Efficient Computational Offloading of Dependent Tasks in Mobile Edge Networks

by

Mohammed Mufareh A Maray

Thesis

Submitted to the University of **W**arwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

December 2021

Contents

Acknowledgments

I would like to express my thanks and deep gratitude to my supervisor and mentor, Dr. Arshad Jhumka, whose guidance, encouragement and support have been invaluable to me during my time at the Department of Computer Science at the University of Warwick. He taught me how to have different thinking strategies and how to have novel solutions. This work would have not been done without his advice and valuable comments. I look forward to maintaining our collaboration in the future.

I would like to thank my advisors Professor Hakan Ferhatosmanoglu and Dr. Gihan Mudalige at the Computer Science department at the University of Warwick for their helpful comments and feedback at my annual review. Also, I would like to thank my research collaborators at Maryland University (USA), and Warwick University. I would like to express my heartfelt appreciation to Professor Mohamed Younis and Dr. Adam Chester. They provided constructive feedback that improved my papers significantly and suggested ideas to make my PhD work more interesting. Also, it led to the successful publication conference paper in 2019.

Last but not least, for their support during this PhD I also want to thank my father Mofareh Alqhtani, and my mother Sharah Alqhtani, who were the permanent source of love, encouragements, and support. Their prayers are the only reason behind my success. Foremost amongst the individuals to whom I am thankful is my wife, Fawziah Alqhtani, for her overwhelming love, ceaseless patience, and continuous motivation. Being away from her parents does not prevent her to all our family with gladness during our long journey to accomplish my goal. My children, Saud, Mofareh, Abdullah and Sharah, are the ones who make my life colourful, enjoyable, and full of fun. I am indebted to my brothers and sisters for their support, help, and love.

Dedication

To my parents....I am proud of you

To my wife.........I am with you

To my children...I am for you

Declarations

1. Publication

Parts of this thesis have been previously published (or accepted) by the author in the following:

- *Maray, M., Jhumka, A., Chester, A., and Younis, M. (2019, October). Scheduling Dependent Tasks in Edge Networks.In 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (pp.1-9). IEEE.*

- *Mohammad Maray, Arshad Jhumka, Adam Chester and Mohamed Younis.Mobility-Aware Scheduling of Dependent Tasks in Mobile Edge Computing.Journal of Parallel and Distributed Computing (JPDC).* under review

- *Mohammed Maray and Arshad Jhumka.A Fully Distributed Computational Offloading Algorithm for Mobile Edge Computing.Future Generation Computer Systems (FGCS).* under review

2. Sponsorships and Grants

The research presented in this thesis was made possible by the support of the following benefactors and sources:

- *King Khalid University (KKU).*

- *Saudi Arabia Embassy in the U.K.*

Abstract

Mobile Edge Network (MEN) is emerging as a novel computing paradigm that puts high storage and computational power within easy reach of mobile users for a range of applications such as big data applications and location-based services. The MENs consist of a number of small base stations, which we call *Cloudlets*, that provide the required services to end-users. Ecosystems are resource-constrained, making execution of resource-hungry applications challenging. Computation offload between ecosystems and cloudlets plays a key role in this vision and ensures that the integration between ecosystem and cloudlet is seamless with better quality of service such as lower latency. Analysis of the available literature relating to currently proposed offloading techniques focuses on *centralised* approaches with a small number of mostly *static* user devices hosting *in-dependent tasks.* In this thesis, we address three major offloading problems: (i) that algorithms consider distributed environment with multi offloading systems, (ii) users with ecosystems are mobile and (iii) tasks are dependent as (DAGs). We develop the *offloading algorithms* for mobile user devices with hosting of *dependent tasks*, where a dependent task cannot start until its immediate predecessor tasks have completed, with the aim of reducing completion latency. We start by formalizing the dependent task offloading problem as a constraint satisfaction problem with all proposed algorithms. While the first objective aims at minimising completion latency with central server in edge, the second objective aims at minimising completion latency with multi systems in distributed environment. We construct optimisation models for both objectives and develop two offloading algorithms to approximate the optimal solution. According to the results with ns-3 simulation, Optimisation CPLEX, and real deployment, our offloading algorithms are able to efficiently produce allocation schemes that are close to optimal during the offloading.

List of Tables

List of Figures

Acronyms

3G Third Generation.

AP Access Point.

API Application Programming Interface.

CISC Complex Instruction Set Computer.

CPU Central Processing Unit.

DAG Directed Acyclic Graph.

DARPA Defense Advanced Research Project Agency.

GHz Giga Hertz Unit.

GPS Global Position System.

ILP Integer Linear Programming.

LTE Long Term Evolution.

MANET Mobile Ad-hoc Network.

MCC Mobile Cloud Computing.

MEC Mobile Edge Computing.

MEN Mobile Edge Network.

MWS Mobile Web Service.

NS-3 Network Simulator Version 3.

QOE Quality of Experience.

RISC Reduced Instruction Set Computer.

sBSs Small Base Stations.

SOA Service Oriented Architecture.

UDP User Datagram Protocol.

Symbols

J	The job as set of tasks
T	The set of tasks
U	The set of users
D	The dependencies between the tasks
C	The set of cloudlets in the edge
μ	A mobility pattern as sequence of sets of cloudlets
S	A scheduling function
r	Denotes the result of the task
s	Denotes the source of the results
d	Denotes the recipient node of the result
ST	The start time of a task
OT	The offloading time of a task
ET	The execution time of a task
WT	The waiting time of a task in the buffer
FT	The finish time of a task
Q	The state of the queue at a given period of time
TT	The transfer time of a task between edge nodes
R	Communication range function
τ	A given time
RT	The ready time of a task
AOT	The actual offloading time of task
ERT	The ready execution time of a task
AET	The actual execution time of a task
DT	The downloading time of a task
CT	The completion time of a task

Chapter 1

Introduction

Nowadays, smart mobile devices such as smartphones are becoming the platform of choice for both personal computing and business needs. It is claimed that, by 2026, the number of smartphone users will be over 7 billion [1]. It is believed that humans, on average, spend 35% of their time on smartphone playing games while they spend over 50% of their time on smart tablets playing games. At the same time, it is projected that the number of connected cars will reach 400 million by 2025, up from 237 million currently [2]. Vehicle users will be expecting their vehicles to be smart, running applications that will make their vehicles safer.

1.1 Motivation

The (predicted) surge in the usage of smart and connected devices is resulting in novel applications being executed on them. For example, the type of applications that people are running on their smartphones or smart devices are compute-intensive, such as live streaming, Augmented Reality/Virtual Reality (AR/VR) and image processing. Similarly, smart cars are expected to run compute-intensive applications such as driving assistance systems and collision avoidance among others, that may require deep learning techniques that are notoriously resource-hungry.

The growth in such resource-hungry and real-time applications has led to a corresponding increase in the computational requirements on hardware (HW) to support such applications as Augmented Reality, driver assistance, Real-Time Analytics, Face Recognition and Gaming [101]. Often, these applications may be hosted on Internet-of-Things (IoT) devices such as smartphones or on relatively resource-constrained HW. The HW limitations of these devices such as battery power (IoT devices) or memory capacity to store data and, most importantly, processing capacity (both IoT and cars) need to be handled properly in the era of computationally-intensive applications [57]. Computation offloading [28] has been proposed as one possible solution, where a remote infrastructure consisting of computers with significant power is used to support the resource-constrained devices, thereby encouraging the users of such devices to transfer their resource-intensive jobs to the remote infrastructure for execution. Such a remote infrastructure can be cloud environment or a similar environment closer to the user, in what is called an edge network [83].

As the type of applications that are becoming increasingly popular (e.g., AR/VR) and also safety-critical (e.g., driver assistance systems) are also time critical, it is important that the communication latency is bounded, to ensure responsiveness of the applications. As such, the main difference between offloading to a cloud computing environment and to an edge network environment lies essentially in the network latency that may be experienced. As cloud computing environments are typically remote, far removed from where users are generally located, it is expected that the communication latency is quite high. This thus makes edge network environments more suitable due to their close proximity to the IoT and vehicular networks. It is expected that, with the commercialisation of 5G networks, the data latency in edge networks can be reduced to less than 10ms, making such deployment suitable for critical real-time applications.

When the users of the IoT devices and smart vehicles are mobile, the edge network (or cloudlet) infrastructure is known as mobile edge computing (MEC) [8]. The problem of computational offloading in the presence of mobile users present some important challenges. For example, a user may offload a task onto an edge node for computation but is no longer within the communication range of that node when the result is ready. Another challenge is that the user has only a partial view of the network, i.e., the user will not know whether there are more suitable edge nodes to offload a task to. These issues can be mitigated if the network topology has a certain structure which, unfortunately, however do not always scale well. For example, it has been typically assumed that some information about the MEC environment is available in a central repository, e.g., [111]. As such, recent works in MEC mostly focused on solving the offloading problem with users being static during task offloading, so that the communication links between mobile devices and edge nodes are always available, e.g., [14, 52, 121, 123]. However, such an availability assumption is not valid, with mobility users presenting a significant challenge in MEC networks that need to be addressed.

Thus, it is becoming essential to be able to address the challenge of computational offloading the mobile edge networks. However, it is also important to determine the type of jobs to be offloaded. Most current works assume that tasks are independent, i.e., there are no data or control dependencies among the tasks. However, a majority of the modern applications are typically composed of dependent tasks. For example, applications with dependent tasks are included in more than 75% of 4 million applications in the Alibaba data-trace [67]. Another example of a video processing application from Facebook has a set of constrained tasks to complete the video [126]. Also, the real world examples of the Global Positioning System GPS navigation and smart parking system have dependent tasks at runtime. As such, the offloading of dependent tasks to the edge networks with make the offloading mechanism more complex and challenging. Most current studies have not solved the problem of offloading of dependent tasks to the edge in the presence of user mobility [52, 67, 98, 121]. The process of uploading, downloading and execution the dependent tasks are seriously affected by constraints of the dependent tasks during the offloading process. For example, the output of an executing task might be used as the input for other tasks that depend on it, which makes the order of execution in the multi edge nodes challenging [71]. More specifically, a task may not start until it receives its inputs from other tasks. By then, a mobile user may no longer be in the range of an edge node to execute the now ready task.

We consider the problem of offloading of jobs that consist of dependent tasks, which we represent as directed acyclic graphs (DAGs), onto heterogeneous computational resources of the edge network so as to minimise the completion time of the job. The task set with dependent tasks has precedence constraints i.e. a child task cannot be begun the execution before the full completion of all its parent tasks.

1.2 Computation Offloading Mechanism

Computation offloading mechanism is a solution which drives the mobile devices to offload the computational functions to a remote-based server. The server has huge computation resources and is able to perform the operations faster than local resources of the mobile device and where remote execution of offloaded computation is performed. Computation offloading mechanism can be done in the cloud side or the edge side to save more energy of smartphones and getting high performance in running the intensive applications that need more resources. Computation offloading can be classified into two categories: fine-grained mobile code offloading structure , which is also known as partial offloading scheme and the coarse-grained offloading which is also known as a full offloading scheme [26, 56]. However it is important for the offloading process to be cognisant of any dependency that may exist between tasks. This is especially important when users are mobile, otherwise significant delay or latency can be introduced due to tasks being blocked waiting for outputs of other tasks. The real world examples of applications with dependent task are the Global Positioning System GPS navigation and smart parking system which have fully dependent tasks in the run time.

1.3 Elastic Resource Provisioning for Cloud

Users have been executing applications in the cloud. However, the cloud platform is not in the vicinity of the mobile devices during the offloading process which causes a huge timing overhead during the communication between nodes. Latency is the main issue to have the offloading to the cloud and also the need for more energy during the offloading process to the cloud. The majority of studies in mobile cloud computing (MCC) addressed mainly the problem of offloading tasks from a mobile device to a cloud server with the assumption of the stable network environment during the offloading process [21, 56, 85]. Most of these offloading frameworks focused on two major problems involved with tasks assignment: (i) the application partitioning problem [22, 23, 54, 55], and (ii) the offloading decision problem [19, 42, 111, 119]. However, the latency during the offloading in MCC makes the offloading process slow with a huge execution time in the cloud.

1.4 Computational Offloading in Edge Networks

To overcome the latency during the offloading to the cloud, there is a new technology which is called edge networks. The edge is considerable computing power resides at the edge of the network and it is closer to the mobile users. The critical feature of edge network is its requirement for small latency and offer of high workload capacity while being near the user and their devices. The transmission and computational delays are found to be minimal in the edge, as these are nearer to the users unlike the remote resources of offloading to the cloud. Edge network provides flexibility in the task of offloading of mobile tasks which otherwise would be subject to delay constraints [81]. Recent works and researches are focusing on making the computation offloading to the edge to meet the requirement of the minimal delay during the offloading process as [62, 64, 111, 130].

A vast number of works have been proposed in the literature. However, most of the algorithms were designed for centralised environments with considering the communication between mobile devices and edge servers are always reliable without considering user mobility and dependent tasks during the offloading, such as in [52, 67, 98, 121]. New efforts are required to develop offloading mechanisms in distributed environment to address the dependent tasks offloading during the user mobility in multi systems.

1.5 Contributions

The problem we solve in this work is *the development of offloading algorithms for jobs with dependent tasks onto edge networks in the presence of device mobility.* As such, the contributions made in this thesis address the novel problem of offloading dependent tasks onto edge networks during the user mobility. Our focus is to minimize the completion time of a job in the edge (compared to the job executing on the device locally).

In support of this thesis the following contributions are made:

- In Chapter (4), we have defined and analysed the current challenges of the problem of computational offloading in distributed environment. We presented a simple case study to show the offloading process with dependent tasks which is represented as a directed acyclic graph[1](DAG).

- In Chapter (5), we make the following contributions: (i) we formulate the scheduling problem of dependent tasks in MEN as a constraint satisfaction problem, (ii) we provide a heuristic offloading algorithm that attempts to reduce the computation time of dependent tasks in MEN, (iii) we develop the Integer Linear Programming (ILP) model to optimize the solution in CPLEX Solver, (iv) We run simulation experiments using ns-3 to gauge the effectiveness of our approach in terms of reduction in computation time compared to execution on the local device and finally, (v) we run a real deployment at Warwick University with Flask server and a Face Recognition Application.

- In Chapter (6), we make the following contributions: (i) we provide a formalisation of the dependent tasks offloading problem as an optimisation problem, (ii) we develop a fully distributed offloading algorithm for the offload-able dependent tasks in a MEC network, (iii) we conduct extensive experiments using the ns-3 simulation engine to evaluate the effectiveness of our distributed offloading algorithm in terms of minimising the task completion time in the edge, (iv) we study the performance of our distributed offloading algorithm compared to the base case of offloading to a central cloudlet, and (v) we also study the bottlenecks caused by the distributed algorithms, in this case, queue waiting time and we study the impact of mobility on the bottleneck.

[1]A directed acyclic graph (DAG) is the a directed graph structure that we use to model the applications in our work and it consists of a set of tasks with dependency constraints as : $A_i = (T_i, D_i)$.

1.6 Thesis Organisation

This chapter has introduced the problem of the offloading of dependent tasks during the user mobility on the edge networks with computational offloading process and stated the contributions this thesis makes. The remainder of this work is organised as follows:

- Chapter 2 presents a background of the previous works performed in developing techniques for computational offloading and other context relevant to task offloading for edge networks.
- Chapter 3 introduces the models used in this work and explains how the models are used in practice when the algorithms are simulated in ns-3 and the deployment.
- Chapters 4 to 6 present the main contributions of this work.
- Chapter 7 discusses the implications of this work and also includes a comparison between the techniques.
- Chapter 8 summarises conclusions and outlines the future direction of work in this area.

Chapter 2

Background and Literature Review

2.1 An Overview on Computational Offloading

Computation offloading technique nowadays is popularly used to tackle the ecosystems limitations and provide effective computation in remote resources i.e. cloud or cloudlets. The delay in task offloading is still found as a delay problem in the current offloading works [53, 70, 82, 111]. Most of the frameworks approach this problem as an optimisation problem with the main objective of minimising the delay time during the execution process. However, previous frameworks have not considered *the optimization of execution latency subject to the dependent tasks with user mobility* as noted by existing studies of offloading [69]. We develop offloading algorithms for multi-systems and dependent tasks with the aim of reducing the completion latency in distributed edge environment. Our focus is to minimize the task delay in the edge that will affect the completion time of offloadable tasks after the offloading decision is made. In this section, we present an overview of computational offloading in edge networks[1] as follows.

2.1.1 Evolution of Computational Offloading

There has been huge advancement and evolution in the field of computing technology. Despite the enhancements, the computational capacity and energy consumption of the ecosystems like smartphones or Internet-of-Things IOT devices are nowhere near that of powerful computing machines that use powerful CPUs. The growth of intensive and real-time applications, such as applications with Augmented Reality, Multimedia, Video Editing, Face Recognition, and Gaming, has increased the computational requirement and energy consumption of these ecosystems. The limitations of ecosystems such as low battery power, low capacity to store data, and most of all limited processing capacity need to be tackled at a fundamental level in the era of intensive applications development

[1]Edge networks in our thesis mean both the mobile cloud computing as a cloud side (MCC) and mobile edge computing as a cloudlet side (MEC).

[57]. Computation offloading mechanism has been the best available solution up to now, driving the ecosystems to offload the intensive computational functions to remote computation resources such as edge-based server as shown in Figure 2.1, which has huge computation resources and can perform the operations faster than local ecosystem resources [21, 28, 34, 56, 85]. As we see in Figure 2.1, the devices could be offload tasks to the cloudlet (edge node) which is close in the distance or to the Service Cloud which is away in the distance.

History of remote computation pointed toward the early 1990s when remote execution and inter process communications were beginning to emerge to utilize the resources in cluster computers at fullest and management of message-passing traffic [12, 40]. Despite the benefit of remote computation, the parallel running challenges diminished the popularity of the concept at that time. Nevertheless, the development of Internet provided a new pathway to develop further the concept of remote execution, which enabled the establishment of a new foundation called Service Oriented Architecture (SOA). Mobile Web Services (MWS) includes SOA along with portable devices, which enabled enhancement of the computation capability and saved energy by allowing the mutual share of services and software between mobile devices and other devices [102]. However, its reliance on a static network produced the drawback of unstable performance. Computation offloading with Mobile Cloud Computing (MCC) started late (2009) and it was based only on the mobile devices side and the main cloud server side to offload tasks [54]. However, in the computation offloading technique of MCC the main cloud side is not close to the mobile devices side during offloading operations, which leads to the latency problem on the middle-ware of media connection and the significant defect of rendering the user mobility impossible during the offloading task [22, 23, 54, 55]. At the end of (2014), Mobile Edge Computing (MEC) was introduced as a means to help resolve the latency problem that happens during the offloading process in MCC [81]. The characteristic feature of MEC is need of small latency and offer of high workload capacity while being near to the user and their devices [28]. The transmission and computational delays are found to be very small in the MECs, as these are nearest to the users unlike the remote resources of traditional computation offloading in MCC.

Computation offloading technique nowadays is popularly used to tackle the smart phone limitations and provide effective computation [57]. Traditional client-server architecture, grid computing, or multiprocessor system are some of the conventional system migrating their computation to their nearest server for the reduction of resources utilization, enhancement of the performance, and load balancing [85]. Since its introduction, utilization of computation offloading technique has been stretched beyond its initial scope. The computation offloading technique of mobile devices differs from traditional computation offloading technique in the sense that it does not utilitise only the resources available nearest to the mobile device. Instead, the offloading is done in the environment which is exclusively outside of the nearest computing environment available.

Computation offloading can be achieved in different levels of migration. These levels of offloading are dependent on the granularities offered by the partitioning mechanism being utilised. Method level, thread level, whole application level, tasks level, class level, object level, or whole virtual machine level are some of the offloading granularities offered by current application partitioning algorithms and offloading frameworks [90, 117]. Use of Java API for Remote Method Invocation, Remote Procedure Call, and .NET remoting played a significant role in enabling the offloading to object level and class level. These offload-able codes are then executed in remote cloud-server environments, where the states of the program are updated through execution, and the update is sent back to the mobile users.

Similarly, offloading mechanism involves the use of offloading engines, which provides functionalities such as partitioning of application, scheduling the partitions, and making suitable offloading decisions based on parameters gathered from profilers used in the offloading frameworks [21]. These engines are embedded local mobile devices or pre-defined devices in networks. These engines recognize the high energy and computation demanding tasks and with the help of profiler prepare suitably annotated partitions of application, and schedule them for remote execution. High bandwidth of the network, high-speed server devices, heavy computation, and lower data weights to be exchanged between two functions or processes are four requirements those should be satisfied for efficient computational offloading which provides enhanced performance speed and energy usage. And when designing frameworks and during comparative studies of developed frameworks, these requirements are made the essential objective to be achieved for optimal performance and energy gain [57].

2.1.2 Offloading Mechanism

Computation offloading mechanism is recognized generally as a transmission of computation data from one computing side to another computing side through various transmission media. Researchers in the area of computing past few years have studied the utilization of remote computation resources for improving performance and energy usage characteristics of mobile devices [38, 50]. Computation-offloading mechanism was the best solution, driving the ecosystems like mobile devices to offload the intensive computational tasks to the remote computation resources, which have huge computation capacity and can perform the operations faster than local mobile device resources such as

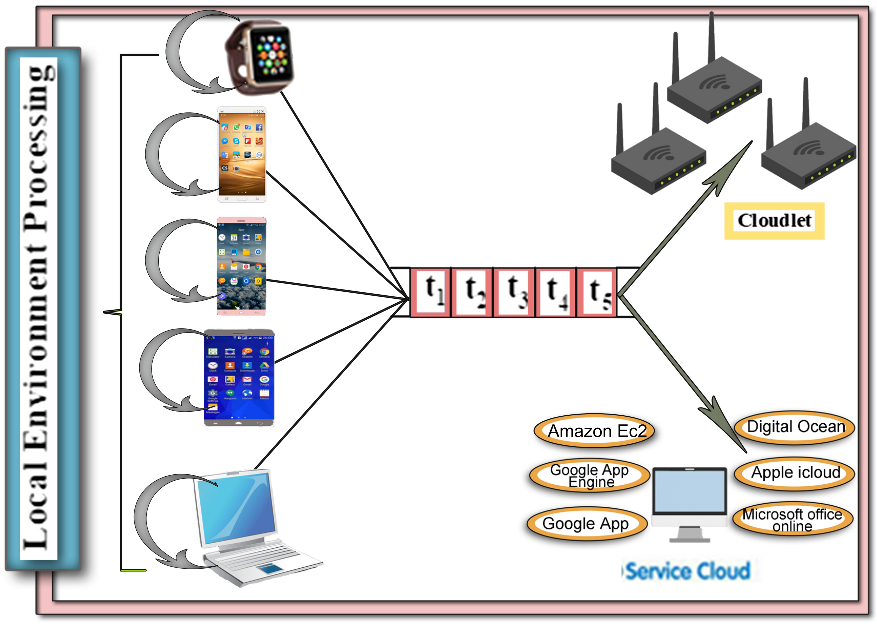


Figure 2.1: Computational Offloading in Cloud and Cloudlet

Mobile Cloud Computing MCC and Mobile Edge Computing MEC as shown in Figure 2.1 with cloudlet/ service cloud.

Although computation offloading in MCC/MEC is a part of remote computing technology, it differs from traditional client-server architecture regarding computational load, where the mobile devices could completely migrates its computation functions to the cloud or edge under any conditions. In addition, computation offloading in MCC/MEC differs also from the migration of computation in multiprocessors system, and grid system as well. Multiprocessors system migrates the computation for load balancing purpose, and the grid system migrates to the nearest resource available which is attached to the grid through some means and within the same environment [57, 130].

