching at the wireless edge. The framework utilises a machine learning-based approach to anticipate user behaviours and content patterns and then prefetch content with expected high popularity. The framework uses a generic Markov prediction model for prediction. The authors state that the machine learning-based approach is useful in improving the cache performance when the popularity distribution fails to follow the Zipf distribution. The proposed algorithm performs better than the LRU algorithm.

Rahman et al [186] have proposed a deep learning predictive caching framework for edge networks. The proposed framework uses a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) model for predicting the popularity of cache objects. The authors have used MovieLens 20M dataset for training the model. The authors assume there will be few changes in the dataset and therefore they have not considered model updates during runtime. The model error rate has been evaluated. However, the proposed algorithm hit rate ratio has not been analysed.

To address the problem of predictive model inaccuracy due to the changing popularity distribution of cache objects, Song et al [187] have proposed a dynamic content placement caching framework. The novel learning framework predicts the temporal cache distribution of future cache contents. The content placement is periodically updated based on future requests. Empirical results show that the algorithm performs better than conventional online caching algorithms.

Finally, for more works on predictive caching in the edge, Wang and Friderikos [188] have provided a comprehensive survey of using deep learning frameworks to predict caching data in edge networks. They have evaluated the different techniques and summarized the research challenges, the benefits and the cost of using deep learning in edge networks.

## 4.3 Co-operative and Hybrid Replacement Caching Algorithm (CHRCA) for MEC

This research section aims to increase the cache performance of the MEC. To accomplish this, a caching algorithm has been designed and developed that can load balance cache data among MEC nodes, reduce latency and bandwidth and increase network robustness and reliability so that content to be constantly available even if the server of the data source is down. This would be achieved by caching the contents from the data source to the MEC layer so that users can fetch data directly from the MEC layer without going to the data source.

The proposed algorithm is a unification of a modified Belady's algorithm [189] and a co-operative caching algorithm. First, to improve cache hits at a MEC node, Optimal Page Replacement Strategy (OPR) also known as Belady's algorithm is leveraged. The downside of Belady's algorithm is the need for a future reference string. To obtain this in this research, historical data are kept for the requests received by a MEC and its relative frequency. These historical data are then used to predict the future occurrences for received requests.

To further maximize the benefits of this solution, a cooperative caching strategy is used to share the cache data of each MEC node with other MEC nodes within the same cluster. Successfully increasing the cache hit rate performance would reduce the cache access time of the MEC platform and overall power consumption [190].

The proposed algorithm depends on the successful prediction of future requests based on past references. Therefore, a polynomial regression algorithm is employed to enable the forecast of the reference string.

### 4.3.1 System Architecture

In this section, the steps to be taken to accomplish the objectives are explained.

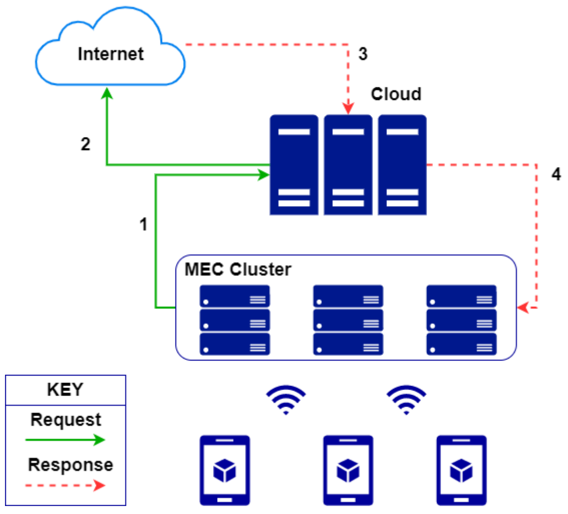

*FIGURE 33 REFERENCE ARCHITECTURE FOR CO-OPERATIVE CACHING*

Figure 33 shows a request-response architecture adopted in this study. The request-response flow for a worst-case scenario is shown to depict when a user request is not in the MEC cluster nor the central cloud servers. In this figure, mobile devices are connected to the edge node through a wireless Access Point (e.g. IEEE 802.11) and the edge node is connected to the cloud servers which are in turn connected to the internet.

The focus of this research is on the MEC layer and how to leverage its distributed architecture to increase its cache efficiency by generating more cache hits while reducing network latency. Here, the aim is to reduce the mobile application's response time by storing popular data at the cache servers of the edge node. This will reduce access to the data repositories since most of the data are stored locally.

### 4.3.2 CHRCA Framework Components

In this section, the different components of the CHRCA framework have been discussed. These components or mechanisms are listed as follows:

#### 4.3.2.1 Co-operative caching

Co-operative caching is a method of caching in a distributed architecture that supports the sharing of cache information. One of the most popular co-operative caching methods is to create a central storage for the distributed servers and allow all connected servers to access cache data from the central storage [103]. This reduces the burden of managing data only on one device, but also creates other challenges such as increased latency due to additional access time in fetching cached data from a remote location. Another problem is finding the optimal location of the central storage device to minimize the access time of all the servers involved. To address this problem, here an algorithm is proposed which utilizes distributed cooperative caching to reduce the impacts of one centralized cache server. The servers in the distributed architecture are allowed to keep their cache data but they also keep a synchronized database of the data available in the MEC cluster cache $M_{cache}$. $M_{cache}$ can be implemented as a HashMap with each MEC ($MEC_j$) as the key and the value is the set of cache saved for each MEC.

$$M_{cache} \leftarrow \{MEC_j \rightarrow \{r_i \,\forall\, r_i \in MEC_j\} \,\forall\, MEC_j \mid j \in \{1,2,,cz\}\} \tag{52}$$

Where $cz$ is the cluster size and $r_i$ cache saved in the MEC. The cluster size is adjusted based on the traffic load and network status to balance the content diversity and spectrum efficiency.

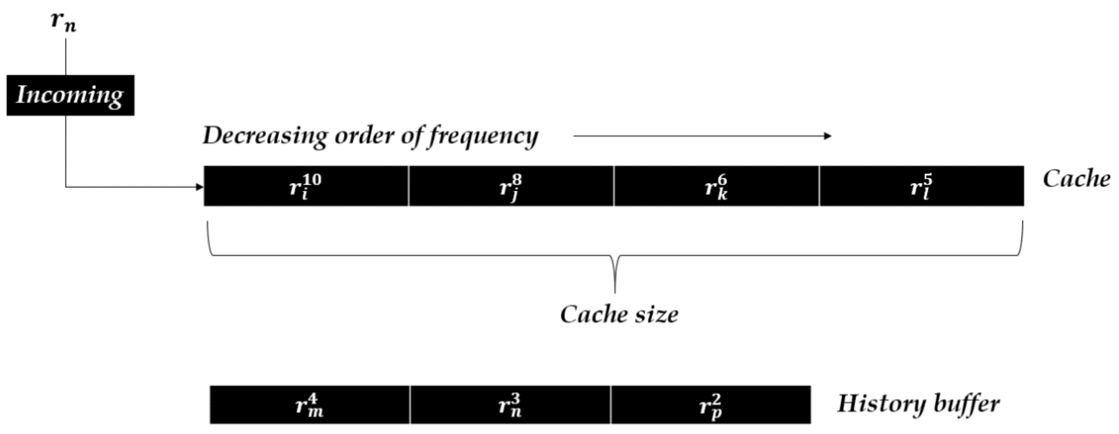

*Figure 34 MLFU Problem Depiction*

Therefore, if a MEC receives a request from an end-user device, it would first search if the request exists in its cache. If the request does not exist, it will search $M_{cache}$. If the requested data is still not obtained, then the MEC initiates a request to obtain the requested data from the data source.

#### 4.3.2.2 Modified Least Frequently Used (MLFU) Cache replacement

LFU requires that the reference with the least count be replaced. To keep a cohesive and reliable frequency record in this proposed modification, a window-size is maintained so that the average frequency of each reference is obtained. LFU only compares the frequency of the references in the cache and not the newly requested data. Therefore, a problem arises when a new request with little or no historic frequency record $r_n$ is used to replace an existing cache data which has the least frequency the cache $r_l$. However, $r_l^5$ has a higher historic frequency record compared to the newly requested data $r_n^3$ as depicted in Figure 34. To address this problem, a modification has been proposed to not only compare the frequencies of the data in the cache but also compare the frequency of the newly obtained request. A decision is made not to cache the newly obtained request if the frequency of the data are less than the frequency of the least frequency in the cache. This method is ideal for caching in MEC in a scenario where the newly requested data are obtained from the MEC cluster. Therefore, if a decision is made not to cache and the data is requested again, it can be obtained from the MEC cluster as the retrieval cost is assumed to be less than the data source.

#### 4.3.2.3 Modified Optimal Page Replacement (MOPR) Algorithm

The Optimal Page Replacement Algorithm (Belady's Algorithm) [189] as its name implies, is selected by researchers as the optimal algorithm for cache replacement since it produces fewer page faults than any other algorithm. However, it is labelled infeasible because it requires data on the occurrence of future requests. This data is not available in the real world. The modification made here is in the process of predicting this future request occurrence. This is achieved by keeping a record of the historic occurrences of the requests and leveraging this data for prediction. Prediction is done using Lagrange extrapolation. Eviction is accomplished by replacing the cache object with the farthest predicted occurrence.

| **System Modelling for CHRCA 4.3.3** | |
|---|---|
| **Notation** | **Meaning** |
| $C_1$ | Cloud |
| $C_2$ | Precache |
| $C_3$ | MEC local cache |
| $M_{cache}$ | MEC cooperative cache |
| $R$ | Received Reference string i.e a list of requests |
| $n$ | Window size of the reference string |
| $rf$ | Relative Frequency |
| $\tau_i$ | Set of timestamps |
| $r_i$ | Request/reference at index $i$ |
| $P(x)$ | Lagrange Interpolating polynomial |
| $d$ | Degree of polynomial |
| $p$ | Time period |
| $R_F$ | Forecasted reference string |
| $\tau_i^{next}$ | the next predicted occurrence of $r_i$ |
| $v$ | Cache victim |
| $FR$ | Relative frequency function |
| $cz$ | MEC cluster size |

## 4.3.3 System Modelling

In this section, the hierarchical content caching placement design has been modelled. This consists of the cloud, the precache, and the MEC.

$$C_3 \subseteq C_2 \subseteq C_1 \tag{53}$$

where $C_1$ represents the cloud, $C_2$ represents the precache and $C_3$ MEC cache. The pre-cache is a HashMap of the previously requested data and its relative frequency $rf$. Each data is uniquely identified by a hash which is obtained from an MD5 hash function of the request.

$$C_2 = \{hash \to rf \ \forall \ hash \in R\} \tag{54}$$

$R = \{r_t \mid t \in [0, n-1]\}$ is a finite, non-empty reference string of received reference instances $r_t$ at time stamp $t$, with window size $n$. $C_2 \subseteq R$ is a HashMap. Hence, a set of distinct elements from the list $R$, with cardinality $|C_2| = m$. The function $FR: r_i \in C_2 \to [0,1]$ calculates the relative frequency $rf$ of occurrence of each reference $r_i$ of pre-cache $C_2$. It measures the share of occurrences of a reference instance over the total window. The function $FR$ is defined below.

$$FR(r_i) = \begin{cases} 0 \ if \ r_i \notin R \\ \frac{f(r_i)}{|R|} \ otherwise \end{cases} \tag{55}$$

Where $f(r_i)$ is the frequency of $r_i$. Hence the sum of the relative frequency of all elements in the pre-cache is unity,

$$\sum_{i=0}^{|C_2|-1} FR(r_i) = 1 \tag{56}$$

The proposed co-operative cache mechanism blends two different cache replacement techniques. First, MOPR, using forecasting, and second, selective caching using relative frequency. The mathematical foundations and modelling of these techniques are as follows. A subset of pre-cache is cache ($C_1 \subseteq C_2$) with a finite size. It comprises of $|C_1|$ most frequently used references from $C_2$.

*MOPR with forecasting* – The traditional OPR algorithm makes use of the future occurrences of the pages to select a victim page during page replacement. To achieve this, future samples must be available. However, in the real scenario, this criterion can be met by forecasting them using their past occurrences. An efficient way of forecasting real-time data is to use a polynomial fit algorithm with Lagrange interpolation [191].

Let $\tau_i$ be a set of timestamps where the reference $r_i \in C_2$ occurs in the reference string $R$. By nature, it is a monotonically increasing sequence that is bounded below.

$$\bigcup_{i=0}^{|C_2|-1} \tau_i \Leftrightarrow \sum_{i=0}^{|c_2|-1} |\tau_i| = n \tag{57}$$

In the above equation, $n$ refers to the window size maintained to prevent memory overflow. Let $P(x)$ be the Lagrange Interpolating polynomial of degree $\leq (d-1)$ which passes through $d$ points $(x_1, \tau_{i1}), (x_2, \tau_{i2}), \dots, (x_n, \tau_{id})$ and is given by

$$P(x) = \sum_{j=1}^{d} P_j(x) \tag{58}$$

Where

$$P_j(x) = \tau_{ij} \prod_{\substack{k=1 \\ k \neq j}}^{d} \frac{x - x_k}{x_j - x_k} \tag{59}$$

Equations (58) and (59) are adapted from [192]. Refer there for the full expansion of the polynomial function.

$P(x)$ returns a polynomial of order $d$ that fits into the given samples of times-stamps with a minimum mean squared error (MSE). The $x_i$ series are the corresponding indices of the $\tau_i$ series. For fitting a curve of order $d$, according to the convergence criteria of Newton's divided difference technique [193], it needs at least $d + 1$ data samples. If enough samples are not available, LFU replacement is used instead.

Hence the forecasting of $\tau_i$ with a period $p$ can be written as,

$$\tau_i' = \{P(|\tau_i| + 1), P(|\tau_i| + 2) \dots P(|\tau_i| + p)\} \tag{60}$$

Hence, the forecasted reference string $R_F$ would be,

$$R_F = \bigcup_{i=0}^{|C_2|-1} \tau_i{}' \tag{61}$$

However, the next time occurrence $\tau_i^{next}$ of a reference $r_i \in C_2$ using $\tau_i$ for forecasting is

$$\tau_i^{next} = P(|\tau_i| + 1) \tag{62}$$

Let at time instance $t$, a page replacement is invoked. Therefore, MOPR will calculate the $\tau_i^{next}$ for all the cache in $C_1$ and select a victim $(v)$ for page replacement. The victim is the $r_i$ with the max $\tau_i^{next}$

$$v = \max(\tau_i^{next} \; \forall \, r_i \in C_1) \tag{63}$$

*Selective caching:* In a classical caching system, a miss in an overflowing cache yields a replacement. It may so happen, that the cached data have less frequency of occurrence than the victim. Consequently, the victim is more likely to appear before the cached data hence it causes a miss. In a MEC environment, fetching data from a remote MEC node is costly, hence it raises the overall access cost significantly. This problem can be tackled with the proposed selective caching where it keeps track of the relative frequency of each reference in pre-cache ($C_2$). The cache ($C_1$) only holds the most relative frequently used references from $C_2$. Therefore, a miss event with a reference $r_j$ will only be served as a replacement if the $FR(r_j)$ is greater than the minimum $FR(r_i)$ in the cache.

$$FR(r_j) > \min(\{FR(r_i)\ \forall\ r_i \in C_1\}) \tag{64}$$

The replacement algorithm used here varies, depending on the origin of the fetched data. The following are the two possible cases of origins.

#### *4.3.3.1* Data from Source*:*

Data obtained from the source are treated with the highest priority because the link between the MEC node and the data source is very costly in terms of latency. MOPR algorithm is used if there is a need for a replacement and the data obtained are cached. The aim is to maintain any data obtained from the point of presence in the local MEC cache units.

#### 4.3.3.2 Data from MEC cluster:

Data obtained from the same MEC cluster are given less priority because the link between the MEC nodes costs less. Therefore, the selective caching approach is used where the data is only cached if equation (64) holds. Otherwise, the data is sent to the user but not cached in the MEC node. By employing such an approach, the key replacement algorithms policies which imply that the existing data are always replaced by the newly requested data upon request when the cache is full is modified. Accessing data from the MEC cluster, would not incur much delay on the system since the access time to get the data from the MEC cluster is less than the time to get the data from the point of presence. This also reduces redundant caching in the MEC cluster.

### 4.3.4 Time Analysis

In this subsection, the time analysis of the data structures maintained, and the time analysis of the replacement algorithm proposed is analyzed.

#### 4.3.4.1 Data structures time analysis

Three data structures are maintained using the proposed algorithm. This includes the set of cache $C_1$, the HashMap of $M_{cache}$ , the HashMap of $C_2$ and the list of time stamps $\tau_i$ . $M_{cache}$ requires insertion, deletion, and search operations. Insertion operation of a known key would be an $O(1)$. Deletion operation will require $O(cache_{size})$ lookup time, and an additional $O(1)$ removal time. Therefore, in worst-case scenario deletion operation will be $O(cache_{size})$. The search operation on $M_{cache}$ on worst-case scenario will require $O(cz * cache_{size})$.

The precache $C_2$ requires an update of relative frequency, insertion, and window size maintenance. Update of relative frequency will require $O(n)$. Insertion and the maintenance of the window size will incur $O(1)$ cost if a linked-list based queue is used as it requires insertion at the tail insertion and head removal.

The list of time stamps $\tau_i$ requires insertion and window size maintenance operations. Similar to the precache, insertion and the maintenance of the window size will incur $O(1)$ cost if a linked-list based queue is used.

The cache $C_1$ requires insertion and a deletion operation. The insertion is an $O(1)$ while deletion will require $O(cache_{size})$ lookup time, and an additional $O(1)$ removal time.

#### 4.3.4.2 Replacement algorithm time analysis

Let us analyse 3 scenarios of using the proposed algorithm. These scenarios represent the possible places where a request could be fetched from.

1. **Local MEC:** If the requested reference is cached locally, 3 operations would be performed. These are, update the relative frequencies, add a new timestamp to $\tau_i$ and search for $r_i$ in $C_1$. The frequency update takes $O(n)$ and the time stamp insertion takes $O(1)$ while the search operation takes $O(cache_{size})$. Therefore, the total cost incurred is $O(n) + O(1) + O(cache_{size})$.

$$time_{local} = O(n) + O(cache_{size}) \tag{65}$$

2. **Remote MEC:** If the data is to be fetched from the remote MEC, there are two possible scenarios. These are the data is cached or the data is not cached. More steps are required if the data is cached. If the data is not cached, the operations required are the same as fetch from local MEC including check if $r_i$ is in $M_{cache}$, check if the cache is full and if equation (64) is True. Checking if $r_i$ is in $M_{cache}$ incurs $O(cz * cache_{size})$ and checking if cache if full requires $O(1)$. However, for equation (64), if a min-heap is maintained the minimum $rf$ in $C_1$ can be found in $O(1)$. Therefore, the total time if data is not cached ($time_{mec_{uncache}}$) is as follows assuming cost of obtaining data from remote MEC is $t_{mec}$ .

$$time_{mec_{uncache}} = O(n) + O(cache_{size}) + O(cz * cache_{size}) + t_{mec} \tag{66}$$

$$time_{mec_{uncache}} = O(n) + O\big(cache_{size}(cz + 1)\big) + t_{mec} \tag{67}$$

However, if the data is cached, additional operations such as find victim with MOPR, delete victim, add to cache and update $M_{cache}$ will be performed. To find a victim, MOPR needs to extrapolate the next sample for each $r_i$ in cache and return the max. Using Lagrange Interpolating polynomial, the time complexity is $O(n^2)$ for each $r_i$. Finding the max using equation (63) requires sorting, thus $O(cache_{size} \log cache_{size})$.

Therefore, total time complexity to find victim using MOPR is $O(cache_{size}(n^2)) + O(cache_{size} \log cache_{size})$. Deleting victim will require find and removal operations. Thus, $O(cache_{size}) + O(1)$. Adding to cache and updating $M_{cache}$ will incur $O(1)$ cost.

Therefore, the total cost if data is cached $(time_{mec_{cache}})$ is as follows assuming the cost of obtaining data from remote MEC is $t_{mec}$ .

$$time_{mec_{cache}} = O(n) + O\big(cache_{size}(cz+1)\big) + O(cache_{size}(n^2)) + O(cache_{size} \log cache_{size}) + O(cache_{size}) + t_{mec} \tag{68}$$

$$time_{mec_{cache}} = O(n) + O\big(cache_{size}(cz+1 + n^2 + \log cache_{size} + 1)\big) + t_{mec} \tag{69}$$

$$time_{mec_{cache}} = O(n) + O\big(cache_{size}(cz + n^2 + \log cache_{size} + 2)\big) + t_{mec} \tag{70}$$

3. **Source/cloud:** If data is obtained from the source, the operations required are the same as fetching from local MEC including check if $r_i$ is in $M_{cache}$, check if the cache is full, find the victim with MOPR, delete victim, add to cache and update $M_{cache}$. These operations have been previously analysed. Therefore, the total time taken for this is as follows assuming the cost of obtaining data from the source is $t_{source}$ .

$$time_{cloud} = O(n) + O\big(cache_{size}(cz + n^2 + \log cache_{size} + 2)\big) + t_{source} \tag{71}$$

ALGORITHM I CO-OPERATIVE AND HYBRID REPLACEMENT CACHING (CHRCA)

```
Input: $n \in \mathcal{N}$    : window size
          $d \in \mathcal{N}$ : Degree of the fitted polynomial
          $cache_{size} \in \mathcal{N}$
Output: None
Data Structure: Multiple Priority Queue
Steps 1: Initialization
$C_1 \leftarrow \phi$                //Cache Memory
$C_2 \leftarrow \phi$                 //Pre-Cache
$M_{cache} \leftarrow \phi$           //MEC cooperative Cache
Step 2: Get a Request
$r_i$ : Incoming reference
Step 3: Calculate Relative Frequency
$rf_i \leftarrow \frac{f(r_i)}{|R|}$
$C_2 \leftarrow C_2 \cup \{hash \rightarrow rf_i\}$
Set 4: Update Timestamps
$\tau_i.add(t_i)$        // add new timestamp to list
Step 5: Fetch Request
if $r_i \in C_i$ do
    $fetch\ locally$  // cache hit
else if $r_i \in M_{cache}$ do
    $fetch\ from\ MEC\ with\ min\ delay$  // cooperative hit
else do
    $fetch\ from\ cloud$      // cache miss
End
Step 6: Replacement
if $|C_1| \geq cache_{size}$ do      // cache full
    if $r_i \in M_{cache}$ do
        if $FR(r_i) > min(\{FR(r_j) \forall r_j \in C_1\})$ do
            $victim \leftarrow max(\tau_j^{next} \forall r_j \in C_1)$ //find victim using MOPR. If not enough samples use LFU. If $rf$ tie use LRU
            $Remove(C_1, victim)$      // remove victim from cache
            $C_1 \leftarrow C_1 \cup \{r_i\}$          // add new data
            $update\ M_{cache}$
        else do
            // do not cache
        End
    End
    if $r_i \in cloud$ do
        $victim \leftarrow max(\tau_j^{next} \forall r_j \in C_1)$ //find victim using MOPR. If not enough samples use LFU. If $rf$ tie use LRU
        $Remove(C_1, victim)$      // remove victim from cache
        $C_1 \leftarrow C_1 \cup \{r_i\}$          // add new data
        $update\ M_{cache}$
    End
else do
    $C_1 \leftarrow C_1 \cup \{r_i\}$          // add new data
    $update\ M_{cache}$
End
```

### 4.3.5 Algorithm and System Design

In this section, the algorithm system design and development has been discussed. The steps that have been taken for the creation of the algorithm, and how it has been deployed in the testbed are also listed in this section.

#### 4.3.5.1 Algorithm Workflow

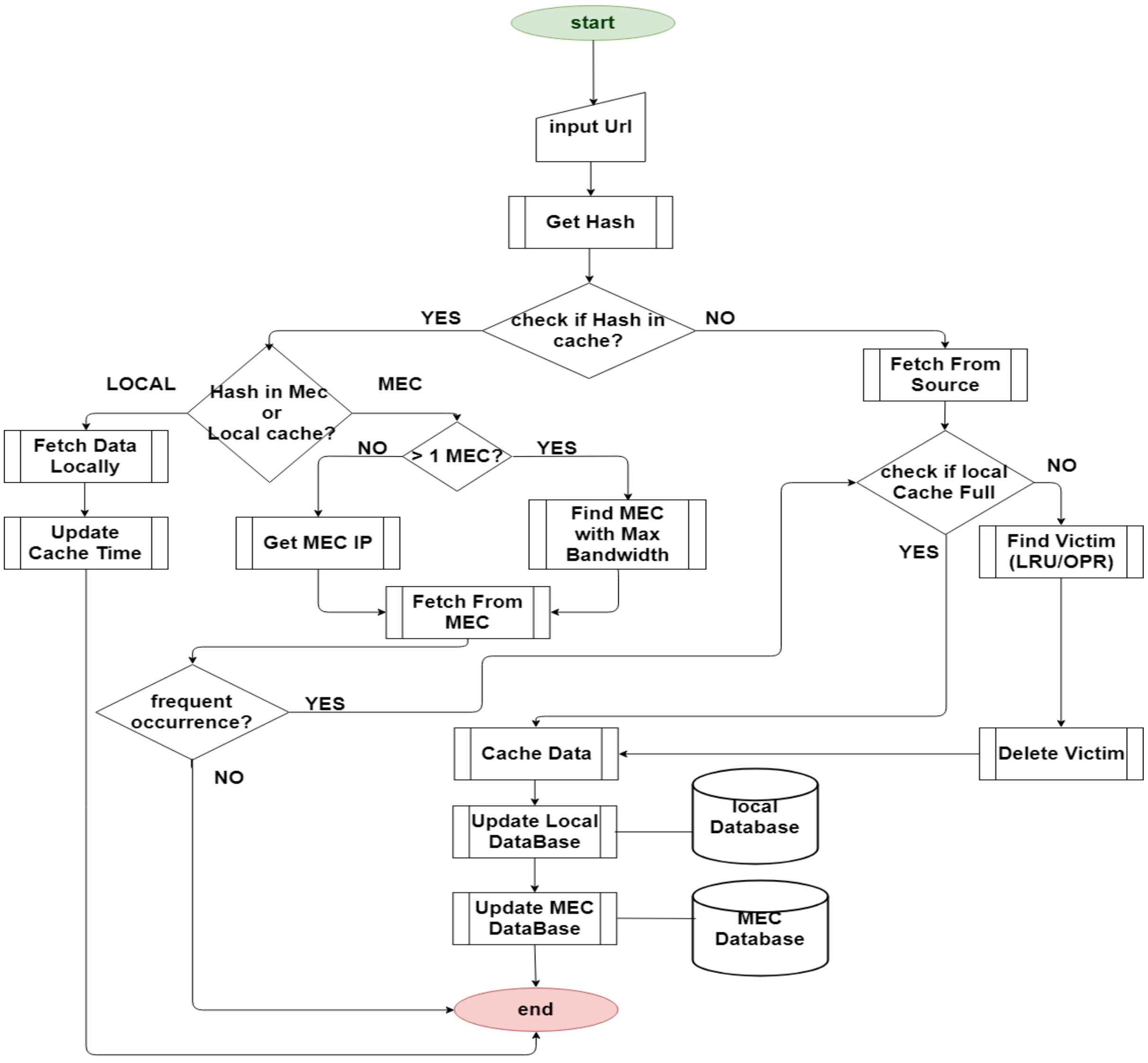

*Figure 35 Algorithm Workflow*

In this subsection, the algorithm workflow is explained. This is a series of steps, processes, and decisions that the algorithm is made up of and how they are linked together to achieve the goal of the algorithm.

Here, a web browsing case study has been explored. On entering a URL, a *Get Hash* process is called. The *Get Hash* process computes a unique id $h_i$ (using an MD5 hash) that identifies the request. The algorithm now checks if $h_i$ is in the cache. If not in the cache, the requested data are fetched from the source. Next, a process is called to determine if the cache is full or not. If the cache is not full, $h_i$ is cached and then recorded both on the local database and in the databases of other MEC's in the cluster.

If the hash $h_i$ is in the cache, another process is then called to determine if the hash is located either locally on the MEC node or in another MEC node. If the cache is located locally on the MEC node, the data are then fetched locally and served to the user. Whereas if the data are not on the local MEC but another MEC, a process is then called to determine if the data is located only on one MEC or more than one MEC. If $h_i$ is located only on one MEC, then the data is obtained from that MEC node and forwarded to the user. However, if the data are located on more than one MEC node, then a process is called to determine which of the MEC nodes have the maximum bandwidth connection on the link. The data are then fetched from that MEC. The next step is to check if the frequency of the fetched data is greater than the least frequency in the cache. If this is true, then the least frequently used data in the cache is replaced. Otherwise, the fetched data are forwarded to the user but not cached. After caching the data, the local database and the MEC database are then updated.

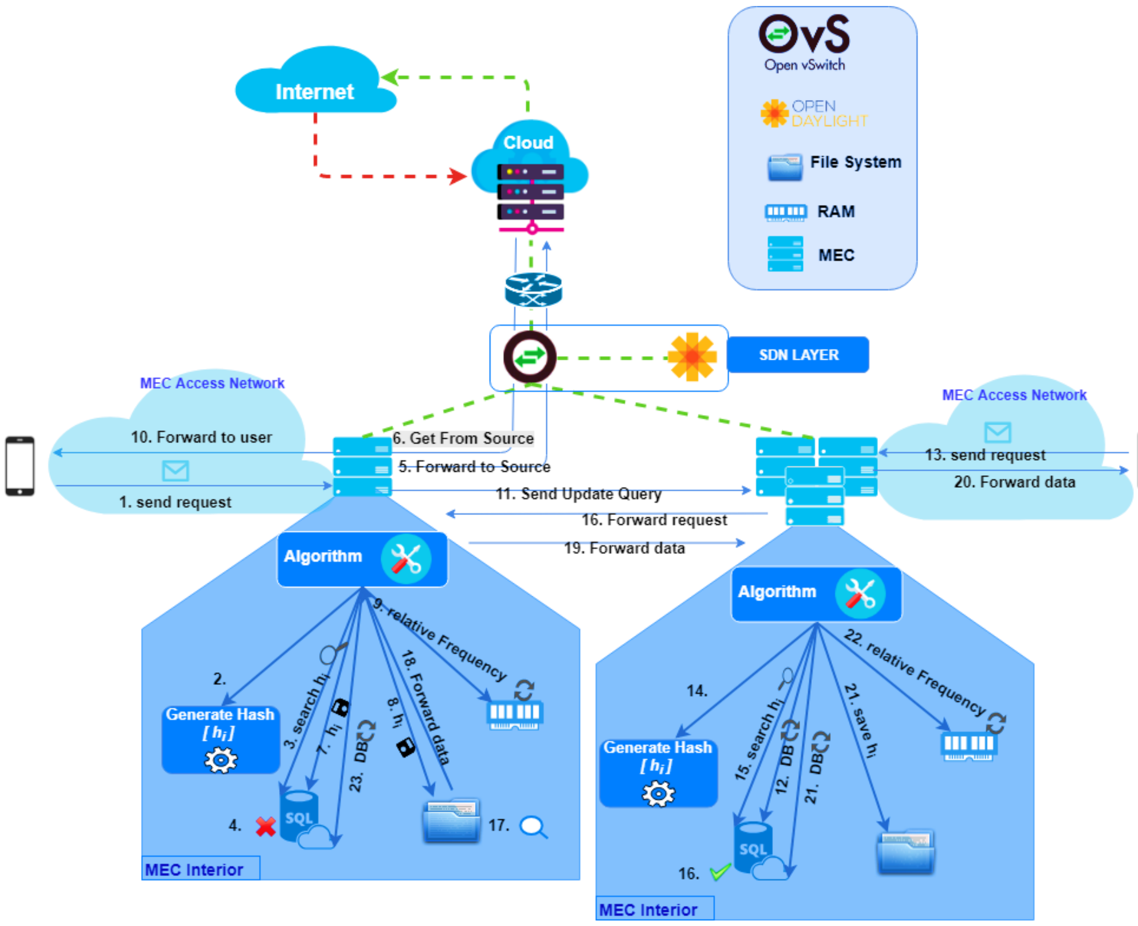

*Figure 36 System Design*

### 4.3.5.2 System design

In this subsection, the system design layout is presented and explained. The interactions between the elements of the system are also detailed and explained.

In Figure 36, the system design layout for the proposed algorithm. In the above diagram, the user is connected to the MEC node via the MEC access network. The MEC cluster is connected via an SDN network. The SDN layer is made up of an Open Vswitch which connects all the MECs and an SDN controller that controls the SDN network. In this scenario, an Open Daylight SDN controller is used in the architecture. The SDN layer is then connected to the cloud and the cloud is connected to the Internet.

From the diagram, the end-user first sends a request (1) from the end device to its closest MEC node. Each MEC node consists of the proposed algorithm, a SQL database and a file system.

Details about the cache data are stored in the SQL database. The actual cache data are stored in the file system while the relative frequency over a specified window size is stored in the RAM. On step (2), the algorithm generates a unique hash $h_i$ for the request using the Hash method. In step (3), the algorithm checks if the hash is in the database. This will return false because it is the first request (4). In step (5), the MEC node forwards the request to the source. The response is obtained in step (6). In step (7), details about the obtained data are recorded in the database. The data are saved in the file system in step (8), and the relative frequency is updated in step (9). In step (10), the response is forwarded to the user. In step (11), the MEC sends an update query to other MECs in the cluster to update their database and the said databases are updated in step (12). This completes one cycle.

Another cycle is started in step (13) when a user that is connected to another MEC node requests the same data. In step (14), a hash $h_i$ that uniquely identifies the data is generated. In step (15) the MEC node checks if it is in the database. In step (16) it returns true and the algorithm then forwards the request to the relevant MEC node that has the requested data. In step (17) the algorithm searches and obtains the data from the file system. In steps (18) and

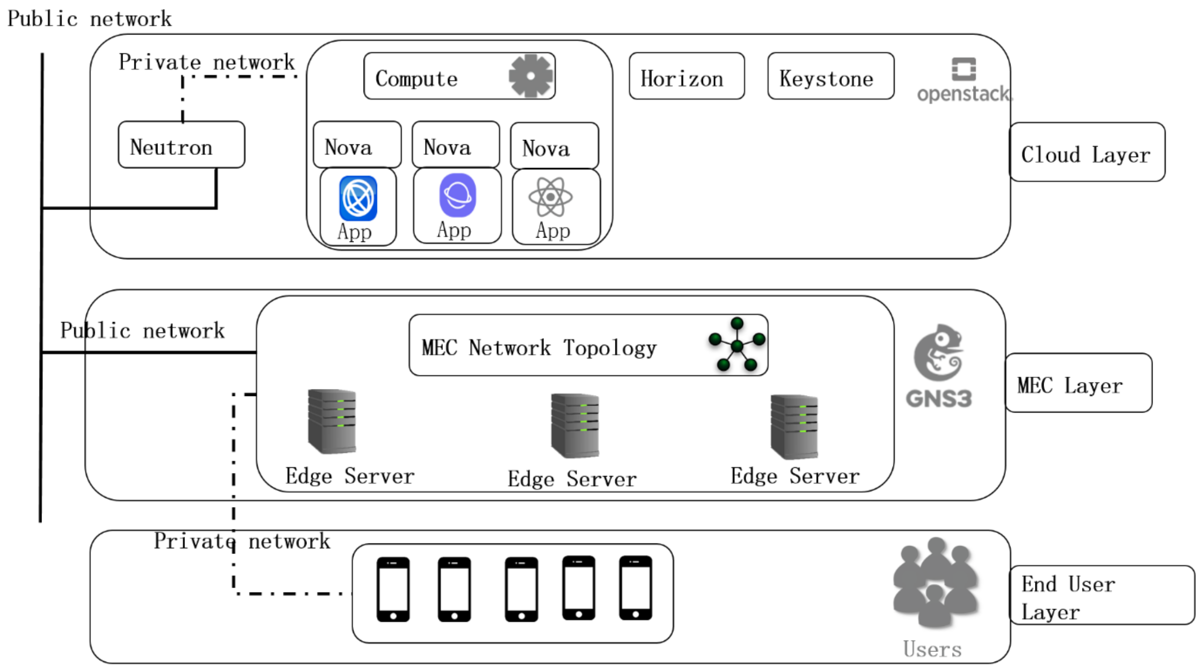

*FIGURE 37 Deployment Set-Up*

(19) the obtained data are then forwarded back to the relevant MEC and the MEC sends the response data to the user. Next, the data are saved in the file system and the database is updated in steps (21) and step (22) and, the relative frequency is updated. This completes another cycle. Notice that in this cycle no interaction has been made to the data source which is the cloud infrastructure.

## 4.3.6 Experimentation Testing

In this section, details are provided of how the proposed algorithm has been tested, which algorithm it has been compared to, and how the algorithm has been deployed. More detail of the experiment setup is available in appendix 2.

### 4.3.6.1 Experimental setup

In this subsection, the deployment set-up is presented. The components that make up the system and what tools and platform that was used are discussed.

The diagram in Figure 37, is made up of three layers, this consists of the Cloud layer, MEC layer, and End-user layer.

#### 4.3.6.1.1 Cloud Layer:

In this research, the OpenStack[1] cloud environment has been used as the cloud platform. OpenStack is an open-source software platform for cloud computing [194]. It provides an infrastructure as a service for users to deploy virtual entities over scalable resources. From FIGURE 37, the main OpenStack components for our test are Neutron, Compute, Horizon, and Keystone. The Neutron manages the virtual networking layout in an OpenStack environment providing a network as a service for vNICs (Virtual Network Interface Card) to Nova

[1] https://www.openstack.org/

instances located at the compute node. In the diagram, neutron provides a private network for the Nova instance to communicate with each other. It also connects the virtual domain to the external world via a public network. The OpenStack Compute provides a platform for virtual machines (Nova instances) to be created. The software applications are deployed inside the Nova instance and assigned the necessary computational resources that support required functionalities. Here, a web application for the proposed test model has been deployed. The horizon provides a GUI dashboard that enables users to monitor and control the virtual machines as well as many other configurations and KPI indicators. OpenStack Keystone provides OpenStack with the identity service for authentication and high-level authorization.

*TABLE 9 SPECIFICATION OF CONTAINERIZED MEC USING DOCKER*

| | |
|---|---|
| Operating System | Ubuntu 16.04 Xenial |
| Architecture | x86_64 |
| CPU(s) | 8 |
| Thread(s) per core | 1 |
| Model name | Intel(R) Xeon(R) |
| CPU (GHz) | 2.6 |
| Total memory (GB) | 2 |

#### 4.3.6.1.2 MEC Layer

GNS3(Graphical Network Simulator- 3)[1] has been used to emulate the MEC cluster. GNS3 is a free software used to emulate complex networks [195]. In GNS3, a cluster of the MEC network has been created. The MEC cluster nodes are interconnected using overlaid networking provided by Open Daylight[2] [196] that also acts as software-defined networking (SDN) controller for the testing platform. Each MEC node is a lightweight docker[3] container.

[1] https://www.gns3.com/
[2] https://www.opendaylight.org/
[3] https://www.docker.com/

A docker container is a lightweight, standalone, executable package of software that has operating system-level virtualization and includes everything needed to run an application. The system specification of the docker container is listed in TABLE 9. The MEC layer is connected to the internet via the public network. A private network is also created within the GNS3 with which the MECs are connected. Quagga [197] is installed on the MEC docker containers and configured to serve as an access point for end-user devices. The proposed algorithm is written in python and deployed as a Docker application in the MEC Docker container.

*TABLE 10 CACHING TEST SPECIFICATION*

| **Specification** | **Amount** |
|---|---|
| Number of MECs in the cluster | 3 |
| Number of requests received per MEC | 500 |
| Cache size of each MEC | 4 |
| Number of content items | 20 |

#### 4.3.6.1.3 End-user layer

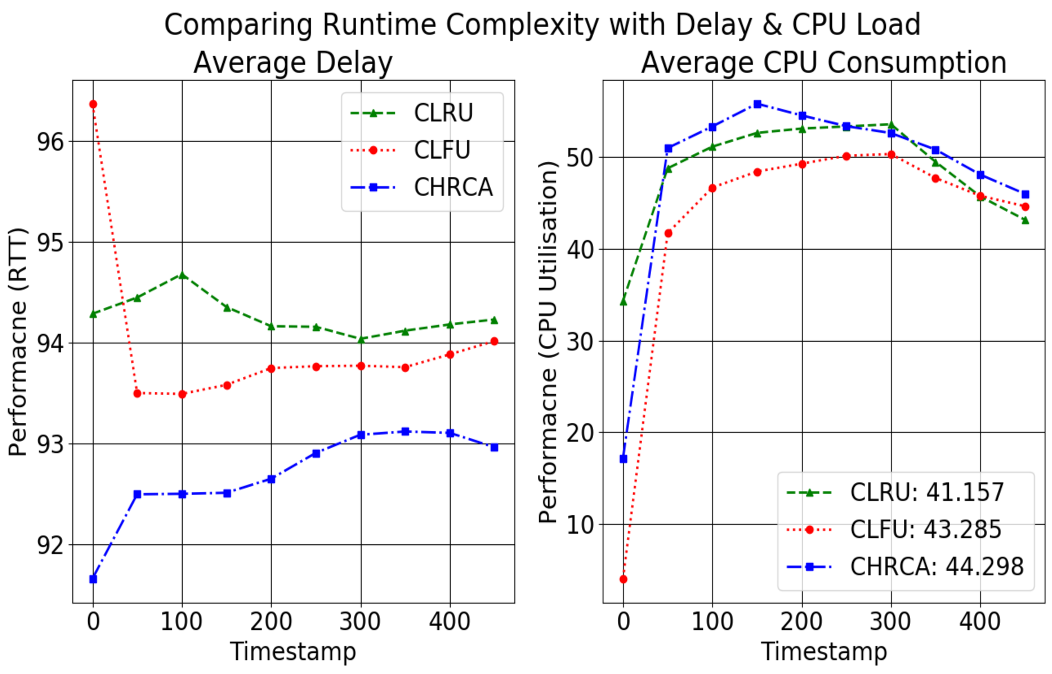

*Figure 38 Lower RTT Implies Reduced Access Time Due To High Local Hits. Less CPU Consumption Implies Computational Efficiency. CHRCA Meets Both Criteria Hence Its Fast And Efficient*

This layer is the final layer and it is made up of the end-user devices. To simulate the users, it is assumed that the sample space of the generated requests uses Gaussian distribution.

#### 4.3.6.2 Experiment Procedure

In this section, performance comparisons are made between the proposed algorithm and two case study algorithms. The first case study algorithm is a combination of Co-operative caching and Least Recently Used (CLRU). LRU algorithm chooses its replacement victim based on which cache reference has been unused for the longest time [138]. While the second case study algorithm is a combination of Co-operative caching and Least Frequently Used (CLFU). The caching test specifications are summarized in Table 10.

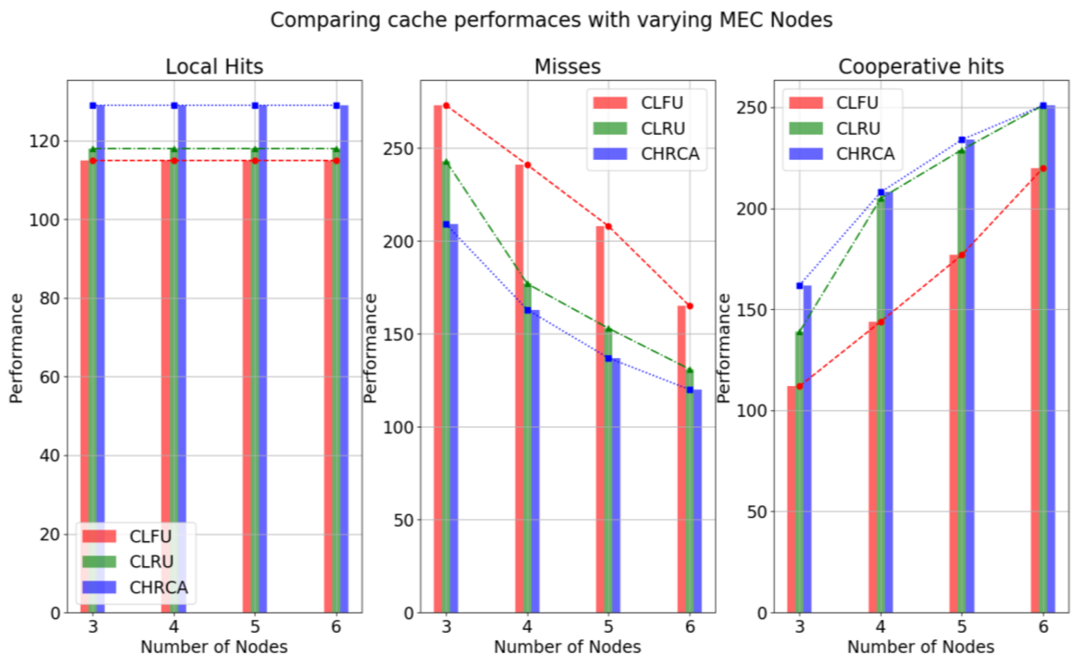

*Figure 39 Number Of Nodes Is Proportional To The Cache Performance. However, CHRCA Shows A Better Convergence (Higher Hits And Lower Misses) Compared To CLFU And CLRU*

#### 4.3.6.3 Experimental Results

In this section, the results obtained after experimentation and testing have been discussed. Here a comparison of the resource utilization which includes the CPU and RTT (this is the RTT between the MEC and the webserver) utilization is made for the Case study algorithm and the proposed algorithm. A comparison is also made for the cache performance which includes local cache hits, cache misses, and total cache hits.

#### 4.3.6.4 Resource utilization comparison

Figure 38 shows the CPU and RTT utilization of each algorithm over a period during the algorithm run time. From Figure 38, it can be deduced that the RTT utilization for all algorithms approximately lies between the same range which is 30ms to 60ms. It is also shown that the CPU Utilization of the proposed algorithm and CLFU stays below 40% for most of the time apart from certain peaks while in CLRU algorithm the CPU utilization stays above 40% most of the time apart from certain falls. It can also be seen that the peak for CPU utilization of all three algorithms is approximately the same. The CPU utilization drops mainly when there are cache hits and the data is fetched locally from the device or the MEC and not from the source. TABLE 11 shows the average resource utilization of each algorithm.

*TABLE 11 AVERAGE RESOURCE (CPU AND RTT) UTILIZATION*

| **Average** | **CLRU** | **CLFU** | **Proposed** |
|---|---|---|---|
| **Average CPU (%)** | 47.49667 | 37.57 | 37.17 |
| **Average RTT (ms)** | 45.4963 | 46.32049 | 41.56043 |

#### 4.3.6.5 Cache performance comparison

Figure 39 shows the local cache hits, cache misses, total cache hit (sum of the local cache hits and cache hit from MEC cluster) respectively of each algorithm over a period during the algorithm process. The delay incurred in using the replacement algorithm is high, therefore less use of the replacement algorithm means less delay in the system. Comparing the two graphs, it can be deduced that the proposed algorithm has a better overall performance as it generates more cache hits, fewer cache misses, more total cache hits, and less replacement.

### 4.3.7 REPLACEMENT ALGORITHM VALIDATION EXPERIMENT

In the previous section, experiments have been conducted to compare the cache performance of the proposed algorithm with varying number of MECs. This section aims a validating the replacement algorithm performance using a higher amount of request, varying cache size and request distribution. The hit ratio of the proposed algorithm is compared to existing algorithms. These algorithms are evaluated with varying Zipf-popularity distribution parameter ($\alpha$). Samples are drawn from a Zipf distribution [198] with specified parameter $\alpha$ > 1. The probability density for zipf distribution is expressed as below.

$$p(x) = \frac{x^{-\alpha}}{\zeta(\alpha)'}$$

where $\zeta$ is the Riemann Zeta function [198]. In this experiment, samples from the Zipf distribution have been generated using the *Numpy* library [199]. Seven algorithms have been evaluated including LFU, LRU, FIFO, MQ, FBR, and OPR. These algorithms have been implemented using *python*.

*TABLE 12 EXPERIMENTAL PARAMETERS*

| ***Parameters*** | ***Values*** |
|---|---|
| $\alpha$ | $\{1.1, 1.3, 1.5\}$ |
| $cache_{size}$ | $\{30,40,50\}$ |
| No of requests | 5000 |
| No of Experiments | 3 |
| No of content | 300 |

*TABLE 13 ALGORITHM PARAMETERS*

| ***Algo.*** | ***Parameter*** | ***Value*** |
|---|---|---|
| *FBR*[162] | $F_{new}$ | 30% |
| | $F_{old}$ | 30% |
| | $A_{max}$ | 100 |
| | $C_{max}$ | 11 |
| *MQ*[159] | $Q_{out}$ *size* | $4 * cache_{size}$ |
| | *No of LRU Q* | 8 |
| *CHRCA* | $d$ | 3 |
| | $n$ | 800 |

The experimental parameters used are displayed in TABLE 12 and the corresponding algorithm parameters are outlined in TABLE 13

### 4.3.7.1 Experiment Results

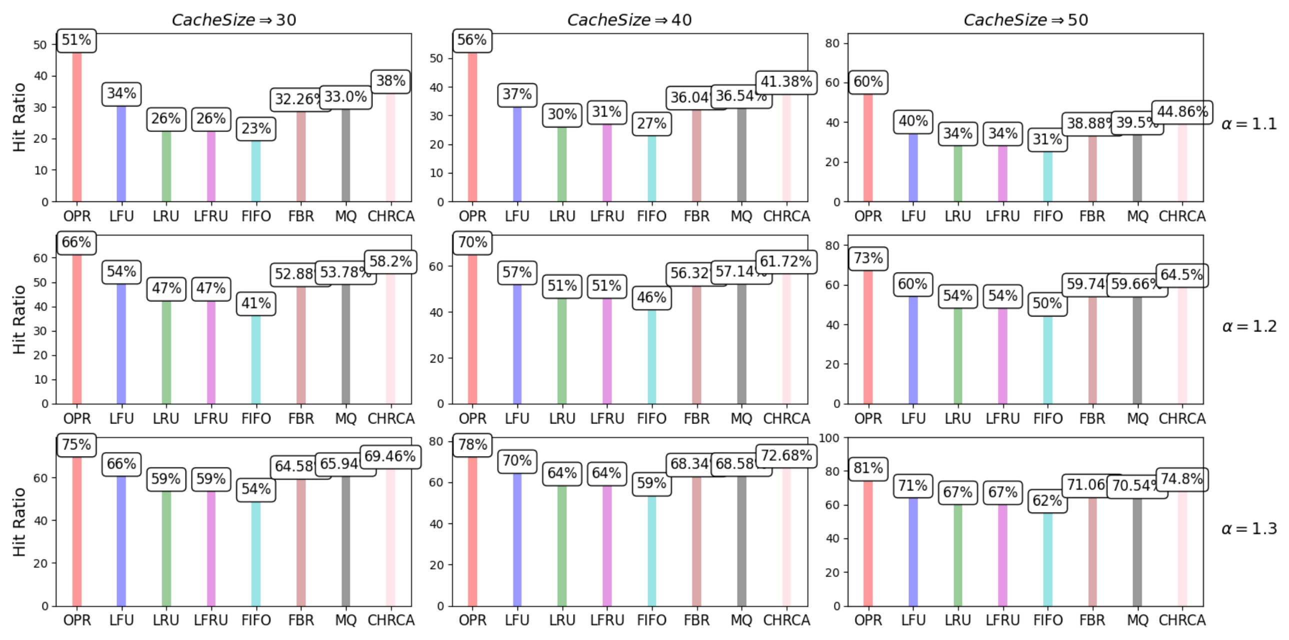

*Figure 40 CHRCA Validation Experimental Results with Varying Cache size and Zipf parameter*

Figure 40 depicts the result obtained from the experiment conducted. It displays a comparison of the resulting hit ratio percentage achieved using varying cache sizes and Zipf parameter.

The proposed algorithm CHRCA attains a 4% lead in hit ratio when compared to other algorithms. However, it does not quite perform as well as OPR. This is because the prediction obtained is an estimation and not 100% accurate. The average prediction accuracy achieved during the experiments is 95% using a polynomial degree of 3 and 800 sample size. The least performing algorithm is the FIFO. Zipf $\alpha$ of 1.1, 1.2 and 1.3, have been used to get a suitable variation of requests to gauge the performance of each algorithm. An increase in $\alpha$ leads to an increase in the hit ratio due to lesser variation in the request distribution. A continuous increase in the $\alpha$ will get to a point where each algorithm almost achieves 100% hit ratio. Hence, not suitable to effectively gauge the performance of the algorithm.

## 4.4 A Novel Predictive-Collaborative-Replacement (PCR) Algorithm for MECs

In the previous section, a predictive caching scheme has been explored that utilizes Lagrange extrapolation for prediction. To improve the prediction accuracy a new predictive caching scheme has been presented in this chapter.

Employing an appropriate caching algorithm is crucial to increase the overall QoE in content distribution systems as a 1% increase in hit ratio can have a very positive impact. The conventional caching algorithms such as FIFO [104], LRU [129], LFU [159], LFRU [161] and their variants [160] [162] [163] [159] follow very specific rules. Therefore, these algorithms alone cannot adapt to the ever-caching user request patterns. Following the increasing popularity of machine learning and data analytics, progress has been made on prediction based caching algorithms [183] [184] [185] [200]. Most of these algorithms use machine-learning schemes that involve data preparation and feature extraction, model training, and finally cache replacement using trained model. The model training is usually very time consuming and therefore mostly done offline. However, once the model is trained with appropriate hyperparameters and adequate feature engineering, it can achieve very high hit ratio. The downside is that the model created is very dependent on the data used in training and hence not very adaptive.

In this chapter, utilizing the capabilities of the MEC, the aforementioned concerns of the caching algorithms have been addressed by proposing a three-fold algorithm solution to improve the cache hit ratio, and access delay. This includes a novel delay-aware replacement caching algorithm that can find a victim in $O(1)$, a proactive online association-based caching strategy that prefetches cache objects based on anticipated user behaviour, and finally a MEC collaborative caching algorithm. Simulation results show that the proposed algorithm outperforms conventional algorithms with regards to hit ratio and experimental results show that it outperforms an offline caching algorithm with a pre-trained model.