2.1.3 Offloading Granularities

Computation offloading can be classified into two granularities. First defines a fine-grained mobile code offloading structure [56], which is also known as partial offloading scheme. This approach relies on developers to annotate the offloading parts, within an application, and the main aim of this approach is to improve the efficiency of energy utilization in mobile devices. This aim is achieved by offloading annotated parts such as methods or thread to gain the energy utilization efficiency. Fine grain granularity is a useful offloading type for the applications that have tasks use the mobile device hardware and not possible to be offloaded outside the mobile devices such as using speaker or the screen of the mobile device. Coarse-grained offloading is the second granularity

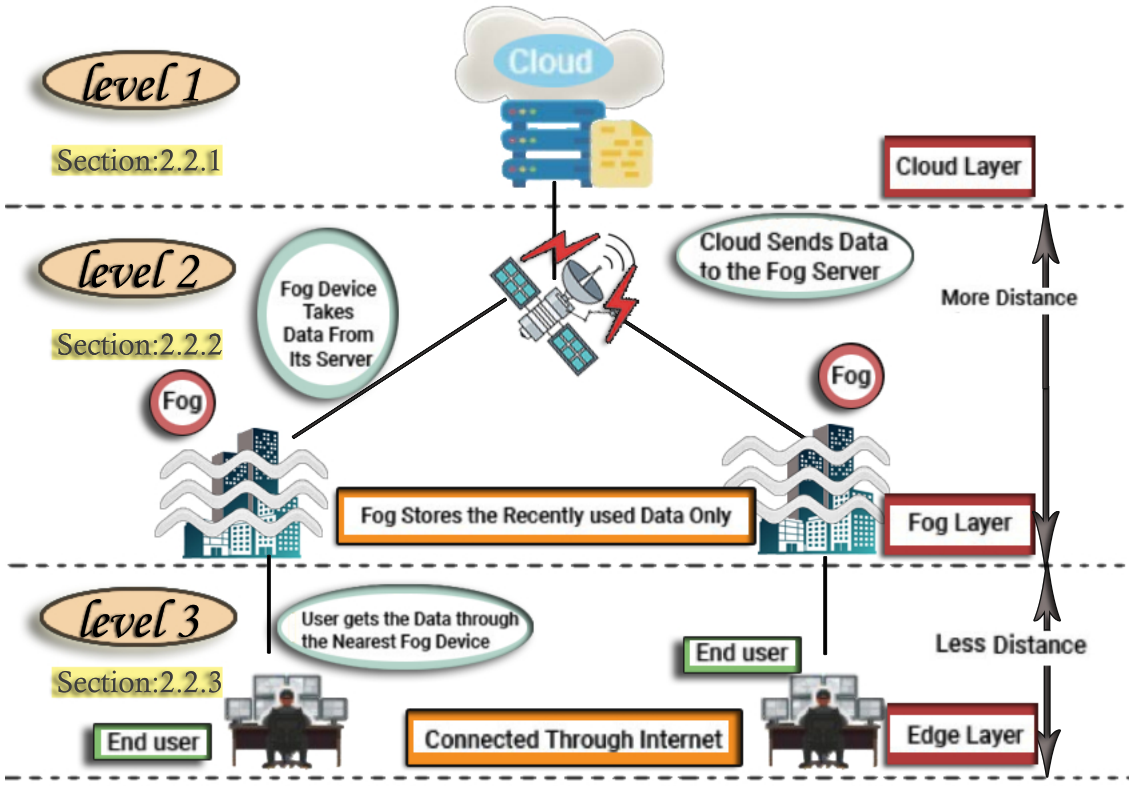

level 1
Section:2.2.1
Cloud
Cloud Layer
level 2
Section:2.2.2
Fog Device Takes Data From Its Server
Cloud Sends Data to the Fog Server
More Distance
Fog
Fog
Fog Stores the Recently used Data Only
Fog Layer
level 3
Section:2.2.3
User gets the Data through the Nearest Fog Device
End user
Less Distance
End user
Connected Through Internet
Edge Layer

Figure 2.2: Remote Computation Resources Levels

of this approach. In this approach, full application/program, or a process, or a whole virtual machine is offloaded to the remote computing resources and it is called full offloading approach [26]. [56][26]

2.2 Remote Computation Resources

In this section, we introduce the concepts of the remote computation resources i.e. Cloud Computing, Mobile Computing, Fog Computing, Mobile Cloud Computing, and Mobile Edge Computing as shown in Figure 2.2. We will introduce concepts of the remote computation resources as follows :

2.2.1 Mobile Cloud Computing (MCC)

Cloud computing is the centralized computation of the computing services within a single environment, allocating the neccessary portion of that environment as per service demand in one of three types of service: Software As a Service [SAAS], Platform As a Service [PAAS], or Infrastructure As a Service [IAAS] [4]. The services provided by the cloud are purely dependent upon what services have been demanded by the users. Service is determined by the type of the device that share resources and the offloaded functions and contents from user devices. This gave birth to the concept of mobile cloud computing. Mobile cloud computing (MCC) is the distributed computation of mobile applications, by offloading some of the computational functions to the cloud via network, within the single environment of the cloud providing the resources as per each user need as shown in level 1 of Figure 2.2. This produces an opportunistic use of Mobile Edge Computing (MEC) surrounding resources for the improvement of MCC functionality in the network issues since the edge is close to mobile devices side and the cloud is so far away from the mobile devices side [93].

Mobile computing is an execution of data and applications in portable devices and mobile devices, while the transfer of data between two or more mobile devices is known as mobile communication. Software, information, applications, and another form of technological instructions are deployed within a small portable device, which is distributed widely and connected through various sorts of wireless connections. The distributed resources, which are centrally located within each device are used, which are connected to each other through the use of mobile computing technology. Increasing popularity of mobile devices among people has increased expectation of quality and service level which they offer [5, 26].

Mobile cloud computing (MCC) is an emerging and innovative technology utilizing the unified resources of different clouds thus exploiting the elastic nature of the cloud computation, providing unlimited ever present services to mobile devices regardless of the location of service demands, and accommodating client service level demands [93]. These services are mutually shared between cloud side and mobile devices side through the network. MCC provides for a wide range of mobile device users an environment where computation processing and storage of mobile device data are done in the cloud which has been allocated exclusively to the particular mobile device rather than within the device concerned, regardless of the kinds of mobile devices being used which provided the MCC services [35]. The driving force behind the development of MCC is to enable limitless computation in mobile devices while minimizing the challenges inherent in the current mobile computation technology.

2.2.2 Fog Computing

Fog computing is a remote computing paradigm that acts as an intermediate layer between cloud and the cloudlet as shown in level 2 of Figure 2.2, so that Cloud-based services can be extended closer to the ecosystems [25]. The cloud datacenters often fail to deal with storage and processing demands of billions of geo-distributed IoT devices and sensors with the consequence of congested networks, high latency in service delivery, and poor Quality of Service (QoS). Edge computing backed by powerful computing resources can reduce the network latency and render the nearby cloudlet accessible by edge users through a one-hop high-speed wireless local area network. In order to reduce the delay during the offloading, cloudlets will be the right offloading decision to get the task result fast with the minimum delay and the cloudlet will be in the edge layer which is closest to the edge users. The fog computing will be in the middle layer between the edge layer (cloudlets) and the cloud layer (cloud) [25].

2.2.3 Mobile Edge Computing (MEC)

Mobile Edge Computing (MEC) is an innovative architecture, which enables the functionalities of cloud computing at the edge of the mobile network. The main idea regarding MEC is to bring resources of cloud computing near the end user and serve request of the end user locally as shown in level 3 of Figure 2.2. MEC helps computation offloading process to get low latency during offloading tasks and reduces the traffic in the network as low requests are accelerated to the cloud server. The MEC architecture is proposed by ETSI where they presumed that cloud functionality like storage and computation would be integrated with the edge network devices like small cell access points, macro base station, radio network controller and macro base station [48]. The idea of Cloudlets was produced at the late of 2009 as a trusted local rich computation resource or multi-core rich resources, which are well linked to the Internet through wireless LAN and are available for use by nearby mobile devices users [94]. Cloudlets use Wi-Fi network for offloading of the computational tasks so that helps to save a considerable amount of energy of mobile devices compared to offloading through 3G/ LTE cellular network [7, 23]. Cloudlet mechanism extends the mobile device battery lifetime thereby reducing the network latency, on the other hand, improves the Quality of Experience (QoE) of the end user [6]. Therefore, deployment of the cloudlets will be similar to that of Wi-Fi hotspot configuration and will be close to the edge users.

2.2.4 How Are MCC / MEC Related?

Mobile Edge Computing was produced to overcome some limitations of Mobile Cloud Computing such as the latency problem during offloading to the main cloud and the energy consuming which accompanied the latency in MCC and the assumption of stable network environment during offloading process in MCC. Latency problem is one of the main limitations regarding Mobile Cloud Computing. It costs a substantial amount of latency to transfer the migration data to the cloud. Latency in transferring the data in MCC raises mainly through three resources, which include latency between, connected access points and mobile devices, between the access point and core network and between the core network and the cloud server. Latency between connected

APs and mobile devices depends on various factors like quality of the wireless channel, path loss, number of bandwidth sharing users and interference. While transferring the data to the core network from the access point, the main reason for latency is backhaul in link capacity due to the low data rate. While the latency between the cloud server and core network depends on the latency of a wide area network which relies on the number of hops and the distance between them.

Once the offloaded task reaches the cloud server, the server undertakes the entire computation required task and transfer the task result back to the mobile device through the core network and APs. Contrarily, in the case of MEC large portion or whole tasks are handled in edge side. This results in a reduction of a significant amount of latency while transferring data to the cloud server side from APs through the core network. Through the deployment of Mobile Edge Computing latency can be reduced from 60% to 90% as per the field trial run by China Telecom. They showed that MEC compared to MCC could reduce the latency by up to 88% for improved reality application [29, 127]. In the case of energy consumption which accompanied with the latency in MCC, computational tasks are offload by the mobile devices to the cloud server through the APs and core network experiencing significant latency. For fulfilling the latency requirements of the real-time applications and intensive computation applications, mobile device offloads a small portion of the task while performing a large portion of tasks locally in the mobile device so that will results in high consumption of mobile device battery power.

By contrast, in MEC, lower latency enables offloading of higher portion or whole of the computation tasks to the edge side, which will help to reduce the mobile battery energy consumption. MEC helps to extend the lifetime of the battery of mobile devices and the MEC saves 42% of energy consumption compared to MCC as stated on [37]. Finally, computation offloading in MCC considers the network as a stable environment that means, after offloading decision is made, the task will migrate to the main cloud side without considering the network fluctuations that could happen during the offloading process like user mobility during offloading that will disconnect the connection between mobile device side and the cloud side [22, 23, 55]. On the contrary, MEC helps to get the best solution in the worst case of the network fluctuations during task offloading and researchers on the area of MEC try to find solutions in various network issues that affect the computation offloading mechanism [18, 107, 111, 119].

2.3 Computational Offloading Optimization

In the edge environments, allocating the best place to offload the tasks is a challenging task because multiple criteria must be taken into account, including limitation of resources and proximity of cloudlets [88]. Methods designed to solve this problem falls into two categories: classical and metaheuristic approaches. Classical approaches can produce better accuracy at the expense of high computational time-consuming. In case the problem is non-linear, or it has a huge size, classical approaches stuck in local optima. Accordingly, researchers shifted towards using metaheuristic as it provides a near-optimal solution with a reasonable computation [11]. Recently, many solutions are advanced regarding the optimization of the offloading process in the edge networks. Offloading task to MCC/MEC platforms has been received lots of attention from the research community [22, 43, 114, 123]. However, published studies have not considered *the optimization of execution latency subject to task precedence with task constraints and user mobility*; this has also been noted by recent works [69, 86, 91, 112].

Existing works on offloading optimisation of the assignment of *tasks* to the edge resources can be categorized based on the optimization objective into: (i) minimizing the response time (delay) in task execution [63, 66, 73, 111], (ii) maximising the energy saving of user equipment [30, 60, 78, 129]. Some studies also considered both energy consumption and delay, opting to strike a balance [20, 70, 116, 118]. Unlike prior work, we focus on optimizing the execution delay in the presence of precedence constraints, i.e., dependent tasks with task constraints, while factoring in the user mobility pattern, which affects reachability to the edge nodes. A comparative summary of related works is shown in ***Table 2.1***[2]. In this section, we are going to summarize some recent studies of optimization problem of the computation offloading approach in Mobile Cloud Computing (MCC) and Mobile Edge Computing (MEC).

2.3.1 Task Offloading Optimization in MCC

The authors in [88] proposed a novel framework that involved queue-based algorithm and hybrid heuristic in optimizing the task assignment process in MCC [88]. The architecture of the framework was divided into two main stages. In the first stage, a queue model is used to represent the clouds and cloudlets into queue structure to reduce the drop rate of the user's tasks. In this stage, Queue based Decision marker (QDM) unit is utilized to estimate the probability of appointing each task to a cloudlet or public cloud. This

[2]This table shows a comprehensive overview about the recent works in *the optimization task offloading studies subject to delay optimisation , energy saving, and both of the energy and delay objectives in edge networks of mobile edge computing and mobile cloud computing.*

is to minimize mean response time. The inputs of this unit are capacity of cloudlets/cloud and all the users requests and initial queue. The functionality of this unit is dependent on the model-driven from the queue theory. The output of QDM and the duration of communication between each user and cloudlets/cloud are the inputs of the subsequent stage. In second stage, two-nature inspired algorithms including Genetic algorithm (GA) and Ant colony optimization (ACO) are hybridized to empower the searching process in finding near-optimal task assignment that considers the duration of communication between each user and cloudlets/cloud with the eventual desired outcome being the minimizing the consumption time of offload-able tasks and power consumption in the mobile battery.

In [53] , computation offloading in MCC is formulated as an optimization problem. Grey Wolf optimizer (GWO) [75] is an optimization algorithm inspired by hunting behavior and leadership hierarchy of GWO in nature. In this paper, researchers proposed an adaptation version of GWO to find the best solution for computation offloading for MCC workflow. In practical, GWO iteratively generated candidate solutions that attempt to minimize the task execution time in workflow and energy consumption in mobile devices. Focusing specifically on a mobile cloud environment, researchers exerted tremendous efforts to gain high-quality assurance and optimal utilization of resources for mobile devices.

Peng et al. in [82] proposed a joint optimization approach based on dynamic voltage and frequency scaling technique and whale optimization algorithm (WOA) [74], to optimize task completing time and energy consumption of mobile devices. In the estimation of these two optimization objectives, several factors are considered, which are task execution position, task execution sequence, and operating voltage and frequency. Moreover, the fitness function utilized in WOA is multi-objectives, where weight scores are assigned for both task completion time and energy consumption. The experimental results proved that the joint optimization approach is a promising and effective approach capable to provide adequate solutions for running the mobile cloud system in a seamless manner with respect to saving energy and parallel task scheduling.

In a recent research [106], an efficient hybridization model based on Queue-Ant Colony Optimization and Artificial Bee Colony Optimization Algorithm, referred as (QAnt-Bee), was proposed as a means to allocate the offloaded tasks to the most accurate cloudlets in MCC environment by optimizing the processing delay of tasks and energy consumption, and the rejected rate of offloaded tasks. Since the resource allocation is considered as NP-hard complete problem.

Ge et al. [41] proposed an improved version of particle swarm optimization (MPSO) to more effectively optimize the resource allocation of task offloading plans in a shortened time. In MPSO, a task movement strategy that allows the movement of task position in current cloudlet to another one. In the context of optimization, this strategy allows the solution to exchange their variables in order to increase the exploration rate and thus avoid becoming stuck in local optima. The experimental results proved that the MPSO algorithm could produce better and effective solutions when compared with PSO.

2.3.2 Task Offloading Optimization in MEC

In mobile edge computing (MEC), several algorithms have been applied to solve the problem of task offloading along with the transmit power allocation. This paper [122] studied the problem of computation offloading for MEC in 5G systems. In particular, this paper focused on improving the energy consumption of system entities offloading the required tasks. The problem was formulated as an optimization problem where the energy consumption is to be minimized, taking into account the delay requirements. Both task transmission (fronthaul and backhaul) and task computation at MEC server were considered in the formulation model. To solve this problem, the authors proposed using an artificial fish swarm algorithm (AFSA). This heuristic algorithm provides a global convergence, obtaining the global optimization solution for the problem under consideration. The efficiency of the proposed algorithm was evaluated and compared with other related algorithms.

Wang et al. [111] studied the problem of task assignment in MEN for multi-task multi-user situations. In particular, this paper considered minimizing the task execution delay on MEN. The problem was formulated as an optimization problem where task properties, user mobility, and network constraints were considered as a constraint satisfaction problem. Then, the authors proposed a heuristic algorithm to solve this problem. The proposed algorithm proceeds as follows. First, users send a message, which includes general information about their tasks, to the central controller of MEN. Particularly, this message contains the data size, execution load, local execution time, and the likely output data size. The central controller , then, allocates each task to a sBS where the delay is the shortest. A sBS, which needs to execute two or more tasks, performs the task with the minimal execution time. Further, the central controller re-allocates those tasks which are not under execution. The process continues until each task is allocated to the optimal sBS. If the local execution time remains shorter than that of the optimal sBS, the task is executed locally at the user end. It is noteworthy that the proposed algorithm considers user mobility prediction during the allocation process. A set of simulation experiments was conducted to evaluate the performance of the proposed algorithm and the results showed that the task execution delay is significantly reduced when the user mobility is considered.

The research in [128] studies the problem of task offloading along with the transmit power allocation in MEC systems. It found that both the execution latency and energy consumption were considered to be reduced so that the overall performance is enhanced. The problem was formulated as an optimization problem aiming at minimizing the weighted sum of executing delay and energy consumption. This paper first used the flow shop scheduling to achieve the optimal task offloading for a given transmit power. Further, it employed convex optimization to determine the optimal transmit power for a given task offloading decision. The results showed that the delay performance improves when both the radio and computational resources are relatively balanced. Further, the proposed algorithm reduces the energy consumption significantly while offering near-optimal delay performance.

In [49], authors study the problem of task offloading and resource allocation in MEC. The problem was formulated as a bilevel optimization problem in which the offloading decision was considered as the upper-level optimization problem whereas the resource allocation was considered as the lower-level optimization problem. Further, the aim of the upper-level problem is to minimize the energy consumption of all users and the aim of the lower-level problem is to minimize the total computations of all users. This bilevel problem, then, was solved using a bilevel optimization approach. In particular, ACS [3] is first used to generate offloading decisions for the upper-level optimization problem. If these decisions are considered feasible, then the monotonic optimization method is employed to calculate the optimal allocations of resources. The performance of the obtained joint solution is evaluated. This process continues until the best combinations have been achieved. The simulation results showed that the probabilistic technique provides efficient solutions for two sets of instances with about 400 mobile users.

Mao et al. [72] consider the problem of task offloading along with resource allocation in MEC systems. The aim is to minimize both the energy consumption and the monetary cost for mobile users. The problem was considered from game theory perspectives. Hence, the authors proposed a game model that includes a cloud and wireless resource allocation algorithm. The simulation results showed that the proposed algorithm minimize the cost with low complexity. Further, compared with existing algorithms, the larger is the size of the task's data is, the less the energy consumption and completion time is. There are other studies which employ the same classification of the optimisation objective like works with minimising the delay [66, 66], works with maximising energy saving [19, 60, 78, 124, 129], and works with the both objectives [17, 70, 72, 116].

[3] Ant Colony System (ACS) is a probabilistic technique for solving computational problems which can be reduced by finding good paths through graphs.

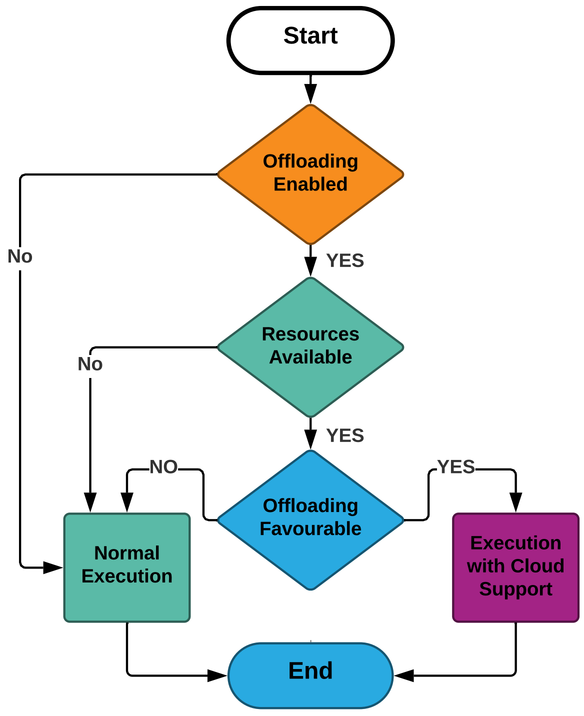


Figure 2.3: Process of Computation Offloading in Mobile Cloud Computing

Apart from the abovementioned studies in ***Table 2.1***, our focus is on minimising the execution latency of dependent tasks in a mobile edge network. The constraints considered in our work are user mobility, multi offloading systems, task precedence as DAG, and distributed edge environment. We develop offloading algorithms that minimise the job completion time with consideration to the user mobility in edge.

2.4 Offloading in Static Environment

Frameworks in Mobile Cloud Computing focused on the problems regarding offloading decision-making and application partitioning in offloading tasks from mobile devices to the main cloud without considering user mobility or changes that could happen in the network connection during offloading operations [22, 54, 55]. They have solved making offloading decision making problem and application partitioning problem by using a mechanism which consists of (1) a Partitioner (section 2.4.1), (2) a Profiler (section 2.4.2), and (3) a Solver (section 2.4.3) [57]. This mechanism helps to decide whether it is favorable to offload the task to the cloud side or just execute it locally in the mobile device as summarized in Figure 2.3 [31].

2.4.1 Partitioner

The partitioner is used to annotate which portion of the application is considered as a offload-able task. Annotated partition is achieved through results from application analysis made on codes of computation. It is determined based on whether the codes are accessing native resources of the mobile environment, or not. Mobile environment comprises either native resources including access to I/O interfaces, GPS, Camera, native services inclusive of the particular mobile environment, or any other hardware embedded to the mobile devices [31].

2.4.2 Profiler

Profiler is used to monitor offloading parameters that will help the framework Solver to make the final decision whether to offload the task or not. Therefore, the profiler will be the important factor in making the final decision in the solver part. Profiler can monitor decision parameters such as CPUs or energy power. Some frameworks used monitor software such as ThinkAir framework which uses power-Tutor software to track various program related parameters. It extracts overall the execution time for a particular method, CPU cycles, and memory allocation of a particular thread, method call numbers, and executed instruction numbers [55]. Other some frameworks used monitor device such as CloneCloud framework [22] utilizes Monsoon Power device to monitor three system variables: CPU activity (active and idle state), Screen (on/off state), and Network interface during active state (transferring/receiving) or idle state.

2.4.3 Solver

Solver of the computation offloading frameworks in MCC is the part which makes the feasible offloading decision based on the available partitions and decision metric developed by using parameters from profiler or directly utilizing profiler parameters for optimizing solution of the decision. The solver can be categorized based on its location whether it is located on the mobile device, or in remote cloud/server, or in both [31]. In this work, the solver is used for assigning a value to decision variable based on minimization of expected cost of a particular partitioned application. It is used to make a final offloading decision by the framework and situated in the mobile device user [22]. Another work uses a linear program solver in both side of mobile device and cloud as shown in Figure 2.4 to solve a global optimization problem developed by using input such as annotation and graphs from the annotated call graph model developed for partitioning model of the framework. Energy used during the local execution, remote execution, time spent for local as well as remote execution are taken as decision making metrics for the solver as shown in Figure 2.4 [23].

Other intensive frameworks in MCC have used the same mechanism of binary

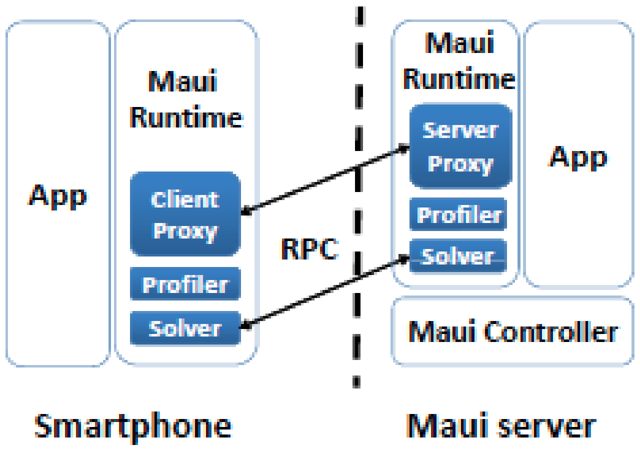


Figure 2.4: Solver in Mobile Device Side and Server Side of MAUI [23] Framework

variable decision without considering the network fluctuations [22, 23, 54, 55, 57]. Mobile Edge Computing was produced to overcome the network fluctuations that will happen after offloading decision is made [69].

2.5 Offloading in Dynamic Environment

Recent works in the Mobile Edge Computing dealing with the dynamic computation offloading are focusing on the task offloading optimization problem in the edge. Offloading optimization problem in computation offloading of MEC can be categorized into three kinds: optimization of reduction of delays in tasks offloading in the edge, optimization of energy consumption in tasks execution in the edge and the combination of the optimization of energy consumption and execution delays of the tasks in the edge.

2.5.1 Minimising Latency

Liu et al. [66] proposed a framework to minimise the execution delays in tasks of MEC like this work of reducing the delay in execution task for a single user, which uses the single dimensioned search algorithm. The result of this algorithm is a policy in making an offloading decision based on the queue state of the application buffer. Alongside with this property of wireless media was considered as well.

Plachy et al. [84] consider the variety in spatial position of sBs while offloading. The sBs chosen by the users are responsible for the execution of tasks offloaded, but the results obtained in the user devices are sent through another sBs having the highest RSSI of the wireless connections. Although the consideration of spatial diversity is remarkable, the work is done considering offloading of the single task only.

On the other hand, user mobility affects the scheduling on the edge so this work proposed a framework to reduce the task execution scheduling in mobile edge network during user mobility. They have considered the information of user mobility and the information of tasks and sBSs resources and have used lightweight heuristics solution to get fast scheduling during task offloading on sBSs with different users equations [111]. The main objective of the task scheduling in this framework is to maximize the using of MEC to reduce the delay time with all users during offloading tasks to the sBS. They have considered a set of users as U within which each user (i) has own computation task (j) that will be assigned to a set of base station as sBS. In the route of user mobility there are a sequence of sBS in user path P_i and (k) belongs to one of the paths in P_i that contains a set of sBS. Each task of $T_{i,j}$ should be executed in the edge once time only along the user trajectory P_i. The problem modeled as an optimization problem as follows:

$$\max_{D^e} \frac{1}{|U|} \sum_{i \in U} \sum_{j \in T_i} d_{i,j}(t^l_{i,j} - t^{edge}_{i,j})$$

$$s.t. \forall t_{i,j}, \sum_{k \in P_i} d^e_{(i,j),k} <= 1$$

2.5.2 Energy Saving

Studies regarding optimization of maximising energy saving, this research [78] demonstrated a framework for reducing the mobile device energy consumption by optimizing the transferring time and the data size offloaded to the edge network AP during offloading process. In [60] authors consider the dynamicity in the state of the channel in transmitting tasks through wireless means and presents a scheme for tasks scheduling and offloading them. The scheme is designed such that it can make proper usage of the wireless connections and user buffers, so that, the energy consumption in task execution is reduced.

Zhang et al. [129] proposed a framework is demonstrated for offloading the computation, in mobile edge computation, for multiple devices and also construct an optimization problem is constructed in order to minimise the energy consumption in these devices. Another study has advanced the work on this topic by considering the possible occurrence of collisions and interference due to multiple users trying in accessing single sBs, which can incur high-energy consumption in the user devices. In a way, the offloading was modeled in the game theory with multiple users, and shown that this is always compatible with Nash equilibrium [19]. This research [124] minimized the mobile devices energy consumption by centralizing framework for multiuser MEC system. They have used orthogonal frequency-division multiple access (OFDMA) and time-division multiple access (TDMA) with the purpose of reducing the energy consumption of mobile devices. Finally, [123] authors proposed a framework to harvest mobile device energy from a base station or able to offload tasks to sBSs for the same purpose of saving energy.

2.5.3 Energy and Latency Optimization

Some other works perform a combined the optimization of energy consumption and execution delays in the tasks. Through these, it is seen that minimization in the task delays, most of which can be executed faster in mobile devices than in the edge network, contribute to the high power consumption in MEC. Some of these works designate a level limit of energy consumption and minimizes the delays in the tasks without crossing the set limit. For example, this research [70] presents a flexible scheme in offloading, considering the single user, for decreasing the delay in execution in energy harvesting devices, where these devices increase the complexity in the offloading algorithms.

Mao et al. [72] proposed a model of task offloading with optimizing the allocation of the power to reduce the delayed weighted sum in computation along with the consumption of energy. They designed an algorithm namely low complexity sub-optimal algorithm. It has been illustrated in this work that the implementation of this algorithm has reached minimum latency in execution with significant energy saving in a device. To find out the optimal tradeoff between complexity and delay, a lightweight approximation is used. This research [116] shows the use of a sequential game model with multiple stages for realizing the concurrent requirements regarding the energy and delay at the same time.

2.6 Open Issues in Edge Distributed Environment

Task offloading in edge networks have been received lots of attention from the research community as we have seen in the related works in ***Table 2.1***. However, existing related works still have open issues that need to be addressed in respect of offload-able tasks in the distributed environment which can be classified into: (i) dependent task-awareness problem in the edge with distributed environment, (ii) mobility-awareness problem in the edge with distributed environment.

2.6.1 Offload-able Tasks with Dependencies in Edge

Offloading applications with concurrent tasks (as shown in Figure 2.5)to MEC makes the offloading more complex. As noted for decades of studies in task scheduling [105, 109], the applications in ecosystems (mobile devices) could consist of a number of tasks with dependencies, which are modeled as a directed acyclic graph DAGs. A task-call graph is used to determine the dependencies among tasks in the application [71]. As we observed in existing works of MCC/MEC, current studies have not solved the problem of offloading the dependent tasks to the edge when considering the user mobility in distributed environment and task constraints [52, 67, 98, 121].

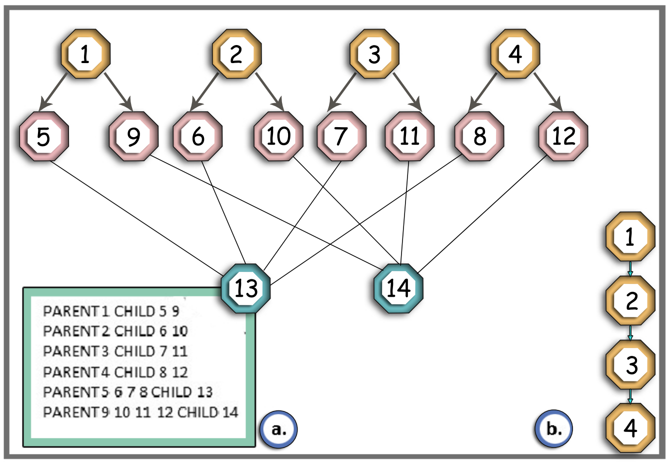


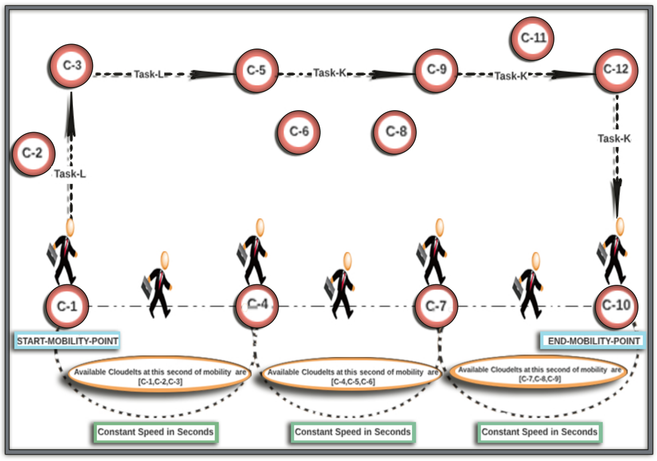
Figure 2.5: Tasks with Dependencies

In the real world, the edge networks have multi-user systems and multi-server systems, which exist studies of offloading dependent tasks, lack this assumption. The process of uploading, downloading and execution in the case of multi-user and multi-server systems are seriously affected by constraints of the dependent tasks during the offloading process. For example, the output of the concurrent task might be used as the input for other tasks, which makes the order of execution in the multi edge nodes sophisticated [71]. A work on [52] is a dependent task scheduling as DAG with multi-user constructed a $(1+\epsilon)$-approximation strategy and they consider only one centralized edge server to minimize the latency.

Liu et al. [67] consider dependent tasks placement in edge with only one user and centralized edge servers to reduce the application completion time. This research [121] proposes industrial application modeled in DAG with multi devices and only a centralized server to minimize the energy consumption of devices and cloud cost. Shu et al. [99] proposed a fine-grained task offloading algorithm with multi-users. The authors however assumed a *centralized* server in the edge that computes the offloading schedule prior to the offloading process. However, such work is not scalable in MEN networks when the number of users is high due to the server becoming a performance bottleneck. To address this open issue, in our work, we develop offloading algorithms for dependent tasks in a distributed edge environment as we will explain in the next chapters.

2.6.2 Mobility- Awareness in Distributed Edge Environment

Mobility awareness is still a significant problem in mobile cloud/edge computing networks because of sending/receiving jobs from different edge nodes as shown in Figure 2.6. The majority of existing works in task assignment to edge network make the assumption that the users are stationary during task offloading and the communication between mobile devices and edge nodes are always available [14, 24, 120]. The authors of [123] discussed the task assignment with resource allocation for multi users in a single edge server while assuming that the users can access the edge server anytime and anywhere, which is unrealistic in the real world. Another study [14] suggested an online solution for the deployment of stream based on the task assignment of multi-user systems in the edge.



C-3
Task-L
C-5
Task-K
C-9
Task-K
C-11
C-12
C-2
Task-L
C-6
C-8
Task-K
C-1
C-4
C-7
C-10
START-MOBILITY-POINT
END-MOBILITY-POINT
Available Cloudelts at this second of mobility are [C-1,C-2,C-3]
Available Cloudelts at this second of mobility are [C-4,C-5,C-6]
Available Cloudelts at this second of mobility are [C-7,C-8,C-9]
Constant Speed in Seconds
Constant Speed in Seconds
Constant Speed in Seconds

Figure 2.6: User Mobility Among Edge Network Nodes

They predict the application response time by using a queueing theory-based model and they then develop an optimization model to reduce the delay. However, they do not consider the user mobility in the edge. This framework [111] assumes all properties are known in advance: task attributes, network conditions, and user mobility (with in-dependent offload-able tasks). They develop an optimization model to reduce the latency in task execution and they consider user mobility with a centralized server as a (static environment) with predefined properties. However, such scenarios are limited in a distributed edge environment.

Ultimately, current works in edge networks (summarized in Table 2.1) have not considered a distributed offloading for offload-able jobs of dependent tasks with task constraints with multi offloading systems in the edge network during the user mobility. In this thesis, we address all these issues together by developing offloading algorithms for user devices mobile with hosting dependent tasks with the aim of reducing completion latency in distributed edge environment. We formalise the offloading problem as a constraint satisfaction problem and we construct optimisation models for the offloading algorithms to approximate the optimal solution.

2.7 Summary

Overall, there are a large number of offloading techniques in edge networks. Two of the biggest categories are (i) offloading in static environment and (ii) offloading in dynamic environment of the edge. The optimisation aim in both of these techniques can be categorized into three kinds: (i) optimization of minimising the delay of the tasks offloading in the edge, (ii) optimization of maximising the energy saving of UE during task offloading to the edge and (iii) the combination of the optimization of energy consumption and execution delays of the tasks in the edge. Existing related works in MCC/MEC still have open issues with (i) dependent tasks offloading in the edge with distributed environment, and (ii) mobility-awareness problem in the edge with distributed environment. We develop heuristic and fully distributed offloading algorithms to minimize the average completion time of offload-able dependent tasks with task constraints while factoring in user mobility which affects reachability to the edge nodes.

FRAMEWORK	OBJECTIVE	TASKS DEPENDENCY	USER MOBILITY	PLATFORM	OPTIMISATION ALGORITHM
Rashidi. [88]	Delay & Energy	In-dependent	N0-Mobility	MCC	QDM & GA & ACO Algorithms
Yang. [115]	Delay	In-dependent	Mobility	MEC	Location-based Offloading Scheme
Yang. [121]	Delay & Eenergy	Dependent	N0-Mobility	MEC	ASO and Pro-ITG Algorithms
Mirjalili. [74]	Delay & Energy	In-dependent	N0-Mobility	MCC	Whale Optimization Algorithm (WOA) Algorithm
Yang. [122]	Delay & Energy	In-dependent	N0-Mobility	MEC	Artificial Fish Swarm (AFSA) AFSA
Kao. [52]	Delay	Dependent	N0-Mobility	MEC	Fully Polynomial Time Approximation (FPTAS)
Wang. [111]	Delay	In-dependent	Mobility	MEC	Heuristic Algorithm
Zhang. [128]	Delay & Energy	In-dependent	N0-Mobility	MEC	Weight-Sum Function
Peng. [82]	Delay & Energy	In-dependent	N0-Mobility	MCC	Whale Optimization Algorithm (WOA) Algorithm
Huang. [49]	Delay & Energy	In-dependent	N0-Mobility	MEC	Ant Colony System (ACS)
Mao. [72]	Delay & Energy	In-dependent	N0-Mobility	MEC	Game Theory
Bi. [9]	Delay & Eenergy	In-dependent	N0-Mobility	MEC	Hybrid Metaheuristic Algorithm
Shu. [98]	Delay	Dependent	N0-Mobility	MEC	Heuristic Algorithm
Kai. [51]	Delay	In-dependent	N0-Mobility	MEC	Successive Convex Approximation (SCA)
Sun. [104]	Delay& Eenergy	In-dependent	N0-Mobility	MEC	Lyapunov Optimization Algorithm
Bi. [10]	Delay& Eenergy	In-dependent	N0-Mobility	MEC	Mixed Integer Non-Linear Programming (MINLP)
Shan. [97]	Delay& Eenergy	In-dependent	N0-Mobility	MEC	Cov-AHP & Nash Equilibrium Algorithm
Erana. [32]	Delay & Eenergy	In-dependent	N0-Mobility	MCC	Energy Estimation Model (RG-EEM)
Raj. [87]	Delay & Eenergy	In-dependent	N0-Mobility	MCC	Ant Colony Optimization (ACO) Algorithm
Gu. [41]	Delay	In-dependent	N0-Mobility	MCC	Particle Swarm Optimization (MPSO) Algorithm
Sundar. [106]	Delay	In-dependent	N0-Mobility	MCC	Queue-Ant Colony (QAnt-Bee) Algorithm
Liu. [68]	Delay & Eenergy	In-dependent	N0-Mobility	MEC	Lyapunov Optimization Algorithm
Liu. [67]	Delay	Dependent	N0-Mobility	MEC	GenDoc Algorithm
Hoang. [47]	Delay	In-dependent	Mobility	MEC	Heuristic Algorithm
Thana. [108]	Delay & Eenergy	In-dependent	Mobility	MEC	Computational Intensity (CI)

Table 2.1: Comparative Summary of Related Works

Chapter 3

Models and Experimental Setup

In order to investigate ways to schedule dependent tasks offloading in multi edge nodes and develop models to provide an efficient offloading algorithm in distributed edge environment, the problem first needs to be formally stated. The offloading process will be achieved in two different platforms i.e. clients (mobile devices or IoT devices as edge users) and cloudlets (which are the remote resources of the edge). In this chapter the formal definition of the offloading and task scheduling problem in distributed edge environment will be described. We will define the important models as shown in Figure (3.1) that we need to construct during the offloading process as: clients model, dependent tasks model, edge nodes model, and the user mobility model.

This chapter describes how ns-3 is applied when testing algorithms experimentally, a process which is undertaken to quantify the performance of the algorithms. There are some parts to show in this section with ns-3 simulation. In the ns-3 simulation allows to have different scenarios during the simulation, but the real deployment in Chapter-5 allows the real scenario to be examined. In order to explain the configuration of the ns-3 simulation in our work a number of points will need to be explained. The two main points are: (i) what main models we use in the offloading process, (ii) what simulator will be used and the configurations. This chapter will detail these points, as well as explained how the simulation and deployment environments were set up, and how the algorithms were performed.

3.1 Offloading Models in Edge Network

In our work, offloading in edge will consist of multiple users, each having a mobile device[1], moving from a starting point to a destination point and a network, called edge network[2] , that contains a number of powerful computing

[1]We will use the terms edge user, client, and user device interchangeably.

[2]We will use the terms edge nodes, remote resources, and cloudlet interchangeably.

nodes known as cloudlets as shown in Figure (3.1). In our work, we model the task graph as DAG to solve the scheduling problem of the offload-able dependent tasks with user mobility as previous works have not covered this problem in distributed edge environment [92, 96, 111]. In our work, our objective is to minimise the finish time of jobs with offload-able dependent tasks for every user. This problem is known to be NP-hard [67]. Such task scheduling is a special case of the problem we pose here, making our problem intractable in the general case. The task scheduling problem is in its general form NP-complete, therefore it is not possible to find an optimal solution in polynomial-time unless P = NP [58, 59]. An optimal assignment indicates that based on some objective function, the mapping method obtains the best solution (schedule) for the problem [79]. We now detail each model as follows:

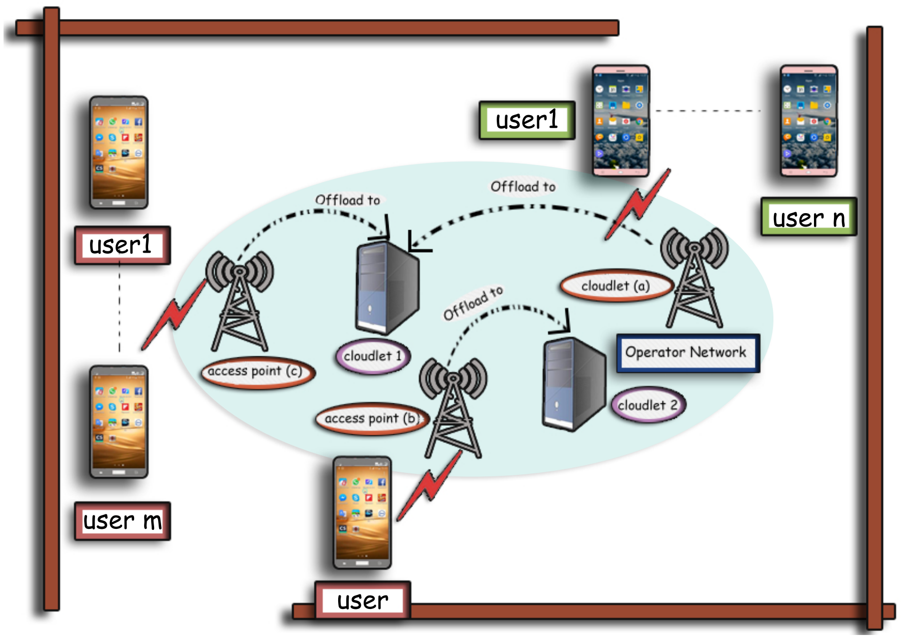


Figure 3.1: Computational Offloading Models

3.1.1 Edge User Model

In client model, we consider IoT devices or mobile devices making the offloading decision for all jobs to be run remotely in the edge side (Cloudlets). Code profiler, system profiler, and decision module in static environment will determine the correct decision regarding which job will be offloaded or not to have the right decision and that will be limited with the user mobility and other challenges in the distributed environment [27, 36]. The main objective of our work is to minimize the execution delay of the job in the edge after making the offloading decision while taking into consideration the distributed variables i.e. user mobility and dependent tasks. We consider a set of n users

$U = \{U_1, \ldots, U_n\}$, each having a mobile device with various forms of network connectivity such as Bluetooth or WiFi. Each device has limited computational resources to run resource-heavy applications (or jobs). We assume each wireless network interface to be associated with its own range. Each user runs a job, which consists of a set of *tasks*. To bridge the gap between the resource demands of the job and the resources available on the user's device, a user will run the job on the edge network to save energy and to potentially boost performance. We assume a user will run only one job at a time.

3.1.2 Application Model

Applications (Jobs) [3] in our work have a set of tasks which can be dependent tasks or independent tasks. Independent tasks can be executed at any time without considering sequence constraints during the offloading process. On the other hand, a job with dependent tasks will have to consider the sequence ordering constraints during the offloading and will be modeled as a directed acyclic graph (DAG). The application with dependent tasks has precedence constraints i.e. a child task cannot begin its execution before the completion of all its parent tasks. The task offloading mechanism will be highly dependent on the type of the task. In our work, we model the task graph as DAG to solve the scheduling problem of the offload-able dependent tasks as previous works have not covered this problem in distributed edge environment.

The root level of the DAG contains independent tasks, while other levels of the DAG contain dependent tasks. In our work, our objective is to minimise the finish time of jobs with offload-able dependent tasks for every user. A job J_i (job of user i) consists of a set of tasks $T_i = \{T_i^1 \ldots T_i^k\}$, with dependence between the tasks. As such, we model a job as $J_i = (T_i, D_i)$ where $D_i \subseteq T_i \times T_i$ and $(T_i^j, T_i^l) \in D_i$ means that task T_i^l depends on task T_i^j. We assume a set of nodes $E_i \subseteq T_i$, called entry nodes, which are tasks that are independent. We also assume a set $X_i \subset T_i$ called exit nodes that are tasks with no successor, i.e. output of those tasks will return to the users. A task can be associated with various metadata and, in this thesis, we focus on the following: $\langle$task size, number of instructions$\rangle$. $T = \bigcup_{i=1}^{n} T_i$ will denote the set of all tasks in the system.

3.1.3 Edge Network Model

A *cloudlet* is a computer that is resource rich, i.e., it has a powerful CPU, sufficient memory and other resources to run resource-hungry applications [111]. We assume a cloudlet to a buffer large enough to be able to queue execution requests [65]. After execution, a cloudlet will take one of three steps: (i) it will

[3]We will use the terms Applications and Jobs interchangeably.

pass the (final) result directly to the user if the user is in range, (ii) pass the (intermediate) result to another cloudlet which has, in its buffer, a task that is dependent on the result or (iii) if the user is out of range, pass the (final) result to another cloudlet which will forward it onto the user. We assume a cloudlet to have a number of communication interfaces, e.g., WiFi, ethernet. We assume each wireless network interface to be associated with its own range. An ecosystem device in our work can communicate with a cloudlet if both fall within the range of each other. We assume an edge network to consist of a set of cloudlets $C = \{C_1 \dots C_m\}$. Cloudlets have a network interconnecting them and we model the edge network as a graph $G = (C, L)$, where $L \subseteq C \times C$ is a set of (symmetric) links between a pair of cloudlets. We assume the network to be heterogeneous, i.e., cloudlets have the same set of computational resources (e.g., memory, CPU) but vary in capabilities or amount [111].

3.1.4 Mobility Model

We consider a mobility model, which tracks users movements and distance change relative to the cartesian coordinates over time. We have used a waypoint mobility model (future positions known a priori) and we have not used a random mobility model (future positions not known a priori) for the complexity analysis of the results. In a waypoint mobility model consists of a list of waypoints that determine the path of the users, and where each waypoint consists of the position and the time at which user reaches that position [15]. While in a random waypoint mobility model, users walk randomly in different directions. When the future positions not known a priori (random waypoint mobility model), the user can take a long time to be in the range of a cloudlet that can make the result analysis impossible in some scenarios.

We assume that the edge users move between edge nodes. As the user is mobile, the communication with edge nodes will have limited range based on the communication means i.e. WiFi connection that changes as the user moves among other cloudlets in the edge. Thus, we model mobility as a user who is mobile among a set of cloudlets C_1... C_i with a sequence of periods during the user mobility. Thus, also we define a function R_i that captures the cloudlets that are in range of the edge user U_i at any given time τ in order to identify a set of cloudlets which are in the range 2^C. It is also to be noted that the set of cloudlets can be null if there is no cloudlet in the range at a given time τ), as follows:

$$R_i : \tau \rightarrow 2^C$$

3.2 Offloading Design and Network Simulation

In the offloading design, we consider three important components: (i) mobile applications with several related tasks, (ii) user devices which run the app during user mobility, (iii) high processing cloudlets distributed along the users path. The mobile application consists of a set of dependent tasks and we assume that each application can be divided into multiple dependency tasks. The user devices are resource-constrained, with a low computation processor CPU and more energy consumption to run heavy applications as augmented reality applications and other applications that require more resources. The Cloudlets are heterogeneous machines that are not resource constrained, i.e., they have a powerful CPU and have sufficient memory to run resource-hungry applications. We denote cloudlets as $C = \{C_1 \ldots C_m\}$ and they have interconnection networks among them a graph $G = (C, L)$, where L is a (symmetric) link between a pair of cloudlets. We simulate our work in a discrete-event network simulator which is primarily used for networking and offloading research purposes [89, 113]. A discrete-event simulation can be used to simulate all layers of computer networks, including signal processing simulation in a physical layer, medium access simulation in a link layer, routing simulation in a network layer, protocols simulation in a transport layer, and has also the ability of varying simulations in an application layer. A discrete-event simulation has an important benefit which is the repeatability, i.e. different algorithms can be simulated and evaluated under exactly the same (random) environment parameters. The discrete-event network simulator which was used in our work is called ns-3 simulation as we will explain in next section.

3.2.1 Network Simulator ns-3

There are several simulator tools that can be used to model the system that is being developed, such as ns-3 and OMNET++. However, in this work, ns-3 was chosen due to the support provided for modelling the network and mobility during the task offloading. The ns-3 simulator was developed to provide an open network simulation platform mainly for use in communication research and networking simulation. ns-3 provides models with a simulator that enable networking simulations with different layers. Using ns-3 is beneficial for simulation of some difficult scenarios that cannot be tested in the real world, studying system performance in a highly controlled environment and learning how networks work. ns-3 simulation gives the users the ability of using a set of libraries that can be combined with external software libraries while some other simulations do not support this feature. Operating Systems like Linux, MacOS and Windows that can build Linux code will be able to run the ns-3 simulation.

C++ and Python are the main languages that ns-3 use for simulation. In addition, the ns-3 simulation is free software which is not supported of any company and it is for research and education purpose [89, 113].

3.2.2 Network Elements in ns-3

The ns-3 simulator has elements for all of the various network parts that simulate in computer network area. We use ns-3 for simulating networks and connections during the offloading process. As ns-3 is basically a discrete network simulator, the events have to be pre-programmed with timings and durations. The events could be anything related to network like setting up connections, sending/receiving data through sockets, or connecting to Wifi/LTE, etc. In particular there are some important components that we used in the simulation:

- Node:
  In networking, a device that connects to a network is called a host or end system. In ns-3 simulator, we do not use the term host as it is related to the Internet or protocols, we use the term Node to express the same meaning of the host during the simulation. Node is the fundamental unit of ns-3 simulator and it constructs the models in our work for the user devices, edge nodes (cloudlets) and other routers or servers in the network.

  Nodes simulates and abstract any computing device in the simulation. We represented this abstraction in C++ by the class Node which provides methods for computing devices representations and management through simulations. Node refers to any entity or object that will be in the network simulation with some behaviors i.e. user devices (IoTDs or mobile devices), and edge nodes (cloudlets or servers). We can simplify some other simulation terms which are related to the node object as follows:

  - Node Container: A container class that can be used to store a set of nodes.
  - NetDevice: This interface defines the API which the IP and ARP layers need to access to manage an instance of a network device layer. It gives a meaning to the node.
  - NetDeviceContainer: A container class that can be used to store a set of net devices.
  - IpV4AddressHelper: This helper class is used to assign ipv4 address to net devices.

- InternetStackHelper: This helper class is used to aggregate IP, TCP, and UDP functionality to existing Nodes. It is used to install internet on the nodes.
- WifiHelper: This class is used to create Wifi Net Device with manual properties.
- YansWifiPhyHelper: It is used to create and manage PHY objects of the YansPhy.
- WifiMacHelper: It is used to create the MAC layer of Wifi Net Device.
- MobilityHelper: This is a helper class which is used to install Mobility Model for the user device node.
- TypeId: It is a unique identifier for interfaces in ns-3. In our simulation, we retrieve things like TCP or UDP socket using TypeId.
- Socket: A low level BSD socket API. We use this mechanism for communication between user device nodes and cloudlet nodes.

According to ns3 simulation, to create a node or multiple of nodes we use NodeContainer topology helper that provides a professional method to create, manage and access Node objects in multi user devices and edge nodes (cloudlets) during the simulation. The container holds a pointer of objects. For example, NodeContainer holds a pointer to a node object, NetDeviceContainer holds a pointer to a net device object. So we declare a smart pointer called deviceNode for the devices. Then we create an object from Node class using CreateObject to create multi nodes for modeling user devices and cloudlets in the edge network.

- Application:
  Software can be classified into two main categories. The first category is the software that handles computer resources such as processor, memory, and network, which is called system software. System software usually does not directly benefit a user. The second category is called application software. Application software uses the resources organized by the system software to accomplish user goals. So we build the ns-3 application to accomplish tasks during the offloading process. After creating the connection means between nodes, the application enables sending and receiving of packets for offload-able tasks.

- Channel:
  All data requires requires a medium through which it can move.The same concept is applied in ns-3 simulation. We model a medium which is called the 'channel to transmit task's packets' between nodes in the offloading

environment. This channel is the basic communication abstraction and ns-3 represented it in C++ by the class 'Channel' to help in communication between nodes (i.e. mobile devices and cloudlets).

- Topology Helpers:
  Topology Helpers comes with many classes which help node configurations and adding of some manual properties. For example, when it is necessary to connect Net Devices to Nodes and Net Devices to Channels for the configurations of user devices or cloudlets, topology helper is the right choice to make this setting. Many operations are done to setup Net Device with many configurations, connect the Net Device to a Channel then install it on a Node. Other operations would be required and complication will be increased if we have multiple devices and multi channels to be connected together. We use topology helper objects that make those operations configure in the professional methods during simulation of the offloading environment.