### 4.4.1 System Model

Let's consider a typical system architecture as shown in Figure 41. Here, there is a set of MECs in a cluster that defines a collaborative space to support the core network. Let $\mathbb{C} = \{C_1, C_2, C_3 .. C_n\}$ denote a set of collaborative spaces. Each collaborative space contains a set of MEC server $C_i = \{M_1, M_2, ... M_n\}$. The MEC is co-located with the base station to provide computational and caching resources. It is assumed that the collaborative space is defined by the network administrator based on the hop count distance between the MEC nodes [170]. This is to reduce the communication delay within the collaborative space and reduce the communication overhead. The users are connected to the MEC in the collaborative space closest to them. The MEC maintains a disjoint one-to-many cardinality with the UE. Here an edge node is connected to many UE, but no UE is connected to multiple MECs. $U_i = \{u_1, u_2, .. u_n\}$ is denoted as a finite non-empty set of UEs connected to a MEC $M_i$. Each $u_i$ request data $r_i$ to be retrieved through the MEC where $r_i \in R$. $R = \{r_1, r_2 .., r_n\}$ is a finite non-empty set of data that can be retrieved from $\mathbb{C}$ *or* $\mathcal{H}$, where $\mathcal{H}$ denotes the cloud platform. Requests that cannot be retrieved from $\mathbb{C}$ is sent to $\mathcal{H}$ through the core network.

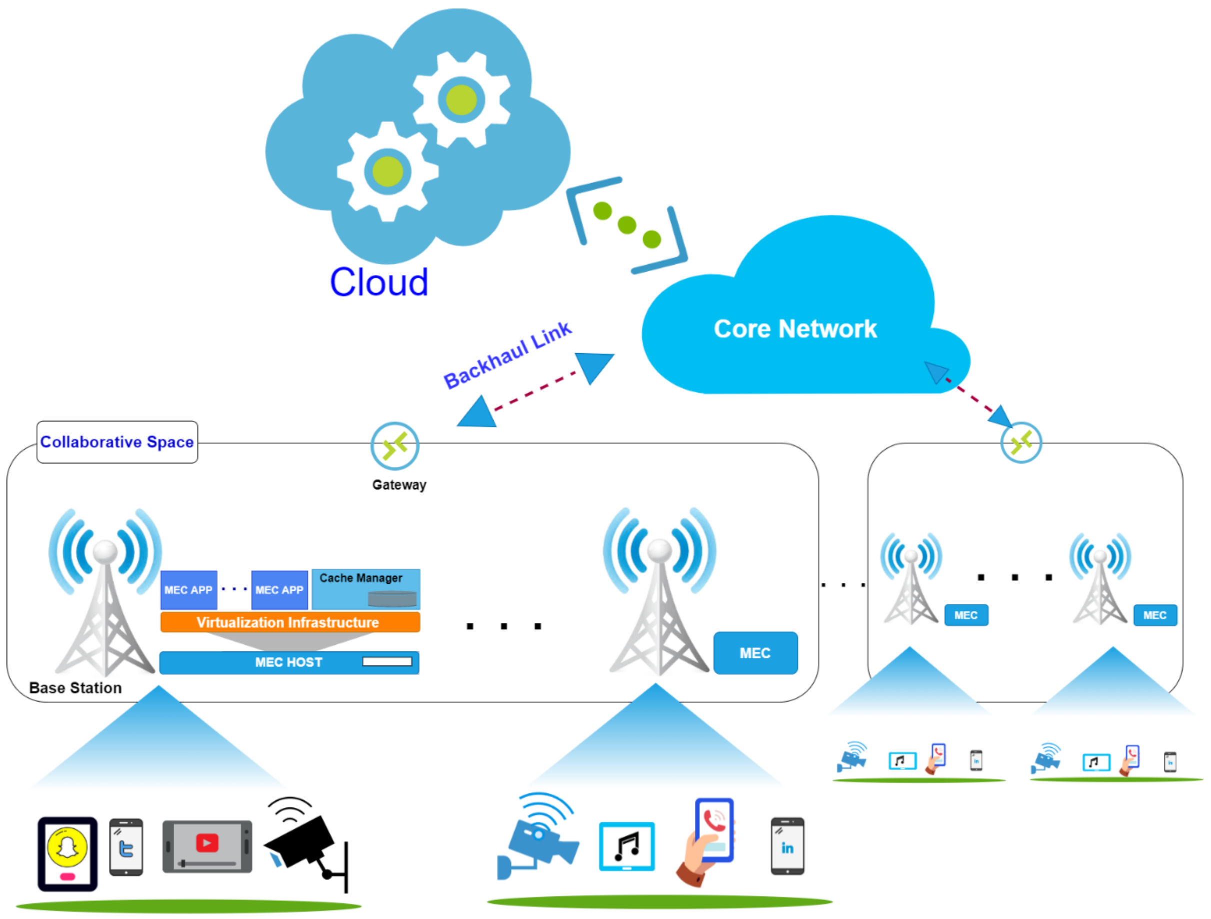

*Figure 41 System Architecture for MEC Collaborative Content Caching*

### 4.4.2 Proposed Algorithms (PCR)

Podlipnig et al [201] has stated that a good caching algorithm must consider 4 factors. The frequency of the cache object, the recency, the size of the cache object, and finally the cost of retrieving the cache object. This is of no surprise as prior to this, Zhou et al [159] have also listed 3 factors a good cache algorithm must-have. Two of them (frequency and recency) have already been mentioned. The third factor is the temporal frequency. This is to solve the problem of cache objects that have been frequently accessed in the past, obtains a very high frequency, and never replaced even if it is no longer popular. An algorithm that addresses these problems is proposed in the content caching subsection. Additionally, a proactive predictive caching scheme is proposed that learns the user's request pattern, anticipates requests, and prefetches the objects. Finally, a collaborative algorithm is proposed for the effective utilization of the global MEC cache storage. In the following sections, the details of the proposed schemes are outlined.

The proposed caching framework is depicted in Figure 42. Following the numbered items in the diagram, the algorithm is initialized with a new user request. If the requested cache object is not in the local cache, then the *MEC Collaborative Scheme* is used to retrieve the object. If it is not in the collaborative cache, then it is retrieved from the content server. The *Replacement Scheme* handles the identification and eviction of the cache victim when the cache is full. Finally, the *Cache Prediction Scheme* generates cache sequential association rules based on the received request patterns. Therefore, when there is a match based on the rules generated, the cache objects are prefetched and stored in the local cache. These three schemes are explained in detail in the following sections.

#### 4.4.2.1 Replacement Scheme: Selective Historic Least Frequently Used (SHLFRU)

The rationale of the proposed content caching algorithm is to design and develop an efficient caching algorithm with competitive time complexity. The basis of SHLFRU is in the combination of LRU and LFU algorithms, in which the least frequent and least recent cache object is replaced. However, two enhancements have been made to this algorithm.

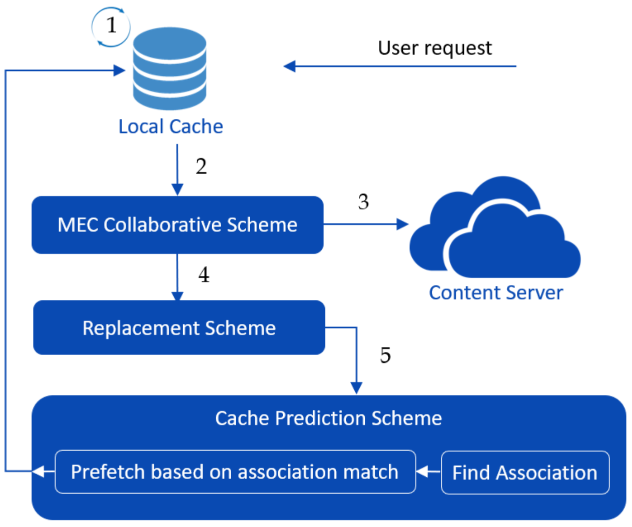

*Figure 42 PCR Caching Framework*

- **Selective caching:** The problem of *cold* cache objects, which are requested once and not requested again for a very long time and therefore not useful of being the cache-store is addressed. To solve this, a selective caching approach is used where the requested cache object $r_i$ is only cached on either of two conditions. The first condition is based on temporal locality thus cache objects with a higher temporal locality have more priority. This has been achieved using a history queue. The second condition prioritises cache objects with higher retrieval costs. This is to reduce the user access cost. The two conditions are represented as in equations (72) and equation (73) below.

$$r_i^{(R-1)} \leq r_j^{(R-1)} \, \forall \, r_i \in H \tag{72}$$

$$r_i^c > r_j^c \tag{73}$$

Here, $H$ is the history queue, $r_i^{(R-1)}$ is the previous recency of the new cache object $r_i$ and $r_j^{(R-1)}$ is the previous recency of the least frequently and recently used object. $r_i^c$ is the cost of retrieval of $r_i$ and $r_j^c$ is the cost of retrieval of the $r_j$.

$$r_i^c = \frac{size}{t_{rate}} \tag{74}$$

Hence, $size$ is the size of $r_i^c$ in bits and $t_{rate}$ is the transmission rate. $r_i^c$ is taken to be the miss penalty, which is the delay in retrieving the cache object if there is a cache miss. The history queue $H$ is a FIFO queue of a finite size $h_{size}$ . It keeps the details of recently evicted blocks but not the cache data.

The selective caching decision is only activated when the MEC cache store $M_i^{store}$ is full.

$$|M_i^{store}| \geq cache_{size} \tag{75}$$

If either eq. (72) or eq. (73) is true, then $r_i$ is removed from $H$ and pushed into $M_i^{store}$.

- **Temporal Frequency:** To address the problem of temporal frequency, a frequency count bounding strategy is used. In this approach, the maximum distance between two consecutive cache objects $r_j$ and $r_i$ in an LRU queue are grouped by their frequency count that is bounded by $f_{max}$. With this approach and $r_i$ being the lead cache object in $M_i^{store}$, the increment of the frequency count of a cache object $r_i^f$ is governed by function $FreqCount(r_i^f)$ stated in the equation below.

$$FreqCount(r_i^f) = \begin{cases} r_i^f + 1, & r_i^f - r_j^f < f_{max} \\ r_i^f, & r_i^f - r_j^f \geq f_{max} \end{cases} \tag{76}$$

Additionally, to avoid the problem of overflow associated with the practical implementation of frequency counts, a similar technique used in FBR [162] is applied. In this approach, the sum of all the frequency count $f_{sum}$ is dynamically maintained. Therefore, every frequency count in $M_i^{store}$ is reduced whenever the following condition occurs.

$$\frac{f_{sum}}{|M_i^{store}|} \geq A_{max} \tag{77}$$

$A_{max}$ is a predefined maximum value which is a parameter of the algorithm. Here, small $A_{max}$ means high-frequency updates. The frequency count of the cache objects in $H$ and $M_i^{store}$ is reduced using the following equation.

$$\left\lceil \frac{r_i^f}{2} \right\rceil \forall\, r_i \in \{M_i^{store}, H\} \tag{78}$$

Using this approach, in a steady state, $f_{sum}$ would lie between $|M_i^{store}| * (\frac{A_{max}}{2})$ and $|M_i^{store}| * A_{max}$. Note that in this reduction a count of one will remain at one, a count of two would be two, a count of three would be two, etc.

The *SHLFRU* algorithm uses multiple LRU queues $\mathbb{Q}$ to achieve LRFU where each LRU queue $Q_i$ contains cache objects with the same frequency count.

$$\mathbb{Q} = \{Q_1, Q_2 \ldots, Q_n\} \; s.t \; \forall \, r_i \in Q_i, r_i^f = i \qquad (79)$$

The reference to the LRU queue that contains the cache objects with the least frequency $Q_L$ is dynamically maintained. Therefore, when a cache needs to be replaced the victim is the cache object in the tail of $Q_L$.

#### 4.4.2.1.1 Time complexity

Maintaining an LRU queue requires a tail insertion/head taking and incurs no overhead. Since $Q_L$ is maintained, a heap data structure is not required to keep the LFU stored. Hence, the replacement victim can be found in constant time. If $Q_L$ changes a maximum of $f_{max}$ queries are required to find the next $Q_L$. $f_{max}$ is a constant that does not depend on the scales. In all, the time complexity of *SHLFRU* is $O(1)$.

Taking these updates into consideration, the proposed algorithm *SHLFRU* in algorithm II.

*ALGORITHM II SHLFRU*

```
Input: $cache_{size}, A_{max}, f_{max}, h_{size}$
Output: None
Initialization:
$M_i^{store} \leftarrow \phi$
$H \leftarrow \phi$
$cache_{decision} \leftarrow True$
/* Procedure to be invoked upon reference to cache object $r_i$ */
1.  if $r_i$ in $M_i^{store}$ then
2.      $Q_i.pop(r_i)$
3.  Else do
4.      $D_1 \leftarrow$ do $eq.$ (72)
5.      $D_2 \leftarrow$ do $eq.$ (73)
6.      if $D_1$ or $D_2$ is True then
7.          $Victim \longleftarrow Q_L.pop()$
8.          $H.push(Victim)$
9.      Else if $r_i$ not in $H$
10.         $H.push(r_i)$
11.         $cache_{decision} \leftarrow False$
12.     Else do
13.         update $r_i^{recency}$
14.         $cache_{decision} \leftarrow False$
15.     end
16. end
17. $k \leftarrow FreqCount(r_i^f)$
18. if $cache_{decision}$ then
19.     if $Q_k$ not in $M_i^{store}$ then
20.         $Q_k \leftarrow \phi$
21.     $Q_k.push(r_i)$
22. end
23. if $eq.(6)$ is True then
24.     do $eq.(7)$
25. end
```

#### 4.4.2.1.2 Simulation Experiments

To evaluate the efficiency of the proposed content caching algorithm, SHLFRU has been compared with existing algorithms. These algorithms have been implemented using *Python*. The simulation implementation project is available on *GitHub* [202]. The simulation platform is also available online [203] for researchers to benchmark their algorithms with existing schemes. More detail of the simulation tool is available in appendix 3. The purpose of the simulation is to evaluate the hit ratio of SHLFRU compared to existing algorithms. These algorithms are evaluated with varying Zipf-popularity distribution parameter ($\alpha$) and

$cache_{size}$. Samples are drawn from a Zipf distribution [198] with specified parameter $\alpha > 1$. The probability density for zipf distribution is expressed as below. where $\zeta$ is the Riemann

$$p(x) = \frac{x^{-\alpha}}{\zeta(\alpha)'}$$

Zeta function [198]. In this experiment, samples from the Zipf distribution have been generated using the *Numpy* library [199]. The experimental parameters are summarized in TABLE 14. Seven algorithms have been evaluated including LFU, LRU, FIFO, MQ, FBR, and OPT. SHLFRU, FBR, and MQ require additional parameters. The algorithm parameters used during the experimentation for this algorithm are summarized in TABLE 15.

*TABLE 14 EXPERIMENTAL PARAMETERS*

| **Parameters** | **Values** |
|---|---|
| $\alpha$ | $\{1.01, 1.20, 1.35\}$ |
| $cache_{size}$ | $\{10,20,30\}$ |
| No of requests | 5000 |
| No of Experiments | 3 |
| No of content | 100 |

*TABLE 15 ALGORITHM PARAMETERS*

| ***Algo.*** | ***Parameter*** | ***Value*** |
|---|---|---|
| *SHLFRU* | $A_{max}$ | 100 |
| | $f_{max}$ | 10 |
| | $h_{size}$ | $4 * cache_{size}$ |
| *FBR*[162] | $F_{new}$ | 30% |
| | $F_{old}$ | 30% |
| | $A_{max}$ | 100 |
| | $C_{max}$ | 11 |
| *MQ*[159] | $Q_{out}$ *size* | $4 * cache_{size}$ |
| | *No of LRU Q* | 8 |

#### 4.4.2.1.3 Results

The results obtained from the simulation are displayed in Figure 43. It can be deduced that

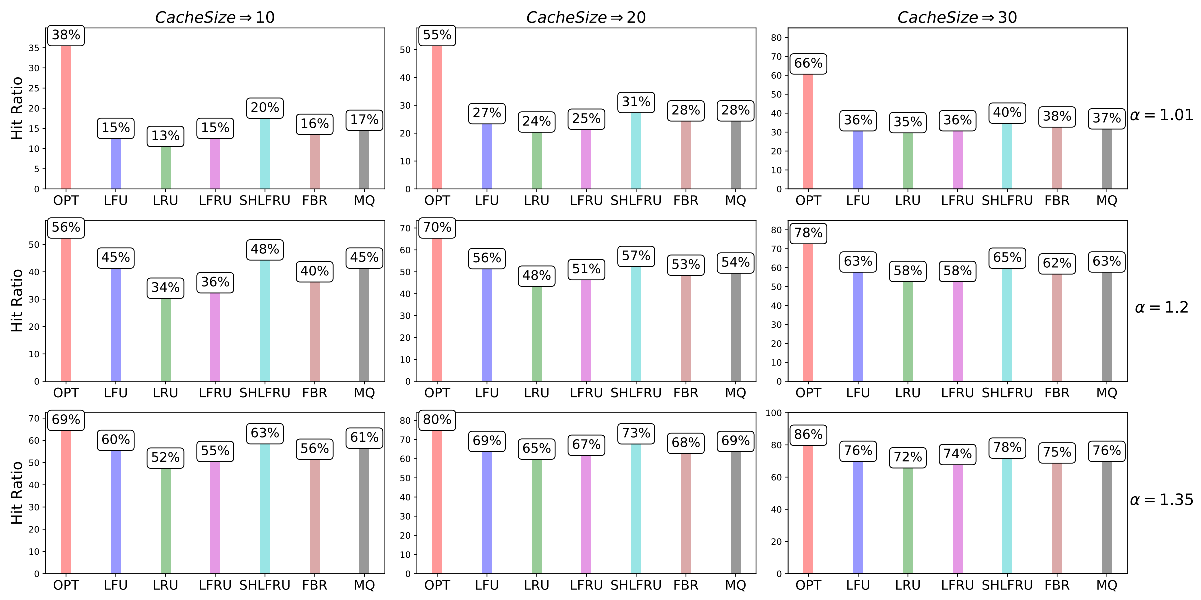

*Figure 43 Simulation Result of The Comparison Of Algorithms With Varying Zipf Parameter (α) And Cache Size*

SHLFRU performs better than the compared algorithms. Its performance is robust for different workloads and cache sizes. It can also be seen that MQ performs better than other algorithms apart from SHLRFU. SHLRFU maintains at least a 3% (3% of 5000 requests is 1500) improvement in hit ratio compared to MQ across the experiment. This improved performance can be attributed to the selective caching of SHLRFU and its ability to quickly identify *cold* cache objects. FBR performs almost as well as MQ across the simulation results. LFU surprisingly performs better than LRU across the simulation results.

#### 4.4.2.2 Cache Prediction Scheme: Proactive Prefetch caching algorithm (PPCA)

The cache store would be efficiently managed if the users' future request pattern is known. This is proved from the simulation results obtained in Figure 43, as OPT performs way better than other algorithms. The best way to predict the future is to study history. The motivation of this proactive caching algorithm is to use a predictive caching strategy based on learning the association patterns between content requests in a MEC environment where the content popularity is time-varying and unknown. Association rule mining techniques are leveraged to identify content requests with close relations.

Here, for a given MEC $M_i$ and time $t_n$, request history $Req$, from time $t_{n-k}$ to $t_n$ is utilized to generate rules ($Rules$) that maps antecedents $rule^{ant}$ to consequents $rule^{cons}$. This approach is employed rather than making predictions for $t_{n+1}$. It is assumed that certain content requests $con_i$ and $con_j$ are often requested together or sequential by users, where $con_i$ and $con_j$ are sets of variable lengths. Therefore, the aim is to classify $con_i$ and $con_j$ as either $rule^{ant}$ or $rule^{cons}$ such that

$$con_i \rightarrow con_j \neq con_j \rightarrow con_i \tag{80}$$

Let $S$ be the FIFO request sequence with length $m$ which has been received by a MEC $M_i$.

$$S = \{r_1, r_2 .. r_m\} \tag{81}$$

This problem is classified as an association sequential pattern mining. Here the order is maintained but no duplicate items may appear in the sequence. To reduce the processing time of the algorithm, two pruning techniques are employed. First, the dimension of the request sequence to be mined $s_i$ is bounded by $k$.

$$s_i = \{r_1, r_2 \dots r_k\}\ s.t\ \ s_i \subseteq S \tag{82}$$

$k$ is dynamically obtained from the number of unique elements $u_{no}$ for a given window size $w_{size}$, such that the sequence bounded by the window size $s_{ws}$ is a subset of $S$. Therefore, $k$ is deduced from the following equation.

$$k = (u_{no})^2 \tag{83}$$

This is to have enough training data that can generate meaningful insights. A similar approach has been used by [185]. In this regard, the training dataset $D$ is a matrix obtained from $S$ with a dimension of $u_{no} * u_{no}$. Therefore, for the association mining to be performed, the following condition must be met.

$$k \leq m \tag{84}$$

Secondly, the minimum support threshold $support_{min}$ is used to reduce the number of itemsets to be evaluated as candidates during the association mining. The support of a given sequence set $con_i \; s.t \; con_i \subset D$ is the number of rows $n_{row}$ in $D$ that contain $con_i$.

$$support(con_i) = \frac{n_{row}}{|D|} \tag{85}$$

Therefore, given $support_{min}$ a set $S_{con}$ that contains all $con_i$ and satisfies $support(con_i) \geq support_{min}$ is sort after.

$$S_{con} = \{con_1, con_2 .. con_n\} \; \forall \; support(con_i) \geq support_{min} \tag{86}$$

Using $S_{con}$, $Rules_{all}$ is generated that contains rules with antecedents which are a subset of $S_{con}$.

$$Rules_{all} = \{rule_1, rule_2 .. rule_n\} \; \forall \; rule_i^{ant} \subseteq S_{con} \tag{87}$$

The generated rules $Rule_{all}$ are then ranked to evaluate the strongest rules using the rule support ($Rule_{support}$) and rule confidence ($Rule_{confidence}$) [204]. $Rule_{confidence}$ of $(con_i \rightarrow con_j)$ is the proportion of transactions in $D$ including both $con_i$ and $con_j$.

$$Rule_{confidence}\left(con_i \rightarrow con_j\right) = \frac{Support\left(con_i \cup con_j\right)}{Support(con_i)} \tag{88}$$

Given the sorted ranked rules $Rule_{ranked}$, the top $p$ rules are selected to be used for prefetch caching.

$$Rule_{output} = \left\{rule_1, rule_2 \ldots rule_p\right\} \; \forall \; rule_i \in Rule_{ranked} \tag{89}$$

$Rule_{output}$ is updated every $Rule_{timeout}$ to ensure that the most relevant rules are stored. After the completion of each user request, the $rule^{cons}$ of the $rule^{ant}$ that matches $Req((t-l) \rightarrow t)$

is prefetched. If there is no match, no prefetch is done. Here, $t$ is the current position in $Req$ and $l$ is the number of elements in $rule^{ant}$. Example if $Req = [r_1, r_2, r_3, r_4, r_5]$, and $l = 2$. Therefore, $Req((t - l) \to t) = Req\big((5 - 2) \to 5\big) = Req(3 \to 5) = [r_4, r_5]$.To ensure quick lookup, the rules in $Rule_{output}$ is stored in a hash table with the key being the number of elements in $rule^{ant}$ and the value is also a hash table with the key being the $rule^{ant}$ and the value being the $rule^{cons}$. The number of rules in the $Rule_{output}$ is bounded by $Rule_{output}^{max}$. This is the maximum number of elements in the $rule^{ant}$ that can be stored in $Rule_{output}$. Thus, the maximum look-up done to find matches is $Rule_{output}^{max}$.

Two major algorithms can be used for association rule mining which are Apriori [205] and FP-Growth [206]. Apriori generates better association with sparse datasets but it is memory intensive due to its breadth-first approach. FP-Growth is quicker and uses a depth-first approach. However, FP-Growth does not do well with sparse datasets [207]. The best of both worlds has been combined by employing both algorithms and then choosing which one to use at runtime based on the sparse density and a given memory threshold $mem_{max}$. In this approach, the default algorithm is FP-Growth. However, the average memory utilized $mem_{avg}$when updating $Rule_{output}$ are stored. Therefore, Apriori is used if the following equation is satisfied, where $M_i^{mem}$ is the current memory utilization of the MEC and $D^{density}$ is the sparse density of the dataset $D$.

$$Apriori \mid M_i^{mem} < mem_{max} \;\forall\; D^{density} > 0.5 \qquad (90)$$

#### 4.4.2.2.1 Time complexity

The worse time complexity of using Apriori association mining [208] is $O(2^{u_{no}})$, where $u_{no}$ is the number of unique elements in $D$. This upper bound can only be reached when your threshold support is zero or support of every set of unique items is greater than threshold. Using eq (82) and (83), the time complexity is bounded by $k$. However, FP-Growth's time complexity is

$O(length\ of\ header\ table \times tree\ depth)$ [208]. Since a matrix of dimension $u_{no} \times u_{no}$ is used, the length of header table is $u_{no}$ and the maximum tree depth is $u_{no}$. Therefore, the time complexity is $O((u_{no})^2)$. This is significantly less than Apriori. The time complexity of ranking the rules generated is $O(|Rules_{all}| \log(|Rules_{all}|))$. Therefore, the total time complexity if Apriori is used is $O(2^k + |Rules_{all}| \log(|Rules_{all}|))$. However, total time complexity if FP-growth is used is $O((u_{no})^2 + |Rules_{all}| \log(|Rules_{all}|))$.

*ALGORITHM III PPCA*

```
Input: $S, w_{size}, m, support_{min}, mem_{max}, Rule_{output}^{max}, p$
Output: $Rule_{output}$
Initialization:
$Rule_{output} \leftarrow \emptyset$
1.From $s_{ws}$ obtain $u_{no}$
2.$k \leftarrow (u_{no})^2$
3.if $k \leq m$ then
4.    $s_i = \{r_1, r_2 \ldots r_k\}$ s.t $s_i \subseteq S$
5.    obtain $D$ from $s_i$ s.t dimension $= u_{no} * u_{no}$
6.    using eq.(15) obtain $S_{con}$
7.    if eq.(19) is False
8.        $Rule_{all} \leftarrow FPGrowth(D)$
9.    Else
10.       $Rule_{all} \leftarrow Apriori(D)$
11.   $Rule_{ranked} \leftarrow sort(Rule_{all})$
12.   $Rule_{output} \leftarrow slice(Rule_{ranked}, p)$
13.   end
14.end
```

#### 4.4.2.3 MEC Collaborative Scheme: Collaborative Greedy algorithm

The motivation behind this scheme is to manage efficiently the cache in the collaborative space by reducing data redundancy and increasing the sharing of cache data between MECs.