3.2.3 Network Models in ns-3

The ns-3 has a solid simulation core which uses many network models for multi-purposes. ns-3 models are well documented, ready to use and debugged, and cater to the needs of the entire simulation workflow, from simulation configuration to trace collection and analysis [89, 113]. We will go through some ns-3 models that we have used in the offloading environment work as follows:

- WIFI Model:
  Ns-3 supports Wi-Fi physical layer model called YansWifiPhy and we use this model in both Mobile Device node and Cloudlet node for their communication in the edge. To install WiFi model, we use ns-3's WifiHelper with standard 802.11n operated at 2.4 GHz. We use default channel width which is 20 Mhz. The bandwidth rate would be HtMCS0 and the bandwidth speed with this configuration comes around 10 to 40 Mbps. The configurations are done using WifiPhyHelper (YansWifiPhyHelper). We set a RangePropagationLoss Model to determine the range of devices connection during the offloading process to be up to 40 meters. Communication range between nodes in the edge environment is very important to be considered as a real world scenario i.e. if any device during the offloading process tries to communicate with a cloudlet where is locate in above 40-m would not be connected and the offloading process will be fail. In short, the mobile device nodes cannot read beacons or detect any signal if the cloudlet is located outside the communication range.

- LTE Model:
  We use LTE model with LTEHelper and EPCHelper to support the communication between cloudlets in the edge. LTEHelper is used for installing LTE Net devices on UE (cloudlets) and eNB (base station) Nodes. We use EPCHelper for creating PGW node and we build only one eNB base station and all UE (cloudlets) are connected to this eNB base station as an edge network. UE Net device is installed on all cloudlets nodes and the communication between cloudlets happens over the LTE network. RemoteHost is created manually with internet installed using InternetStackHelper and later linked wired to PGW using PointToPointHelper. Suitable routing is done using IpV4StaticRoutingHelper .

- Point to Point Model:
  We use ns-3 point-to-point model to simulate a wired connection between eNB base station and the PGW in the edge network. The model of Net Device has the following attributes during the communication: address (for mac address of the devices), data rate (for the data transmit rate), InterframeGap (for delay time between frames), Rx (for tracing received packets), and drop (for tracing dropped packets).

- Mobility Model:
  Mobility Model tracks objects movements and distance change relative to the cartesian coordinates over time. There are subclasses of the base class 'MobilityModel' that are used for different movements types. MobilityModel uses the class PositionAllocator to set the start position of objects. After running the simulation, PositionAllocator only will be used again to determine the future position points. We have used two mobility models, one for stationary cloudlets nodes ConstantPositionMobilityModel and the another model for moving mobile device nodes WayPointMobilityModel. The ConstantPositionMobilityModel is installed on cloudlet nodes since they are stationary. We would give them a position during installation and they will remain the same throughout the simulation.The Waypoint Mobility Model is installed on mobile devices to simulate them walking like users and we set a list of waypoints that determines the path of the users, each waypoint consists of the position and the time at which device reach that position. There are two methods for the waypoint mobility model that enable users to access the next waypoint: NextWaypoint and the number of waypoints that have been passed WaypointsLeft.

- Propagation Model:
  Signal propagation faces various limitation that affect signal transmission

during the offloading process. We use Propagation Loss Model basically to define how the signal of WiFi will be affected between two connected devices in the edge. Propagation loss model takes into consideration two factors when computing the received signal power, the transmitted signal power and the distance between transmitter to receiver. Propagation loss model can be designed to be a serial chain of different loss models and the total power received is the power after passes all the loss models. There are several propagation losses models, our focus will be on RangePropagationLossModel. As its name indicated this loss model affected only by the traveling distance (distance from the transmitter to the receiver). We use the MaxRange method (in meters) to determine the range distance in meters as (40-m) during the communication between mobile devices and cloudlets. If signal travels within MaxRange it will not face any losses and the received power will be exactly equals the transmitted power. If we do not install a loss model, the wifi signal is going to remain constant also if the device is near to cloudlet or too far away from the cloudlet during the offloading process. Therefore, in this model the wifi signal is affected by the distance between the two WiFi devices based on the range of communication i.e. in case of the user device is far away from the cloudlet, the signal is going to be bad and vice versa.

- Packet Socket Class:
  Socket is a gate that enables data transmission between devices in the offloading environment. We use TCP socket and UDP socket for send and receive data between nodes in the edge network during the offloading process. TCP socket is used for 'Reliability'. TCP uses three way handshake to initiate the communication (SYN, ACK, SYN-ACK). In Socket API, Connect() function is used to connect to the destination IP address, and Send() function is used to send a packet to the endpoint. To receive the same, TCP/UDP socket sink (endpoint) has to be installed. Hence, Packet sinks are installed (using PacketSinkHelper) on mobile device nodes and cloudlet nodes. TCP communications take place between cloudlet to cloudlet transmissions. 'UDPSocketSender' is a helper class made to handle UDP transmissions. UDP is a connectionless protocol and C++ ns-3 callbacks are implemented to know when or where the data is transferred (sent/received). We use UDP socket for the communication between mobile device to cloudlet and from cloudlet to mobile devcie nodes. We use basically an Internet Protocol IpV4 which identify the unique IP Address for all endpoint nodes in the edge environment. In ns-3, Ipv4Address instance is used to store IP Address in nodes. We locally store the instance of IpV4Address inside both cloudlet nodes and

user device nodes instance to keep hold of their LTE/Wifi address. The IP addresses of the cloudlet/mobile device nodes have to be known before we can communicate through TCP/UDP socket and the ARP (Address Resolution Protocol) helps to make this process.

3.2.4 Application Representation in ns-3

We represent the application [4] by a directed acyclic graph $J = (T, D)$, where T is a set of tasks with dependencies and D is the set of edges between dependencies as we see in the Figure 3.2. Each edge $(i, j) \in D$ represents the precedence constraints between tasks during the offloading process. The task T_i as a parent task should be completed before the task T_j as a child task in the graph theory that we model in the ns-3. Output data as a task result is a $(t\ x\ t)$ matrix of communication data between dependencies during the offloading, where data $T_{i,r}$ is the amount of data required to be sent from task T_i to T_j in edge nodes. In a given task graph, tasks in root level of the graph are called entry tasks and the tasks in the last level without any child task are called exit tasks. In our work, we modeled many different DAGs in random graph generation and fixed generation which they have different properties in each graph.

We modeled three main elements in DAG: the parent task, the child task which depends on the parent task and the finish time of parent task which is also the start time of child task. We also design Edge Class that holds the three elements then design DAG Class that uses the Edge Class to manage the task generation. The main methods in DAG Class: AddEdge Method (to build up the DAG by adding nodes), ComputeParentNodes Method (to get the root tasks (tasks of level 0), SetWeight Method (to set the start time of child node), and GetNextNode Method (to get the child task id with least start time). In ns-3 simulation, application with set of tasks are represented as dummy tasks with size of bytes. During the offloading process we upload/download tasks with size of bytes through socket between nodes of mobile devices and cloudlets. Tasks will be ordered during the offloading based on the DAG logic and each task will have metadata i.e. task-id, task-size, task-number-of-instructions, dependencies and etc. The parent tasks will start execution time in free ready time but the child tasks will start the execution time after the completion time of the parent tasks. We will go in details of DAGs we use in our work in next chapters.

[4]We will use the terms Applications and Jobs interchangeably.

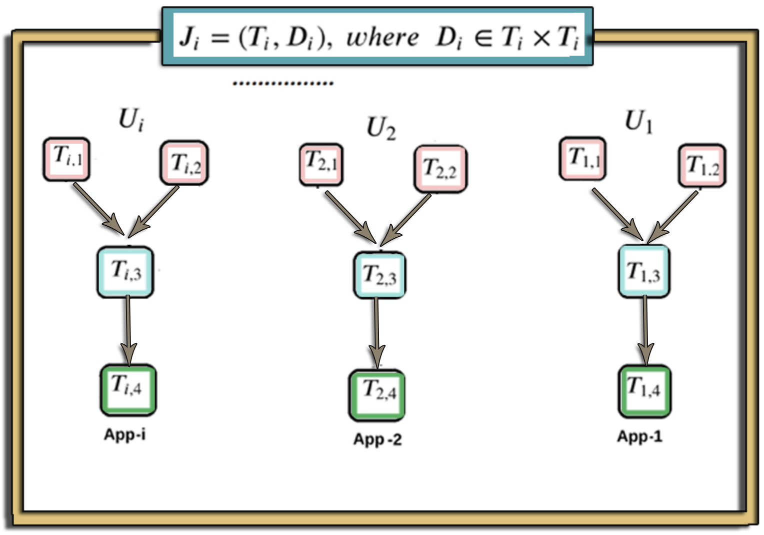


Figure 3.2: Directed Acyclic Graph (DAG) Design

3.2.5 Simulation Configuration and Experiments

In this section the simulation environment and the configurations that were used in ns-3 to generate the results are described. We first define the values and parameters that we used :

- Map Area: 0.5 x 0.5 km
- Number of User Devices: 10 to 50 Devices
- Number of Edge Cloudlets : 30 Cloudlets
- Cloudlet uplink/downlink: 40 Mbps / 40 Mbps
- Cloudlet CPU frequency: 2 GHz & CISC
- Cloudlet coverage radius: 40-50 meters
- Central Cloudlet CPU frequency: 2 GHz & RISC
- CPU Cycles Required by One Task: $2 \times 10^9 \sim 2 \times 10^{10}$
- Acceleration Ratio between CISC & RISC: 10~50
- Input Data Size: 1.0~4.0 MB
- Mobility Model: WayPoint Mobility Model
- User Moving Speeds: 0.5, 1.0, 1.5, 2, 3 [m/s]

We used the WiFi-Phy 802.11-n standard configuration in ns-3 as we model the cloudlet network as a mobile edge network [46]. The WiFi frequency is set at 2.4 GHz. The protocol used for the communication between an user device nodes and a cloudlet nodes is UDP socket with Type-Id. The cloudlet transmission range is set to 40-50 meters and the communication between cloudlets happens over an LTE network [110]. We consider an edge network with 30 heterogeneous cloudlet nodes that are uniformly distributed over the setup area. All cloudlets have fixed positions in all the experiments. and they are directly connected to a base station. All cloudlets can communicate with each other via the base station via LTE. We run thousands of experiments with some changes in the simulation settings as we will explain in the next chapters. The cloudlet has a task queue which works on scheduling algorithms i.e. first comes first serves basis FCFS or priority algorithm with non-preemptive scheduling algorithm [65]. A newly uploaded task is added to the back (end) of the queue based on the type of scheduling algorithm. The front task of the queue is popped (removed) on the following conditions: a) the task is solved and the results are sent to the child tasks b) the task is solved and the task is an end task (has no child tasks), so in this case, the results are delivered to the user-device/cloudlet on request.

We use a waypoint mobility model, where each user devcie moves in straight lines between end points with a given time at a given user speed [61]. Given the uniform distribution of cloudlets over the map area, and the user device nodes uniformly distributed over the setup area. Users move between edge nodes (cloudlets) with defining communication range as 40 meters to be able to offload tasks in this range. User know the starting position and the end position and they move in speed constant during the journey. We install WiFi module on each user device and we create $10 \ldots 50$ users moving at a constant speed in a given run [46]. Based on previous research, e.g., [111], we set the input data size of each task to be in the region of 1 MB to 4 MB. The CPU cycles requirement ranges from 2×10^9 cycles per task (cpt) to $\sim 2 \times 10^{10}$ cpt. Besides the difference in CPU frequencies between the cloudlets and user device, we additionally set the impact of acceleration rate on CISC and RISC processors as the same CPU settings in [111]. We consider users to travel at 5 different speeds, namely $0.5, 1.0, 1.5, 2.0,$ and $3\ m/s$ during the experiments. The user device uploads a given task to the cloudlet that will minimise the completion of the task. It will then either obtain the results from the same cloudlet or a cloudlet close to it when the result is ready.

3.2.6 Example of Offloading in ns-3

In this example, we used the same settings as defined in the previous section. A mobile device moves at a constant speed of 1 m/s to offload tasks in different sizes to the edge nodes. The same sequence of sets of edge nodes was laid along the path of the mobile device, from a start position to the end position. We used a total job size as 10 MB and 35 MB in 10 tasks. The job has the same precedences, i.e., the job can be represented using the same DAG. The reason for this example is to explain the impact of completion time with changes of task sizes as we see in Figures: (3.3 and 3.4). As we see in this example we have different job sizes with the same user speed. The X-axis represents the task id, while the Y-axis represents the completion time. We compare the results between Local (execute the tasks with the same settings locally in CPU processor as RISC) and Offload (execute tasks in the edge nodes).

As can be observed from this example, when the the total job size is small (10 MB), it is finished earlier in the edge nodes and it takes more time to be executed locally (i.e., on the mobile device). This is due to the case that the simulation setup of the cloudlet resources are powerful with the CISC and it is faster than RISC five times. On the other hand, when the size of the job is higher, then the jobs takes longer to execute as we see in the example of (35 MB) Figure 3.4. From a total job of size 10 to 35 MB, the completion time becomes more significant as the job size increases in local execution and in the edge. In next chapters, we are going to explain in details how the user speeds affect the offloading with considering the delay time, different user speeds, different job sizes and other important factors with multi user-device/edge-nodes during the offloading process.

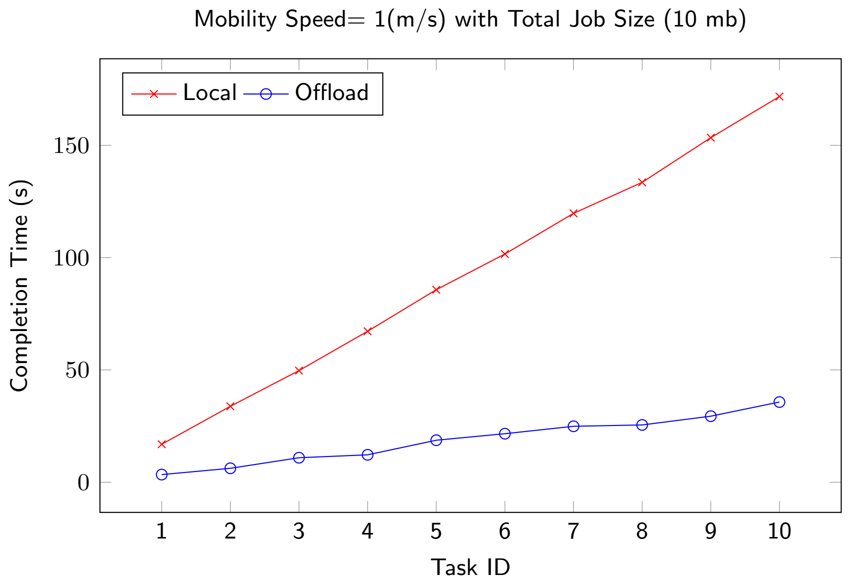


Figure 3.3: Total Job Size 10 MB with User Speed 1 m/s

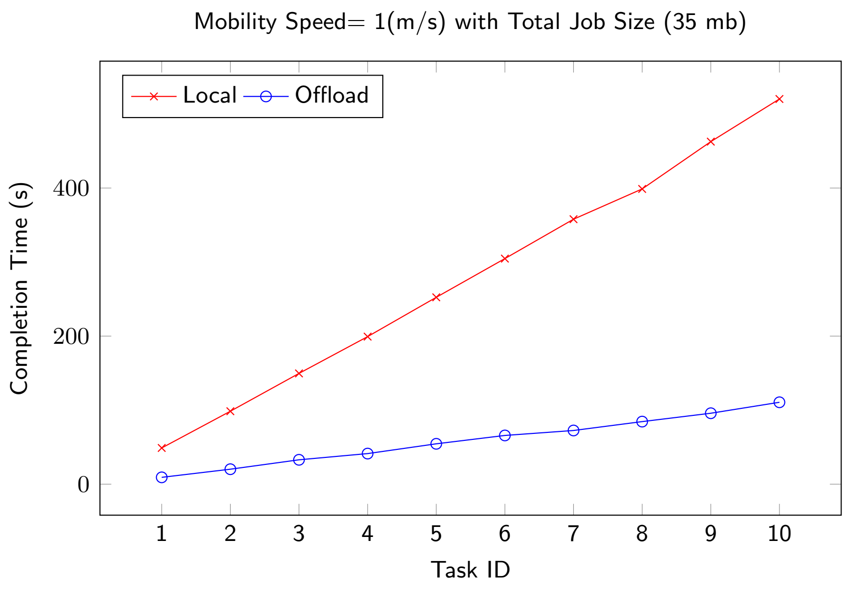


Figure 3.4: Total Job Size 35 MB with User Speed 1 m/s

3.3 Summary

In this chapter, we defined the models of the offloading dependent tasks problem in distributed edge networks. We explained the important models that we should construct during the offloading process as: clients model, dependent tasks model, edge nodes model, and the user mobility model. Also, we described how the abstract models in ns-3 simulation are applied when testing algorithms experimentally, which is undertaken to quantify the performance of the algorithms. Overall, we discussed main points on what main models we use in the offloading process, and what simulator will be used and the configurations.

Chapter 4

Problem Definition of Computational Offloading

The growth in resource hungry and real-time applications has led to a corresponding increase in the computational requirements on hardware (HW) to support such applications as augmented reality, multimedia, video editing, face recognition, and gaming [101]. Often, these applications may be hosted on devices such as smartphones. However, the limitations of smartphones on battery power, memory capacity to store data and, most importantly, limited processing capacity need to be tackled properly in the era of computationally intensive applications development [57]. Computation offloading mechanism is one possible solution, which encourages the users of mobile devices to transfer the computational jobs to a remote-based computer such as a cloud (in mobile cloud computing - MCC) or cloudlet (in mobile edge computing - MEC) [56][21]. The remote server has huge computation resources and is able to perform the operations faster than smartphones [28].

The main difference between cloud and cloudlet lies in the computing latency, making cloudlet more suitable. Recent works in MEC mostly focused on solving the offloading problem with users being static during task offloading and the communication links between mobile devices and edge nodes are always available [14, 52, 121, 123]. However, such assumptions are not valid with mobility users being a significant challenge in MEC networks that need to be addressed. This challenge is exacerbated when a job contains dependent tasks, i.e., tasks that cannot execute without a previous one has completed. Several real-world applications are available, where dependent tasks exist: for example, a GPS navigator application [99], smart access control based on face recognition [3], smart vehicular networks [103], and some virtual reality applications [33]. Most current works on offloading in MEC networks focus on independent tasks, where tasks are offloaded according to their suitability for an edge node. An important issue is to resolve then is offloading the constrained tasks on heterogeneous servers during the user mobility. When such dependencies exist between tasks, i.e., a task t depends on task s, then task t can only be executed after s has completed. Current works in offloading suffer from at least one of these several limitations: (i) assume independent tasks, (ii) assume static users, (iii) assume a central server that performs the offloading on behalf of users or (iv) focus on a single user. On the other hand, in this thesis, we address all these challenges together by developing the *algorithms* for offloading *dependent* tasks by *multiple mobile* users. However, the precedence constraints on tasks while considering the user mobility in a distributed edge environment have not been taken considered in similar previous works, e.g.,[67, 98, 111, 121].

1) Complex Offloading of Dependent Tasks to Edge: Modern applications are typically composed of constrained tasks (including heavy computational instructions) that can be represented as a Directed Acyclic Graph (DAG) as task models, as in Figure 4.1. For example, applications with dependent tasks are included in more than 75% of 4 million applications in the Alibaba data-trace [67]. Another example of a video processing application from Facebook has a set of constrained tasks to complete the video [126]. Specifically, the majority of applications could be dependent due to various precedence constraints, i.e. a child task cannot be begun before the completion of all its parent tasks. Moreover, the high data transmission between cloudlets in the MEC networks will typically occur when child/parent tasks are placed on different cloudlets. The dependent tasks scheduling in the edge with communication between heterogeneous edge resources make the constrained task offloading more complex and challenging in MEC. Previous works consider the communication between mobile devices and edge servers are always reliable without user mobility affection during task offloading, such as in [52, 67, 98, 121]. In this thesis, we model the application[1] as a DAG with a sequence of dependent tasks and we consider the MEC as a distributed environment with multiple users mobility in multi-server systems. In MEC offloading, the main aim is to reduce the offloaded task completion time by selecting the most suitable cloudlet in the execution.

2) How to Map Subtasks for Multi-User Systems with Multi-Server Systems: In the edge side, the uploading time and execution time play a significant role in the average execution time for offloaded tasks, and may have a direct effect on the decisions to offload. Therefore, to minimise the average latency during offloading, we need to organize all of the offloading techniques of various users. However, the issue of offloading subtasks of multi-user systems and multi-server systems becomes even more difficult as:

[1]We will use the terms Applications and Jobs interchangeably.

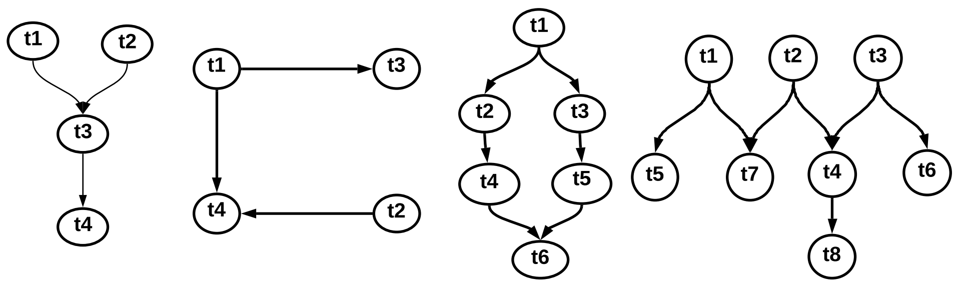


Figure 4.1: Dependency Task Models

(**a**) the offloading time may be influenced by the number of contending users decided by the offloading decisions of other nearby users to the same edge server and (**b**) each user can offload a set of tasks, which will also have a major effect on the scheduling of dependent tasks on each server. To address these difficulties, We develop the fully distributed offloading algorithm for dependent tasks hosted on a large number of mobile devices, with the aim of reducing completion latency.

3) How User Mobility Affecting Task Offloading in Edge Networks: The user mobility during task offloading is the main property of MEC, which often presents a significant impact on the efficiency of the task execution. We assume that the mobile edge users will move between edge nodes during a given mobility period and the future positions and mobile patterns are known a priori. There are two main ways by which the mobility of users affects the execution times. Firstly, dynamic workload distribution, which varies with time, is due to the mobility of users, and some connected mobile users in each edge server vary. Secondly, the uploading server and task-executing server in mobile environments are possibly different servers, due to the internal communication of MEN and workload of computation, and rarely the same. Still, it has been observed that the work being done in the field of MEC, often found to disregard user mobility. The intuitive model based mobility of users has been included in few works [73, 111], yet these are found to be weak in including the exact impact of user mobility, contributing to the poor efficiency in dependent task executions with multi-user systems in heterogeneous edge nodes. Our thesis will present a novel approach of dependency tasks scheduling algorithms with user mobility in MENs to minimize the delay of task execution during the offloading process.

Apart from previous studies in chapter 2 either in the computation offloading of the dynamic network environment (MEC) or the stable network environment (MCC), mobility aware scheduling for dependent offloaded tasks in the distributed edge environment is still not covered in the area of research.

In this thesis, we are focusing to minimize the dependent offloaded tasks delay in the edge that will affect the completion time of offload-able tasks after offloading decision is made. For the above challenges, we present algorithms for developing an offloading schedule that attempts to minimize the average completion time of offloaded jobs as we will explain in details in next chapters.

4.1 Problem Challenges

In the traditional computation offloading [22, 23, 54, 55], dynamic allocation of the tasks in MENs for estimated delay time is unknown. After offloading decision is made, the offload-able task execution in MEN is hidden from mobile users. In addition, the traditional offloading considered MEN as a stable environment with no user mobility, which is a main feature of the MEC. Current studies have not covered the issue of offloading tasks with dependencies in distributed edge environment with considering user mobility [17, 49, 66, 111].

In our thesis, we address these problems, either the resource usage estimation should be done in MEN end C, or a feedback mechanism should be established for letting the mobile device users U know about the MENs resources (cpu, queue delay, ..etc). In addition, we consider dependent tasks in Directed Acyclic Graph $J_i = (T_i, D_i)$ as computation intensive tasks of the edge, which is a sensitive case during user mobility of computation offloading in the MEN. Since the communication between edge user U and the edge node C, will be within the defined range μ_i.

As a result, to provide the storage facility as well as a computation platform for mobile device users, the Cloudlets C are constructed in an edge network. The users are mobile, and their connection may transfer from one Cloudlet C to another Cloudlet C due to their mobility μ_i while executing dependent tasks $J_i = (T_i, D_i)$ in the MENs.

For example, an augmented reality navigator should be able to load the map of some indoor location and also be able to note the mobility of users around the location. The information regarding the location should be displayed in the user augmented reality display, but as the user is roaming from one Cloudlet C to another Cloudlet C throughout the location, the outcome of computation of the augmented reality application should be delivered to the user through a different Cloudlet C than that chosen for the execution of the task.

In this situation, scheduling of dependent tasks should be done carefully so that the independent-tasks (root tasks in level 0) are executed in the most suitable Cloudlet C, which is near to the user U and must offer the sufficient resource for the user for finishing the execution of the independent-task to release and allow the dependent task (child task) to be fired.

To solve this issue, we defined start time ST and finish time FT with each task T in a directed acyclic graph (DAG) to schedule dependent task during offloading in the edge. So, when a lightweight task is offloaded, this can be executed within the designated period, i.e. as long as the user U establish a connection to a particular Cloudlet C, the task should be scheduled such that it is executed in the connected Cloudlet and must be returned to the user immediately. But if the task offloaded is of heavy weight, which would be quite large to finish in a single currently, connected Cloudlet of the user, the task is to be executed in one Cloudlet and then transfer a task result to another nearby Cloudlet C which fall in the user trajectory. So, through this mechanism, the Cloudlet C, which is connected to the user when they roam around the location, can be used for downloading the task result to the user. This mechanism will help to make the computation offloading in MEN during user mobility more efficient and effective with reducing the delay time during the offloading process.

Current studies and recent studies [17, 49, 66, 111, 111] assume at least one of these assumptions to avoid the challenges that we solve and cover in our thesis:

- Assuming independent tasks (without considering the issue of offload-able dependent tasks),
- Assuming static users (without considering the user mobility during the offloading process and task roaming between edge nodes),
- Assuming a central server that performs the offloading on behalf of users,
- Focusing on a single user (without considering the multi systems during the offloading process from both sides: edge users side and edge nodes side).