*ALGORITHM IV COLLABORATIVE GREEDY ALGORITHM*

```
Input: $w$
Output: $None$
Initialization:
$M_i^{names} \leftarrow \phi$
$M_i^{store} \leftarrow \phi$
$C_i^{store} \leftarrow \emptyset$
$TNM = w * |C_i|$
/* Procedure to be invoked upon reference to cache object $r_i$ */
1. $\textbf{if } r_i \in M_i^{store} \textbf{ not True}$
2.     $n_i \leftarrow M_i^{names}(r_i)$
3.     $q \leftarrow Cf(C_i^{store}, n_i)$
4.     $\textbf{if } q \textbf{ equals } 1 \textbf{ then}$
5.         $\textbf{if } |M_i^{n_i}| > 1 \textbf{ then}$
6.             $r_i \leftarrow Retrive(n_i, M_d)$
7.         $\textbf{else}$
8.             $r_i \leftarrow Retrive(n_i, M_j)$
9.         $\textbf{end}$
10.        $\textbf{if } |M_i^{n_i}| < TNM \textbf{ then}$
11.            $M_i^{store}.push(r_i)$
12.        $\textbf{end}$
13.    $\textbf{end}$
14. $\textbf{else}$
15.    $use\ Algorithm\ I\ to\ fetch\ r_i$
16.    $Send\ push\ request\ to\ nrs\ with\ r_i$
17.    $send\ cache\ update\ to\ C_i$
18. $Use\ Algorithm\ II\ to\ prefetch\ cache$
19. $\textbf{end}$
```

The aim here is to increase the efficiency of the global collaborative cache by improving the efficiency of the individual edge node. Let's assume content-centric networking for sharing cache data within the collaborative space. Therefore, contents are retrieved using a named identifier $n_i$. Additionally, let's assume a name resolution server $nrs$ are located in each collaborative space $C_i$. The contents in $nrs$ are populated by the MECs in $C_i$ after each content retrieval. To ensure security and integrity, the contents in the $nrs$ is stored in a distributed ledger similar to the blockchain. $M_i$ stores $M_i^{names} \mid M_i^{names} \subset nrs$ locally to improve efficiency. $M_i^{names}$ is a FIFO queue with a limited size. For simplicity, let's assume each MEC $M_i$ has homogeneous storage capacity and belongs to a collaborative space $C_i$. The total content stored in the collaborative store is denoted by $C_i^{store}$. Thus, $M_i^{store} \subset C_i^{store}$. The $C_i^{store}$ is populated by event-driven updates sent by MECs. Furthermore, a binary cache function $Cf(cache_{store}, r_i)$ that indicates if a cache object is available in a cache store has been defined.

$$Cf(cache_{store}, n_i) \in \{0,1\} \tag{91}$$

From eq. (*91*), 1 implies that $n_i$ is cached in a given $cache_{store}$ and 0 otherwise. If a named content is not stored in either $M_i^{store}$ or $C_i^{store}$, it is retrieved from the content provider in the cloud $\mathcal{H}$. However, if $n_i$ is in more than one MEC in $C_i$ then $n_i$ is retrieved from the MEC with the least network delay. Let's denote the network delay between two MECs $M_i\ and\ M_j$ as $M_{i\to j}^{delay}$. If $M_i^{n_i}$ denotes a set of MECs that have $n_i$ in the cache and $M_d$ is the MEC with the least network delay, the collaborative cache retrieve function $Retrive(n_i, M_d)$ is defined in eq. (*92*).

$$Retrive(n_i, M_d) \leftarrow \ \forall\, n_i \in M_i^{n_i}\ \ s.t\ \left|M_i^{n_i}\right| > 1 \tag{92}$$

To reduce cache redundancy, the number of MECs that can store $n_i$ is capped at $w$ percent. Therefore, the total number of MECs $TNM$ that can store $n_i$ is represented in eq. (*93*)

$$TNM = w * |C_i| \tag{93}$$

### 4.4.3 Experimentation

To evaluate the efficiency of the proposed algorithm PCR has been implemented in an emulation environment that consists of a varying number of MECs $\{6, 8, 10\}$ and a content server. The content server has been deployed on *Netlify* [209] and the MECs have been deployed in the GNS3 platform. Each MEC is a Linux Server using docker as its Virtualization infrastructure. Communication between the MECs is achieved using a messaging broker.

The proposed algorithm has been compared with a contemporary deep learning-based predictive edge caching algorithm [186]. The authors have used a RNN model, Long Short-Term Memory (LSTM) to create a model that can make decisions on what to cache on the edge based on popularity. LSTM[11] is an artificial recurrent neural network architecture used in the field of deep learning well suited for making decisions based on a time series dataset. Henceforth, this algorithm has been referred to as *C-LSTM*. *C-LSTM* has been trained using the MovieLens 20M dataset. Therefore, for fairness, the same dataset has been used for this experimental comparison. From the MovieLens 20M dataset, the focus was on the movie IDs that have been used in [186] and movie IDs in the range of 1 to 1070. After data preparation and filtering out movie Ids with little references, 444 models have been generated. The generated models have been then deployed on the MECs as per the author's specifications.

The algorithms have been implemented using $Python$ and the project is available on *Github* [202]. In the experimentation, it has been assumed that the arrival time of the user requests on the MEC server follows Poisson distribution $\lambda = 1 \mid \lambda \in \mathbb{N}$ . The parameters used for the experiment have been summarized in the table below. For simplicity, it has been assumed that $cache_{size}$ is a natural number $\mathbb{N}$. More detail of experiment setup is available in appendix 4.

[11] https://en.wikipedia.org/wiki/Long_short-term_memory

*TABLE 16 EXPERIMENTAL SETUP PARAMETERS*

| **General Setup** | | |
|---|---|---|
| **Parameters** | **Value** | **Meaning** |
| $\lvert C_i \rvert$ | $\{6, 8, 10\}$ | No of MEC in the collaborative space |
| $\lvert R \rvert$ | 20,000 | Total no of requests |
| $No\ of\ content$ | 1070 | No of unique content |
| $\lambda$ | 1 | Poisson parameter |
| **Algo 1 parameter** | | |
| **Parameters** | **Value** | **Meaning** |
| $A_{max}$ | 100 | Max average frequency |
| $f_{max}$ | 20 | frequency count bounding parameter |
| $cache_{size}$ | 50 | Cache size |
| $h_{size}$ | $4 * cache_{size}$ | History size |
| **Algo II parameter** | | |
| **Parameters** | **Value** | **Meaning** |
| $w_{size}$ | $cache_{size} * .5$ | Window size |
| $m$ | $(w_{size})^2$ | Length of history request $S$ |
| $support_{min}$ | 0.45 | Minimum support |
| $mem_{max}$ | 70% | Maximum memory threshold |
| $Rule_{output}^{max}$ | 4 | Max length $rule^{ant}$ |
| $p$ | 10% | No of the top rules selected |
| **Algo III parameter** | | |
| **Parameters** | **Value** | **Meaning** |
| $w$ | 10% | No of MEC that can store a cache |

#### 4.4.3.1 Experimental Results

In this section, the results obtained from the comparison of the two algorithms with respect to hit ratio, access delay, CPU, and memory utilization are discussed.

- **Hit Ratio:** Figure 44 shows the hit ratio comparison of PCR and C-LSTM with varying number of MECs. It can be seen that PCR achieves a better hit ratio than C-LSTM in all caches with at least a 25% increase. This high increase is due to the effective use of the collaborative cache and its selective caching approach. Additionally, this is also attributed to the efficient identification and replacement of cold cache objects and the ability to learn and predict cache association patterns.

- **Access Delay:** The comparison of access delay is depicted in Figure 44. It can be deduced that PCR obtains a lower access delay than C-LSTM. This is due to the increase in hit ratio as the access delay is dependent on the hit ratio. There is less access delay incurred if the user request is served from the local cache or MEC cache compared to obtaining the request from the content server. Therefore, a more hit ratio would lead to lesser access delay. This reduction would indirectly improve the QoS for the end-users and a step closer to achieving URLLC.
- **CPU Utilization:** The CPU utilization comparison is shown in Figure 44. C-LSTM uses considerably lower CPU utilization than PCR. This is because C-LSTM is an offline algorithm. Hence, the model has already been trained with the dataset offline and the trained model is then used for caching decisions. Therefore, not much CPU utilization is required for prediction. However, PCR is an online algorithm, therefore, the association prediction is done during runtime and hence obtains a higher CPU utilization. The CPU utilization obtained is stable and predictable. Therefore can be accounted for during live deployment.
- **Memory Utilization:** The memory utilization for both C-LSTM and PCR can also be seen in Figure 44. C-LSTM obtains a higher memory utilization than PCR. However, it is a low percentage compared to the overall memory. The higher memory utilization is because C-LSTM must load each trained model into memory which is then used for prediction. Although PCR will have to store a lot of the parameters in memory, these parameters are capped to prevent overflow and hence lower memory utilization. Low memory utilization is essential in real MEC environments due to the limited computation resources of MEC nodes. The memory utilization obtained during the

experiment proofs that the proposed framework can be deployed in a real MEC environment.

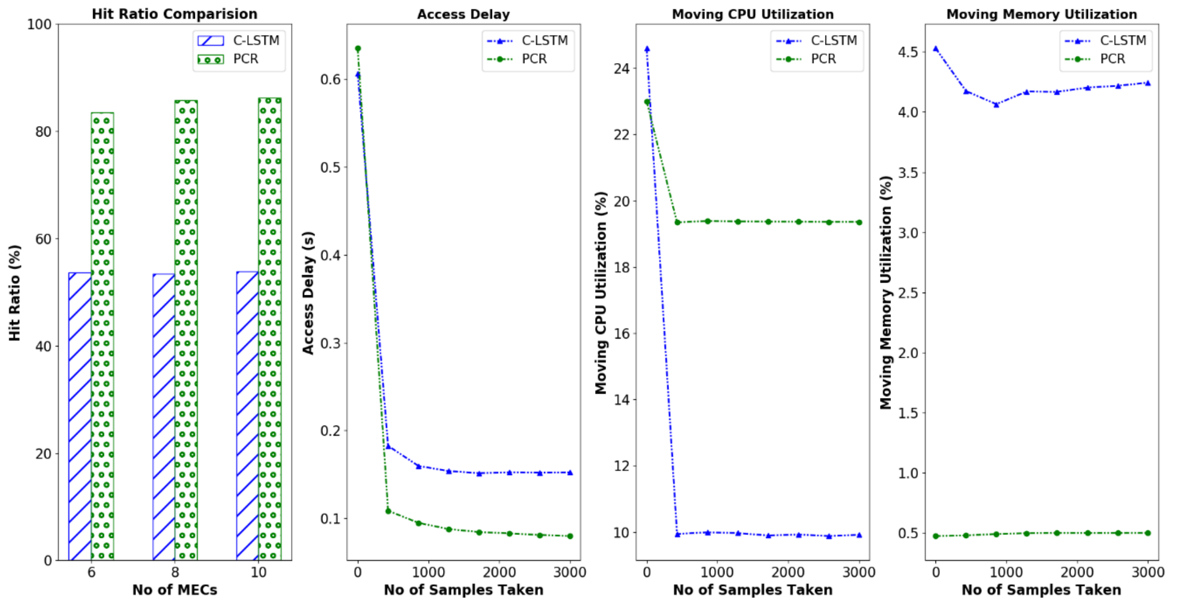

*Figure 44 Comparison Of Hit Ratio, Access Delay, CPU And Memory Utilization Of Both PCR And C-LSTM*

## 4.5 Conclusion

In this chapter, caching resource management in MEC has been explored. Following extensive literature, it has been identified that the user request pattern fluctuates with time. Therefore, the rule-based caching schemes are not suitable to dynamically adapt to user request patterns [104]. Additionally, there is a need to design a cooperative caching algorithm for MECs to effectively utilise the cache storage and reduce data redundancy in the edge. Hence, to address these problems, two predictive and cooperative caching schemes for MEC have been explored in this chapter to improve cache resource management in MEC. Table 17 shows how the two proposed schemes (bottom two) designed and evaluated in this chapter, compares with other existing algorithms studied in the literature review of this chapter in terms of techniques used. Ageing is this context refers to identifying cold cache objects quickly.

TABLE 17 BENCHMARKING CACHING TECHNIQUE COMPARISON

| *Algorithms* | Frequency | Recency | Size | Latency | Ageing | Prediction | Cooperative |
|---|---|---|---|---|---|---|---|
| *FIFO* | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| *LFU* | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| *LRU* | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| *LRU-K* | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| *LRFU* | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| *FBR* | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| *2Q* | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| *MQ* | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| *GD-Size* | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ |
| *LUV* | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ |
| *LLF* | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| *Size* | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| *Size-Adjusted LRU* | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ |
| *LRU-SP* | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| *Log(size)+LRU* | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| *Pitkow/Recker* | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ |
| *Liu et al [172]* | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| *MixCo[176]* | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ |
| *Sajeev et al [183]* | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ |
| *Dutta et al [184]* | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| *Chan et al [185]* | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| *Rahman et al [186]* | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| *CHRCA* | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| *PCR* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

A hybrid content caching algorithm for MEC has been proposed to improve caching efficiency. The problems addressed include the design of a sub-optimal Belady's algorithm that can be used in practice and reduce cache redundancy in the MECs. The result is a three-fold MEC caching framework that includes a modified LFU algorithm. The proposed algorithm is a convergence of a modified Belady's algorithm and a distributed co-operative caching algorithm. Polynomial regression algorithm has been leveraged to predict the future occurrences of requests using historical data of relative frequency of the requests. Cache redundancy in cooperative caching has been reduced by employing a selective caching approach. Experimental results have been obtained when the proposed algorithm is compared with two case study algorithms (a merge of Co-operative caching and LRU (CLRU) and a merge of Co-operative caching and LFU (CLFU)) on RTT, CPU utilization and cache performance show that the proposed algorithm obtains more cache hits and lesser average CPU utilization due to its selective caching approach. However, the prediction technique needs to be improved as the overall efficiency of the algorithm depends on the prediction accuracy to be near perfect which is very unlikely. Therefore, a new prediction technique could be explored.

To improve the prediction technique of the proposed scheme, a new predictive caching scheme has been presented. Here, PCR which is a three-fold caching solution to increase the collaborative hit ratio in the MEC platform and reduce the access delay incurred with obtaining request data and improve the prediction technique has been presented. The first problem addressed in the study is to design an algorithm that can identify cold cache objects quickly and reduce latency with a very little overhead cost. This has been achieved by the design and development of a caching algorithm that considers access delay, frequency, and recency during cache replacement to optimize the hit ratio. The novel replacement algorithm can identify a victim to be replaced in constant. Additionally, to adjust dynamically to the ever-changing user request pattern a proactive predictive caching algorithm to learn cache associations and prefetch cache objects when a user request is anticipated have been presented. The algorithm employs two sequential rule-based association mining techniques to learn and predict user request patterns. The rules generated are then used to effectively manage the MEC cache store. Finally, to increase the total hit ratio in the MEC platform a collaborative caching algorithm for MECs has been proposed. The algorithm relies on the sharing of cache information between MEC nodes. Comparisons have been made of the proposed algorithm PCR with an existing offline caching algorithm C-LSTM [186] and extensive experimentation shows that PCR is better than C-LSTM and other conventional algorithms with regards to the hit ratio and reduction of access delay. This improved hit ratio and reduction in access delay would improve the users' QoS.

# CHAPTER 5 CONCLUSION AND FUTURE WORK

## *5.1 SUMMARY*

The next generation of mobile networks (5G) is almost here, with many mobile operators in the deployment phase at the time of writing this thesis. Existing 4G networks are getting overwhelmed with the continuous increase of connected devices [1] due to the rise of IoT and mMTC and therefore would not be adequate for future needs. 5G aims at addressing the problems of 4G and additionally create a network environment that would be the base for emerging applications such as virtual and augmented reality, autonomous vehicles, and remote surgery. 5G would be beneficial for both the MNO and end-users. With regards to the MNOs, 5G would increase the network capacity to meet the increasing demand from the increase of connected devices and hence reduce the burden on 4G. Additionally, it will reduce the cost of deployment of future mobile network generations with network services entirely decoupled from the physical hardware. Therefore, making upgrades software specific rather than hardware-specific with the introduction of NFV and VNFs. Users also stand to benefit from 5G with the realization of URLLC which would support end-user applications such as 360 video streaming, driverless cars, etc.

MEC is essential in the realization of URLLC in 5G. This is because MEC offers cloud services at the edge of the network thereby reducing end-to-end latency for UE, easing traffic in the core network, enabling computation-intensive applications, and the use of resource-constrained devices. MEC aims at decentralising the cloud infrastructure. Therefore, by eliminating the distance and time it takes to send data to centralized sources, the speed and performance of data transport can be improved. To facilitate enhancements to the existing network infrastructure and improve the QoE, MEC provides two main services at the edge of the RAN. These include computational and caching services. Hence, to realize the full potential of MEC, it is important to effectively manage the computational resource and the caching resource. Therefore, this thesis has been aimed to improve the QoE of the network by designing and developing efficient algorithms that would effectively manage the caching and computational resources of MEC.

There have been increased computation capacity for end devices over the years. However, end devices still have inadequate computational capacity to execute emerging computation-intensive applications such as face recognition, etc. However, utilizing the computational resources available on the MEC, the end devices can offload their computationally intensive tasks to be executed in the MEC. This would yield major benefits including latency reduction, low energy consumption, and reduced traffic in the core network. The MEC servers have smaller computational resources compared to the cloud servers. Additionally, there would be a huge demand for these computational services due to the increasing number of connected devices [99]. This excessive demand might cause overprovisioning of resources in MEC and may even lead to deadlock in the system if adequate measures are not in place [100]. Computational offloading and resource provisioning have been explored in the context of MEC [63] [101] [77]. However, the issue of deadlock in MEC during resource provisioning has not been addressed. This problem has been fully explored in this thesis. The problem has been modelled and solved algorithmically by proposing a deadlock aware resource provisioning algorithm for MEC. The proposed algorithm utilises a modified resource request Banker's algorithm which can also re-distribute tasks to satisfy the latency constraints of offloaded workloads. Extensive simulation experiments have been carried out to validate the hypothesis of the proposed algorithm. Results have shown that system deadlock can be avoided by applying the proposed algorithm. To further validate the framework proposed, experimental comparisons have been made between different deadlock algorithms for MEC. The metrics used in the comparison include Round-trip time, Queue waiting time, CPU utilization, the ratio of tasks that meet the deadline during the experiment for each case study algorithm, and

Ratio of Local execution to cooperative MEC to cloud. Results show that each case study algorithm performs better when given specific constraints. This method will be very useful for applications like autonomous vehicles that require no failure rate to ensure passenger safety during commute.

Caching is another key service MEC offers that will aid mobile end devices to achieve low latency for delay-sensitive applications and improve the QoS. There has been a continuous increase in the amount of traffic in wireless networks due to the increasing demand for mobile video streaming and the increasing popularity of social networking [102]. Therefore, Caching in the MEC would help in reducing the transmission distance, reduction in transmission delay, reduction in energy consumption which would ultimately lead to improvement of user QoS. However, to ensure that this increasing demand does not overwhelm the MEC, an efficient caching algorithm is required to effectively manage the cache resource of the MEC. To address this problem, two novel intelligent caching algorithms have been explored in this thesis. First, a hybrid content caching algorithm for MEC is presented to improve caching efficiency. The proposed algorithm is a convergence of a modified Belady's algorithm and a distributed co-operative caching algorithm. Polynomial regression algorithm has been leveraged to predict the future occurrences of requests using historical data of relative frequency of the requests. The efficiency of the algorithm has been shown through extensive experimentations and comparisons with existing schemes. Additionally, another novel Predictive-Collaborative-Replacement scheme PCR has been explored. PCR is a three-fold caching solution to increase the collaborative hit ratio in the MEC platform and reduce the access delay incurred when obtaining requests. Comparison between PCR and an existing offline caching algorithm using LSTM for prediction [186] which has been aliased as C-LSTM in this report has been made. Extensive experimentation shows that PCR is better than C-LSTM and other conventional algorithms with regards to the hit ratio and reduction of access delay. This framework will be very useful for applications like e-health and streaming applications like Netflix, Amazon Prime Video, Disney Plus etc. These applications are latency-critical. Therefore, the framework could be used to proactively predict and cache popular items based on changing trends.

The two MEC resources (caching and computation) explored in this research have both contributed to the improvement of QoS and the realization of URLLC in 5G. The research outcomes have been summarised in section 5.2 and finally, the future works suggestions have been presented in section 5.3.

## *5.2 RESEARCH OUTCOMES*

This section summarizes the research outcomes of this thesis.

### 5.2.1 Deadlock Aware Resource Provisioning In MEC

A deadlock avoidance resource provisioning algorithm has been proposed for IoT devices using MEC platforms to ensure higher reliability of network interactions. The proposed scheme incorporates a Banker's resource-request algorithm using software-defined networking to reduce communication overhead. Extensive simulation results have shown that system deadlock can be prevented by applying the proposed algorithm which ultimately leads to a more reliable network interaction between mobile stations and MEC platforms.

### 5.2.2 Comparative Analysis For Deadlock Avoidance And Prevention for MEC

Here, deadlock avoidance and prevention mechanism are compared using Bankers resource request avoidance algorithm, wound wait algorithm, and wait-die algorithms. Additionally, a deadlock-aware cooperative decision algorithm has been proposed. The effectiveness of deadlock avoidance and prevention algorithms in real-time scenarios has been evaluated. The metrics used for comparison include Round-trip time, Queue waiting time, and CPU utilization. The effect that an increase in the number of MEC nodes has on the algorithm convergence has also been analyzed. Results obtained show that the avoidance algorithm does better in the percentage of tasks executed locally and the overall percentage of tasks executed on time while prevention algorithms obtain better CPU utilization. It has been concluded that the optimum difference between these two algorithms with regards to the framework is the ability to keep the system in a safe state and thus, eradiate deadlock. Thereby, improving the overall QoS of the system. Additionally, there was no clear optimal algorithm that is the best using the metrics employed for evaluation. In contrast, a clear winner can be selected by comparing each metric individually. Therefore, it has been deduced that the wound wait algorithm using RMS for task scheduling achieve the best CPU utilization. The Wait-die algorithm using RMS for task scheduling achieves the best percentage of tasks meeting the deadline while Wait Die using EDF as scheduling and Bankers using RMS as scheduling both achieve the best comparison in terms of total tasks executed locally and not re-offloaded.

### 5.2.3 Predictive Caching Using Lagrange extrapolating Polynomial

A novel hybrid content caching replacement algorithm in MEC to increase its caching efficiency where future request references are predicted using a polynomial fit algorithm along with Lagrange extrapolation has been proposed. Additionally, a distributed co-operative caching algorithm to improve data access within MECs has been presented. Experimental results have shown that the proposed scheme obtains more cache hits and lesser average CPU utilization due to its selective caching approach when compared with existing traditional cache replacement algorithms. The validation experiment confirms that the proposed approach achieves a 4% higher hit ratio than other compared algorithms excluding OPR.

### 5.2.4 PCR: A Novel Predictive-Collaborative-Replacement Algorithm for MECs

Here, three novel schemes (proactive predictive scheme, a collaborative scheme, and a replacement scheme (PCR)) have been proposed to address the underlined caching problem. The proposed algorithm has been tested using a real dataset (MovieLens20M dataset). Finally, it has been compared with an existing contemporary algorithm and results show that the proposed algorithm performs better in terms of hit ratio, achieving a 25% increase during comparison. Additionally, network access delay was reduced by approximately 40% during comparison.

## *5.3 FUTURE WORK*

The following should be considered for further research as an extension of the work done in this thesis.

### 5.3.1 Intelligent Deadlock Detection For MEC

Future work on deadlock in the MEC area may be to explore deadlock prediction in MEC using data collected over time. In this scenario, the Resource Allocation Graph (RAG) maps resource consumption over time for each MEC. Additionally, the fluctuation of resource consumption for each edge is monitored which results in a time series. Furthermore, an autoregressive function may be used to estimate the probability of the RAG to form a cycle. Hence, a proactive solution to prevent deadlock is employed. Therefore, the algorithm can deprioritize processes that are more likely to result in a deadlock as a preventive measure.

### 5.3.2 Distributed Deadlock Aware Algorithm For MEC

The proposed RPA is an algorithm for distributed systems and not a distributed algorithm. Further works on this topic will be to improve the algorithm into a distributed algorithm for MEC which can map the WFG of all the MEC nodes together rather than individual MEC WFG node mapping. Here decision-making can be done using a consensus algorithm.

### 5.3.3 Predictive Caching in ICN

ICN is advocated to shift the communication focus from data location to the data itself by making the named data the priority in the network. Therefore, data can be sourced from the internet using the named data and not the data location or IP address. Exploration analysis can be done to determine if the proposed caching algorithms can be adapted in the context of

ICN. In this context, the routers would be used instead of MEC for caching. Therefore, an analysis should be done on where to carry out model training for predictions to optimise latency and avoid overloading of the router's computational resources.

### 5.3.4 Distributed Predictive Caching Algorithm

The proposed novel scheme PCR is a proactive distributed caching framework that shares its cache details among collaborative MECs. However, it employs a centralised learning approach where each MEC maintains its predictive model. A complete predictive model of the popular cache objects for each collaborative space can be obtained if a distributed learning scheme is utilized such as Federated Learning. Hence, further research can be done to determine how the proposed predictive algorithm can be decentralised using federated learning.

# REFERENCES

[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022 White Paper," *Cisco*. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html (accessed Dec. 31, 2019).

[2] 5G PPP Architecture Working Group, "View on 5G architecture," 2, 2016. Accessed: Aug. 02, 2019. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2018/01/5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf

[3] I. F. Akyildiz, S. Nie, S.-C. Lin, and M. Chandrasekaran, "5G roadmap: 10 key enabling technologies," *Computer Networks*, vol. 106, pp. 17–48, Sep. 2016, doi: 10.1016/j.comnet.2016.06.010.

[4] T. O. Olwal, K. Djouani, and A. M. Kurien, "A Survey of Resource Management Toward 5G Radio Access Networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 1656–1686, thirdquarter 2016, doi: 10.1109/COMST.2016.2550765.

[5] S. Chaudhari, R. S. Mani, and P. Raundale, "SDN network virtualization survey," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Mar. 2016, pp. 650–655. doi: 10.1109/WiSPNET.2016.7566213.

[6] S. F. Abedin, Md. G. R. Alam, S. M. A. Kazmi, N. H. Tran, D. Niyato, and C. S. Hong, "Resource Allocation for Ultra-Reliable and Enhanced Mobile Broadband IoT Applications in Fog Network," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 489–502, Jan. 2019, doi: 10.1109/TCOMM.2018.2870888.

[7] P. Popovski *et al.*, "Wireless Access in Ultra-Reliable Low-Latency Communication (URLLC)," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5783–5801, Aug. 2019, doi: 10.1109/TCOMM.2019.2914652.

[8] C. D. Alwis *et al.*, "Survey on 6G Frontiers: Trends, Applications, Requirements, Technologies and Future Research," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 836–886, 2021, doi: 10.1109/OJCOMS.2021.3071496.

[9] A. Dogra, R. K. Jha, and S. Jain, "A Survey on beyond 5G network with the advent of 6G: Architecture and Emerging Technologies," *IEEE Access*, pp. 1–1, 2020, doi: 10.1109/ACCESS.2020.3031234.

[10] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017, doi: 10.1109/COMST.2017.2705720.

[11] E. W. Dijkstra, "Cooperating Sequential Processes," in *The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls*, P. B. Hansen, Ed. New York, NY: Springer New York, 2002, pp. 65–138. doi: 10.1007/978-1-4757-3472-0_2.

[12] M. Wolf, "Chapter 4 - Processes and Operating Systems," in *High-Performance Embedded Computing (Second Edition)*, M. Wolf, Ed. Boston: Morgan Kaufmann, 2014, pp. 201–241. doi: 10.1016/B978-0-12-410511-9.00004-6.

[13] A. Gupta and R. K. Jha, "A Survey of 5G Network: Architecture and Emerging Technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015, doi: 10.1109/ACCESS.2015.2461602.

[14] "5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf." Accessed: Oct. 05, 2018. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2018/01/5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf

[15] "Next Generation 5G Wireless Networks: A Comprehensive Survey - IEEE Journals & Magazine." https://ieeexplore.ieee.org/document/7414384 (accessed Jan. 16, 2020).

[16] S. Denazis, E. Haleplidis, J. H. Salim, O. Koufopavlou, D. Meyer, and K. Pentikousis, "Software-Defined Networking (SDN): Layers and Architecture Terminology." https://tools.ietf.org/html/rfc7426 (accessed Jan. 22, 2020).