4.2 Model Definitions

In this section, we introduce a number of extra concepts that are required for defining the problem and are based on those introduced in Chapter 3.

4.2.1 System Model

We have explained the system model in Chapter:3 with details of edge user model, application model, edge network model and mobility model.

Definition 1 (Schedule)

$$S : T \times \tau \to C \cup \{\perp\} \tag{4.1}$$

The scheduler S essentially captures the time at which which a given task $T_i^j \in T$ is resident on a cloudlet $C_k \in C$.

Definition 2 (Decision Variable) *We define a decision variable $d(T_i^j, C_k)$ to ensure that all the jobs can be offloaded to the cloudlets as follows:*

$$d(T_i^j, C_k) = \begin{cases} 1, & \textit{if task } T_i^j \textit{ is executing on MEC } C_k \\ 0, & \textit{otherwise} \end{cases} \tag{4.2}$$

where $d(T_i^j, C_k) = 1$ representing that task T_i^j has been offloaded and will execute on the edge node C_k.

4.3 Problem Statement

We view the task offloading problem as a scheduling problem whereby a job can be offloaded to an edge node (cloudlet) only if the mobile device is within communication range of that node during the offloading process. Given a set of users, each with a job to execute and an edge network, our offloading algorithms will offload and schedule the the different tasks of a job onto cloudlet nodes such as to minimize the offloaded job completion time. More specifically, we should decide 1) the task placement to best edge node during the offloading, 2) the user mobility at different speeds during offloading, and 3) the uploading, execution and downloading order of the jobs on edge nodes. The main objective is to develop offloading algorithms for scheduling a job that consists of a set of dependent tasks onto an edge network so as to minimise the completion time of the job.

Formally, the problem specification is as follows.
Given:

- A network $G = (C, L)$ where
- C is the set of all cloudlets in the edge.
- L is the set of links between cloudlets in the edge.
- U is the set of all users with offloading tasks.
- Tasks are represented as directed acyclic graph $J_i = (T_i, D_i)$

Each cloudlet $C_i \in C$ has processor capabilities, queue store and queue capacity to execute tasks during the offloading process. Edge users will move between edge nodes and each user will offload a task to a cloudlet, where the job is represented as Directed Acyclic Graph (DAG) $J_i = (T_i, D_i)$. Each job has a set of dependency tasks and each user has at least one job J_i during the mobility as :

$$\begin{aligned} U = \{U_1, \cdots, U_n\}, &\text{where each } U_i = (J_i, \mu_i) \\ J_i = (T_i, D_i), &\text{with } T_i = \{T_i^1, \cdots, T_i^{|T_i|}\} \end{aligned} \tag{4.3}$$

Mobility is a function that takes a time instant and returns the set of cloudlets in mobile user range as $\mu_i : \tau \rightarrow 2^C$.

To capture the completion of a job under a schedule S (Definition 1 - Eq 4.1), we define the finish time (FT) of a job J_i (of user U_i) under schedule S, as follows:

$$FT(J_i, S) = max_{t \in T_i}(FT(t, S)) \tag{4.4}$$

We define the offloading time of a task T_i^j of user U_i on cloudlet C_k, denoted by $OT(T_i^j, C_k)$, as the minimum time t when $S(T_i^j, t) = C_k$, i.e.,

$$OT(T_i^j, C_k) = min\{t | S(T_i^j, t) = C_k\} \tag{4.5}$$

We say that a task T_i^j has not been offloaded if $\forall t \geq 0, ST(T_i^j, t) = \perp$. The scheduler will take a set of tasks from all users and the time and will return the cloudlet where a task has been offloaded to. Our main objective is to minimize the finish time job J_i of a user i during offloading process to the edge as:

$$argmin_S \ FT(J_i, S), \forall i, 1 \leq i \leq n \tag{4.6}$$

We also define a binary variable as follows, to capture the offloading schedule of a task on a cloudlet:

$$T_i^{j,k} = \begin{cases} 1, & \text{if } \exists t > 0, S(T_i^j, t) = C_k \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

After that, we develop offloading algorithms as we will explain in next chapters. We also consider the high level constraints of such offloading schedules and subsequently consider the actual offloading timing process in our algorithms.

4.4 Computational Offloading Constraints

In this work, we have developed a computational offloading model to calculate the total time required for a job to finish execution during the offloading process. The total time consists of five times, namely the start time, the queue waiting time, the execution time, the reply transfer time and the result downloading time.

The start time of a task is the time at which a task is ready to to be

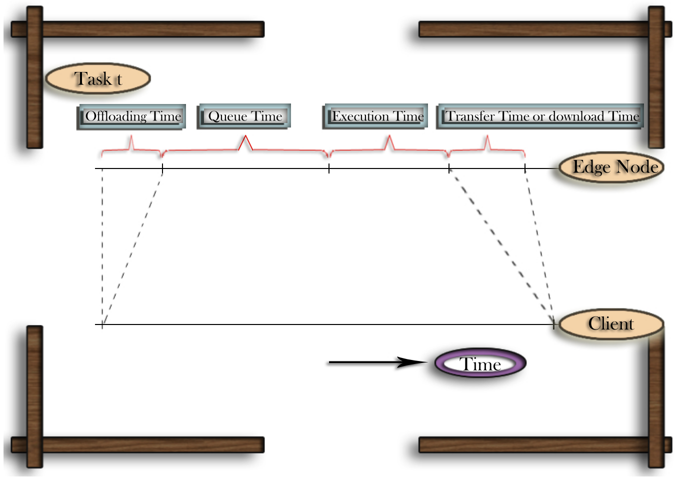


Figure 4.2: The relationship between different times during offloading.

executed. For a dependent task t, the start time can only occur after all of t's parents have completed and their results transferred to the cloudlet where t resides. The queue time is the time the task t is waiting in the queue of the cloudlet to be executed. The task execution time is the time the CPU takes to complete the execution of task t in the cloudlet. The (result) transfer time is the time it takes to transfer the result of the execution from one cloudlet to another, based on the type of dependency.The result download time is the time of sending the job result back to the IoTD and we consider the type of dependency during the downloading process. Next, we will explain the edge time in more details.

1) Start Time: The start time of a (dependent) task on a given cloudlet can only occur after (all) its parent(s) tasks have completed.

$$\begin{aligned} \forall U_i \in U, \forall (T_i^k, T_i^l) \in D_i, \exists C_m, C_n \in C, T_i^{k,m} = 1 \\ \wedge T_i^{l,n} = 1 \Rightarrow ST(T_i^l, C_n) \geq FT(T_i^k, C_m) \end{aligned} \tag{4.8}$$

2) Queue Time: The queue time $(Q(T_i^j, C_k))$ of a task T_i^j on a cloudlet C_k is the time during which that task is in the queue of the cloudlet before execution.

$$Q(T_i^j, C_k) = \sum_{t=1}^{idx(T_i^j, C_k)} E(t, C_k), \quad t \in T_i^j \tag{4.9}$$

3) Execution Time: The execution time (ET) of task T on cloudlet C_k is the time it takes for the task to complete execution on C_k. Denoting the CPU frequency of cloudlet C_k by $C_k.freq$, the execution time of the number of instructions of the task is thus given by

$$ET(T, C_k) = \frac{(T.num\_of\_inst.)}{(C_k.freq)} \tag{4.10}$$

4) Result Transfer Time: The result transfer time (TT) is the time to transfer the result from one cloudlet to another. The task result r will be divided by uploading and downloading rates of the nodes in the edge which is ($T^{u,d}$) :

$$\begin{gathered} TT(T_i^j.r, C_s, C_d) = \frac{\text{size}(T_i^j.r)}{T^{u,d}(C_s, C_d)} \\ C_s, C_d \in C \end{gathered} \tag{4.11}$$

($T_i^j.r$) is the task result that will be sent to the cloudlet destination C_d that requires the result. C_s is the source cloudlet that will send the result to C_d.

5) Downloading Time: The transferring time (DT) is the time to download the result from a cloudlet to IoTD as:

$$DT(T_i^j.r, C_n) = \frac{\text{size}(T_i^j.r)}{T^d(C_n)} \tag{4.12}$$

The **edge completion time** CT for the task T_i^j will be the total of the five edge times and will depend on the type of dependency between tasks. There are two scenarios to model the edge completion time.

The first scenario is the completion time of a (parent-task):

$$\begin{aligned} T_i^{j,k} \wedge T_i^{l,m} \wedge (T_i^j, T_i^l) \in D_i \Rightarrow CT(T_i^j, C_k) = OT(T_i^j, C_k) \\ + Q(T_i^j, C_k) + ET(T_i^j, C_k) + TT(T_i^j.r, C_s, C_d) \end{aligned} \tag{4.13}$$

The second scenario is the completion time of a (leaf-task):

$$\begin{gathered} T_i^{j,k} \wedge \forall n \in T_i, (T_i^j, n) \notin D_i \Rightarrow \\ CT(T_i^j, C_k) = OT(T_i^j, C_k) + Q(T_i^j, C_k) + ET(T_i^j, C_k) \\ + \ TT(T_i^j.r, C_s, C_d) + DT(T_i^j.r, C_n) \end{gathered} \tag{4.14}$$

4.5 An Example as a Case Study

In this section, a simple example as a case study is presented to show the offloading process (see Figure 4.3, and Figure 4.4). The application of user

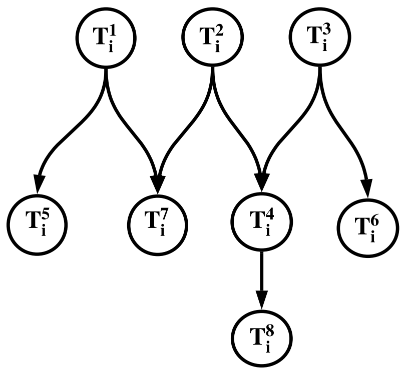


Figure 4.3: Case Study: Example DAG for a job with 8 Tasks for user i.

U_i, J_i[2], consists of a set of eight tasks, $J_i = \{T_i^1, \ldots, T_i^8\}$, with dependency constraints, as shown in Figure 4.3. The job J_i has a set of entry tasks $E_i = \{T_i^1, T_i^2, T_i^3\}$ and a set of exit tasks $X_i = \{T_i^8\}$. Task T_i^4 is dependent on tasks T_i^2 and T_i^3 while task T_i^8 is dependent on it. During the offloading process, task T_i^4 cannot start until both tasks T_i^2 and T_i^3 have completed. Likewise, T_i^8 cannot complete until T_i^4 has finished. When T_i^8 has finished, then the user U_i can download the result.

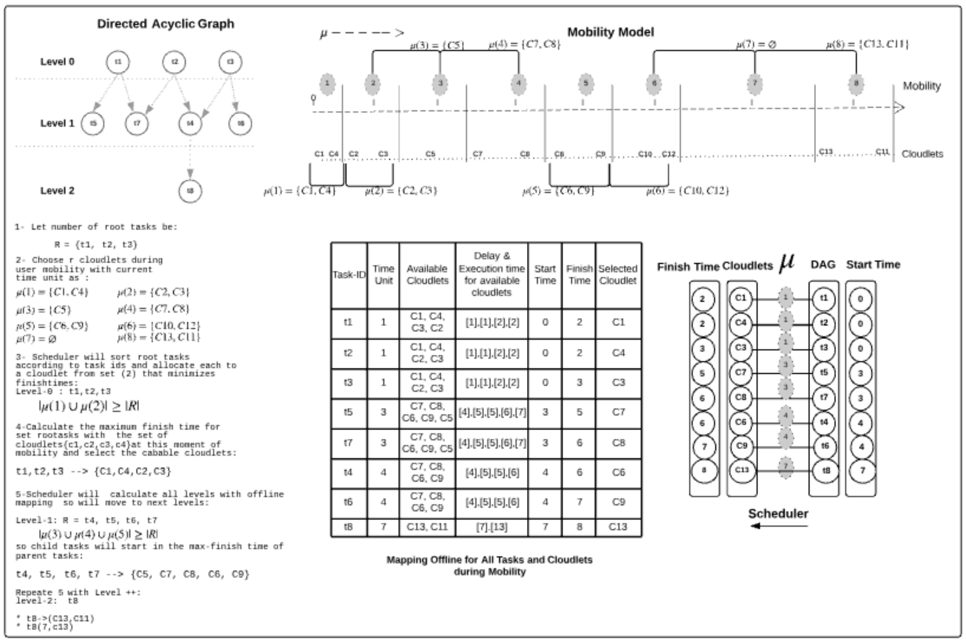
In this example case study, the edge network has 13 cloudlets, denoted $C_1, \ldots, C_{13}$, as in Figure 4.4. The mobility model is the waypoint model and, at specific times, a given set of cloudlets are in range, e.g., during the first part of the movement, denoted by $\mu(1)$, the cloudlets that are in range are $\{C_1, C_4\}$, during the second part, it is cloudlets $\{C_2, C_3\}$.

At the beginning the scheduler will offload the root tasks (T_i^1, T_i^2, T_i^3) to different cloudlets, as shown in Figure 4.4. Scheduler start allocating the tasks at the root level of DAG (as one by one) (T_i^1, T_i^2, T_i^3) in such a way as to minimise the completion time of each task. Since root tasks (T_i^1, T_i^2, T_i^3) do not have predecessor tasks, their respective start times ST will be equal to zero (i.e., they are ready to execute right at the start˙.The algorithm will then proceed in a breadth-first-search fashion to allocate tasks to cloudlets.

To simulate waiting time, we add the delay time on the available cloudlets C_1, C_4, C_3, and C_2, as shown in Figure (4.4): (1s, 1s, 2s, and 2s) respectively. Finished time FT for the root tasks will be taken on the account the delay time and the total time of the uploading, execution, downloading and transferring times between edge nodes during the offloading process. The uploading, execution, downloading and transferring times are all set to 0.25s, giving a total equal of 1.0s to be added to the delay time during mapping process.

For example, the start time of task T_i^5 is equal to 3.0s and the delay time

[2]We will use the terms Applications and Jobs interchangeably.



```
1- Let number of root tasks be:

      R = (t1, t2, t3)

2- Choose r cloudlets during
user mobility with current
time unit as :
```

$\mu(1) = \{C1, C4\}$ $\mu(2) = \{C2, C3\}$
$\mu(3) = \{C5\}$ $\mu(4) = \{C7, C8\}$
$\mu(5) = \{C6, C9\}$ $\mu(6) = \{C10, C12\}$
$\mu(7) = \emptyset$ $\mu(8) = \{C13, C11\}$

```
3- Scheduler will sort root tasks
according to task ids and allocate each to
a cloudlet from set (2) that minimizes
finishtimes:
Level-0 : t1,t2,t3
```

$|\mu(1) \cup \mu(2)| \geq |R|$

```
4-Calculate the maximum finish time for
set rootasks with  the set of
cloudlets{c1,c2,c3,c4}at this moment of
mobility and select the cabable cloudlets:

t1,t2,t3 --> {C1,C4,C2,C3}

5-Scheduler will  calculate all levels with offline
mapping  so will move to next levels:

Level-1: R = t4, t5, t6, t7
```

$|\mu(3) \cup \mu(4) \cup \mu(5)| \geq |R|$

```
so child tasks will start in the max-finish time of
parent tasks:

t4, t5, t6, t7 --> {C5, C7, C8, C6, C9}

Repeate 5 with Level ++:
level-2:  t8

* t8->(C13,C11)
* t8(7,c13)
```

Task-ID	Time Unit	Available Cloudlets	Delay & Execution time for available cloudlets	Start Time	Finish Time	Selected Cloudlet
t1	1	C1, C4, C3, C2	[1],[1],[2],[2]	0	2	C1
t2	1	C1, C4, C2, C3	[1],[1],[2],[2]	0	2	C4
t3	1	C1, C4, C2, C3	[1],[1],[2],[2]	0	3	C3
t5	3	C7, C8, C6, C9, C5	[4],[5],[5],[6],[7]	3	5	C7
t7	3	C7, C8, C6, C9, C5	[4],[5],[5],[6],[7]	3	6	C8
t4	4	C7, C8, C6, C9	[4],[5],[5],[6]	4	6	C6
t6	4	C7, C8, C6, C9	[4],[5],[5],[6]	4	7	C9
t8	7	C13, C11	[7].[13]	7	8	C13

Mapping Offline for All Tasks and Cloudlets during Mobility

Figure 4.4: Illustration for Proposed Case Study

of the best cloudlet C_7 is equal to 4.0s, so the updated delay time will be equal to 1.0s. As the user is moving, we will add it to the start time of mapping with the defined time of uploading, execution, downloading, transferring to give a completion time of (5.0s), as seen in Figure 4.4.

4.6 Summary

In this chapter, the distributed offloading problem in edge networks has been addressed and we explained the main challenges with task offloading in distributed environment. Current works in offloading suffer from at least one of these several limitations: (i) assume independent tasks, (ii) assume static users, (iii) assume a central server that performs the offloading on behalf of users or (iv) focus on a single user. On the other hand, in this thesis, we address all these challenges together by developing the *algorithms* for offloading *dependent* tasks by *multiple mobile* users. Overall, we give a simple example as a case study to explain more the offloading process with dependent tasks and the user mobility.

Chapter 5

Scheduling of Dependent Tasks in Edge Networks Using a Centralised Delay Server

Cloud computing has enabled the on-demand availability of computing resources and power to end users. Often, the cloud resources predominant today have functions that are distributed over multiple locations, remote from the actual users. Due to issues of communication latency, these services are now being put at the periphery of the network, now called edge computing. When these services are being accessed by mobile users, such a network is called a Mobile Edge Network (MEN). A MEN consists of a number of base stations or cloudlets that execute tasks on behalf of users, i.e., users offload tasks to the MEN nodes which then execute them before returning the results to the users. This chapter focuses on the novel problem of scheduling *dependent tasks* on the MEN with the objective of reducing the completion time of the job. We formalise the problem as a constraint satisfaction problem and we provide a heuristic for scheduling the dependent tasks. We conduct simulation experiments in the ns-3 network simulator to study the performance of the proposed heuristic and we have implemented the datasets in CPLEX (PULP) optimization. We run a deployment with Face Recognition app (android studio) and Flask server side. Our results show that (i) it is impossible to obtain a schedule when the speed is very high, (ii) when the speed is low, it is better to allow the tasks to run locally on the mobile device and (iii) when a MEN schedule is obtained, the reduction in completion time is proportional to the size of a job. For instance, in our work, we obtained completion times reduction of $\approx 45\%$.

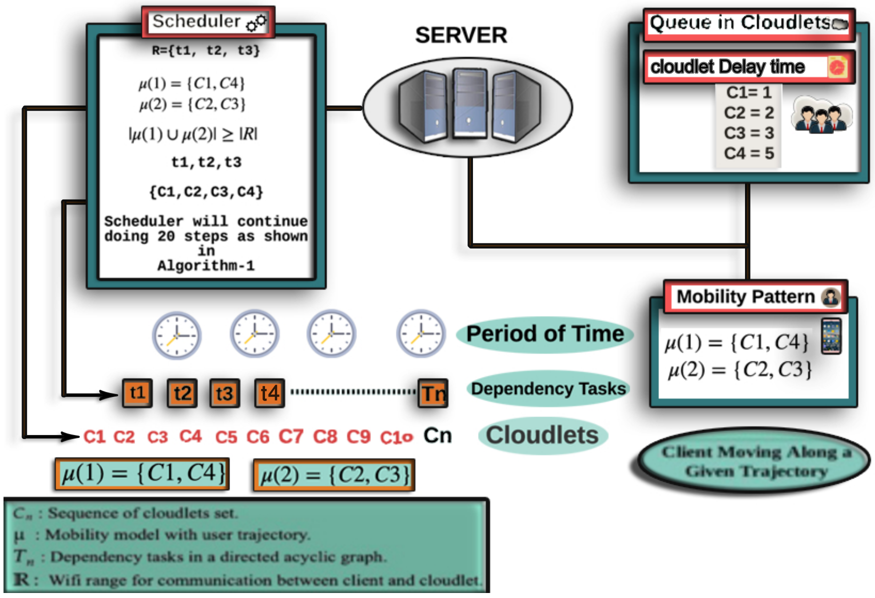

Scheduler
R={t1, t2, t3}
μ(1) = {C1, C4}
μ(2) = {C2, C3}
|μ(1) ∪ μ(2)| ≥ |R|
t1,t2,t3
{C1,C2,C3,C4}
Scheduler will continue doing 20 steps as shown in Algorithm-1
SERVER
Queue in Cloudlets
cloudlet Delay time
C1= 1
C2 = 2
C3 = 3
C4 = 5
Mobility Pattern
μ(1) = {C1, C4}
μ(2) = {C2, C3}
Period of Time
Dependency Tasks
Cloudlets
t1 t2 t3 t4 Tn
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 Cn
Client Moving Along a Given Trajectory
μ(1) = {C1, C4}
μ(2) = {C2, C3}
Cn : Sequence of cloudlets set.
μ : Mobility model with user trajectory.
Tn : Dependency tasks in a directed acyclic graph.
R : Wifi range for communication between client and cloudlet.

Figure 5.1: System Model

5.1 Models

In this chapter, we developed the model as a constraint satisfaction problem with a user device (client), the edge nodes (cloudlets), and a mobility model which considers different speeds of movement. We provide a heuristic algorithm for scheduling the dependent tasks during user mobility and we perform extensive simulations using ns-3 with three different job graphs. We model the problem using Integer Linear Programming (ILP) in CPLEX and PULP [13, 45] and obtain the optimal offloading strategy. Finally, we create a small testbed by using Flask server and obtain a set of dependent tasks using a Face Recognition Application [39, 100]. We provide further details in next sections.

5.1.1 System Model

The system model is depicted in Figure 5.1. The system consists of a client moving along a given trajectory and MEN of cloudlets. Given that the scheduling problem is intractable, exponential complexity seems unavoidable (unless P=NP). An optimal solution to the scheduling problem can be obtained when network-wide information is available. However, to avoid the exponential complexity and to obtain a good schedule in reasonable time, our system model will provide network-wide information and we seek a heuristic approach. In the following sections, we provide information about the various aspects of the network that are needed.

Delay Server: To gather network wide information, we further assume that there is a server that, when queried at time τ, returns the delay associated with each cloudlet at τ. This server is required as it will allow the user to decide whether to postpone offloading a task to the MEN to a later cloudlet (rather than one which is currently in range). In essence, this delay server allows the user to get network-wide delay, i.e., the delay server simulates the case where there are multiple users in the network.

The client, cloudlet and mobility models are as explained in Chapter 4

5.1.2 Offloading Model

Definition 3 (Schedule) *We first define a schedule as follows: a schedule is function $S : T \times \tau \rightarrow C$ that takes a task $T_i^j \in T$ and a given time t and returns the cloudlet c onto which T_i^j needs to be at t.*

Thus, the problem is as follows: Given a job $J = (T, D)$, where T is the set of tasks and D the dependencies between the tasks, a set C of cloudlets, a mobility pattern (sequence of sets of cloudlets), which we denote by μ, develop a schedule such that certain constraints are satisfied, which we detail below. Tasks and the node of the edge have the following metadata:

- node: The node on which the task will be offloaded to, based on the schedule.
- size: The size of the task, in terms of code footprint and data.
- result: The result of the task after completion.
- offloading time: The time at which the task is offloaded to the selected node.
- start time: The time at which the task execution starts on the selected node.
- execution time: The (worst case) execution time of the task on the selected node.
- finishtime: The time at which the task completes on the selected node.
- uprate: the rate at which data can be uploaded, i.e., upload speed.
- freq: The frequency of the node's CPU.
- delay: The delay associated with the node at a given time.

Henceforth, we will denote by $t.x$ (resp. $n.x$), the value of attribute x of task t (resp. node n). We will also denote a task using small letters whereas we will use capital letters to denote a set of tasks.

Given a job $J = (T, D)$, we define a function called $FinishTime_S$, the finish time of the job T under schedule S as follows:

$$FinishTime_S(T) = max\{t.finishtime \mid t \in T\} \tag{5.1}$$

The objective of the scheduling problem is to minimize the execution time of the job, i.e., maximising the reduction in execution time as compared to a local computation. Thus, we formulate the objective as follows:

Minimize:

$$argmax_{MS}|FinishTime_{MS}(T) - FinishTime_{LS}(T)| \tag{5.2}$$

Subject to:

$$\forall t \in T, S(t) \neq \perp \tag{5.3}$$

$$\mu(FinishTime_{MS}(T)) \neq \emptyset \tag{5.4}$$

$$\forall t_1, t_2 \in T, t_1.node \neq t_2.node \tag{5.5}$$

$$\forall t \in T \cdot t.node \in \mu[t.OffloadTime] \tag{5.6}$$

$$\forall t_1 \to t_2 \in D, t_2.starttime > t_1.finishtime \tag{5.7}$$

As we can see with the above constraints, the first condition captures the fact that all tasks need to be offloaded to an edge node. The second constraint is that, the client needs to be within communication range of a cloudlet when the job finishes so it can receive the final result. Since μ is the sequence of sets of cloudlets (mobility model), this means that, at the time the job completes, there should still be at least one cloudlet within the range. The third constraint means no two tasks can be offloaded to the same cloudlet. The fourth one means a task can only be offloaded to a node with which it is in direct communication with. For the last constraint, where there are two task dependencies, the dependent task can only start after the main task completes.

Local Finish time: The objective is to maximise the difference in completion time between an offloaded job and the job being run locally on the device. The local completion time of the job is given by :

$$F(T, L) = \sum_{t_i \in T} e(t_i), \tag{5.8}$$

Finish Time in Edge: We now detail how the various attributes of a task is computed when a job is offloaded. The following are two important constraints:

$$\begin{aligned}\forall t_1 \rightarrow t_2 \in D \cdot t_2.starttime \geq (t_1.finishtime + TransferringTime(&\\ t_1.result.size, t_1.node, t_2.node))&\end{aligned} \tag{5.9}$$

$$\begin{aligned}\forall t_1 \rightarrow t_2 \in D \cdot (t_2.offloadtime \leq t_1.finishtime + TransferringTime(&\\ t_1.result.size, t_1.node, t_2.node))&\end{aligned} \tag{5.10}$$

The first condition means that the start time of a dependent task t_2 has to be after the finish time of the main task t_1 and after the result has been transferred from the node executing t_1 to the node executing t_2. The second condition means that a dependent task t_2 has to have been already offloaded to a selected node before its main task has finished executing and the result has been transferred to the required node, i.e., the result will "wait" for the task. The offload time of a task t on cloudlet c is given by:

$$\begin{aligned} t.offloadtime = \tau + \lceil \frac{(t.size)}{(c.uprate)} \rceil \\ t.node = c, \tau = min\{\alpha | c \in \mu(\alpha)\} \end{aligned} \tag{5.11}$$

A task is offloaded onto a cloudlet as soon as the cloudlet becomes in direct communication with the mobile device and it is resident on the edge node. The completion time of a task is given by:

$$t.finishtime = t.starttime + t.executiontime \tag{5.12}$$

After a task has been offloaded, the latest it will wait before starting execution is either when the delay is expired or when the results it depends on become available:

$$\begin{aligned} &t.starttime = t.offloadtime + max(t.node.delay, t'.resulttime+ \\ &TransferringTime(t'.result.size, t'.node, t.node)), t' \rightarrow t \end{aligned} \tag{5.13}$$

The execution time of a task on an edge node will depend on the number instructions of the task and the cloudlet computation resources:

$$t.executiontime = \lceil \frac{(t.size)}{(t.node.freq)} \rceil \tag{5.14}$$

The result time of a task is the time at which all the results needed from its parents tasks become available. This is computed as follows:

$$\begin{aligned} t.resulttime = max\{&p.starttime + p.executiontime(p.node) + \\ &TransferTime(p.result, p.node, t.node) | p \rightarrow t\} \end{aligned} \tag{5.15}$$

The last term, TransferTime, denotes the time taken for transferring results between nodes of the edge network, as the MEN is completely connected. TransferTime takes in three parameters, (r, s, d), where r denotes the result of the task, s denotes the source of the results and d denotes the recipient node of the result. Thus, TransferTime is given as follows:

$$TransferringTime(r, s, d) = \frac{(r.size)}{(Tx.rate(s, d))} \tag{5.16}$$

5.2 Integer Linear Programming (ILP) Model:

We develop ILP (Integer Linear Programming) Model to find the optimum allocations of the offloading dependency tasks to the edge with the minimum finish time as shown in Figure 5.2. We defined the linear objective function, constraints, decision variables and the solver. The objective function was represented as a linear equation to find the optimal dependency tasks allocation in the edge with the best minimum execution time during the allocation process.We defined the constraints as linear inequalities to delimit the solutions to the problem and the optimal solution that meets the conditions sought by the main objective function. We defined three constraints: (i) each task among dependencies should be allocated to only one cloudlet, (ii) the cloudlets in the edge cannot be reused more than one time, (iii) representing dependency in the DAG with parent and child tasks as shown in the DAG of Figure 5.2. Since we consider the user mobility as a sequence of a set of cloudlets in the path of the user, the child task should be allocated to the cloudlet node that follows the parent task's cloudlet node. So, we represent the user mobility paths as a sequence of cloudlets that have unique (IDs) in ascending order, where the selected cloudlet node ID for the child task should be greater than the cloudlet ID for the parent task. We have represented the decision variable in the model as a binary variable (0,1) to allocate the dependency tasks to the cloudlets in a simplex method using the CPLEX Solver [13].

The resource allocation problem is as follows: Given a job $J = (T, D)$, where T is the set of tasks and D the dependencies between the tasks, a set C of cloudlets, a mobility pattern (sequence of sets of cloudlets with unique (IDs) in ascending order to represent the user mobility). Tasks and the nodes of the edge (cloudlets) have the following metadata that are represented as numbers in the ILP model. Cloudlet (C): the edge node to which the task will be allocated based on the allocation decision, where it has unique CPU capabilities ($c_{freq.}$) for each node in the edge. Task size (t_i): the number of task instructions that will be executed on the edge node. Server delay time (c_d): the delay associated with the edge node (cloudlet) at a given time. After defining the variables with numbers , we defined the allocator (ALOC) as a binary decision variable with (0, 1):

$$ALOC_C^T = \begin{cases} 1, & \text{if tasks T allocate to the node on the edge C} \\ 0, & \text{Otherwise} \end{cases} \tag{5.17}$$

Given a job J = (T,D), we define a function called Completion Time (CT) , the total completion time of the task T under the allocator ALOC as follows:

$$\begin{gathered} CT_{t,c} = max\{Execution\ Time\ _{(t,c)} + Edge\ Delay\ Time\ (c_d)\} \\ where, Execution\ Time\ _{(t,c)} = (\ t_i\ /\ c_{freq.}) \end{gathered} \tag{5.18}$$

The objective of our allocation problem is to minimize the execution time of the job and allocate the dependency tasks to the best cloudlet in the edge that satisfying the objective. Thus, we formulate the objective as follows:
Minimize:

$$min \sum_{\forall t \in T, \forall c \in C} ALOC_c^t \cdot CT_{t,c} \tag{5.19}$$

Subject to:

$$\sum_{\forall c \in C} ALOC_C^t = 1 \tag{5.20}$$

$$\sum_{\forall t \in T} ALOC_c^t \leq 1 \tag{5.21}$$

$$\begin{gathered} \sum m + ALOC_{CID}^{t_l} + ALOC_{C\grave{I}D}^{t_k} = 1 \\ CID < C\grave{I}D, \quad \forall (CID, C\grave{I}D) \in C \\ StartTime(t^l) \geq FinishTime(t^k), \quad \forall (t^k \rightarrow t^l) \in D \\ where,\ m\ is\ defined\ as\ a\ binary\ variable\ (0,1) \end{gathered} \tag{5.22}$$

5.3 Integer Linear Programming (ILP) Evaluation

We installed Python 3.6 with the PULP module [44, 45, 76, 77]. The experiment was carried out using Linux (Ubuntu) with processor CPU: Intel i5-2400 4 cores, GPU: NVD9, and RAM: 16 GiB. We modeled the dependency tasks with DAG of (8-tasks) as shown in (Figure 5.2), and we modeled the edge network as (30-cloudlets) which have unique IDs. Each task has its unique number of instructions and each cloudlet has unique CPU capability and delay time groups. We defined global variables that keep track of the total tasks and cloudlets and arrays to store datasets of cloudlet CPU, cloudlet delay times and the number of instructions in the tasks. To define the optimization problem in PULP we used the function LpProblem to define the ILP problem and LpMinimize to specify it as a minimization problem based on our objective. We created a new array called "completion˙time" to store the computations of the total completion time for a given task to a given cloudlet as mentioned above in equation (5.18). We defined the allocator (ALOC) as a binary decision variable to represent the strategy of resource allocation based on our objective as mentioned in equation (5.17).

We have represented the dependency tasks of DAG with child task and parent task as the order of the DAG in (Figure 5.2) shows. To model the mobility model in the ILP, we have to make sure the cloudlet selected for the child node was be greater than that of the parent nodes. All cloudlets have a unique (IDs) in ascending order which helped to determine the order of the cloudlets. We used the same datasets of the delay groups in ILP model and

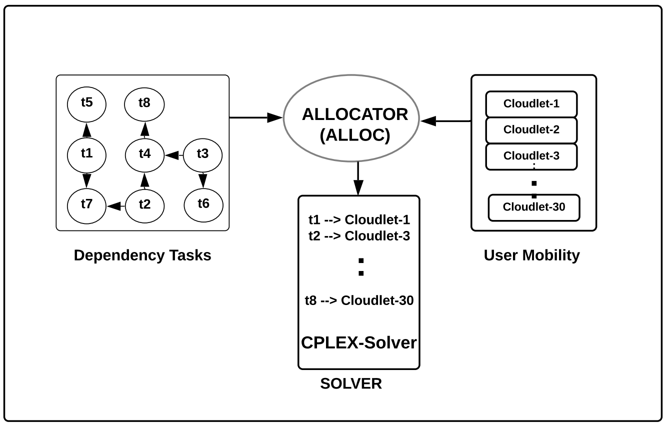


Figure 5.2: ILP System Model

ns-3 simulation with the Heuristic algorithm. We have compared the results of the ILP model as optimal solutions and the Heuristic algorithm results that we got in ns-3. In the ILP model, we run the experiment with different delay groups (10 delay groups) with 5 increased groups and 5 decreased groups as shown in (Figure:5.3, and Figure 5.4).

We used two different sets of delay time groups as follows: (i) the further away a cloudlet is from the starting point, the higher is the delay. We call this set the increasing delay set and, (ii) the further away a cloudlet is from the starting point, the lower is the delay. We call this set the decreasing delay set. The increasing delay sets are labelled delay D-0 to delay D-4, while the decreasing delay sets are labelled delay D-5 to delay D-9.

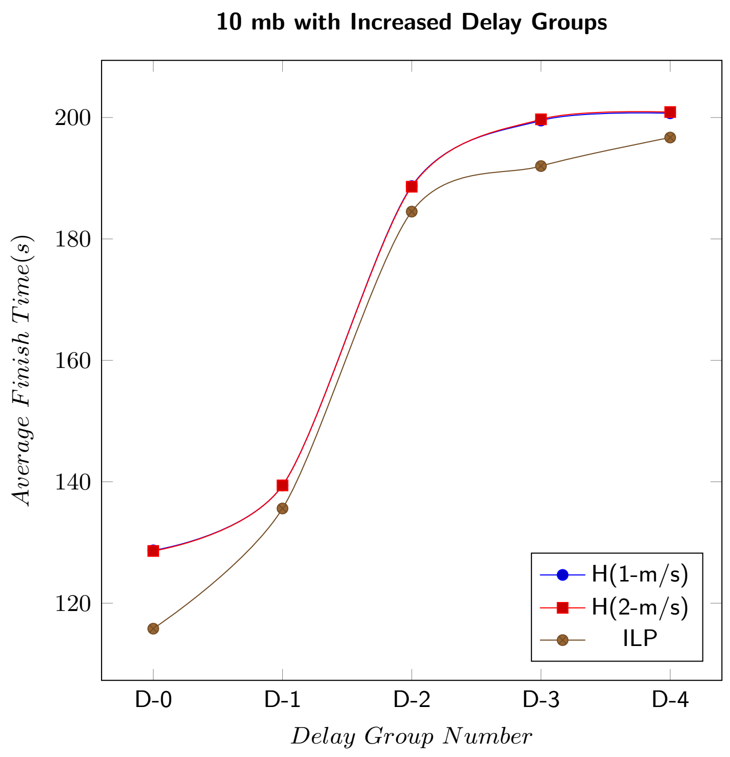
Increased delay groups with heuristic algorithm with different user speeds were close to the optimal results from the PULP. With delay groups, D-1 to D-4 with respect to the network variables and user speeds, our heuristic algorithm was almost optimal as shown in (Figure 5.3). This is due to the fact that the smallest task completion times happen at the start of the journey, thereby reducing the job completion time. So with small delay groups in the edge it is better to offload the task early and make it run on the edge.

On the contrary, as observed for the decreased delay groups, D-5 to D-9, our heuristic algorithm was far from the optimal solution as shown in (Figure 5.4), even accounting for the network fluctuation and user speeds. This is due to the fact that a user cannot know that shorter delay will happen at a later time and can thus defer the offload, i.e., due to the user having only local delay information. As a result, it is better to run the task locally in the mobile device when the edge has a pressure of the delay time.

5.4 Heuristics Offloading Algorithm

In this section, we present a heuristic offloading algorithm that returns a schedule for a job with dependent tasks in an edge system. As previously mentioned, the mobile device will query the delay server for the delays at the various edge nodes. Our heuristic will then use the delays and the edge network to determine an offloading schedule that will reduce the computation time of the job. The multiprocessor scheduling problem with precedence constraints is known to be an NP-complete problem. To circumvent the complexity of the problem, we developed a heuristic that would provide good enough solutions for several instances of the problem. The heuristic attempts to reduce the completion time of the job by reducing the finish times of individual jobs. The heuristic is shown in *Algorithm 1.* We will explain the algorithm in details as follows:

- (Lines 1-15): To attempt to reduce the completion time, the heuristic shown in *Algorithm 1* offloads tasks as soon as possible onto an edge node even before the parents have completed. Since no dependency exists for the root tasks, they can be scheduled as soon as they are ready. Thus, if there are r tasks, then the first r edge nodes need to be selected and tasks assigned to them in such a way that reduces their completion times.
- (Lines 16-22): The scheduling algorithm then uses this information to decide when and where to schedule child tasks. The set of possible nodes where a task T can be scheduled is calculated based on the computation times of its parents. Specifically, the set of nodes will comprise those



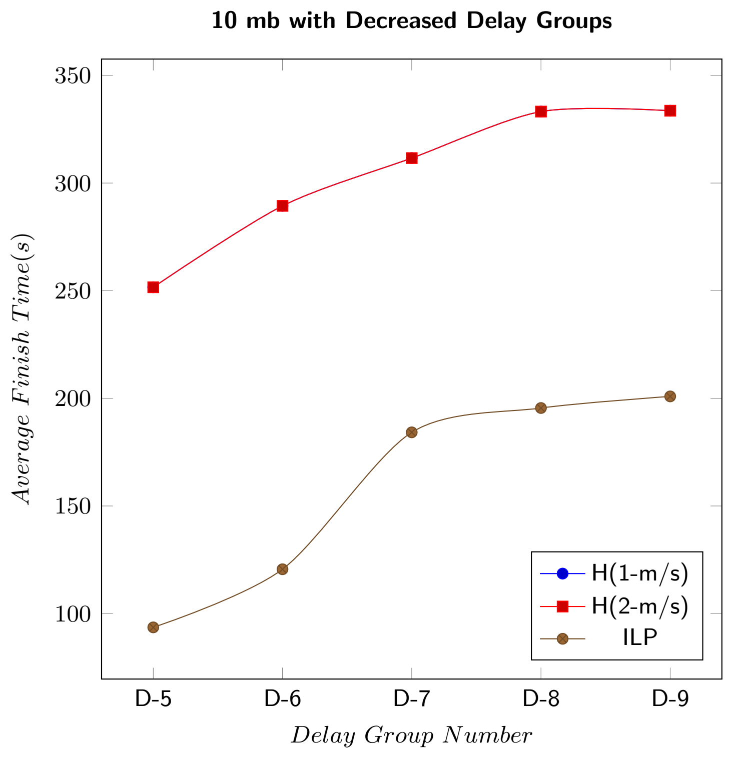
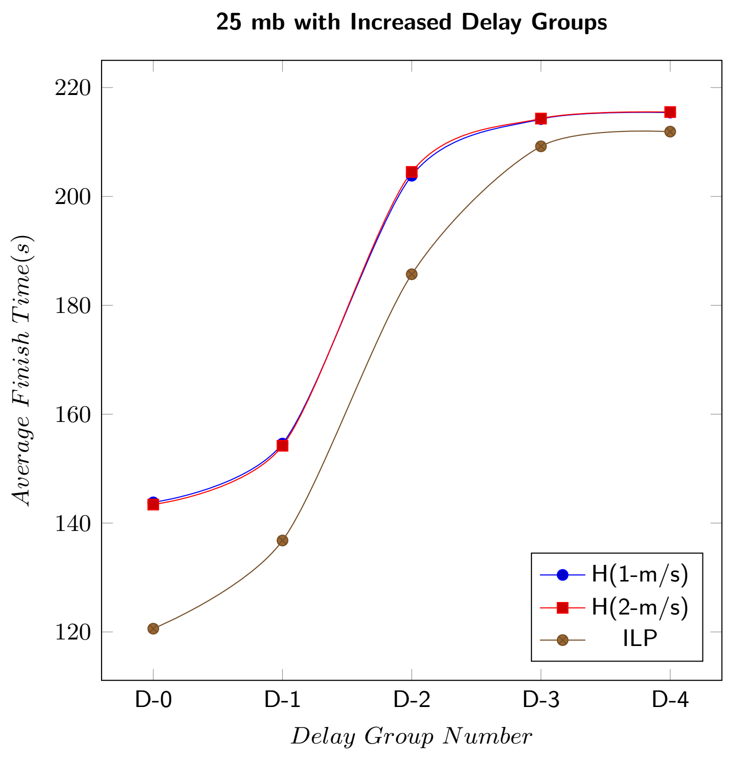


Figure 5.3: Completion Time with ILP Optimal Solution and (Heuristic Algorithm with Different User Speeds) in 10-mb and (5-Increased, and 5-Decreased Delay Groups in the Edge)



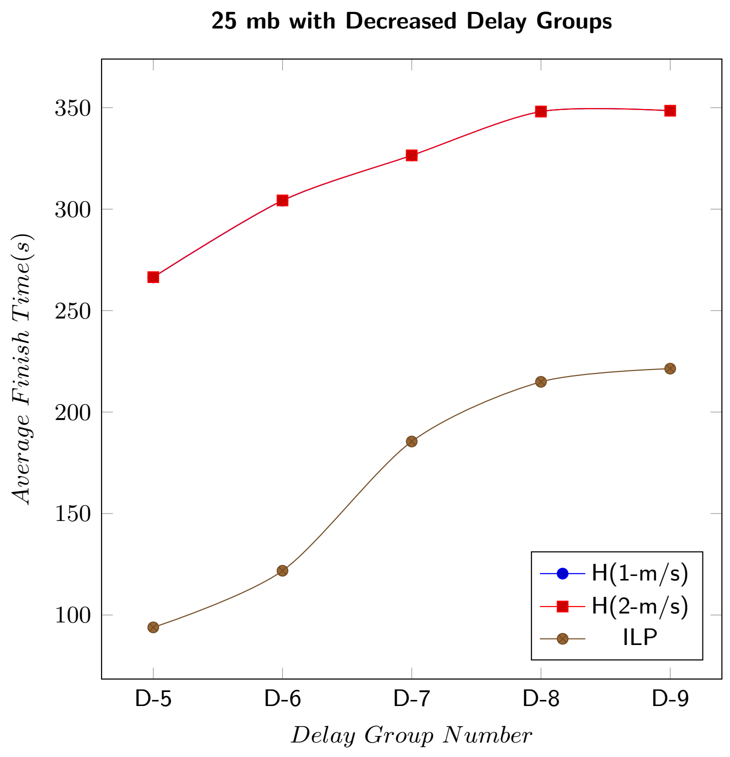


Figure 5.4: Completion Time with ILP Optimal Solution and (Heuristic Algorithm with Different User Speeds) in 25-mb and (5-Increased, and 5-Decreased Delay Groups in the Edge)

Algorithm 1 High Level Description of Heuristic Offloading Algorithm

```
 1: procedure SCHEDULE($J, \mu, \mathbb{D}$)        J: Job, $\mu$: Seq of nodes set, $\mathbb{D}$ : Delays
 2:
 3:     Let number of root tasks be $r$
 4:
 5:     Choose $r$ cloudlets from $t = 1 \ldots \tau$, where $|\bigcup_{p=1}^{\tau} \mu[p]| \geq r$
 6:
 7:     Assign cloudlets to root tasks so as to reduce completion time.
 8:         No two tasks share same edge node.
 9:
10:     $current\_level := 1$
11:
12:     while ($current\_level \leq J.depth$) do
13:
14:         $current\_level + +$
15:
16:         while ($\exists\, T \in J, T.level = current\_level$) do
17:
18:             $start := min\{t | p.starttime = t, p \rightarrow T\}$ (Equation (5.1))
19:
20:             $end := max\{t | p.finishtime = t, p \rightarrow T\}$ (Equation (5.12))
21:
22:             $nodes := \bigcup_{\tau \in [start \ldots end]} \mu[\tau]$
23:             if ($nodes = \emptyset$) then
24:
25:                 "No Schedule Exists!"
26:
27:             else
28:                 $T.node :=$ node $c \in nodes$ that reduces finish time of $T$
29:
30:             $nodes := nodes \setminus \{T.node\}$
31:
32:             $T.offloadtime :=$ Equation (5.11)
33:
34:             $T.starttime :=$ Equation (5.13)
35:
36:             if $\exists T, T.level = J.depth \wedge T.finishtime \geq$ (local completion time of $J$ -
37:                 Equation (5.8) then
38:
39:                 "Better not offload"
40:
41:
```

nodes that are directly reachable from when one of its parents first starts executing till the last of its parents completes.

- (Lines 23-41): The last check, i.e., the decision whether to offload or not, is based on whether the completion time is smaller than the local computation time. This can happen, for example, when the delays on the edge network are commensurate with the mobility time. Among the possible nodes, we assign nodes to tasks so as to reduce the completion time of the task (Line 28). Of course, it is possible that no schedule exists if, for example, the user is moving too fast and runs out of edge nodes. Then, the algorithm assigns values to task attributes according to equations derived earlier.

We will illustrate the algorithm with an example of DAG (8-tasks) as shown in Figure 5.5, which consists of three levels and 8 dependency tasks as : $(t_1 \rightarrow t_5)$, $((t_1, t_2) \rightarrow t_7)$, $((t_2, t_3) \rightarrow t_4)$, $(t_3 \rightarrow t_6)$ and $(t_4 \rightarrow t_8)$. We consider in this example the user mobility pattern has a sequence of set of cloudlets in different time unit as: $\mu(1) = \{C1, C4\}$, $\mu(2) = \{C2, C3\}$, $\mu(3) = \{C5\}$, $\mu(4) = \{C7, C8\}$, $\mu(5) = \{C6, C9\}$, $\mu(6) = \{C10, C12\}$, $\mu(7) = \{\emptyset\}$, $\mu(8) = \{C13, C11\}$. Scheduler will check the set of cloudlets are bigger than or equal to the set of tasks before mapping process (Line 1-6). In level 0 of DAG we have root tasks: (t_1, t_2, t_3) and on the first two unit times of mobility model we have $\mu(1) = \{C1, C4\}$, $\mu(2) = \{C2, C3\}$. Scheduler will sort root tasks according to task ids and allocate each to set cloudlets in mobility pattern after check: $|\mu(1) \cup \mu(2)| \geq |R|$ whereas, $|R|$ is the set of root tasks in level 0 (Line 7-8). In level 1 of DAG we have set of dependent tasks (t_5, t_7, t_4, t_6) with start time bigger than or equal to the maximum finish time of the root tasks (t_1, t_2, t_3). In the level 2 (t_8) will start after max finish time of (t_4) and the process of mapping will be completed with this task as it is the last leaf node of the DAG. Scheduler will calculate maximum finish time for set tasks and do mapping with set capable cloudlets in different unit time. After calculating the expected delay time and execution time for set tasks with set of cloudlets, scheduler will assign tasks to set of cloudlets that help to reduce delay time with dependency tasks (Line 10-21). To start with the new level of the DAG, scheduler will check the DAG to offload all tasks at the current level to the available cloudlets with repeating previous steps (Line 22-34). The offloading to the edge will be recommended when the completion time in the local device take long time of the task execution (Line 36-41).

We implemented the heuristics algorithm in a ns-3 simulation as shown in (Section-5.5) and we implemented it with a real life application of dependency tasks as shown in (Section-5.6). In the real deployment, we configured two parts of the system: client (mobile device) as one user and six edge nodes as (cloudlets). The deployment was run at the University of Warwick with user walking mobility. We created a Facial Recognition application in Android studio modelling the recognized pictures as dependency tasks in six levels of the DAG. We will go with details in the next sections.

5.5 ns-3 Simulation

In this section, we first present the experimental setup for the simulation experiment conducted to evaluate the proposed heuristic algorithm. We then present a sample of the results of the evaluation with ns-3 with different DAG(s) as shown in Figure 5.5. We will summarize them as follows:

5.5.1 ns-3 Experimental Setup

The scheduler was written in C++. It takes the following as input: (i) the job (as a directed acyclic graph), (ii) the delays of the various edge nodes, (iii) the mobility model and (iv) the edge system topology. The scheduler then returns the offloading schedule for the task set. The client then used the generated schedule to offload the task in real-time. For the actual offloading, we used ns-3, an open and extensible discrete network simulator for internet systems. It provides various communication technologies, such as Wi-Fi functionality, in order to simulate a wide variety of communication scenarios. The parameters and the values used in the simulations are summarized in Table 5.1.

Table 5.1: Simulation Settings in Chapter 5

Parameter	Value / Range
Total Job Size	10 ~35 MB
Mobility Model	WayPoint Mobility Model
Cloudlet Position	Constant Position Mobility
Moving Speed of the Mobile User	0.5, 1.0, 1.5, 2, 3 [m/s]
Cloudlets Uplink/Downlink	DsssRate 5 ~ 11 Mbps
Intra MEN Transmission Rate	1000 Mbps
Cloudlet CPU Frequency & Instruction Set	2.0 Ghz & CISC
Mobile Device CPU Frequency & Instruction Set	2.0 Ghz & RISC
Acceleration ratio between CISC and RISC	10 ~ 50
CPU Cycles Required by One Task	2×10^8 ~ 2×10^9
The ratio of input/output data size	2:1

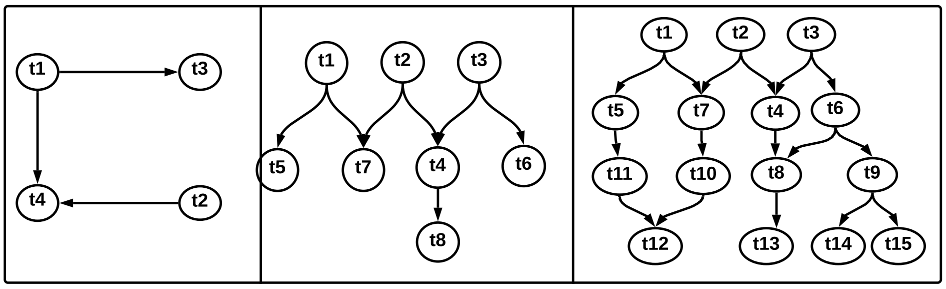


Figure 5.5: Directed Acyclic Graphs with (4 -Tasks, 8 -Tasks, and 15 - Tasks)

We used the WiFi-Phy 802.11-b standard configuration in ns-3 as we have modelled the cloudlet subsystem as a mobile edge network (MEN) in our work [46][89]. The data transmission uplink/downlink speed between a mobile device and a cloudlet is set to be in the range of 5 Mbps to 11 Mbps with the WiFi-Phy model. Furthermore, the data transmission uplink/downlink between cloudlets is set at 1 Gbps, as an Ethernet connection in a point-to-point model. We have used the waypoint mobility model for the client moving with the mobile device.