[17] L. Ma, X. Wen, L. Wang, Z. Lu, and R. Knopp, "An SDN/NFV based framework for management and deployment of service based 5G core network," *China Communications*, vol. 15, no. 10, pp. 86–98, Oct. 2018, doi: 10.1109/CC.2018.8485472.

[18] K. Hasan and S.-H. Jeong, "Efficient Caching for Delivery of Multimedia Information with Low Latency in ICN," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2019, pp. 745–747. doi: 10.1109/ICUFN.2019.8806137.

[19] E. J. Kitindi, S. Fu, Y. Jia, A. Kabir, and Y. Wang, "Wireless Network Virtualization With SDN and C-RAN for 5G Networks: Requirements, Opportunities, and Challenges," *IEEE Access*, vol. 5, pp. 19099–19115, 2017, doi: 10.1109/ACCESS.2017.2744672.

[20] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, “Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions,” *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, thirdquarter 2018, doi: 10.1109/COMST.2018.2815638.

[21] W. Yu *et al.*, “A Survey on the Edge Computing for the Internet of Things,” *IEEE Access*, vol. 6, pp. 6900–6919, 2018, doi: 10.1109/ACCESS.2017.2778504.

[22] “Advanced Mobile Phone System (AMPS).” http://www.wirelesscommunication.nl/reference/chaptr01/telephon/amps.htm (accessed May 18, 2020).

[23] “The mobile phone adventure,” *Telenor Group*, Jan. 05, 2012. https://www.telenor.com/the-mobile-phone-adventure/ (accessed May 18, 2020).

[24] “TACS (1st Generation Mobiles).” https://www.engagingwithcommunications.com/Technology/Mobiles/TACS/tacs.php (accessed May 18, 2020).

[25] “GSM (Global System for Mobile Communications) - Global Telecoms Insight.” http://www.mobilecomms-technology.com/projects/gsm/ (accessed May 18, 2020).

[26] A. B. Forouzan, *Data communications & networking (sie)*, 4th ed. Tata McGraw-Hill Education, 2007.

[27] “General Packet Radio Service (GPRS) and EDGE,” in *From GSM to LTE-Advanced Pro and 5G*, John Wiley & Sons, Ltd, 2017, pp. 71–109. doi: 10.1002/9781119346913.ch2.

[28] T. K. Proctor, “Evolution to 3G services: provision of 3G services over GERAN (GSM/EDGE radio access network),” in *4th International Conference on 3G Mobile Communication Technologies*, Jun. 2003, pp. 78–82. doi: 10.1049/cp:20030341.

[29] S. Y. K. Ting and T. T. Chai, “WCDMA Network Planning and Optimisation,” in *2008 6th National Conference on Telecommunication Technologies and 2008 2nd Malaysia Conference on Photonics*, Aug. 2008, pp. 317–322. doi: 10.1109/NCTT.2008.4814295.

[30] A. Samukic, “UMTS Universal Mobile Telecommunications System development of standards for the third generation,” in *Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (Cat. No.98TH8361)*, Sep. 1998, vol. 2, pp. 978–984 vol.2. doi: 10.1109/PIMRC.1998.734707.

[31] T.-H. Liew and L. Hanzo, “High Speed Downlink and Uplink Packet Access,” in *3G, HSPA and FDD versus TDD Networking*, John Wiley & Sons, Ltd, 2008, pp. 87–117. doi: 10.1002/9780470754290.ch2.

[32] D. A. December 06 and 2007, “Evolution-Data Optimized,” *ITProPortal*. https://www.itproportal.com/2007/12/06/evolution-data-optimized/ (accessed May 18, 2020).

[33] “3G UMTS Handover: Hard Soft Softer » Electronics Notes.” https://www.electronics-notes.com/articles/connectivity/3g-umts/handover-handoff-hard-soft-softer.php (accessed May 18, 2020).

[34] U. Varshney, “4G Wireless Networks,” *IT Professional*, vol. 14, no. 5, pp. 34–39, Sep. 2012, doi: 10.1109/MITP.2012.71.

[35] M. Rumney, “Looking Towards 4G: LTE-Advanced,” in *LTE and the Evolution to 4G Wireless: Design and Measurement Challenges*, Wiley, 2013, pp. 567–600. doi: 10.1002/9781118799475.ch8.

[36] W. Kabir, “Orthogonal Frequency Division Multiplexing (OFDM),” in *2008 China-Japan Joint Microwave Conference*, Sep. 2008, pp. 178–184. doi: 10.1109/CJMW.2008.4772401.

[37] D. T.R. and I. Jose, “A Survey on 5G Standards, Specifications and Massive MIMO Testbed Including Transceiver Design Models Using QAM Modulation Schemes,” in *2019 International Conference on Data Science and Communication (IconDSC)*, Mar. 2019, pp. 1–7. doi: 10.1109/IconDSC.2019.8816942.

[38] F. Courau, M. Baker, S. Sesia, and I. Toufik, “Beyond LTE,” in *LTE – The UMTS Long Term Evolution*, John Wiley & Sons, Ltd, 2009, pp. 585–590. doi: 10.1002/9780470742891.ch24.

[39] C. Sun, X. Gao, S. Jin, M. Matthaiou, Z. Ding, and C. Xiao, “Beam Division Multiple Access Transmission for Massive MIMO Communications,” *IEEE Transactions on Communications*, vol. 63, no. 6, pp. 2170–2184, Jun. 2015, doi: 10.1109/TCOMM.2015.2425882.

[40] “5G-PPP.” https://5g-ppp.eu/ (accessed Jan. 11, 2020).

[41] S. Aglianò, M. Ashjaei, M. Behnam, and L. L. Bello, “Resource management and control in virtualized SDN networks,” in *2018 Real-Time and Embedded Systems and Technologies (RTEST)*, May 2018, pp. 47–53. doi: 10.1109/RTEST.2018.8397078.

[42] X. Liang and X. Qiu, “A software defined security architecture for SDN-based 5G network,” in *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, Sep. 2016, pp. 17–21. doi: 10.1109/ICNIDC.2016.7974528.

[43] J. Gil Herrera and J. F. Botero, “Resource Allocation in NFV: A Comprehensive Survey,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sep. 2016, doi: 10.1109/TNSM.2016.2598420.

[44] J. Fu and G. Li, “An Efficient VNF Deployment Scheme for Cloud Networks,” in *2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN)*, Jun. 2019, pp. 497–502. doi: 10.1109/ICCSN.2019.8905349.

[45] B. Gerő *et al.*, “The orchestration in 5G exchange — A multi-provider NFV framework for 5G services,” in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2017, pp. 1–2. doi: 10.1109/NFV-SDN.2017.8169865.

[46] G. Baldoni *et al.*, “Edge Computing Enhancements in an NFV-based Ecosystem for 5G Neutral Hosts,” in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2018, pp. 1–5. doi: 10.1109/NFV-SDN.2018.8725644.

[47] M. Gharbaoui *et al.*, “Demonstration of Latency-Aware and Self-Adaptive Service Chaining in 5G/SDN/NFV infrastructures,” in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2018, pp. 1–2. doi: 10.1109/NFV-SDN.2018.8725645.

[48] Jihoon Sung, Minseok Kim, Kyongchun Lim, and June-Koo Kevin Rhee, “Efficient cache placement strategy for wireless content delivery networks,” in *2013 International Conference on ICT Convergence (ICTC)*, Oct. 2013, pp. 238–239. doi: 10.1109/ICTC.2013.6675348.

[49] D. P. Evangeline and P. AnandhaKumar, “From P2P to cloud based P2P for live media streaming — A survey,” in *2015 Seventh International Conference on Advanced Computing (ICoAC)*, Dec. 2015, pp. 1–13. doi: 10.1109/ICoAC.2015.7562809.

[50] C. Fang, F. R. Yu, T. Huang, J. Liu, and Y. Liu, “A Survey of Green Information-Centric Networking: Research Issues and Challenges,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1455–1472, thirdquarter 2015, doi: 10.1109/COMST.2015.2394307.

[51] K. Ueda and A. Tagami, “ICN Performance Enhancing Proxies for the Global Content Distribution,” in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, Sep. 2018, pp. 253–254. doi: 10.1109/ICNP.2018.00039.

[52] H. Jung and S. Kim, “A Multiple Hash Routing Scheme for Fast Data Retrieval in ICN,” in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, Oct. 2018, pp. 1562–1565. doi: 10.1109/ICTC.2018.8539379.

[53] “Internet of things,” *Wikipedia*. Jan. 14, 2020. Accessed: Jan. 16, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=935732964

[54] B. Ó. hAnnaidh *et al.*, “Devices and Sensors Applicable to 5G System Implementations,” in *2018 IEEE MTT-S International Microwave Workshop Series on 5G Hardware and System Technologies (IMWS-5G)*, Aug. 2018, pp. 1–3. doi: 10.1109/IMWS-5G.2018.8484316.

[55] E. Kapassa, M. Touloupou, P. Stavrianos, and D. Kyriazis, “Dynamic 5G Slices for IoT Applications with Diverse Requirements,” in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, Oct. 2018, pp. 195–199. doi: 10.1109/IoTSMS.2018.8554386.

[56] S. Dahmen-Lhuissier, “ETSI - Open Source Mano | Open Source Solutions | Mano NFV,” *ETSI*. https://www.etsi.org/technologies/open-source-mano (accessed May 21, 2020).

[57] A. E. C. Redondi, A. Arcia-Moret, and P. Manzoni, “Towards a Scaled IoT Pub/Sub Architecture for 5G Networks: the Case of Multiaccess Edge Computing,” in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Apr. 2019, pp. 436–441. doi: 10.1109/WF-IoT.2019.8767268.

[58] D. Wang, D. Chen, B. Song, N. Guizani, X. Yu, and X. Du, “From IoT to 5G I-IoT: The Next Generation IoT-Based Intelligent Algorithms and 5G Technologies,” *IEEE Communications Magazine*, vol. 56, no. 10, pp. 114–120, Oct. 2018, doi: 10.1109/MCOM.2018.1701310.

[59] “Real-Life Use Cases for Edge Computing,” *IEEE Innovation at Work*, Jun. 25, 2019. https://innovationatwork.ieee.org/real-life-edge-computing-use-cases/ (accessed May 21, 2020).

[60] S. Shahzadi, M. Iqbal, T. Dagiuklas, and Z. U. Qayyum, “Multi-access edge computing: open issues, challenges and future perspectives,” *J Cloud Comp*, vol. 6, no. 1, p. 30, Dec. 2017, doi: 10.1186/s13677-017-0097-9.

[61] X. Wei *et al.*, “MVR: An Architecture for Computation Offloading in Mobile Edge Computing,” in *2017 IEEE International Conference on Edge Computing (EDGE)*, Jun. 2017, pp. 232–235. doi: 10.1109/IEEE.EDGE.2017.42.

[62] Y. Li and S. Wang, “An Energy-Aware Edge Server Placement Algorithm in Mobile Edge Computing,” in *2018 IEEE International Conference on Edge Computing (EDGE)*, Jul. 2018, pp. 66–73. doi: 10.1109/EDGE.2018.00016.

[63] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, “Zenith: Utility-Aware Resource Allocation for Edge Computing,” in *2017 IEEE International Conference on Edge Computing (EDGE)*, Jun. 2017, pp. 47–54. doi: 10.1109/IEEE.EDGE.2017.15.

[64] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, “The Road Towards 6G: A Comprehensive Survey,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021, doi: 10.1109/OJCOMS.2021.3057679.

[65] "Edge computing vs. fog computing: Definitions and enterprise uses," *Cisco*. https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html (accessed Apr. 20, 2020).

[66] Z. Wu, J. Zhang, W. Xie, and F. Yang, "CDN Convergence Based on Multi-access Edge Computing," in *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, Oct. 2018, pp. 1–5. doi: 10.1109/WCSP.2018.8555887.

[67] H. J. Roh, Y. Kim, K. S. Ko, and Y. I. Eom, "Design of a Content Replacement Scheme using the p-based LRFU-k algorithm in Contents Delivery Networks," in *2008 10th International Conference on Advanced Communication Technology*, Feb. 2008, vol. 3, pp. 2067–2070. doi: 10.1109/ICACT.2008.4494194.

[68] B. M. R. Wilson, B. Khazaei, and L. Hirsch, "Enablers and Barriers of Cloud Adoption among Small and Medium Enterprises in Tamil Nadu," in *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Nov. 2015, pp. 140–145. doi: 10.1109/CCEM.2015.21.

[69] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-Aware Elastic Cloud Provisioning for Scientific Workloads," in *2015 IEEE 8th International Conference on Cloud Computing*, Jun. 2015, pp. 971–974. doi: 10.1109/CLOUD.2015.130.

[70] O. Min, C. Park, J. Lee, J. Cho, and H. Kim, "Issues on supporting public cloud virtual machine provisioning and orchestration," in *13th International Conference on Advanced Communication Technology (ICACT2011)*, Feb. 2011, pp. 270–273.

[71] A. Hendre and K. P. Joshi, "A Semantic Approach to Cloud Security and Compliance," in *2015 IEEE 8th International Conference on Cloud Computing*, Jun. 2015, pp. 1081–1084. doi: 10.1109/CLOUD.2015.157.

[72] M. Aazam and E. N. Huh, "Inter-cloud Media Storage and Media Cloud Architecture for Inter-cloud Communication," in *2014 IEEE 7th International Conference on Cloud Computing*, Jun. 2014, pp. 982–985. doi: 10.1109/CLOUD.2014.151.

[73] M. Bahrami, "Cloud Computing for Emerging Mobile Cloud Apps," in *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Mar. 2015, pp. 4–5. doi: 10.1109/MobileCloud.2015.40.

[74] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," in *2010 IEEE 3rd International Conference on Cloud Computing*, Jul. 2010, pp. 450–457. doi: 10.1109/CLOUD.2010.37.

[75] D. A. Rodríguez-Silva, L. Adkinson-Orellana, F. J. Gonz'lez-Castaño, I. Armiño-Franco, and D. Gonz'lez-Martínez, "Video Surveillance Based on Cloud Storage," in *2012 IEEE Fifth International Conference on Cloud Computing*, Jun. 2012, pp. 991–992. doi: 10.1109/CLOUD.2012.44.

[76] Z. Zhang and S. Li, "A Survey of Computational Offloading in Mobile Cloud Computing," in *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, Mar. 2016, pp. 81–82. doi: 10.1109/MobileCloud.2016.15.

[77] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Communications and Mobile Computing*, vol. 16, no. 16, pp. 2572–2589, 2016, doi: 10.1002/wcm.2709.

[78] S. Farrugia, "Mobile Cloud Computing Techniques for Extending Computation and Resources in Mobile Devices," in *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, Mar. 2016, pp. 1–10. doi: 10.1109/MobileCloud.2016.26.

[79] F. AlShahwan and M. Faisal, "Mobile Cloud Computing for Providing Complex Mobile Web Services," in *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Apr. 2014, pp. 77–84. doi: 10.1109/MobileCloud.2014.12.

[80] "The Tactile Internet," p. 24, 2014.

[81] "ITU: Committed to connecting the world." https://www.itu.int/en/Pages/default.aspx (accessed May 22, 2020).

[82] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009, doi: 10.1109/MPRV.2009.82.

[83] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," *Journal of Network and Computer Applications*, vol. 62, pp. 18–40, Feb. 2016, doi: 10.1016/j.jnca.2015.11.009.

[84] Z. Pang, L. Sun, Z. Wang, E. Tian, and S. Yang, "A Survey of Cloudlet Based Mobile Computing," in *2015 International Conference on Cloud Computing and Big Data (CCBD)*, Nov. 2015, pp. 268–275. doi: 10.1109/CCBD.2015.54.

[85] E. Cuervo *et al.*, “MAUI: making smartphones last longer with code offload,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, San Francisco, California, USA, Jun. 2010, pp. 49–62. doi: 10.1145/1814433.1814441.

[86] A. Balasubramanian, R. Mahajan, and A. Venkataramani, “Augmenting mobile 3G using WiFi,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, San Francisco, California, USA, Jun. 2010, pp. 209–222. doi: 10.1145/1814433.1814456.

[87] C. C. Byers, “Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks,” *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, Aug. 2017, doi: 10.1109/MCOM.2017.1600885.

[88] M. Mukherjee, L. Shu, and D. Wang, “Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges,” *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 1826–1857, thirdquarter 2018, doi: 10.1109/COMST.2018.2814571.

[89] H. L. Cech, M. Großmann, and U. R. Krieger, “A Fog Computing Architecture to Share Sensor Data by Means of Blockchain Functionality,” in *2019 IEEE International Conference on Fog Computing (ICFC)*, Jun. 2019, pp. 31–40. doi: 10.1109/ICFC.2019.00013.

[90] T. Pfandzelter and D. Bermbach, “IoT Data Processing in the Fog: Functions, Streams, or Batch Processing?,” in *2019 IEEE International Conference on Fog Computing (ICFC)*, Jun. 2019, pp. 201–206. doi: 10.1109/ICFC.2019.00033.

[91] M. Bouet and V. Conan, “Mobile Edge Computing Resources Optimization: A Geo-Clustering Approach,” *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 787–796, Jun. 2018, doi: 10.1109/TNSM.2018.2816263.

[92] S. Dahmen-Lhuissier, “ETSI - Multi-access Edge Computing - Standards for MEC,” *ETSI*. https://www.etsi.org/technologies/multi-access-edge-computing (accessed May 22, 2020).

[93] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, “Edge Computing Enabled Smart Cities: A Comprehensive Survey,” *IEEE Internet of Things Journal*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.2987070.

[94] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile Edge Computing: A Survey,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb. 2018, doi: 10.1109/JIOT.2017.2750180.

[95] A. C. Baktir, A. Ozgovde, and C. Ersoy, “How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, Fourthquarter 2017, doi: 10.1109/COMST.2017.2717482.

[96] K. Ha *et al.*, “The Impact of Mobile Multimedia Applications on Data Center Consolidation,” in *2013 IEEE International Conference on Cloud Engineering (IC2E)*, Mar. 2013, pp. 166–176. doi: 10.1109/IC2E.2013.17.

[97] S. Ghosh, T. Dagiuklas, and M. Iqbal, “Energy-Aware IP Routing Over SDN,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–7. doi: 10.1109/GLOCOM.2018.8647764.

[98] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, “Fog Computing: A Platform for Internet of Things and Analytics,” in *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis and C. Dobre, Eds. Cham: Springer International Publishing, 2014, pp. 169–186. doi: 10.1007/978-3-319-05029-4_7.

[99] A. Sheshasaayee and R. Megala, “A study on resource provisioning approaches in autonomic cloud computing,” in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Feb. 2017, pp. 141–144. doi: 10.1109/I-SMAC.2017.8058325.

[100] E. E. Ugwuanyi, S. Ghosh, M. Iqbal, and T. Dagiuklas, “Reliable Resource Provisioning Using Bankers’ Deadlock Avoidance Algorithm in MEC for Industrial IoT,” *IEEE Access*, vol. 6, pp. 43327–43335, 2018, doi: 10.1109/ACCESS.2018.2857726.

[101] Y. Mao, J. Zhang, and K. B. Letaief, “Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016, doi: 10.1109/JSAC.2016.2611964.

[102] M. Mehrabi, D. You, V. Latzko, H. Salah, M. Reisslein, and F. H. P. Fitzek, “Device-Enhanced MEC: Multi-Access Edge Computing (MEC) Aided by End Device Computation and Caching: A Survey,” *IEEE Access*, vol. 7, pp. 166079–166108, 2019, doi: 10.1109/ACCESS.2019.2953172.

[103] A. Abouaomar, A. Filali, and A. Kobbane, “Caching, device-to-device and fog computing in 5 lt;sup gt;th lt;/sup gt; cellular networks generation : Survey,” in *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Nov. 2017, pp. 1–6. doi: 10.1109/WINCOM.2017.8238174.

[104] C. Zhang *et al.*, “Toward Edge-Assisted Video Content Intelligent Caching With Long Short-Term Memory Learning,” *IEEE Access*, vol. 7, pp. 152832–152846, 2019, doi: 10.1109/ACCESS.2019.2947067.

[105] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, “Delay-optimal computation task scheduling for mobile-edge computing systems,” in *2016 IEEE International Symposium on Information Theory (ISIT)*, Jul. 2016, pp. 1451–1455. doi: 10.1109/ISIT.2016.7541539.

[106] A. Al-Shuwaili and O. Simeone, “Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications,” *IEEE Wireless Communications Letters*, vol. 6, no. 3, pp. 398–401, Jun. 2017, doi: 10.1109/LWC.2017.2696539.

[107] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, “A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications,” *IEEE Access*, vol. 5, pp. 6757–6779, 2017, doi: 10.1109/ACCESS.2017.2685434.

[108] C. Sanchez, H. B. Sipma, and Z. Manna, “Generating Efficient Distributed Deadlock Avoidance Controllers,” in *2007 IEEE International Parallel and Distributed Processing Symposium*, Mar. 2007, pp. 1–8. doi: 10.1109/IPDPS.2007.370357.

[109] C.-K. Tham and R. Chattopadhyay, “A load balancing scheme for sensing and analytics on a mobile edge computing network,” in *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2017 IEEE 18th International Symposium on*, 2017, pp. 1–9.

[110] R. Beraldi, A. Mtibaa, and H. Alnuweiri, “Cooperative load balancing scheme for edge computing resources,” in *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*, 2017, pp. 94–100.

[111] M. Altamimi, “A task offloading framework for energy saving on mobile devices using cloud computing,” 2015.

[112] J. Doherty, *SDN and NFV simplified: a visual guide to understanding software defined networks and network function virtualization*. Addison-Wesley Professional, 2016.

[113] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System,” *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978, doi: 10.1145/359545.359563.

[114] K. M. Chandy, J. Misra, and L. M. Haas, “Distributed Deadlock Detection,” *ACM Trans. Comput. Syst.*, vol. 1, no. 2, pp. 144–156, May 1983, doi: 10.1145/357360.357365.

[115] H. H. C. Nguyen, H. V. Dang, N. M. N. Pham, V. S. Le, and T. T. Nguyen, “Deadlock Detection for Resource Allocation in Heterogeneous Distributed Platforms,” in *Recent Advances in Information and Communication Technology 2015*, 2015, pp. 285–295.

[116] H. H. C. Nguyen and V. S. Le, “Detection and avoidance deadlock for resources allocation in heterogenenous distributed plaforms,” *Int. J. Comput. Sci. Telecommun.(IJCST), English*, vol. 6, no. 2, pp. 1–6, 2015.

[117] J. Lim, T. Suh, and H. Yu, “Unstructured deadlock detection technique with scalability and complexity-efficiency in clouds,” *International Journal of Communication Systems*, vol. 27, no. 6, pp. 852–870, 2014, doi: 10.1002/dac.2638.

[118] “A novel scheduling strategy for an efficient deadlock detection - IEEE Conference Publication.” https://ieeexplore.ieee.org/document/5383067 (accessed Nov. 07, 2019).

[119] “An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems - IEEE Conference Publication.” https://ieeexplore.ieee.org/document/1562667 (accessed Nov. 07, 2019).

[120] A. Izumi, T. Dohi, and N. Kaio, “Deadlock Detection Scheduling for Distributed Processes in the Presence of System Failures,” in *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, Tokyo, Japan, Dec. 2010, pp. 133–140. doi: 10.1109/PRDC.2010.49.

[121] K. S. Rashmi, V. Suma, and M. Vaidehi, “Enhanced Load Balancing Approach to Avoid Deadlocks in Cloud,” *arXiv:1209.6470 [cs]*, Sep. 2012, Accessed: Jan. 12, 2019. [Online]. Available: http://arxiv.org/abs/1209.6470

[122] O. Mahitha and V. Suma, “Deadlock avoidance through efficient load balancing to control disaster in cloud environment,” in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, Jul. 2013, pp. 1–6. doi: 10.1109/ICCCNT.2013.6726823.

[123] D. Malhotra, “Deadlock Prevention Algorithm in Grid Environment,” *MATEC Web of Conferences*, vol. 57, p. 02013, 2016, doi: 10.1051/matecconf/20165702013.

[124] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, “System level concurrency control for distributed database systems,” *ACM Trans. Database Syst.*, vol. 3, no. 2, pp. 178–198, Jun. 1978, doi: 10.1145/320251.320260.

[125] L. Lou, F. Tang, I. You, M. Guo, Y. Shen, and L. Li, “An Effective Deadlock Prevention Mechanism for Distributed Transaction Management,” in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Jun. 2011, pp. 120–127. doi: 10.1109/IMIS.2011.109.

[126] K. N. Mishra, “An efficient voting and priority based mechanism for deadlock prevention in distributed systems,” in *2016 International Conference on Control, Computing, Communication and Materials (ICCCCM)*, Oct. 2016, pp. 1–6. doi: 10.1109/ICCCCM.2016.7918267.

[127] J. Ding, H. Zhu, H. Zhu, and Q. Li, “Formal Modeling and Verifications of Deadlock Prevention Solutions in Web Service Oriented System,” in *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, Mar. 2010, pp. 335–343. doi: 10.1109/ECBS.2010.48.

[128] Z. Chuanfu, L. Yunsheng, Z. Tong, Z. Yabing, and H. Kedi, “A Deadlock Prevention Approach based on Atomic Transaction for Resource Co-allocation,” in *2005 First International Conference on Semantics, Knowledge and Grid*, Nov. 2005, pp. 37–37. doi: 10.1109/SKG.2005.4.

[129] A. Silberschatz, G. Gagne, and P. B. Galvin, *Operating System Concepts*. John Wiley & Sons, 1994.

[130] H. Badri, T. Bahreini, D. Grosu, and K. Yang, “Risk-Based Optimization of Resource Provisioning in Mobile Edge Computing,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct. 2018, pp. 328–330. doi: 10.1109/SEC.2018.00033.

[131] N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, “Optimized Provisioning of Edge Computing Resources With Heterogeneous Workload in IoT Networks,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, Jun. 2019, doi: 10.1109/TNSM.2019.2894955.

[132] P. Chang and G. Miao, “Resource Provision for Energy-Efficient Mobile Edge Computing Systems,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–6. doi: 10.1109/GLOCOM.2018.8648008.

[133] S. Yu and R. Langar, “Collaborative Computation Offloading for Multi-access Edge Computing,” in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Apr. 2019, pp. 689–694.

[134] Z. Zhou, S. Yu, W. Chen, and X. Chen, “CE-IoT: Cost-Effective Cloud-Edge Resource Provisioning for Heterogeneous IoT Applications,” *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8600–8614, Sep. 2020, doi: 10.1109/JIOT.2020.2994308.

[135] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, “Mobility-Aware and Delay-Sensitive Service Provisioning in Mobile Edge-Cloud Networks,” *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020, doi: 10.1109/TMC.2020.3006507.

[136] Y. Ma, W. Liang, M. Huang, Y. Liu, and S. Guo, “Virtual Network Function Service Provisioning for Offloading Tasks in MEC by Trading off Computing and Communication Resource Usages,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 1–7. doi: 10.1109/INFOCOMWKSHPS47286.2019.9093758.

[137] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, “Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019, doi: 10.1109/JIOT.2019.2903191.

[138] “Operating System Concepts, 9th Edition,” *Wiley.com*, 1994. https://www.wiley.com/en-us/Operating+System+Concepts%2C+9th+Edition-p-9781118063330 (accessed Sep. 11, 2018).

[139] S. Davidson, I. Lee, and V. F. Wolfe, “Deadlock prevention in concurrent real-time systems,” *Real-Time Syst*, vol. 5, no. 4, pp. 305–318, Oct. 1993, doi: 10.1007/BF01088833.

[140] S.-D. Lang, “An extended banker’s algorithm for deadlock avoidance,” *IEEE Transactions on Software Engineering*, vol. 25, no. 3, pp. 428–432, May 1999, doi: 10.1109/32.798330.

[141] F. Tricas, J. M. Colom, and J. Ezpeleta, “Some improvements to the Banker’s algorithm based on the process structure,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, Apr. 2000, vol. 3, pp. 2853–2858 vol.3. doi: 10.1109/ROBOT.2000.846460.

[142] J. J. Lee and V. J. Mooney, “A Novel O(n) Parallel Banker’s Algorithm for System-on-a-Chip,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, pp. 1377–1389, Dec. 2006, doi: 10.1109/TPDS.2006.164.

[143] P. M. El-Kafrawy, “Graphical Deadlock Avoidance,” in *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Dec. 2009, pp. 308–312. doi: 10.1109/ACT.2009.83.

[144] P. Li, K. Agrawal, J. Buhler, R. D. Chamberlain, and J. M. Lancaster, “Deadlock-avoidance for streaming applications with split-join structure: Two case studies,” in *ASAP 2010 - 21st IEEE International Conference on Application-specific Systems, Architectures and Processors*, Jul. 2010, pp. 333–336. doi: 10.1109/ASAP.2010.5540957.

[145] A. Mohammadi and S. G. Akl, “Scheduling Algorithms for Real-Time Systems,” p. 49, 2005.

[146] S. Reveliotis and Z. Fei, “Robust deadlock avoidance for sequential resource allocation systems with resource outages,” in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, Aug. 2016, pp. 864–871. doi: 10.1109/COASE.2016.7743492.

[147] Y. Yang and H. Hu, “Distributed deadlock avoidance in automated manufacturing systems with forward conflict free structures using Petri nets,” in *2016 European Control Conference (ECC)*, Jun. 2016, pp. 2345–2352. doi: 10.1109/ECC.2016.7810641.

[148] Z. Huang and Z. Wu, “A new distributed deadlock avoidance strategy for flexible manufacturing systems using digraph models,” in *2006 8th International Workshop on Discrete Event Systems*, Jul. 2006, pp. 276–281. doi: 10.1109/WODES.2006.1678442.

[149] V. Kate, A. Jaiswal, and A. Gehlot, “A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock,” in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, Mar. 2016, pp. 1–6. doi: 10.1109/CDAN.2016.7570873.

[150] A. B. Forouzan, “CHAPTER 12 MULTIPLE ACCESS: Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA),” in *Data Communications and Networking*, 4th ed., 1221 Avenue of the Americas, New York, NY: McGraw-Hill, 2007, pp. 377–379.

[151] A. Bazzi, C. Campolo, B. M. Masini, A. Molinaro, A. Zanella, and A. O. Berthet, “Enhancing Cooperative Driving in IEEE 802.11 Vehicular Networks Through Full-Duplex Radios,” *IEEE Transactions on Wireless Communications*, vol. 17, no. 4, pp. 2402–2416, Apr. 2018, doi: 10.1109/TWC.2018.2794967.

[152] K. McClaning and T. Vito, “Additive white Gaussian noise,” in *Radio Receiver Design*, Noble Publishing Corporation, 2000, pp. 99–105, 361. [Online]. Available: https://books.google.co.uk/books/about/Radio_Receiver_Design.html?id=6hEfAQAAIAAJ&redir_esc=y

[153] J. Lehoczky, L. Sha, and Y. Ding, “The rate monotonic scheduling algorithm: exact characterization and average case behavior,” in *[1989] Proceedings. Real-Time Systems Symposium*, Dec. 1989, pp. 166–171. doi: 10.1109/REAL.1989.63567.

[154] Andrew Coleman, Julien Duponchelle, and Ricar Ganancial, “GNS3 documentation,” *GNS3 Documentation*, 2019. https://docs.gns3.com/ (accessed Dec. 05, 2019).

[155] T. H. Team, “Quality of Service 0,1 & 2 - MQTT Essentials: Part 6.” https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels (accessed Jun. 22, 2020).

[156] E. E. Ugwuanyi, “emylincon/deadlock_project,” *Deadlock implementation Python Project*, Dec. 05, 2019. https://github.com/emylincon/deadlock_project (accessed Dec. 05, 2019).

[157] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, “Cache in the air: exploiting content caching and delivery techniques for 5G systems,” *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, Feb. 2014, doi: 10.1109/MCOM.2014.6736753.

[158] D. L. Willick, D. L. Eager, and R. B. Bunt, “Disk cache replacement policies for network fileservers,” in *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*, May 1993, pp. 2–11. doi: 10.1109/ICDCS.1993.287729.

[159] Y. Zhou, J. Philbin, and K. Li, “The Multi-Queue Replacement Algorithm for Second Level Buffer Caches,” in *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, USA, Jun. 2001, pp. 91–104.

[160] E. J. O’Neil, P. E. O’Neil, and G. Weikum, “The LRU-K page replacement algorithm for database disk buffering,” *SIGMOD Rec.*, vol. 22, no. 2, pp. 297–306, Jun. 1993, doi: 10.1145/170036.170081.

[161] Donghee Lee *et al.*, “LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies,” *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, Dec. 2001, doi: 10.1109/TC.2001.970573.

[162] J. T. Robinson and M. V. Devarakonda, “Data cache management using frequency-based replacement,” in *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, Univ. of Colorado, Boulder, Colorado, USA, Apr. 1990, pp. 134–142. doi: 10.1145/98457.98523.

[163] T. Johnson and D. Shasha, “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA, Sep. 1994, pp. 439–450.

[164] N. E. Young, “The K-Server Dual and Loose Competitiveness for Paging,” *Algorithmica*, vol. 11, no. 6, pp. 525–541, Jun. 1994, doi: 10.1007/BF01189992.

[165] P. Cao and S. Irani, “Cost-aware WWW proxy caching algorithms,” in *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, Monterey, California, Dec. 1997, p. 18.

[166] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, “Evaluating content management techniques for Web proxy caches,” *SIGMETRICS Perform. Eval. Rev.*, vol. 27, no. 4, pp. 3–11, Mar. 2000, doi: 10.1145/346000.346003.

[167] H. Bahn, S. H. Noh, S. L. Min, and K. Koh, “Using full reference history for efficient document replacement in web caches,” in *Proceedings of the 2nd conference on USENIX Symposium on Internet Technologies and Systems - Volume 2*, Boulder, Colorado, Oct. 1999, p. 17.

[168] Kai Cheng and Y. Kambayashi, “LRU-SP: a size-adjusted and popularity-aware LRU replacement algorithm for web caching,” in *Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000*, Oct. 2000, pp. 48–53. doi: 10.1109/CMPSAC.2000.884690.

[169] H. Wu, J. Li, J. Zhi, Y. Ren, and L. Li, “Edge-oriented Collaborative Caching in Information-Centric Networking,” in *2019 IEEE Symposium on Computers and Communications (ISCC)*, Jun. 2019, pp. 1–6. doi: 10.1109/ISCC47284.2019.8969688.

[170] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, “Collaborative cache allocation and computation offloading in mobile edge computing,” in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Sep. 2017, pp. 366–369. doi: 10.1109/APNOMS.2017.8094149.

[171] X. Zhang and Q. Zhu, “Collaborative Hierarchical Caching over 5G Edge Computing Mobile Wireless Networks,” in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6. doi: 10.1109/ICC.2018.8422371.

[172] J. Liu, D. Li, and Y. Xu, “Collaborative Online Edge Caching With Bayesian Clustering in Wireless Networks,” *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1548–1560, Feb. 2020, doi: 10.1109/JIOT.2019.2956554.

[173] Y. M. Saputra, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and D. I. Kim, “Distributed Deep Learning at the Edge: A Novel Proactive and Cooperative Caching Framework for Mobile Edge Networks,” *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1220–1223, Aug. 2019, doi: 10.1109/LWC.2019.2912365.

[174] E. K. Markakis, K. Karras, A. Sideris, G. Alexiou, and E. Pallis, “Computing, Caching, and Communication at the Edge: The Cornerstone for Building a Versatile 5G Ecosystem,” *IEEE Communications Magazine*, vol. 55, no. 11, pp. 152–157, Nov. 2017, doi: 10.1109/MCOM.2017.1700105.

[175] S. Kim, “5G Network Communication, Caching, and Computing Algorithms Based on the Two-Tier Game Model,” *ETRI Journal*, vol. 40, no. 1, pp. 61–71, Feb. 2018, doi: 10.4218/etrij.2017-0023.

[176] N. Wang, W. Shao, S. K. Bose, and G. Shen, “MixCo: Optimal Cooperative Caching for Mobile Edge Computing in Fiber-Wireless Access Networks,” in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, Mar. 2018, pp. 1–3.

[177] X. Huang, Z. Zhao, and H. Zhang, “Cooperate Caching with Multicast for Mobile Edge Computing in 5G Networks,” in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, Jun. 2017, pp. 1–6. doi: 10.1109/VTCSpring.2017.8108600.

[178] C. Yang, H. Li, L. Wang, and D. Tang, “Exploring the behaviors and threats of pollution attack in cooperative MEC caching,” in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2018, pp. 1–6. doi: 10.1109/WCNC.2018.8377317.

[179] Y. Chen, Y. Liu, J. Zhao, and Q. Zhu, “Mobile Edge Cache Strategy Based on Neural Collaborative Filtering,” *IEEE Access*, vol. 8, pp. 18475–18482, 2020, doi: 10.1109/ACCESS.2020.2964711.

[180] J. Xu, K. Ota, and M. Dong, “Energy Efficient Hybrid Edge Caching Scheme for Tactile Internet in 5G,” *IEEE Transactions on Green Communications and Networking*, vol. 3, no. 2, pp. 483–493, Jun. 2019, doi: 10.1109/TGCN.2019.2905225.

[181] K. Qi and C. Yang, “Popularity Prediction with Federated Learning for Proactive Caching at Wireless Edge,” in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, May 2020, pp. 1–6. doi: 10.1109/WCNC45663.2020.9120586.

[182] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, “Reinforcement Learning-Based Optimal Computing and Caching in Mobile Edge Network,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2343–2355, Oct. 2020, doi: 10.1109/JSAC.2020.3000396.

[183] G. P. Sajeev and M. P. Sebastian, “Building a semi intelligent web cache with light weight machine learning,” in *2010 5th IEEE International Conference Intelligent Systems*, Jul. 2010, pp. 420–425. doi: 10.1109/IS.2010.5548373.

[184] S. Dutta, A. Narang, S. Bhattacherjee, A. S. Das, and D. Krishnaswamy, “Predictive Caching Framework for Mobile Wireless Networks,” in *2015 16th IEEE International Conference on Mobile Data Management*, Jun. 2015, vol. 1, pp. 179–184. doi: 10.1109/MDM.2015.14.

[185] C. A. Chan *et al.*, “Big Data Driven Predictive Caching at the Wireless Edge,” in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2019, pp. 1–6. doi: 10.1109/ICCW.2019.8756663.

[186] S. Rahman, Md. G. R. Alam, and Md. M. Rahman, “Deep Learning-based Predictive Caching in the Edge of a Network,” in *2020 International Conference on Information Networking (ICOIN)*, Jan. 2020, pp. 797–801. doi: 10.1109/ICOIN48656.2020.9016437.

[187] H.-G. Song, S. H. Chae, W.-Y. Shin, and S.-W. Jeon, "Predictive Caching via Learning Temporal Distribution of Content Requests," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2335–2339, Dec. 2019, doi: 10.1109/LCOMM.2019.2941202.

[188] Y. Wang and V. Friderikos, "A Survey of Deep Learning for Data Caching in Edge Network," *arXiv:2008.07235 [cs]*, Aug. 2020, Accessed: Dec. 14, 2020. [Online]. Available: http://arxiv.org/abs/2008.07235

[189] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966, doi: 10.1147/sj.52.0078.

[190] P. Panda, G. Patil, and B. Raveendran, "A survey on replacement strategies in cache memory for embedded systems," in *2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, Aug. 2016, pp. 12–17. doi: 10.1109/DISCOVER.2016.7806218.

[191] J. H. Mathews and K. K. Fink, *Numerical Methods Using Matlab: United States Edition*, 4 edition. Upper Saddle River, N.J: Pearson, 2003.

[192] E. W. Weisstein, "Lagrange Interpolating Polynomial." https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html (accessed Apr. 29, 2021).

[193] "(3) (PDF) Newton's Divided Difference Interpolation formula: Representation of Numerical Data by a Polynomial curve," *ResearchGate*. https://www.researchgate.net/publication/308576265_Newton's_Divided_Difference_Interpolation_formula_Representation_of_Numerical_Data_by_a_Polynomial_curve (accessed Oct. 04, 2020).

[194] "Open source software for creating private and public clouds.," *OpenStack*. https://www.openstack.org/ (accessed Sep. 12, 2018).

[195] "GNS3 | The software that empowers network professionals." https://www.gns3.com/ (accessed Sep. 12, 2018).

[196] "Welcome to OpenDaylight Documentation — OpenDaylight Documentation Fluorine documentation." https://docs.opendaylight.org/en/stable-fluorine/ (accessed Oct. 14, 2018).

[197] "Quagga Software Routing Suite." https://www.quagga.net/ (accessed Oct. 31, 2018).

[198] G. K. Zipf, "Selected Studies of the Principle of Relative Frequency in Language — George Kingsley Zipf | Harvard University Press." https://www.hup.harvard.edu/catalog.php?isbn=9780674434929 (accessed Apr. 19, 2021).

[199] "numpy.random.zipf — NumPy v1.20 Manual." https://numpy.org/doc/stable/reference/random/generated/numpy.random.zipf.html (accessed Apr. 19, 2021).

[200] E. E. Ugwuanyi, S. Ghosh, M. Iqbal, T. Dagiuklas, S. Mumtaz, and A. Al-Dulaimi, "Co-Operative and Hybrid Replacement Caching for Multi-Access Mobile Edge Computing," in *2019 European Conference on Networks and Communications (EuCNC)*, Jun. 2019, pp. 394–399. doi: 10.1109/EuCNC.2019.8801991.

[201] S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, Dec. 2003, doi: 10.1145/954339.954341.

[202] E. E. Ugwuanyi, "emylincon/caching," *MEC Content and Predictive Caching using Association*, Jul. 22, 2020. https://github.com/emylincon/caching (accessed Jul. 22, 2020).

[203] "CacheTest." https://cachetrace.herokuapp.com/ (accessed Sep. 01, 2020).

[204] N. Zhou, M. Qiao, and J. Zhou, "BI_Apriori Algorithm: Research and Application Based on Battery Production Data," in *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, Jul. 2019, pp. 1–5. doi: 10.1109/ICEIEC.2019.8784491.

[205] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA, Sep. 1994, pp. 487–499.

[206] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, Jan. 2004, doi: 10.1023/B:DAMI.0000005258.31418.83.

[207] B. Wu, D. Zhang, Q. Lan, and J. Zheng, "An Efficient Frequent Patterns Mining Algorithm Based on Apriori Algorithm and the FP-Tree Structure," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, Nov. 2008, vol. 1, pp. 1099–1102. doi: 10.1109/ICCIT.2008.109.

[208] T. Pang-Ning, M. Steinbach, and K. Vipin, "Chapter 6. Association Analysis: Basic Concepts and Algorithms," in *Introduction to Data Mining. Addison-Wesley*, 2005.

[209] "Netlify: All-in-one platform for automating modern web projects," *Netlify*. https://www.netlify.com/ (accessed Nov. 08, 2020).

# APPENDIX SECTION

## APPENDIX 1 DEADLOCK TESTBED

The purpose of this testbed setup is to create a system that can record event traces during the deployed algorithm runtime. Details of how to deploy and run the algorithms are available on the GitHub page[12].

### 1.1 DESIGN

The network architectural emulation has been done using the GNS3[13] platform. This allows the design of complex networks using routers, switches, firewalls, end devices etc. However, the deployed algorithms have been implemented using python.

#### 1.1.1 GNS3 ARCHITECTURE

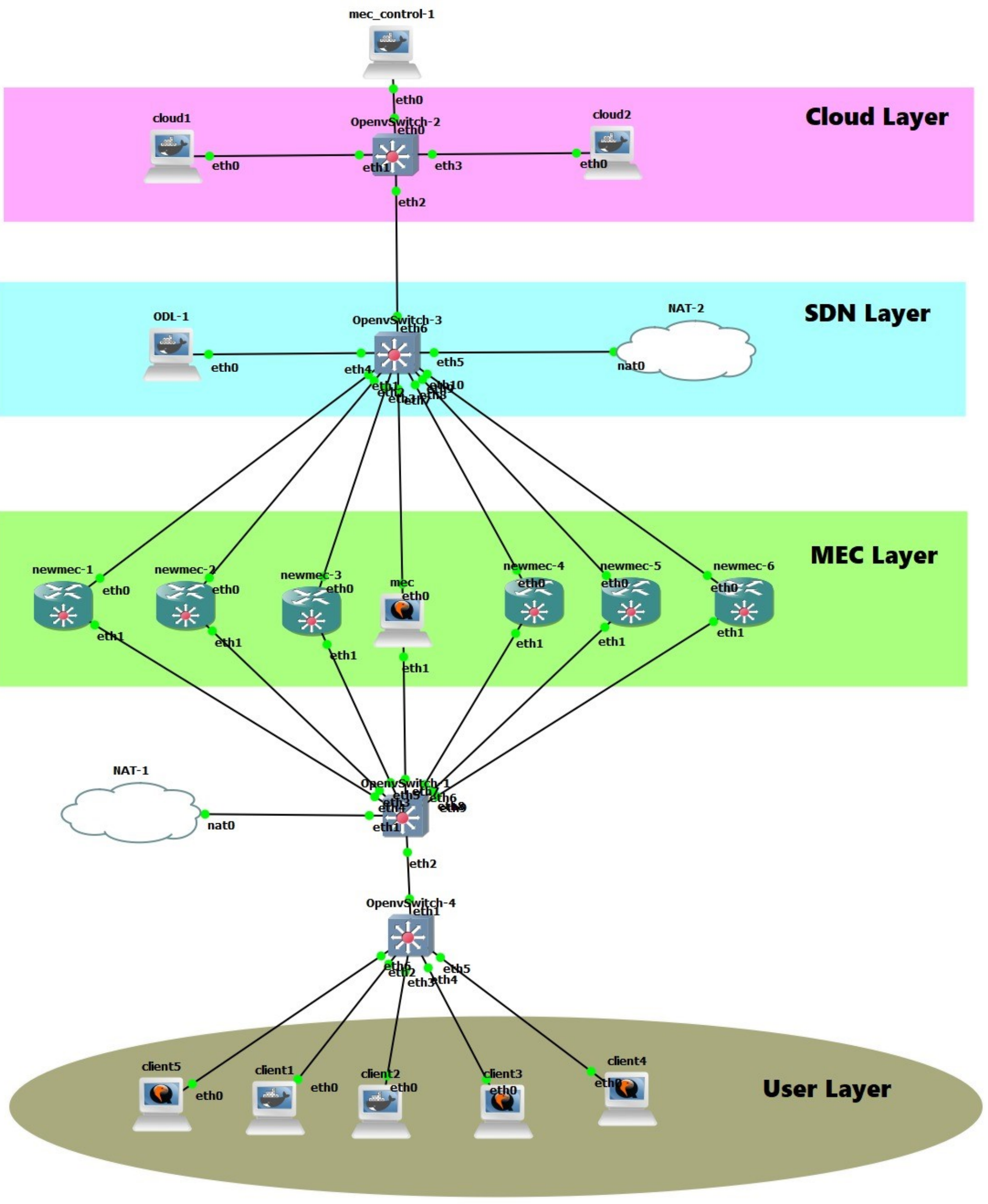

*Figure 45 Deadlock Testbed Architecture*

[12] https://github.com/emylincon/deadlock_project
[13] https://www.gns3.com/

Figure 45 depicts the architecture used for testing the deployed algorithms. The first layer is the cloud layer. It has been emulated using Linux containers[14]. The second layer consists of the SDN controller and the NAT-2 network. The Gns3 NAT node[15] creates a local network which additionally allows the created topology internet access. Opendaylight[16] has been used as the SDN controller for the testbed. The MECs are Linux containers running an OpenVswitch[17] service. This allows communication with the SDN controller using OpenFlow[18]. OpenFlow is an SDN northbound communication protocol between SDN controllers and the forwarding plane of network devices. The testing algorithms have been deployed in the MECs for comparison. Finally, the end-user layer is also Linux desktops that submit tasks to the MEC for processing. The experiment has been conducted using two physical compute nodes which specifications can also be seen in Table 18. Each of the compute nodes has an Intel(R) Core (TM) i7-8550U processor. Compute 1 (*C1*) runs the GNS3 emulator software and GNS3 VM1 while GNS3 VM2 runs on compute 2 (*C2*).

*Table 18 System Specifications During Experiment*

| **Name** | **Operating System** | **CPU Cores** | **Memory** |
|---|---|---|---|
| ***Compute1 (C1)*** | Windows (x64) | 16 | 32GB |
| ***Compute2 (C1)*** | Windows (x64) | 8 | 16GB |
| ***MEC*** | Ubuntu (x64) | 0.5 | 512MB |
| ***End Device*** | Alpine (x64) | 0.4 | 400MB |
| ***GNS3 VM1 (C1)*** | Ubuntu 18.04 (x64) | 8 | 16GB |
| ***GNS3 VM2 (C2)*** | Ubuntu 18.04 (x64) | 4 | 8GB |
| **Software** | **GNS3 Emulator** | | |

## *1.2 IMPLEMENTATION*

In this section, the deployment and implementation setup are discussed. This includes a setup for each of the nodes in the 3-tier architecture.

### *1.2.1 UNIQUE IDS*

Unique ids have been used to identify each task that is sent through the system. These ids include the task id, MEC id and the client id.

[14] https://linuxcontainers.org/
[15] https://docs.gns3.com/docs/using-gns3/advanced/the-nat-node/
[16] https://www.opendaylight.org/
[17] https://www.openvswitch.org/
[18] https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/

**TASK ID**: task id is denoted by $task_{id}$. The tasks are stored in a hash table which holds the capacity, period and deadline of the tasks. The $task_{id}$ is the key in the hash table. From the example below the keys in the hash table are $'t1',' t2', etc$. These keys are the $task_{id}$. There are 5 unique types of tasks.

```
tasks = {'t1': {'wcet': 3, 'period': 20, 'deadline': 15},
         't2': {'wcet': 1, 'period': 5, 'deadline': 4},
         't3': {'wcet': 2, 'period': 10, 'deadline': 8},
         't4': {'wcet': 1, 'period': 10, 'deadline': 9},
         't5': {'wcet': 3, 'period': 15, 'deadline': 12}
         }
```

**MEC ID**: $mec_{id}$ uniquely identifies a given MEC node. The IP address can be used as the $mec_{id}$ to make sure each id is unique. However, in this experiment, the nodes have been configured in such a way that the last octet of the IP address is unique. Therefore, this has been used because this was enough to uniquely identify each node in this experimental set-up.

**CLIENT ID**: The $client_{id}$ is used to uniquely identify each user end device. This is obtained the same way the $mec_{id}$ is obtained. The last octet of the IP address. The Ip address has been statically configured to ensure that these unique IDs are not duplicated in this experiment.

**SEQUENCE NO:** $sequence_{id}$ is the sequence number of each task. When sending tasks from the client to the MEC for execution, the client starts allocating sequence from 0 to n. where n is the last batch of tasks sent.

### *1.2.2 USER LAYER*

Initialization of the task is done in the user layer. To ensure that each task that is sent by the user is uniquely identified and can travel through the network layers and make it back to the client sender, a unique naming nomenclature was used.

Task name: the $task_{name} = task_{id}.mec_{id}.client_{id}.sequence_{no}$. An example of this is $t2.110.170.1$

Where $t2\ is\ task_{name}$, $110\ is\ the\ mec_{id}$, $170\ is\ the\ client_{id}$ and 1 is the $sequence_{no}$. Therefore, the task is sent from client 170 to MEC 110.

#### *1.2.2.1 TASK GENERATION:*

For a task to be sent to a MEC node for execution a record of the task must be stored in $task_{record}$. The appropriate metadata necessary for the task execution must also be recorded and sent with the task to the MEC. The metadata includes:

- **RMS scheduling requirements**: this includes the capacity, period and deadline. This is stored in the $tasks$ hash table which holds the requirement where the key is the $task_{id}$ and the values is another hash table that contains details about the task capacity, period and deadline. Therefore, the requirements of a task type can be obtained from the hash table if the task type ($task_{id}$) is known. Therefore the $task_{id}$ is sent with the task. $task = \{task_{id}: \{capacity: c, period: p, deadline: d\}\}$

- **Deadlock requirements**: These are parameters required by the deadlock algorithm which is the resource need. Each task type ($task_{id}$) has a maximum need stored in a need hash table. The need hash table stores a vector of the types of resources required for the task to be executed. In this experiment setup, we assume that the resource type required for each task type include CPU memory and storage. Therefore, $need = \{task_{id}: [CPU, Memory, Storage]\}$
- **Time constraint**: each task has two, time constraints which include execution time and the latency requirement. Since each task is simulated, therefore the execution time of the task and the latency requirement is also simulated. These time constraints are assumed to follow a generic distribution. Therefore, each task, $task_{id} = [execution_{time}, latency]$. For this experimental setup, these times are assumed to be in milliseconds.

#### *1.2.2.2 TASK METADATA*

With the above metadata of the RMS, deadlock and time requirements, the task is sent to the MEC for execution.

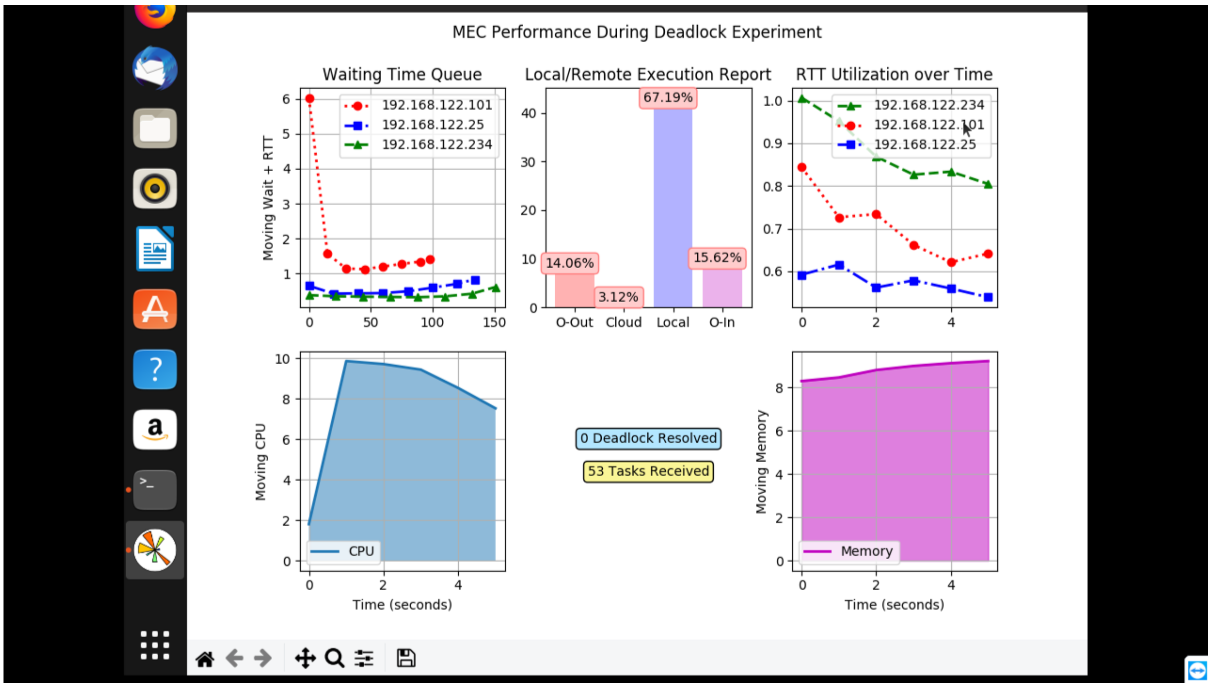

*Figure 46 MEC Runtime Event Monitoring Screen*

$$task_{meta_{data}} = \left\{ task_{id}: \left\{ \begin{matrix} scheduling: \{capacity: c, period: p, deadline: d\}, \\ time: [execution_{time}, latency], \\ deadlock: [CPU, Memory, Storage] \end{matrix} \right\} \right\}$$

Here the metadata is a JSON object that contains all the details of the task.