Each object in a waypoint mobility model determines its speed and position at a given time from a set of waypoint objects. The protocol used for the communication between a mobile device and a cloudlet is UDP socket with Type-Id.

Client Setup: Our simulation setup also includes a client with a mobile device moving at a constant speed at a run. Five different speeds, namely $0.5, 1, 1.5, 2, and$ 3 m/s, are used in our experiments. The same sequence of sets of cloudlets was laid along the path of the mobile device, from a start position to an end position, and the mobile device offloads tasks according to the schedule generated by the scheduler. We used jobs of sizes[1] $[10 \ldots 35]$ MB, in steps of 5MB. The jobs have the same precedences, i.e., the jobs can be represented using the same DAG. The reason for this is to understand the impact of different parameters on the resulting schedule.

Network Setup: The simulation process was repeated for all tasks in (DAG) with different groups of delay times, speeds, and task sizes. We focused on two different sets of delay times, measured in minutes, as follows: (i) the further away a cloudlet is from the starting point, the higher is the delay. We call this set the increasing delay set and, (ii) the further away a cloudlet is from the starting point, the lower is the delay. We call this set the decreasing delay set. The increasing delay sets are labelled delay 0 to delay 4, while the decreasing delay sets are labelled delay 5 to delay 9. We used 30 cloudlets, with each cloudlet (c_i) being labelled with the distance from the starting point. For example, cloudlet (c_5) is 120 m away from the starting point. We used 10 delay sets, with 5 increasing delay sets and 5 decreasing delay sets; the delay associated with each cloudlet is indicated in the delay array. We run a total of 300 experiments during the simulation (5 speeds, 6 job sizes and 10 delay distributions), excluding the baseline computation carried out on the local device.

5.5.2 ns-3 Evaluation and Results

In this section, we now present the results of our simulation experiments. We observed that there is an increased relation between the completion time and the completion distance through the results in (Figures: 5.6, 5.7, 5.8) and (Figures: 5.9, 5.10, 5.11). Also we observed that there is an increased relation between the completion distance and the speed through the results in (Figures: 5.14, 5.15) and (Figures: 5.12, 5.13). Due to reasons of space, we will present only a sample of the results as follows.

Completion Time: We first investigate the impact of speed on the

[1]We use the terms Job Size and Total Task interchangeably

completion time. The results are shown in the Figure 5.6 for DAG with 4 tasks, Figure 5.7 for DAG with 8 tasks, and Figure 5.8) for DAG with 15 tasks. In all figures, the X-axis represents the size of the job completed (in MB), while the Y-axis represents the average completion time. As can be observed, when the job is fairly small (10-20 MB) in Figures: 5.6, 5.7, 5.8, it is better to execute the job locally (i.e., on the mobile device). This is due to the case that the delay associated with each cloudlet induces a large enough temporal overhead compared to the computation time of the jobs. If all the delays are small enough to compare with the execution times of the tasks (i.e., the queues are small), then it may be possible to schedule a job in the MEN. On the other hand, when the size of the job is higher in Figures: 5.6, 5.7, 5.8, then the heuristic suggests that the job is scheduled in the MEN, as the jobs takes longer to execute. Then, the total computation time of the job in the cloudlet is less than the local computation time. From a job of size 20-35 MB, the reduction in completion time becomes more significant as the job size increases.

Completion Distance: In this part, we have the distance graphs which are related to the same graphs in the completion time part: (Figures: 5.6, 5.7, 5.8).

The Figure 5.9 (for DAG with 4 tasks), Figure 5.10 (for DAG with 8 tasks), and Figure 5.11 (for DAG with 15 tasks) show the type of schedule that is being produced. As we see in (Figures: 5.9, 5.10, 5.11) when the size of a job increases, the completion distance increases too. Similar to completion times, this is due to the fact that it takes longer for job to complete and thus the user needs to move further.. Also, as the speed increases, it can be clearly observed that the completion distance increases. This is due to the fact that, as the speed increases, by the time a task completes, the user has already moved a significant distance. As such, the later tasks are executed on cloudlets that are further away from the starting point.

Impact of Very High Speed: In Figures: 5.12, and 5.13), the X-axis represents the total task size (in MB) (it is a total job size for all 8 tasks), while the Y-axis represents the average completion time. In Figures: 5.14, and 5.15), the X-axis represents the total task size (in MB) (it is a total job size for all 8 tasks), while the Y-axis represents the distance in meter.

From Figures: 5.14, and 5.15), we can see that the completion distance increases direct relation to the user's speed. This suggests that there may be a cut-off speed at which no schedule exists. Focusing on a small 10MB job (see the first graph in (Figure 5.12), it can be observed that all 8 tasks are scheduled when the delay increases as the user moves along the path (delay groups 0-4). This suggests that, before the delay got large, all the tasks were already scheduled. It can also be observed that it was only better for the job to be scheduled in the MEN for delay groups 0 and 1, i.e., for very small delays. When the delays get larger, it becomes more beneficial to execute the job

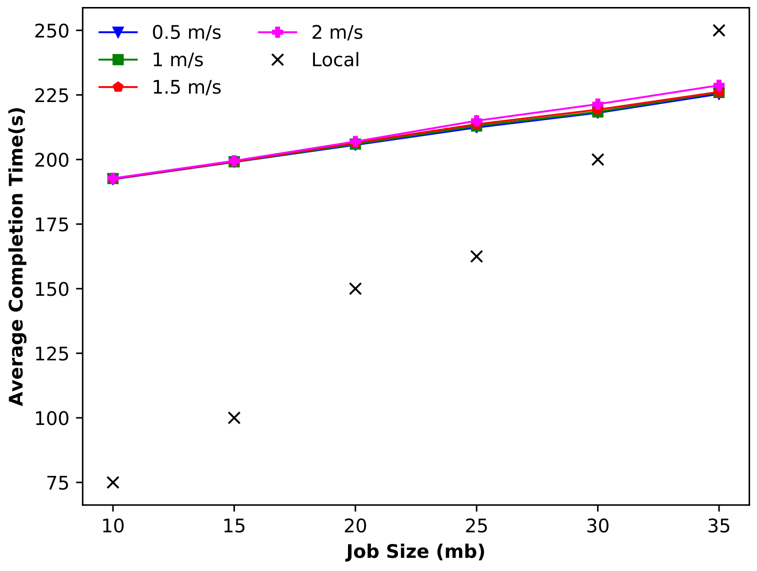


Figure 5.6: Average Completion Time for (DAG with 4 - Tasks)

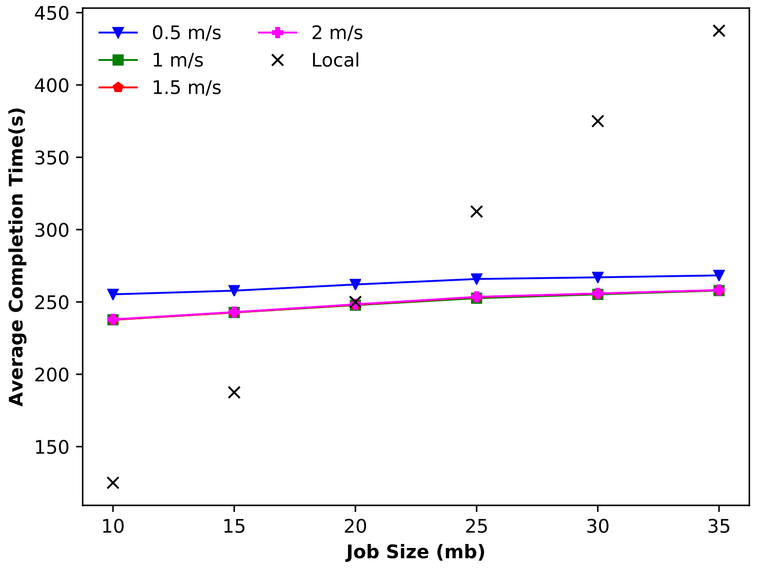


Figure 5.7: Average Completion Time for (DAG with 8 - Tasks)

locally. On the other hand, when the delay decreases as the user moves (delay groups 5-9), some tasks were not scheduled. This can be better understood by looking at the completion distance of the tasks. For the decreasing delay group, tasks are scheduled later (on later cloudlets) and the user's journey is over before all tasks could complete.

As similar scenario is observed when a large job is executed (see Figure

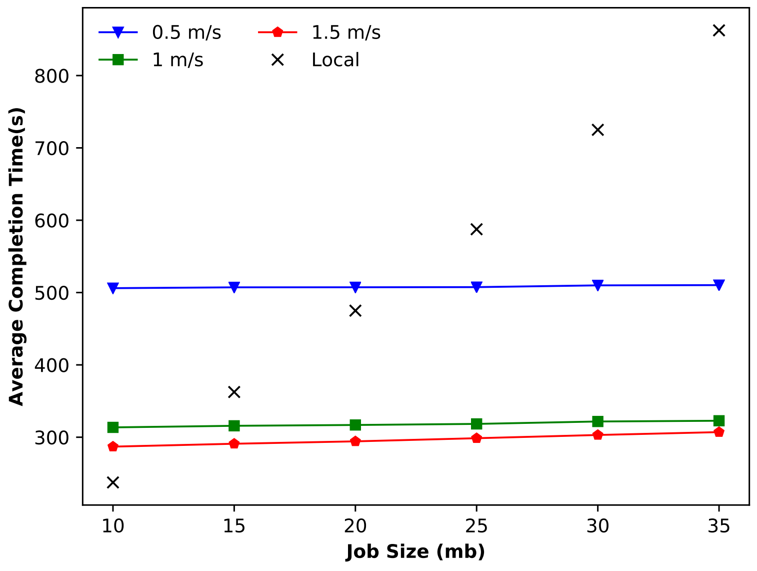


Figure 5.8: Average Completion Time for (DAG with 15 - Tasks)

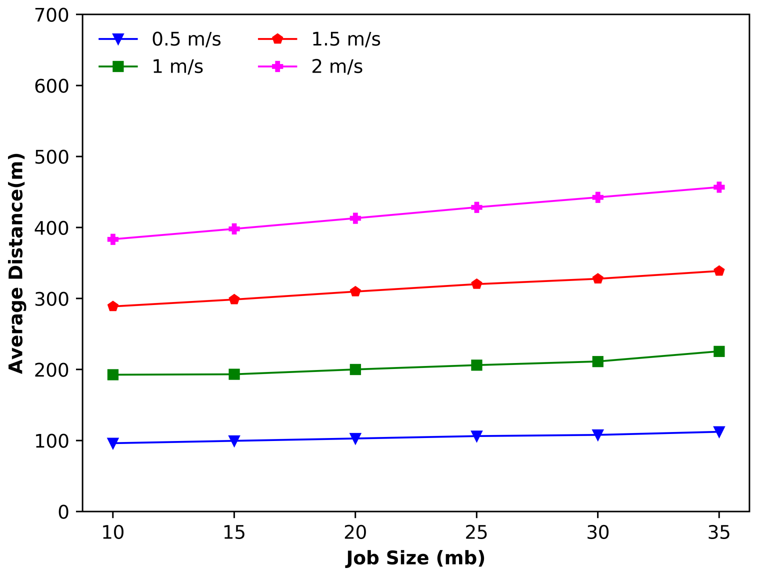


Figure 5.9: Average Distance (meter) for (DAG with 4 - Tasks)

5.12). When increasing delays are encountered, scheduling becomes impossible. In contrast to results (Figure 5.13), as the size of the job is larger, it is more beneficial to execute in the MEN for larger delays as it takes longer to execute the tasks locally. On the other hand, when the decreasing delay groups are used, some tasks could not be scheduled, while the tasks that were scheduled completed when the user was far from its starting point.

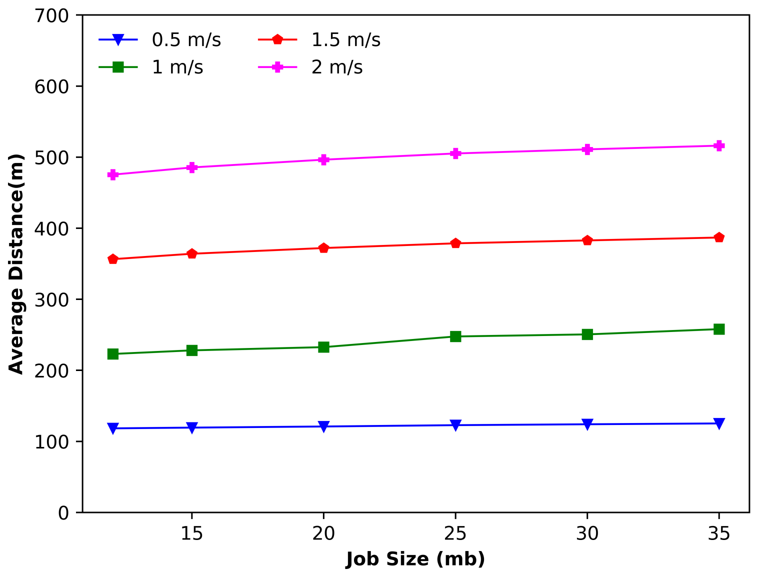


Figure 5.10: Average Distance (meter) for (DAG with 8 - Tasks)

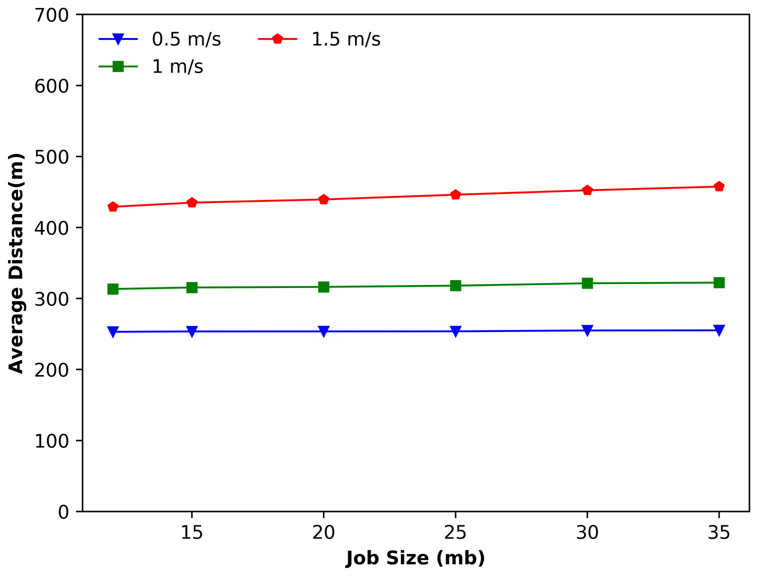


Figure 5.11: Average Distance (meter) for (DAG with 15 - Tasks)

5.6 Real Deployment with Face Recognition Application and Flask Server

This area of work was carried out with the deployment of one user and six edge nodes at the department of Computer Science in the University of Warwick as shown in Figure 5.18. For this we have used a Facial Recognition

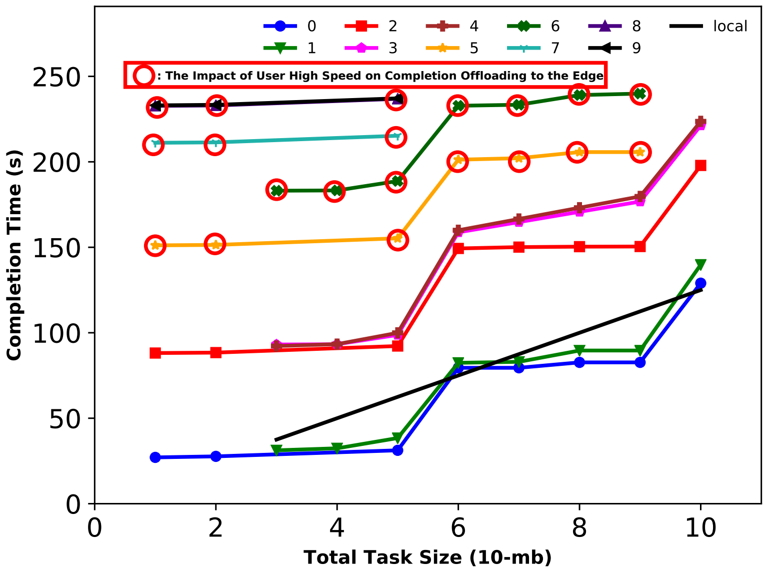


Figure 5.12: High Speed with Job Size (10-mb)

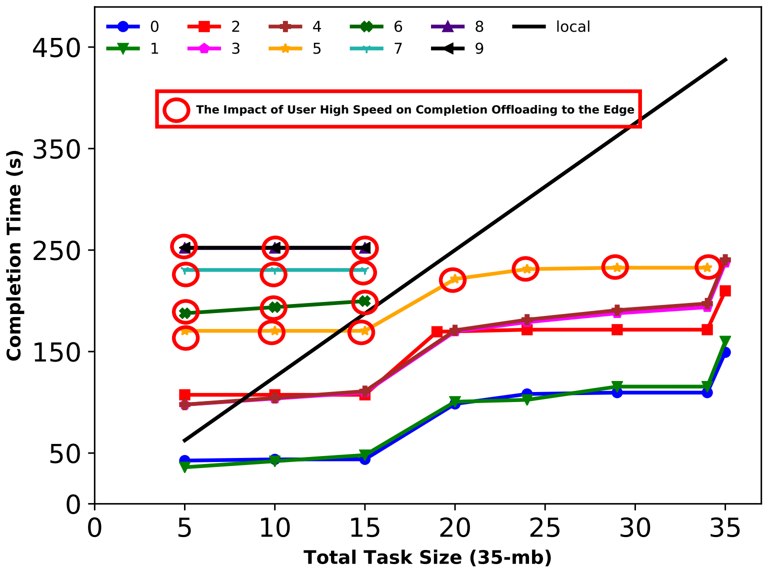


Figure 5.13: High Speed with Job Size (35-mb)

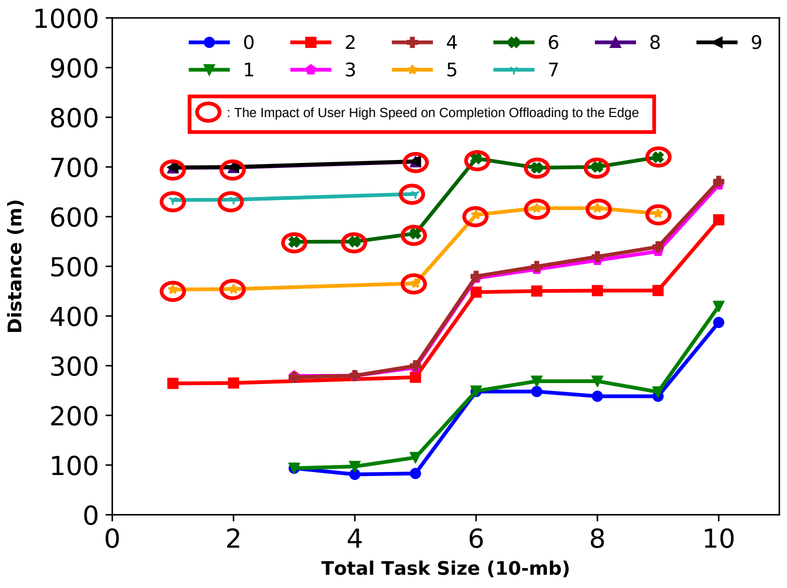


Figure 5.14: High Speed (distance) with Job Size (10-mb)

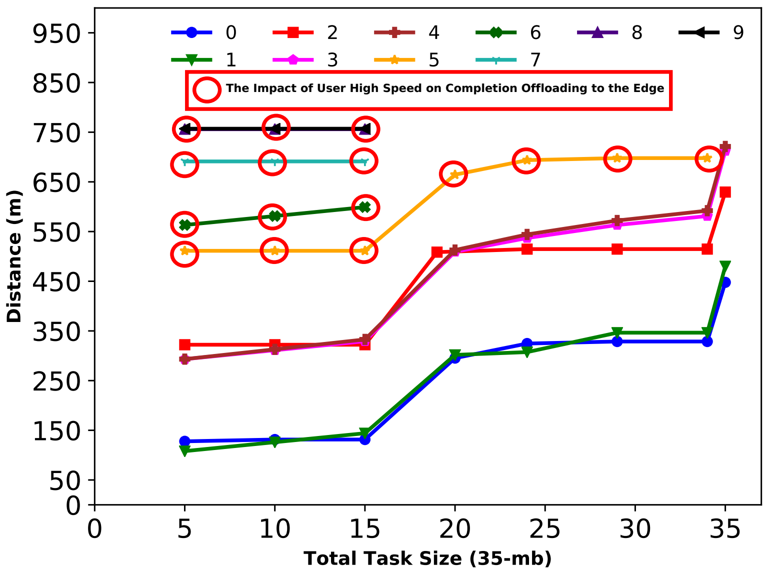


Figure 5.15: High Speed (distance) with Job Size (35-mb)

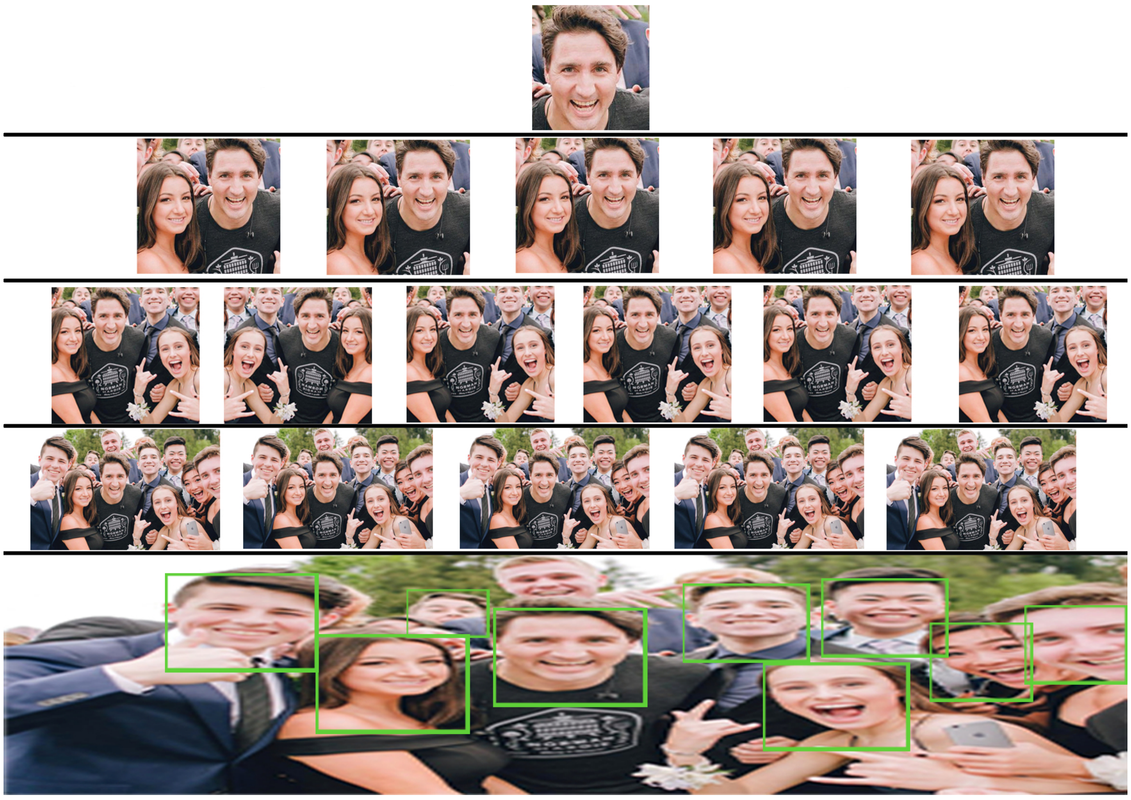


Figure 5.16: Representation of A Directed Acyclic Graph (DAG) with Datasets of Pictures

application with the dependency relating to pictures in six levels of the DAG. The deployment was in two environments: the mobile device (Android) acting as one user and personal computers (Linux with Flask Server) acting as cloudlets of edge nodes [39, 125]. A cloudlet here is a node which is connected to the Internet and which can be accessed remotely for various purposes like computation, website, server purposes, etc. It can be of any configuration (GPU, CPU, Memory, OS, etc.). We decided to use Facial Recognition as dependency tasks based on our core DAG framework. We started by uploading one image first, which typically contains one face in the image as shown in the figures (5.16 and 5.17), the face from the first image will be trained and store as a reference for all next levels of the DAG.

We created datasets with sizes of: 10, 15, 20, 25, 30, 35 (MB) and each dataset was considered as a dependency task. The dataset consisted of a collection of images and each dataset size amounted to the net sum of the size of all the images in the same dataset. The core logic lies in the processing of the dataset in the mobile device/ cloudlet. After receiving the dataset, it is processed by reading each image one by one. The faces from each image are extracted and compared to previously recognized faces (in the first instance it will only make a comparison with the face extracted from the first image as a reference).