#### *1.2.2.3 TASK RECORD*

The client node keeps a record of all the tasks sent, when the task is sent and, what is the maximum deadline. It uses this record to compare when the task is executed and received back to ascertain if the task executed within the deadline.

### *1.2.3 MEC LAYER*

Each MEC maintains a task queue where tasks are dequeued and executed. Before execution, the tasks are scheduled using the given real-time algorithm scheduler (RMS or EDF) and then checked for deadlock using a given prevention or avoidance mechanism. After this, the waiting time for each task is calculated and then checked with the task deadline constraint. If this is feasible in the MEC then the task is executed in the MEC else, the task is sent to a MEC where the deadline constraint is satisfied. If no such MEC is found, the task is sent to the cloud for execution. Figure 46 shows the live event monitoring screen for each MEC during algorithm runtime.

### *1.2.4 CO-OPERATIVE MEC COMMUNICATION*

For the MECs to make appropriate decisions on which MEC node to re-offload tasks to for execution, they maintain a communication mechanism where they send their current average queue waiting time, RTT and current resource utilization to each MEC in the area periodically. The waiting time is crucial during the algorithm run time because it is one of the factors used by each MEC to determine which of the neighbouring MEC is suitable for task re-offload if need be. The waiting time here is obtained for each of the MEC periodically, similar to how the RTT is obtained. The waiting time is determined from the perspective of one MEC to another. If a MEC $M_i$ requires the waiting time for another MEC $M_j$. Then the waiting time $WT_{m_i \to m_j} = rtt_{m_i \to m_j} + Q_{mj}$ . Where $rtt_{m_i \to m_j}$ is the round-trip time from $M_i\ to\ M_j$ and $Q_{m_j}$ is the queue waiting time for $M_j$. $Q_{m_j}$ is periodically sent from each MEC to all neighbouring MEC for waiting time calculation. During reoffloading, the MEC with the lowest waiting time is always selected to evenly balance the load across the MEC platform.

If the MEC with the lowest waiting time cannot satisfy the constraint of the task, the task is then sent to the cloud for execution.

### *1.2.5 BROKER BASED COMMUNICATION*

Communication between MEC and user node and communication between MEC and cloud is done with broker-based communication. For simulation and experimental purposes this setup was chosen but any means of communication would work as well.

### *1.2.6 MULTICAST COMMUNICATION*

Exchange of cooperative messages Communication within the MEC platform is done with multicast communication. These messages include the periodic waiting item and resource utilization

### *1.2.7 CLOUD LAYER*

In the cloud layer, tasks that cannot be executed in the MEC node is executed and sent back to the appropriate MEC which then forwards the task back to the user.

## *APPENDIX 2 CACHING TESTBED CHRCA*

The aim of the testbed is to create a system that can record event traces during the deployed caching algorithm runtime. The source code and details of how to deploy and run the algorithms are available on the GitHub page[19].

### *2.1 GNS3 ARCHITECTURE*

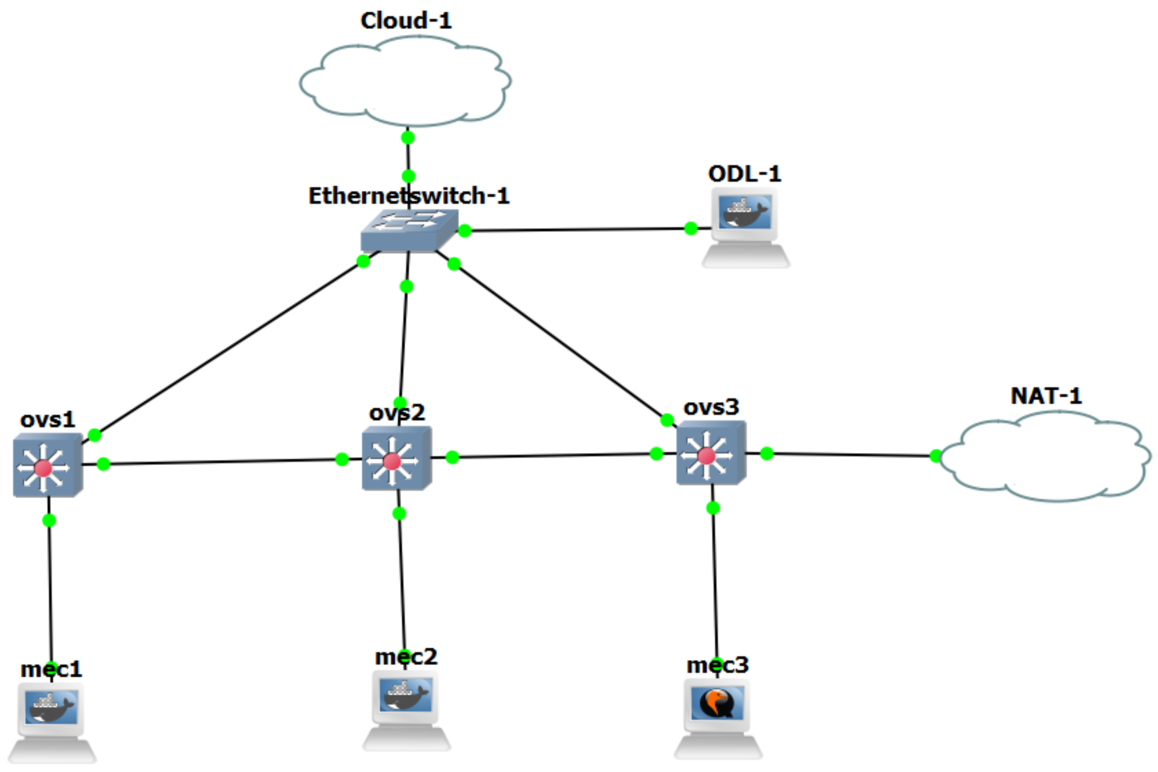

*Figure 47 GNS3 Caching Testbed for CHRCA*

Figure 47 depicts the architecture used for testing the deployed algorithms. An SDN network has been used for this testbed. SDN network is not a requirement for the testing nor the algorithms. However, it is a feature of the testbed. The figure shows 3 MEC nodes connected to a multilayer OVS switch. The management port of each OVS switch is connected to the ODL SDN controller. Each of the MEC nodes is Linux containers running the algorithms to be compared. Two networks are used in the testbed supplied by the GNS3 NAT and GNS3 Cloud. The GNS3 cloud network is used to connect the OVS management ports to the ODL SDN controller. However, the GNS3 NAT gives internet access to the testbed which is used by the MEC nodes to send a request to the webserver deployed on OpenStack Compute if a request is not found in the collaborative MEC.

### *2.2 IMPLEMENTATION*

The algorithms deployed have been written using python. The two main components of the testbed are discussed in this section. These are the webserver setup and the MEC algorithm deployment.

[19] https://github.com/emylincon/caching_project

#### 2.2.1.1 WEBSERVER

The webserver was deployed on the OpenStack Compute. A VM was created with 1 vCPU and 2GB memory. Nginx has been used as the webserver. The Nginx configuration used can be seen in GitHub[20]. Arbitrary webpages were created on the webserver to serve as the total content pool of requests for the MEC nodes.

### 2.2.2 MEC

The MEC nodes are Linux docker containers running the compared algorithms. Each algorithm has been written in python and it consists of 4 major sections which include the initialization, the communication, the event record, and the cache database.

#### 2.2.2.1 INITIALIZATION

This includes the initialization of the variables of the algorithm previously discussed. This includes the number of requests, cache size, window size, and number of MEC. Additionally, the initialization includes the discovery of MEC nodes in the cluster to establish a connection to share cache information. Two communication modes have been used during the experiment which would be explained in the next section.

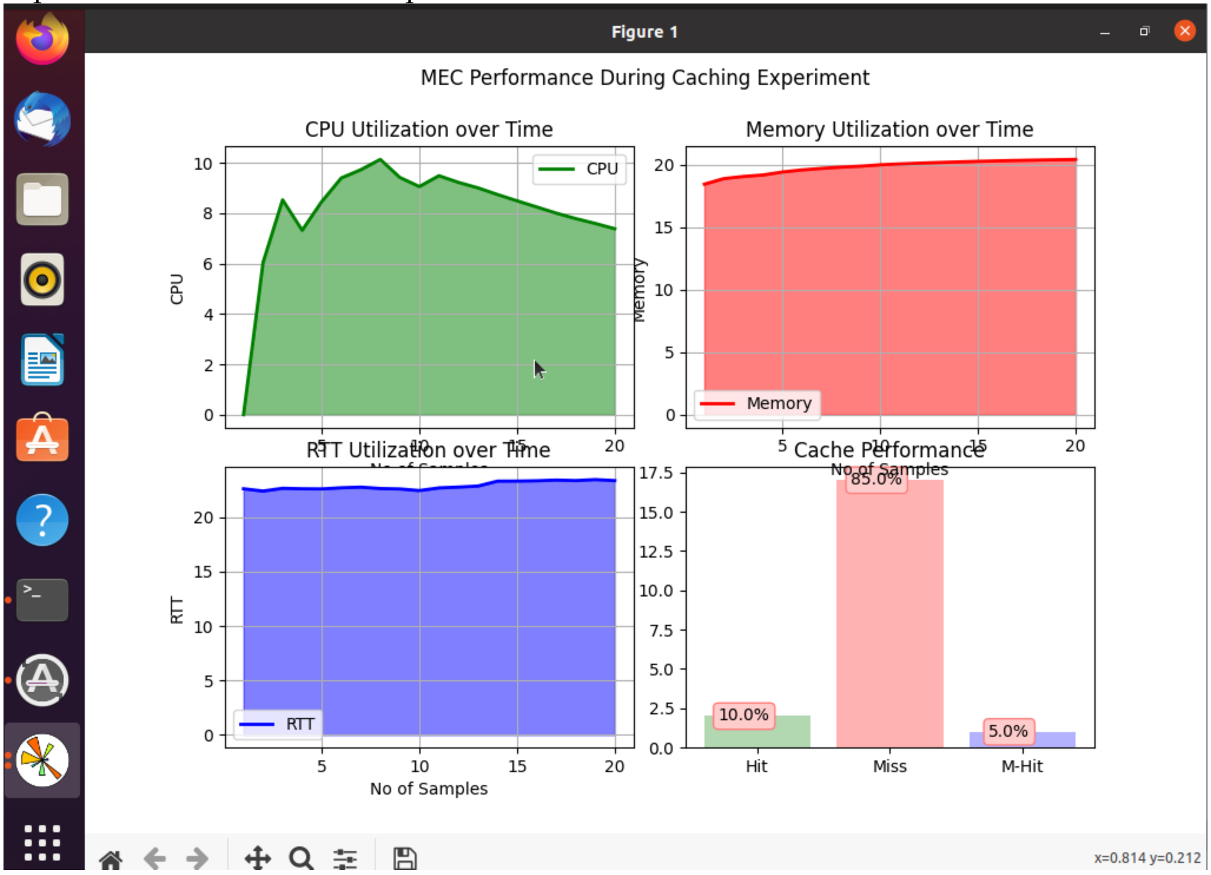
Figure 1
MEC Performance During Caching Experiment
CPU Utilization over Time
CPU
Memory Utilization over Time
Memory
RTT Utilization over Time
RTT
No of Samples
Cache Performance
85.0%
10.0%
5.0%
Hit
Miss
M-Hit
x=0.814 y=0.212

*Figure 48 Event Monitoring display for CHRCA Testbed*

#### 2.2.2.2 COMMUNICATION

During the algorithm runtime, the MEC requires communication with other nodes in the MEC cluster for node discovery. Additionally, communication is required to share event-driven updates about each MEC cache status. These events are triggered during cache evictions and

[20] https://github.com/emylincon/nginx

cache insertions. Communication during MEC node discovery is achieved using a multicast group. Each MEC in the cluster is subscribed to the same multicast group. In this scenario, each MEC node sends a hello message during the initialization phase to the multicast group. The hello message includes the IP address and the hostname of the MEC. This information is used to identify each MEC for further communication. Event-driven updates are achieved using Secure Shell (SSH) connection communication. The SSH connection has been implemented using python's Paramiko[21] module. In hindsight, a broker based communication would have been more appropriate in this situation.

#### *2.2.2.3 EVENT RECORD*

The event-driven records are used to monitor the experiment progress and present a report at the end of each experiment. The events monitored include cache hits, cache misses, CPU utilization and RTT. The RTT here refers to the RTT from the MEC node to the webserver measured in milliseconds. Figure 48 shows how the live event monitoring display looks like during the algorithm runtime.

#### *2.2.2.4 CACHE DATABASE*

A distributed cache database has been used in this scenario. Here each MEC maintains its own database. However, the database is updated during runtime based on the event-driven updates by other MECs in the cluster regarding cache status updates.

[21] http://docs.paramiko.org/en/stable/

## *APPENDIX 3 CACHE-TRACE - CACHING TOOL SIMULATOR*

The Cache-Trace tool simulator has been developed to benchmark cache replacement algorithms. The aim of the tool has been to create an easy-to-use tool where users can easily enter their experimental parameters and obtain an appropriate result. The tool has been deployed and available online at Cache-Trace[22]. Figure 49 depicts the home page of the tool. The caching algorithms available for comparison in the application include Least Frequently Used (LFU), Least Recently Used (LRU), First In First Out (FIFO), Least Frequently Recently Used (LFRU), Multi Queue (MQ), Frequency Based Replacement (FBR) and Optimal Page Replacement (OPR). Each algorithm has been written in python. The source code of the project is available on GitHub[23].

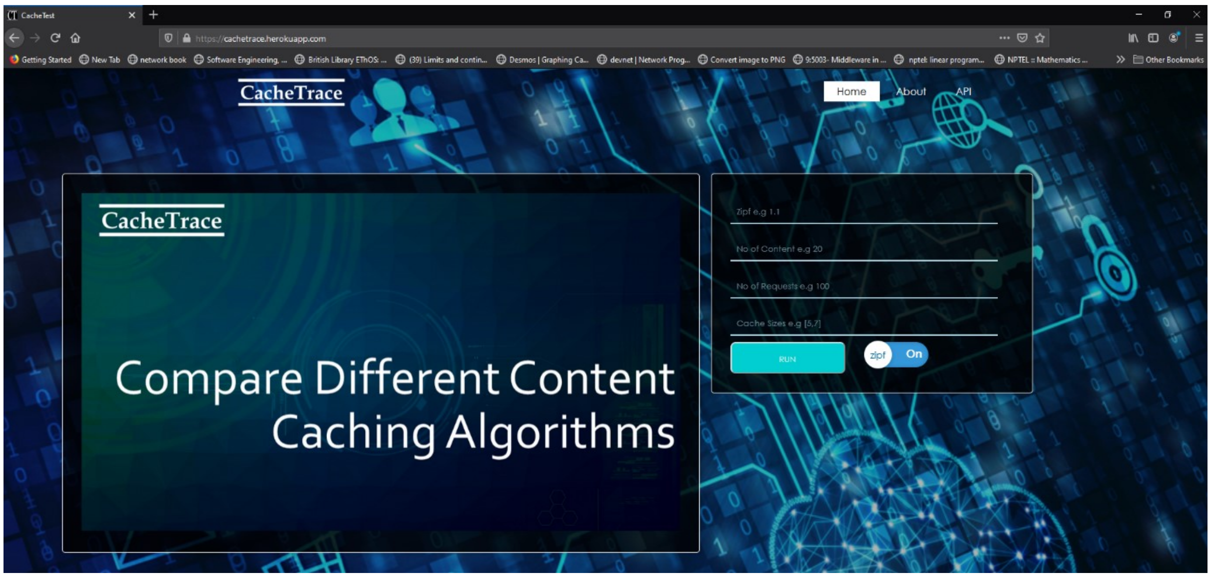

*Figure 49 Cache-Trace Tool Home Page*

### *3.1 HOW TO USE*

The application works by specifying the parameters as wanted. This includes the Zipf parameter, number of contents, number of requests and cache sizes specified in an array format as shown in Figure 50.

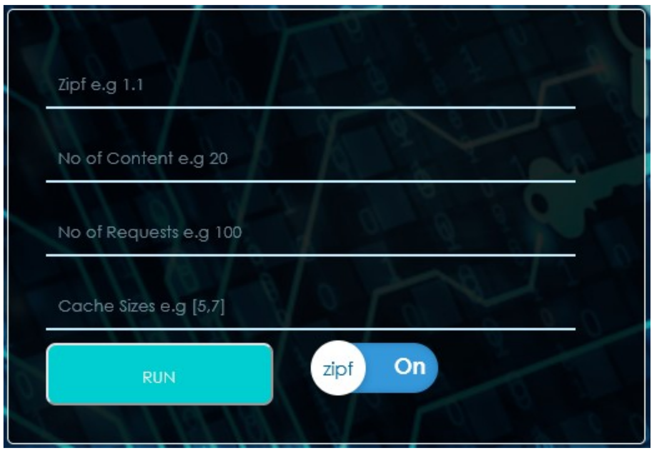

*Figure 50 Cache-Trace Parameter Input Form*

[22] https://cachetrace.herokuapp.com/
[23] https://github.com/emylincon/cachetrace

If you wish to upload custom requests, you can do so by turning off the Zipf switch. Only a CSV file with a limit size of 1MB can be uploaded. Additionally, the CSV file must have a column with the name 'requests' which is a series with integer datatype.
The obtained result will be displayed as seen in Figure 51. A downloadable JSON file is available after each run as seen in Figure 51. API support for the tool is available and the documentation can be accessed on the website[24].

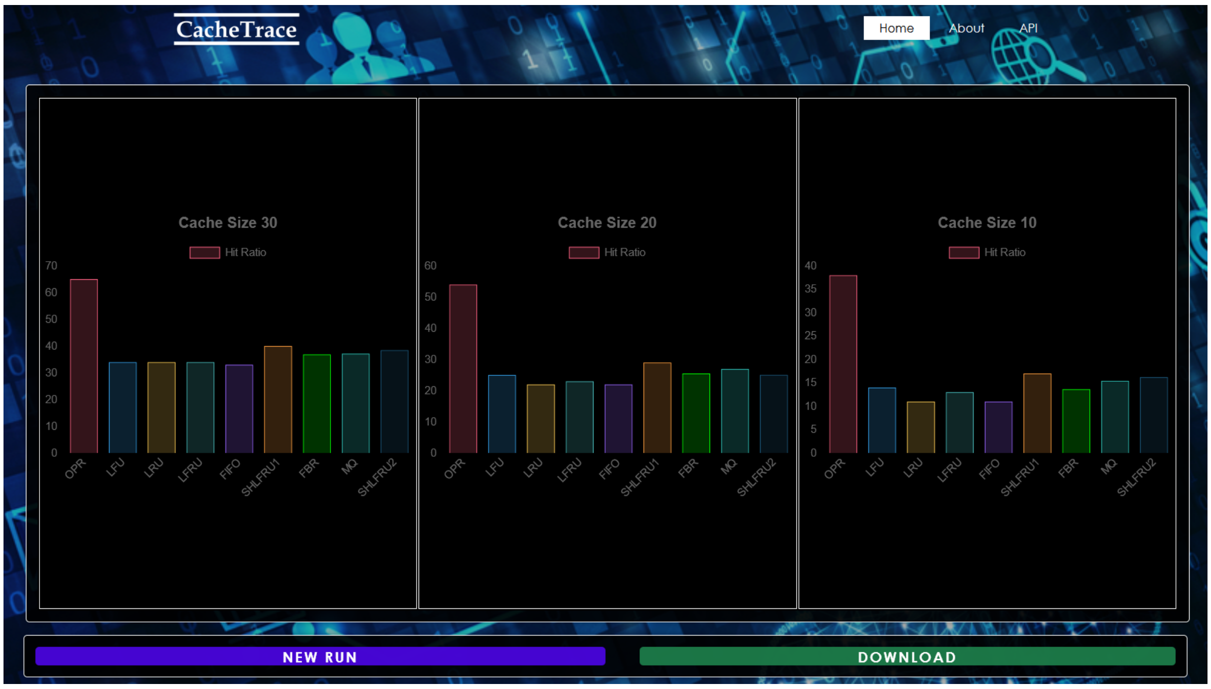

*Figure 51 Cache-Trace Result Display Page*

[24] https://cachetrace.herokuapp.com/app-api

## *APPENDIX 4 CACHING EXPERIMENT FOR PCR COMPARISON*

The aim of the testbed is to create a system that can record event traces during the deployed caching algorithm runtime. Details of how to deploy and run the algorithms are available on the GitHub[25].

### *4.1 GNS3 ARCHITECTURE*

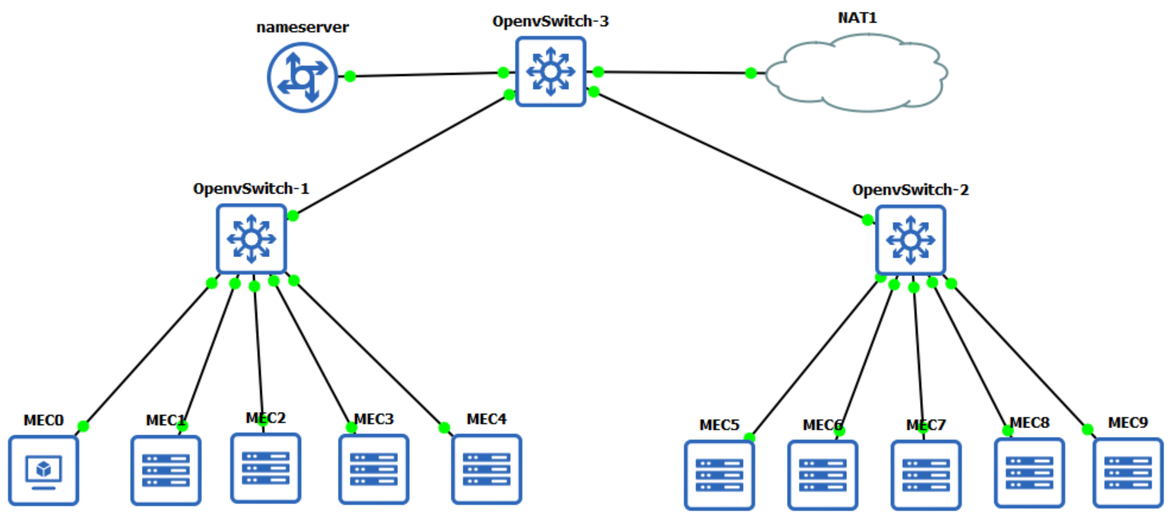

*Figure 52 PCR Testbed GNS3 Architecture*

Figure 52 depicts the GNS3 architecture used for the testbed. The figure shows 10 MEC nodes that are connected to OVS switches and a nameserver. The GNS3 NAT gives internet access to the testbed which is used by the MEC nodes to send the request to the webserver. The MEC nodes and the nameserver are docker containers.

### *4.2 IMPLEMENTATION*

The algorithms deployed have been written using python. Two frameworks have been tested using this testbed. Therefore, the implementation of both frameworks will be discussed. The first framework is a deep learning-based predictive edge caching algorithm[26] that has been aliased as C-LSTM and the second one is the proposed framework PCR.

#### *4.2.1 C-LSTM*

This caching framework uses a Recurrent Neural Network (RNN) model using Long Short-Term Memory (LSTM) cells trained on the MovieLens 20M dataset[27]. LSTM is a deep learning RNN architecture that uses feedback connections, unlike the standard feedforward neural networks. LSTM network is a well-known solution to overcome the vanishing gradient problem. The C-LSTM replacement framework works by replacing the least predicted popular cache object in the cache. The popular cache is determined using the number of clicks by users on a video/movie content (which is the hit rate count). This has been estimated in the research as the cumulative user ratings count for each movie. The implementation source

[25] https://github.com/emylincon/caching
[26] https://ieeexplore.ieee.org/document/9016437
[27] https://grouplens.org/datasets/movielens/20m/

code of this framework is available on GitHub[28]. The next section discusses the implementation of the LSTM model.

#### 4.2.1.1 LSTM MODEL DESIGN

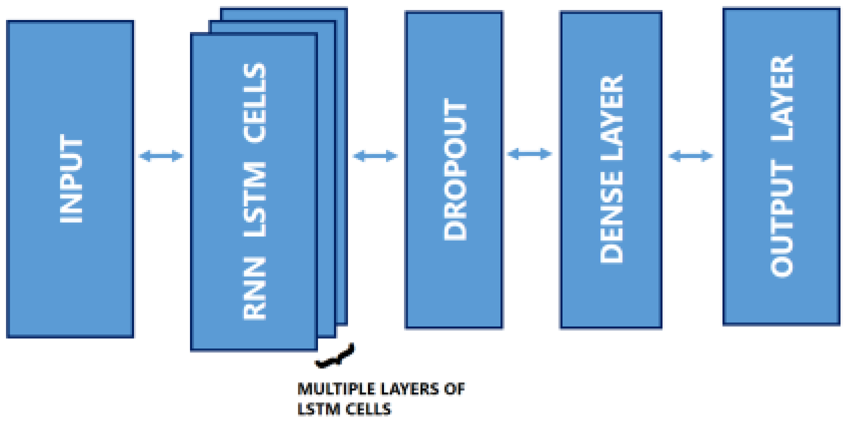

*Figure 53 LSTM Model Design for C-LSTM*

Figure 53 depicts the LSTM model design for C-LSTM framework to predict the popularity of each movie based on the cumulative ratings achieve. The design includes an input layer, multiple LSTM layers, a dropout layer, a dense layer, and an output layer. The hyperparameter values are the same as the authors of the paper specified and are summarised as follows:

- 2 LSTM layer containing 50 LSTM cells.
- 1 Dropout Layer with value 0.075
- batch size of 5000
- 1100 epochs
- Nadam optimizer

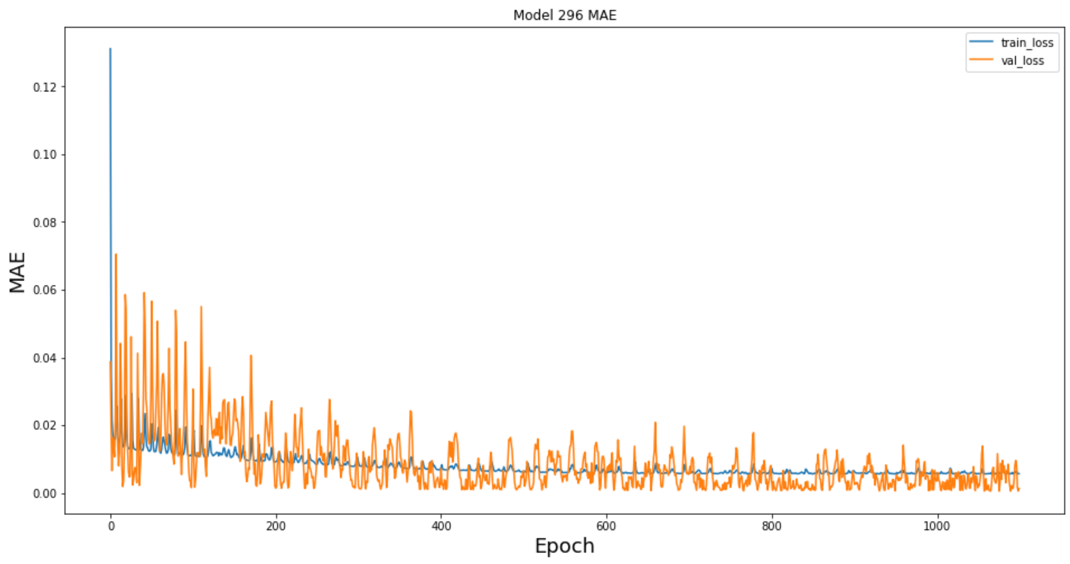

*Figure 54 Plot of change of Mean Absolute Error during training of Model 296 for 1100 Epochs for training and validation.*

[28] https://github.com/emylincon/caching/tree/master/LSTM_caching

#### *4.2.1.2 MODEL PERFORMANCE VALIDATION*

In this section, the performance of the models obtained has been evaluated in terms of validation Mean Absolute Error (MAE) and validation Mean Square Error (MSE). Additionally, the trained model is used to predict values from the holdout test data. The notebook used to generate the model is publicly available in Google Colab[29].

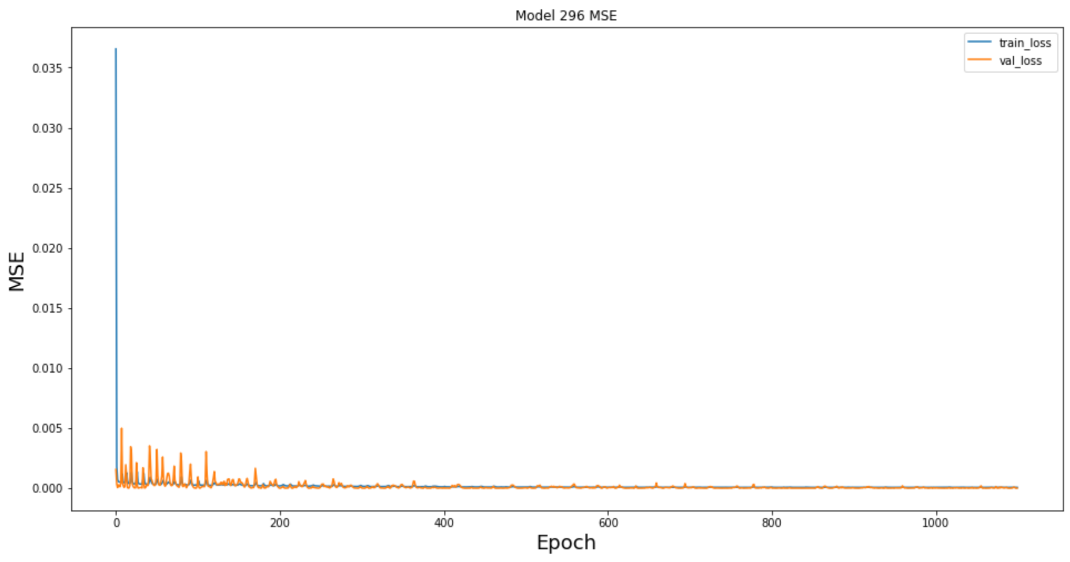

*Figure 56 Plot of change of Mean Square Error during training of Model 296 for 1100 Epochs for training and validation*

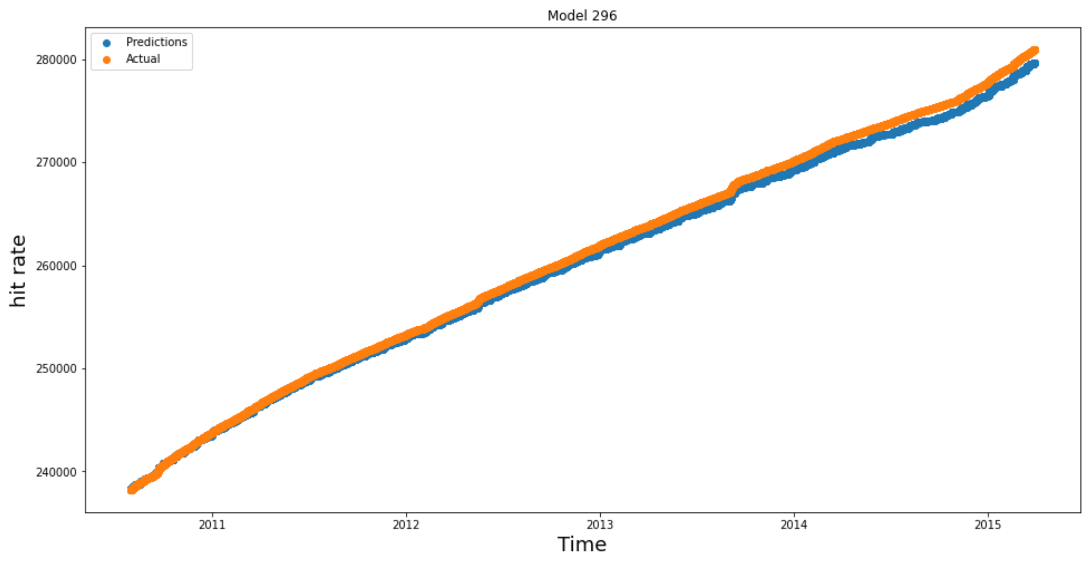

*Figure 55 Plot of the Test data with the prediction results to validate the performance of the model 296 on the holdout unseen data.*

Figure 54, Figure 58, and Figure 59 shows a plot of change in the Mean Absolute Error during training over 1100 Epochs. It can be observed that the MSE reduces as the number of Epochs increases. This same behaviour has been observed for Mean Square Error in Figure 56, Figure

[29] https://colab.research.google.com/drive/1Ak8A7wQVMGiWBk-1gBpYikgfPUPUCpJG?usp=sharing

57, and Figure 62. Furthermore, Figure 55, Figure 60, and Figure 61 show the comparison between the validation data and the predicted data using the trained models. It can be observed that the obtained predicted results are close to the actual result.

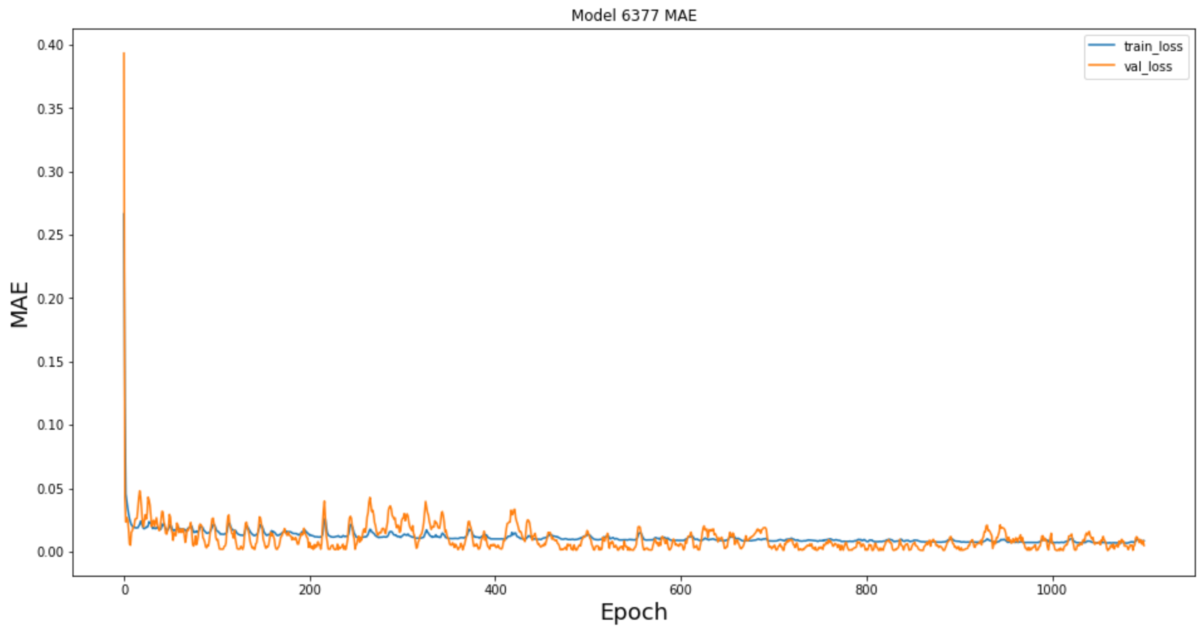

*Figure 58 Plot of change of Mean Absolute Error during training of Model 6377 for 1100 Epochs for training and validation loss.*

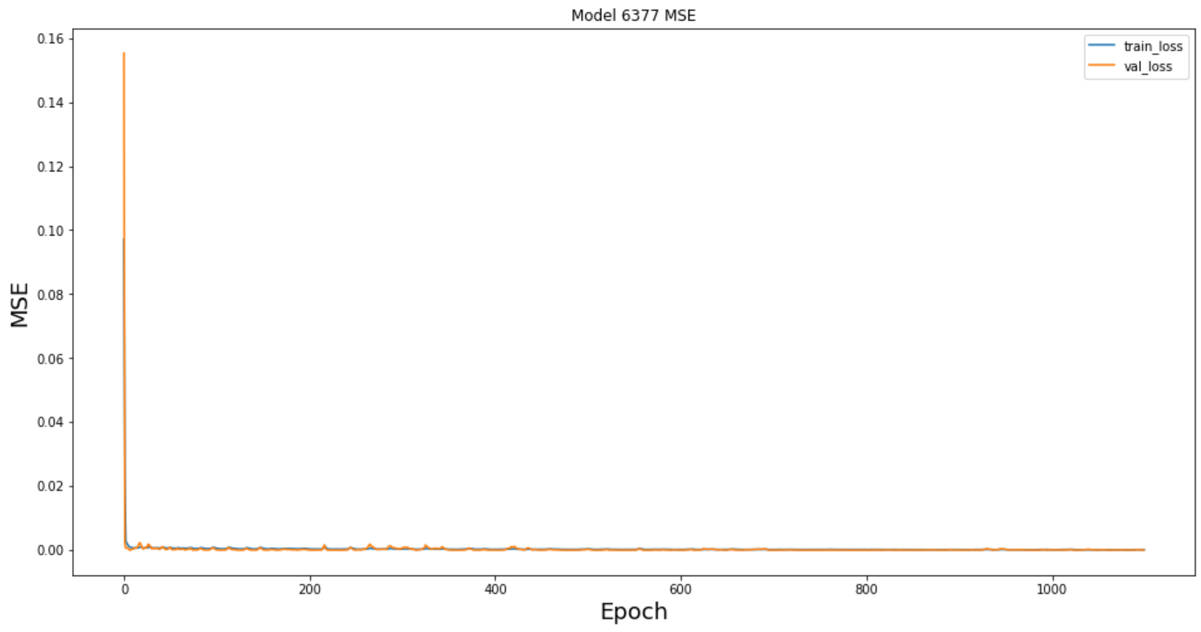

*Figure 57 Plot of change of Mean Square Error during training of Model 6377 for 1100 Epochs for training and validation loss.*

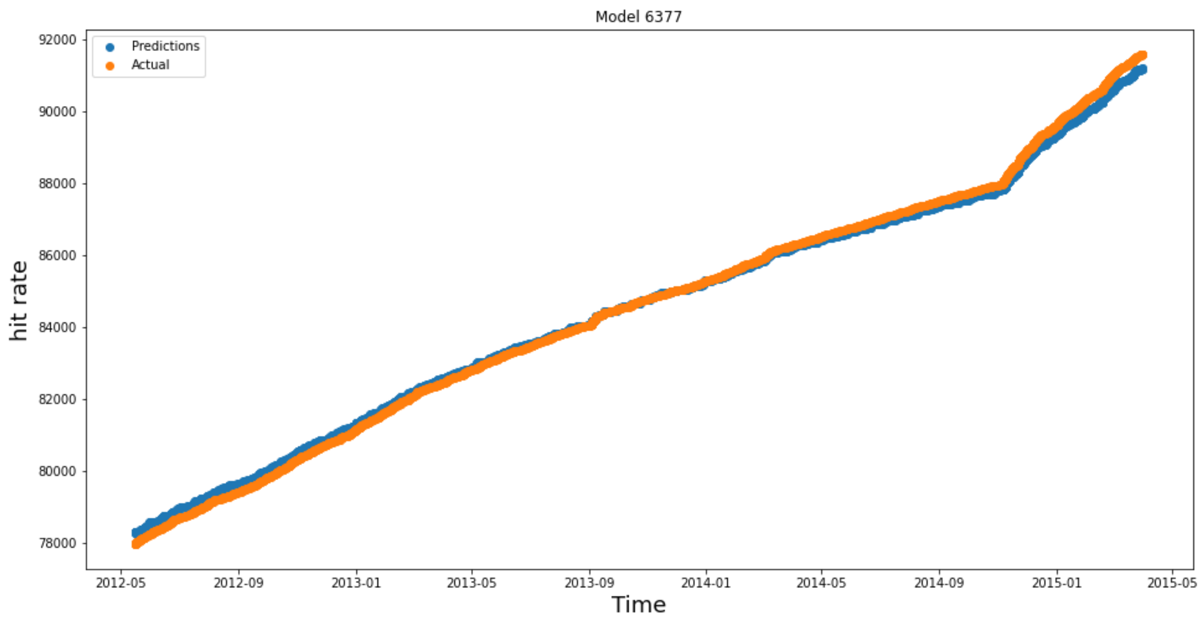

*Figure 60 Plot of the Test data with the prediction results to validate the performance of the model 6377 on the holdout unseen data.*

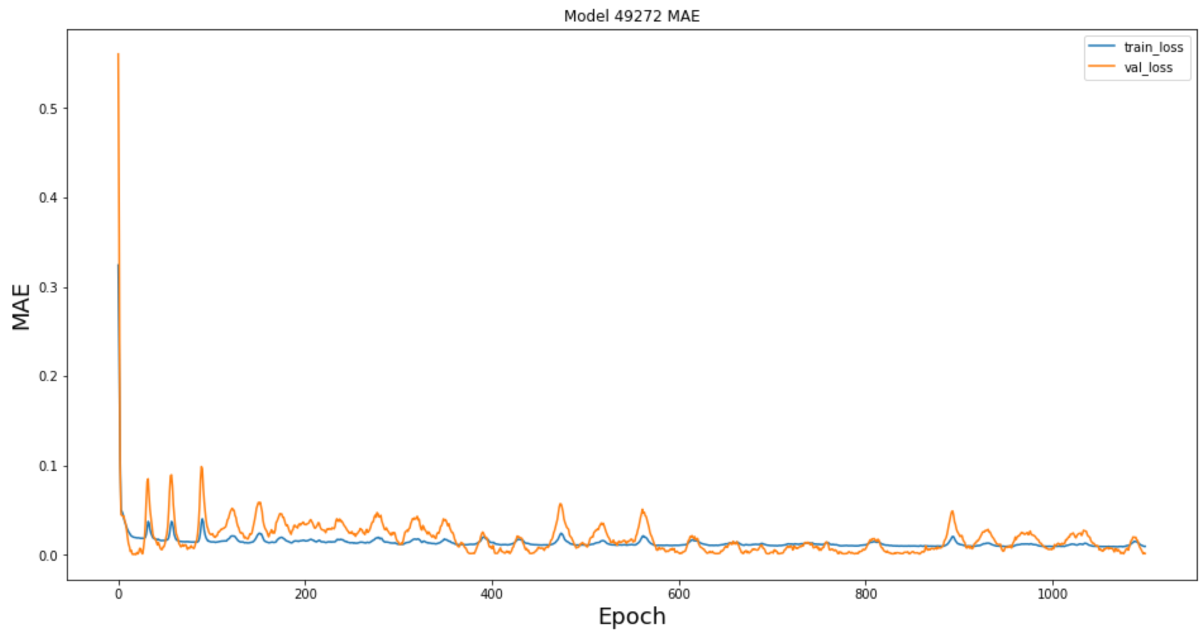

*Figure 59 Plot of change of Mean Absolute Error during training of Model 49272 for 1100 Epochs for training and validation loss.*

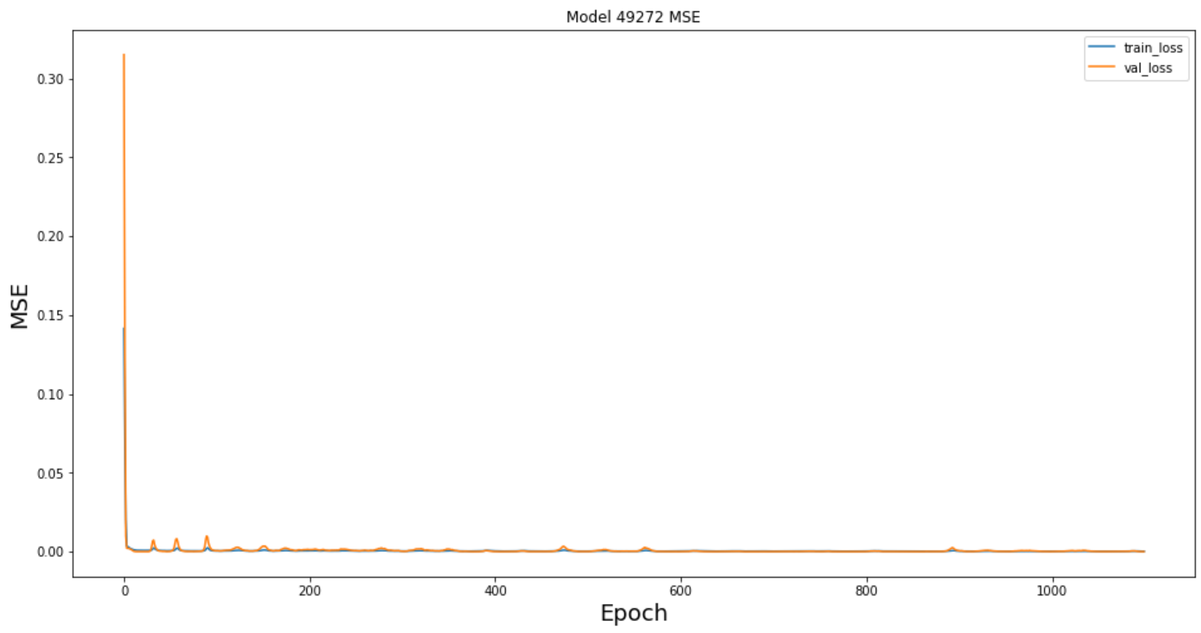

*Figure 62 Plot of change of Mean Square Error during training of Model 49272 for 1100 Epochs for training and validation loss.*

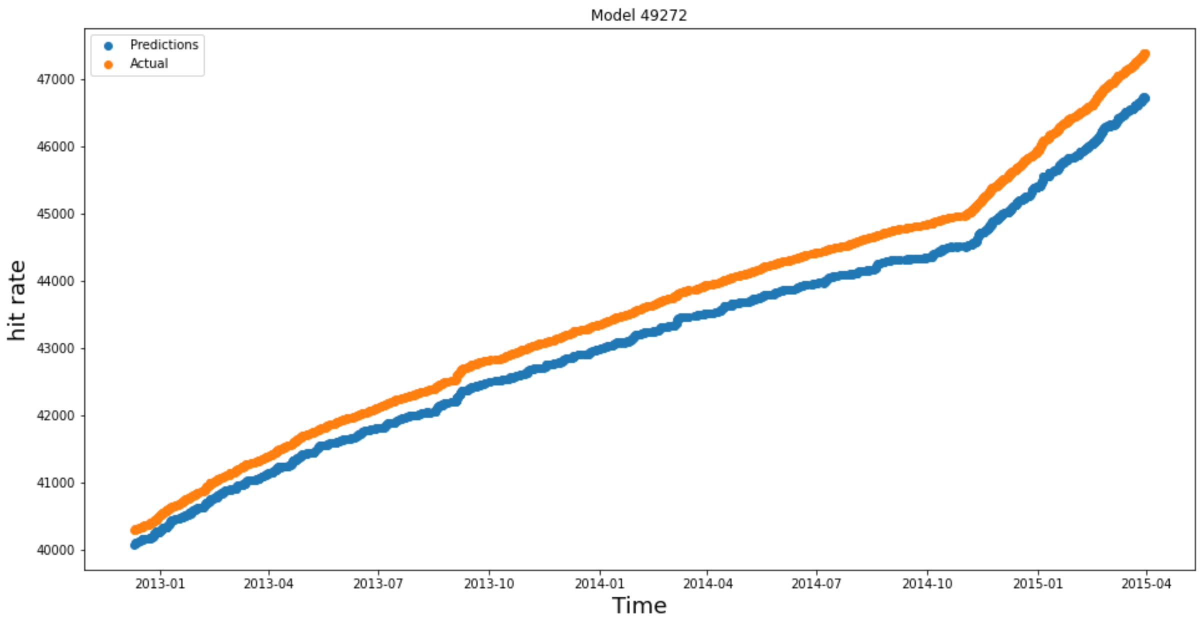

*Figure 61 Plot of the Test data with the prediction results to validate the performance of the model 49272 on the holdout unseen data.*

| | **C-LSTM Paper Performance** | | **Trained model Performance** | |
|---|---|---|---|---|
| **Model** | **Val MSE** | **Val MAE** | **Val MSE** | **Val MAE** |
| MovieID 296 | 0.00015 | 0.0087 | $2.0058^{-6}$ | 0.0012952 |
| MovieID 6377 | 0.00014 | 0.0084 | $2.4488^{-5}$ | 0.0048711 |
| MovieID 499272 | 0.00021 | 0.010 | $4.4619^{-6}$ | 0.0016634 |

*Table 19 Comparison of the trained Models performance metrics with the obtained metrics from the C-LSTM paper*

Table 19 shows the numeric comparison of the trained Models performance metrics with the obtained metrics from the C-LSTM paper. It can be seen that the performance of the trained models is slightly better than the performance of the model from the C-LSTM paper.

### 4.2.2 PCR

In this section, the key implementation components of PCR are discussed. This includes the nameserver implementation, communication, cache file sharing and event record.

#### 4.2.2.1 WEBSERVER

A webserver has been used to host the web contents that would be requested by the MEC nodes during the algorithm runtime. The webserver hosts static contents of similar sizes which have been numbered from page 0 to page 1070. The web contents have been hosted using *netlify*[30] free service. The hosted web contents are available online using the following footnote[31]. The Figure 63 depicts the hosted website home page with the static pages accessible on the right side of the page.

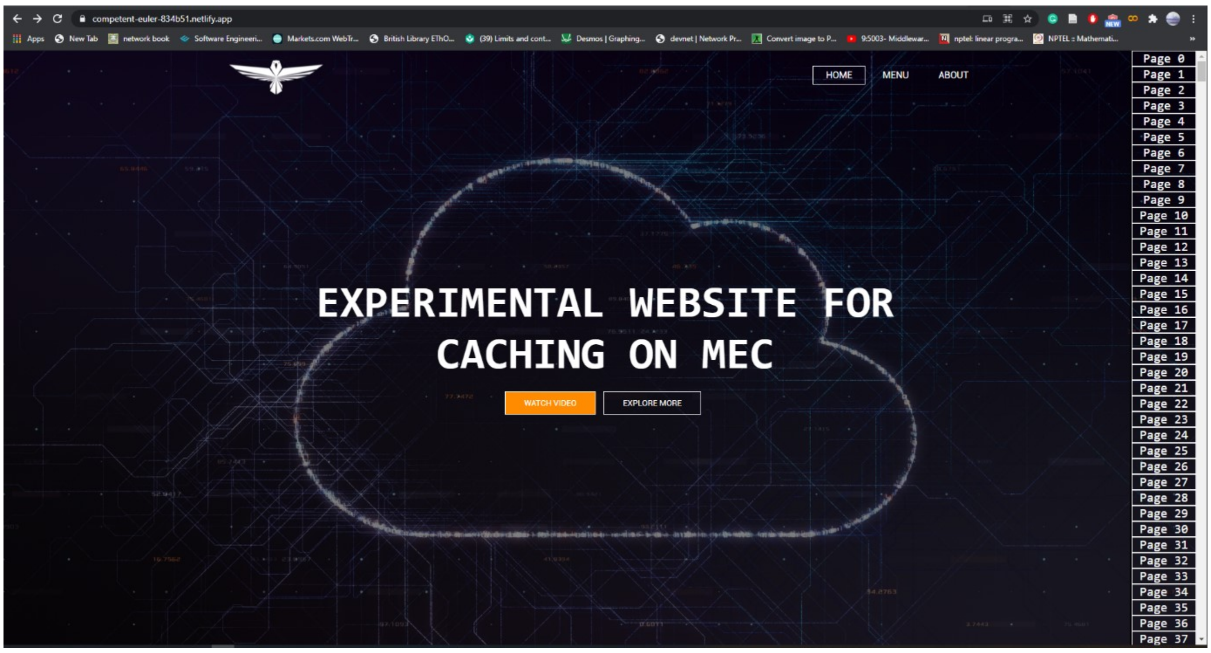

*Figure 63 Website with hosted static web contents for the experiment. The static pages can be seen as outlined in the right*

#### 4.2.2.2 NAMESERVER

The purpose of the nameserver is to serve as a cache hash lookup similar to how the DNS works. It translates a given content hash to the equivalent URL and vice versa. Additionally, it can add a new record to the database given the appropriate content hash and the URL. This has been implemented as a RESTFUL API using python. Therefore, a HTTPS request can be sent to the nameserver to the appropriate endpoint for a response. The source code and usage of the API are available on GitHub[32].

[30] https://www.netlify.com/
[31] https://competent-euler-834b51.netlify.app/
[32] https://github.com/emylincon/caching_chain

#### *4.2.2.3 COMMUNICATION AMONG MEC*

Communication among the MEC nodes in the cluster is achieved using a messaging broker. This communication is used to update the other MEC nodes in the cluster about the data stored in the cache. Update messages are sent after insertion and replacement from the cache. MQTT[33] messaging broker has been used to achieve this. MQTT communication is based on a publish-subscribe architecture. In this scenario, each client subscribes to a topic, and they will receive all messages that have been broadcasted to that topic.

#### *4.2.2.4 FILESHARING*

File sharing is required to share saved cache objects among other MEC nodes for collaborative caching. This has been achieved using File Transfer Protocol (FTP). In this scenario, each MEC hosts an FTP server on which it stores its saved cache objects. Therefore, each MEC node can request a file using the corresponding content hash ID.

#### *4.2.2.5 EVENT RECORD*

The event-driven records are used to monitor the experiment progress and present a report at the end of each experiment. The events monitored include cache hits, cache misses, CPU utilization, memory utilization, and RTT. The RTT here refers to the RTT from the MEC node to the webserver measured in milliseconds. Figure 48 shows how the live event monitoring display looks like during the algorithm runtime.

[33] https://mqtt.org/