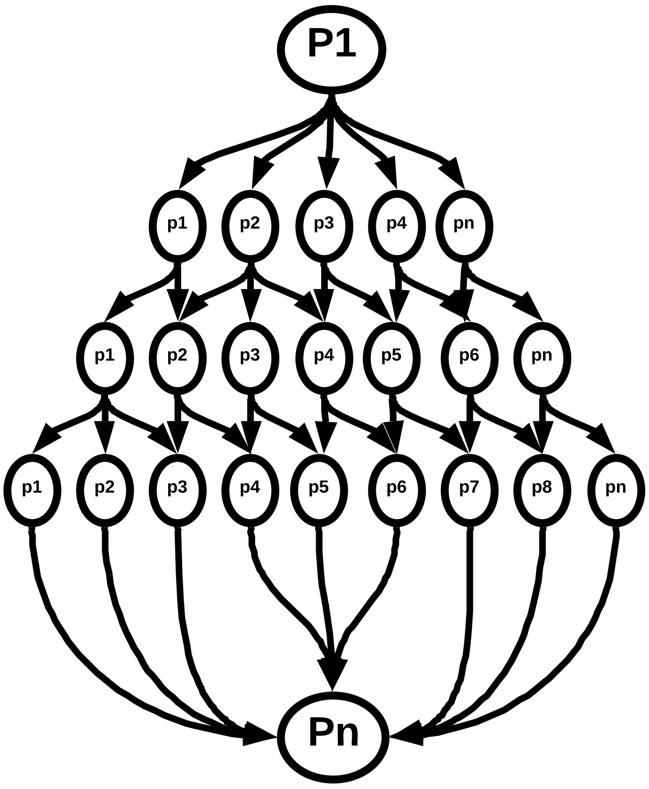


Figure 5.17: Representation of A Directed Acyclic Graph in Deployment

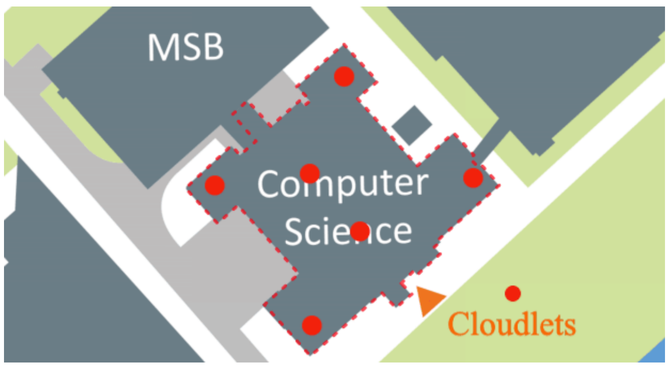


Figure 5.18: The Map of Cloudlets Deployment at University of Warwick

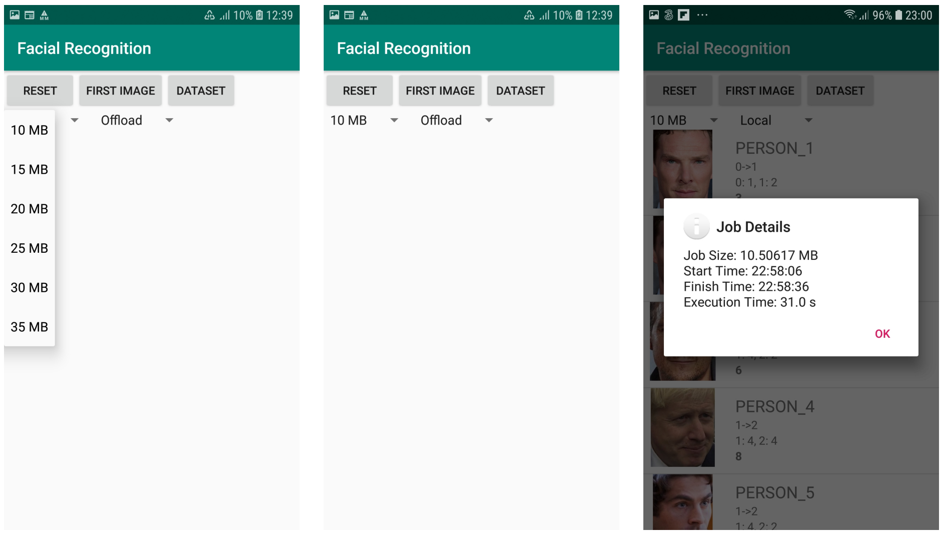
When a particular face matches (found/recognized) with previous faces, we will train the neighbor faces of the same image. If there are no matches found we will simply ignore the image and proceed to the next image in the dataset. We keep iterating this procedure till the end of dataset. Another case is when faces are repeated multiple times, we keep track of count of repetition of faces and we also assign a unique generated name for each of them. Then, the same procedure was repeated for the next dataset on the DAG. At the end, we display the results of all faces recognized along with their count and display the details of the task as shown in figure (5.19).

We have used a library called FaceNet for face recognition, which was developed by Google in the early 2015 and was later open sourced [80, 95]. It was very useful since it had the option of one shot learning. One shot learning was essential here since the entire network can be trained by only one facical image. In order to simplify the use of Facenet model, we generated the model manually with complicated procedures and exported it as (.h5) and (.pb). The (.h5) port was created for server side of the cloudlet, which uses Keras on top of Tensorflow. The (.pb) port was created for Android, which has support for Tensorflow only. In our work, we have used the Facenet model to take one input and produce one output. The inputs here are the pixels of face of image, which have been arranged in a particular order. The output produced is an encoding/embedding and we stored this encoding of the person in the memory. Later, if we want to predict someone's face we should compare their encoding with the encoding of all previously determined faces (encodings) to see if they match. For the arrangement of pixels, we consider the image to have 3 channels Red, Green, and Blue channels. So for each location (2d) on the image we get access to 3 channels to denote the pixels. We have fixed size for face (96 x 96) in order to make one shot learning accurate. If the face detected is larger or smaller than this resolution, I will be scaled to 96 x 96. So the total data would be around 96 x 96 x 3 = 27648. We store them in an array by default order from face detection using MTCNN. Later we "Transpose" them with Permutation of (2, 0, 1) and later divide all the elements of array by 255. Now this array is considered as an input for our Facenet model.

We use Android Studio software from Google as an environment for developing the face recognition application [100]. We start the creation by creating the UI layout. We use RelativeLayout as a base for creating layout as it offers more flexibility for alignment of inner elements compared to LinearLayout. We use the default Google material design components for UI and we add buttons for local, offload, uploading first image, and similarly for dataset. Later, we have added Recyclerview to display results to our own design (padding, divider) and content (count, face image, name). Now we start adding the Tensorflow-Android Gradle to the framework, which contains helper classes for implementing the facenet model into any android device. We have implemented our FaceNetHelper class to facilitate loading, unloading, preprocessing, processing, and computing purposes. The Facenet models are stored as Assets on Android. They are stored on Assets directory, which will be loaded by FacenetHelper. For communication with server side, we use a library called OkHttp with Picaso for working with Flask server on the cloudlet. It provides a layer to handle get request, post requests along with uploading of datasets to the cloudlet. In the server side with edge nodes, we have used a web framework, which is called Flask. Flask is written in Python and it helps to facilitate the interaction between the cloudlet and mobile device. Flask is basically a web application framework that could be used to serve web content (http) over the Internet. In the beginning we initialize all the modules like Keras Engine, Facenet (loading keras model) and other things before we initialize the flask web server. Apart from this we create two directories. One for storing the images to be detected "Todetect" and another is for storing the trained faces "Images". The face images are labeled as assigned person name (PERSON˙1, PERSON˙2, ... etc).

5.6.1 Testbed Evaluation and Results

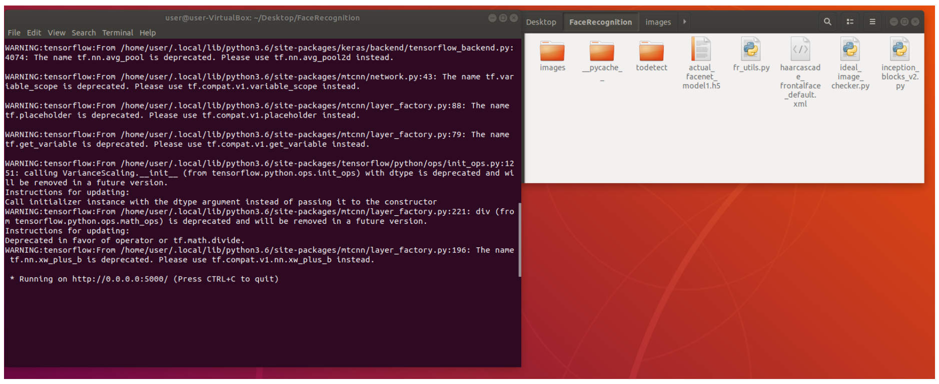
In our work, we have two main parts in the implementation: mobile device (as a user) and the edge nodes (as a cloudlets). We have used Android Studio in Java to build the face recognition application for the mobile device side. We have used Galaxy J7 Prime model with Exynos 7870 octa-core processor to run the experiment of the user side. In the edge side, we have used a Flask server, which is installed on each node. We have used six heterogeneous processors with six edge nodes of the edge as personal computers with [Linux OS]. We have deployed all nodes in the path of user mobility within 200 meters. We have installed Flask servers in all of cloudlets and we generated the APK file to the android mobile device. We test the offloading as a user walking within the range of communication all nodes in the edge. We run the experiment many times with respecting the network fluctuations. The sizes of the offloaded tasks were: 10,15,20,25,30, and 35 (MB) respectively. We created datasets with sizes of: 10, 15, 20, 25, 30, 35 (MB) and each dataset was considered as a dependency task. The dataset consisted of a collection of images and each dataset size amounted to the net sum of the size of all the images in the same dataset. We run the experiment many times to get the average of completion time during the user mobility in different network conditions. As we see in figure (5.21), we investigate the impact of user mobility on the completion time. The result is shown in the first graph as a good condition of the signal communication that means user was close to the edge during the mobility. The X-axis represents the



Job Size: 10.50617 MB
Start Time: 22:58:06
Finish Time: 22:58:36
Execution Time: 31.0 s

(a) First Screen (b) Second Screen (c) Third Screen

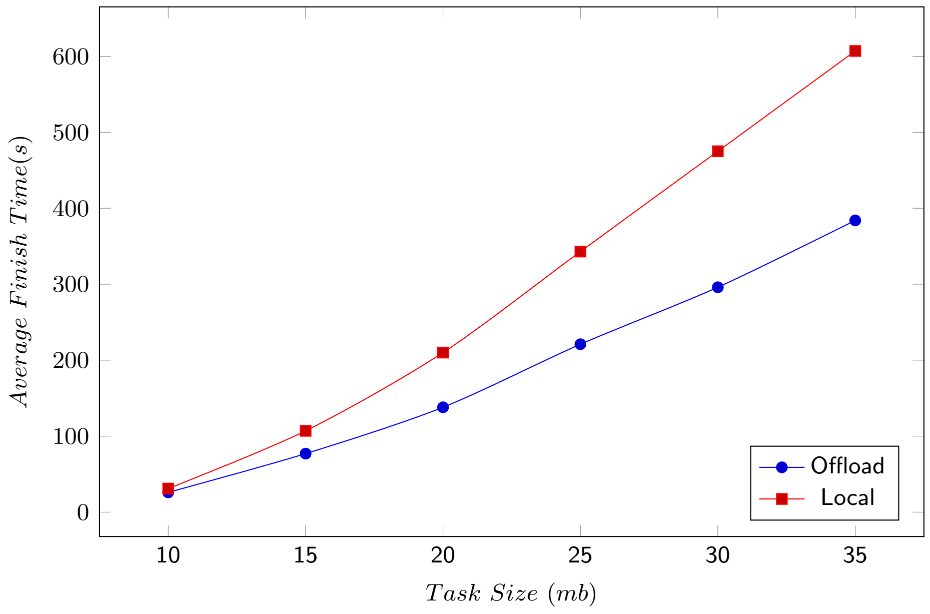
Figure 5.19: Face Recognition Application in Client Side



```
WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:
4074: The name tf.nn.avg_pool is deprecated. Please use tf.nn.avg_pool2d instead.

WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/mtcnn/network.py:43: The name tf.var
iable_scope is deprecated. Please use tf.compat.v1.variable_scope instead.

WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/mtcnn/layer_factory.py:88: The name
tf.placeholder is deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/mtcnn/layer_factory.py:79: The name
tf.get_variable is deprecated. Please use tf.compat.v1.get_variable instead.

WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/tensorflow/python/ops/init_ops.py:12
51: calling VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and wi
ll be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/mtcnn/layer_factory.py:221: div (fro
m tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
WARNING:tensorflow:From /home/user/.local/lib/python3.6/site-packages/mtcnn/layer_factory.py:196: The name
 tf.nn.xw_plus_b is deprecated. Please use tf.compat.v1.nn.xw_plus_b instead.

 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figure 5.20: The Flask Server in Cloudlet Side

size of the job completed (in MB), while the Y-axis represents the completion time. As can be observed, when the job is fairly small (10-20 MB), it is better to execute the task locally on the mobile device. This is due to the case that the communication time for offloading/downloading-required time that will not count locally. If the network stable during communications then it may be possible to schedule the smaller sized tasks in the edge. On the other hand, when the size of the task is bigger, it is better that the task is executed in the edge, as the tasks takes longer to execute locally in a mobile device. As we see from the results, the total completion time of the tasks in the cloudlet is less than the local completion time in mobile device handling larger task sizes. The result of the ns-3 simulation with the big size task matches the result of the real deployment.



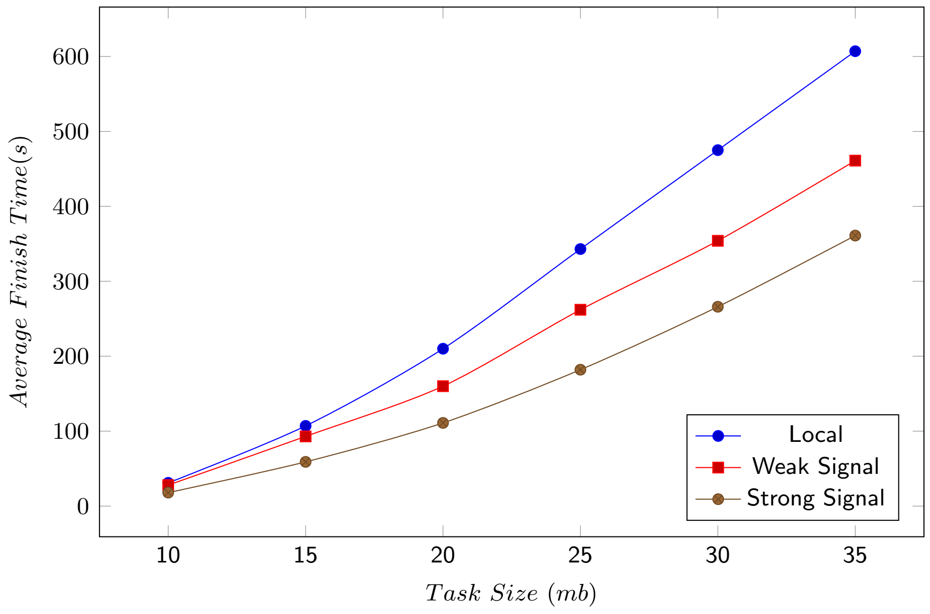


Figure 5.21: Testbed: Face Recognition Application and Flask Server with Tasks Dependency during Network Fluctuation

5.7 Summary

In this chapter, we focused on scheduling a mobile job that consists of dependent tasks on a set of cloudlets involving user mobility on the edge. Given that the problem is in general intractable, we provided a heuristic that returns a schedule that will potentially reduce the computation time in the cloudlet. We developed an ILP model with the optimal solution and implemented it in CPLEX solver with the same datasets using a ns-3 simulation. We conducted a range of simulation experiments using ns-3 with a range of job sizes, mobility speed and delay distributions. Finally, we run a real deployment of the heuristic with a Face Recognition Application and Flask Server. Our results show that, under certain conditions, it is impossible for the heuristic to return a proper schedule. We have studied the relationship between delay, speed and job size and shown that the completion time gain increases with the size of the job. Overall, we have presented the a heuristic algorithm that returns a schedule that will potentially reduce the computation time of dependent mobile tasks in edge computing networks.

Chapter 6

A Fully Distributed Computational Offloading Algorithm for Mobile Edge Computing

Mobile edge computing (MEC) is an emergent technology that helps bridge the gap between resource-constrained IoT devices (IoTD) and the ever-increasing computational demands of the mobile applications they host. Edge network enables the IoTDs to offload computationally expensive tasks to the nearby edge nodes for better quality of service such as lower latency. Most of the proposed offloading techniques focus on *centralised* approaches with a small number of mostly *static* IoTDs hosting independent tasks. In this chapter, we address three major problems: (i) that algorithms assume some central nodes where information is recorded, (ii) nodes are not mobile and (iii) tasks are independent. To the best of our knowledge, we develop the *first* fully distributed offloading strategy for mobile devices hosting *dependent tasks*, where a dependent task cannot start until its parent has completed, with the aim of reducing completion latency. The main contributions in our work are summarized as follows: (i) we provide a formalisation of the dependent tasks offloading problem as an optimisation problem, (ii) we develop a fully distributed algorithm for offloading the dependent tasks in the edge network, (iii) we conduct extensive experiments using the ns-3 simulation engine to evaluate the effectiveness of our distributed algorithm in terms of minimising the task completion time in the edge, and (iv) we also study the bottlenecks caused by the algorithms, in this case, queue waiting time and we study the impact of mobility on the bottleneck. Experiment results show that our offloading algorithm provides significant improvements in performance compared to the base case of offloading to a central cloudlet. We provide in-depth profiling of the network and observe the impact of mobility patterns on completion times.

6.1 Models

Recent works in edge networks mostly focused on solving the offloading problem with users being static during task offloading and the communication links between mobile devices and edge nodes are always available, e.g., [14, 52, 121, 123]. However, such availability assumption is not valid, with mobility users being a significant challenge in MEC networks that need to be addressed. Also, to ease the development of offloading strategies, it is typically assumed that some information about the edge environment is available in a central repository, e.g., [111]. Similarly, such assumption is limiting as such a server becomes a bottleneck in the system. These challenges are exacerbated when a job contains *dependent* tasks, where a task cannot start executing without its parent task having completed.

The system consists of multiple users, each having a mobile internet of thing (IoT) device[1], moving from a starting point to a destination point and a network, called an edge network, that contains a number of powerful computing nodes known as cloudlets as shown in Figure 6.1. We detailed each component of the system in Chapter:3 and we have some assumptions that we added in Edge Network Model as we will explain in this section.

6.1.1 Cloudlet and Edge Network Model

A *cloudlet* is a computer that is resource rich, i.e., it has a powerful CPU, sufficient memory and other resources to run resource-hungry applications [111]. We assume a cloudlet to have a large enough buffer which queues execution requests. We assume a cloudlet will execute tasks in the buffer in a *FIFO* fashion [65]. After execution, a cloudlet will take one of three steps: (i) it will pass the (final) result directly to the user if the user is in range, (ii) pass the (intermediate) result to another cloudlet which has, in its buffer, a task that is dependent on the result or (iii) if the user is out of range, pass the (final) result to another cloudlet which will forward it onto the user. We assume a cloudlet to have a number of communication interfaces, e.g., WiFi, ethernet. We assume each wireless network interface to be associated with its own range. A IoT device can communicate with a cloudlet if both fall within the range of each other. We assume an edge network to consist of a set of cloudlets $C = \{C_1 \dots C_m\}$. Cloudlets has an interconnection network among them and we model the edge network as a graph $E = (C, L)$, where $L \subseteq C \times C$ is a set of (symmetric) links between a pair of cloudlets. We assume the network to be heterogeneous, i.e., cloudlets have the same set of computational resources (e.g., memory, CPU) but vary in capabilities or amount [111]. We also assume

[1]We will use the terms user and device interchangeably.

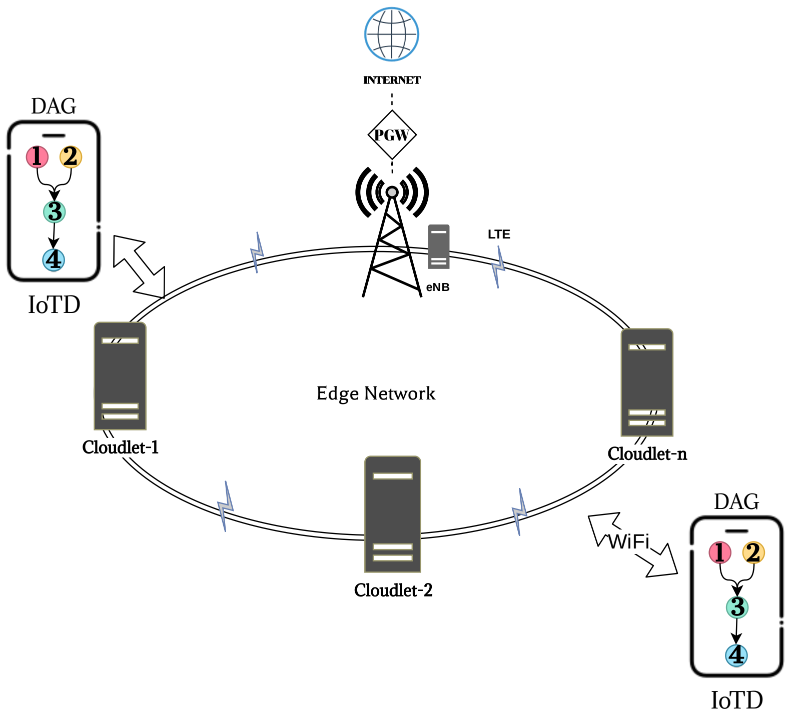


Figure 6.1: System Model

the existence of a base station to which all cloudlets are connected and is responsible for the inter-cloudlet communication, making the edge network fully connected.

6.2 Problem Formalisation

Our main objective is to develop a *fully* distributed algorithm for offloading a job of dependent tasks onto an edge network so as to minimise the completion time of the job.

6.2.1 Problem Definition

Given a set U of users, with each user U_i having a job J_i to execute, and an edge network E, our distributed offloading algorithm allocate the dependent tasks onto cloudlet nodes so as to minimise the job completion time. Some of the challenges are: (i) a mobile user can only communicate with cloudlets that are in range, and (ii) a user can offload a task only after all of its parent tasks have been offloaded.

Table 6.1: Key Notations in Chapter 6

Symbols	Definitions
$OT(T_i^j, C_k)$	Offloading time of a task T_i^j on a selected cloudlet C_k
U	The set of all users with offloading tasks
$ST(T_i^j, t)$	Start time of a task T_i^j of user U_i in a given time τ
C	The set of all Cloudlets in the edge
$ET(t, C_k)$	The execution time of a task t on a selected cloudlet
T_i^j	j^{th} task of user i
$WT(T_i^j, C_k)$	Waiting time of T_i^j in the buffer of selected cloudlet
S	Offloading scheduling function
τ	A given time.
$FT(J_i/(t), S)$	Finish time (FT) of J_i (/task t) under schedule S
Q^τ	The state of the queue at a given period of time τ
$TT(r_i^j, s, d, T)$	Time to transfer result of T_i^j from cloudlet s to d for task T.

6.2.2 General

We define an offloading scheduling function as $S : T \times \tau \rightarrow C \cup \{\perp\}$ and define the finish time (FT) of a job J_i under schedule S, as follows:

$$FT(J_i, S) = max_{t \in T_i}(FT(t, S)), \forall i, 1 \leq i \leq n \tag{6.1}$$

The scheduler essentially captures the times during which a task is resident on a cloudlet. We then define the offloading time of a task T_i^j on cloudlet C_k, denoted by $OT(T_i^j, C_k)$, as the minimum time τ when $S(T_i^j, t) = C_k$, i.e.,

$$OT(T_i^j, C_k) = min\{t | S(T_i^j, t) = C_k\} \tag{6.2}$$

We say that a task T_i^j has not been offloaded if $\forall t \geq 0, ST(T_i^j, t) = \perp$. Our main objective is to minimise the finish time of job J_i:

$$argmin_S \ FT(J_i, S), \forall i, 1 \leq i \leq n \tag{6.3}$$

We define a binary variable as follows, to capture the offloading schedule of a task on a cloudlet:

$$T_i^{j,k} = \begin{cases} 1, & \text{if } \exists t > 0, S(T_i^j, t) = C_k \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

6.2.3 Constraints

We initially consider the high level constraints of such offloading schedules and subsequently consider the actual offloading timing process.

6.2.3.1 High Level Constraints

The offloading process has a number of properties, namely (i) fairness, (ii) completeness, (iii) no duplication and (iv) validity.

- *Fairness*: The scheduler considers some notion of fairness during task offloading, preventing a user from hogging a given cloudlet. Specifically, a user cannot offload more than one of its tasks to the same cloudlet:

$$\begin{gathered} \forall U_i \in U,\ \forall C_k \in C,\ \forall l, l', 1 \leq l, l' \leq |J_i|, l \neq l' \cdot \\ T_i^{l,k} = 1 \Rightarrow T_i^{l',k} = 0 \end{gathered} \tag{6.5}$$

- *Completeness*: The scheduler will also allocate all tasks of a job to a cloudlet, for example, to reduce energy consumption on local devices (though energy usage minimisation is not an objective).

$$\forall U_i \in U, \forall T_i^l \in T_i, \exists C_k \in C \cdot T_i^{l,k} = 1 \tag{6.6}$$

- *No duplication*: The scheduler will also ensure that a task in only allocated once to a cloudlet.

$$\begin{gathered} S(T_i^j, t') \neq \perp \wedge S(T_i^j, t'') \neq \perp \wedge 0 < t' \leq t'' \Rightarrow \\ S(T_i^j, t') = S(T_i^j, t'') \end{gathered} \tag{6.7}$$

- *Validity*: The offload scheduler also considers the communication range during offloading of tasks to the edge network. We assume a function $\mathcal{R}_i$ that captures the nodes that are in range of a user U_i at a given time, as follows:

$$\mathcal{R}_i : \tau \rightarrow 2^C \tag{6.8}$$

Then, a user will only offload to a cloudlet if that cloudlet is in range of the user at the given time.

$$OT(T_i^j, C_k) = t \Rightarrow C_k \in \mathcal{R}_i(t) \tag{6.9}$$

6.2.3.2 Timing Level Constraints

Now that we have provided the high level properties of the offloading scheduler, we now consider the finer (i.e., timing) details of the offloading process.

- *Start time*: The start time of a (dependent) task on a given cloudlet can only occur after (all) its parent(s) tasks have completed.

$$\begin{gathered}\forall U_i \in U, \forall (T_i^k, T_i^l) \in D_i, \exists C_m, C_n \in C, T_i^{k,m} = 1 \\ \wedge T_i^{l,n} = 1 \Rightarrow ST(T_i^l, C_n) \geq FT(T_i^k, C_m)\end{gathered} \tag{6.10}$$

- As a cloudlet will have waiting tasks in its buffer, the start time (ST) of a task on a cloudlet is equal to the sum its offload time onto that cloudlet and its waiting time on that cloudlet. WT is the queue waiting time.

$$T_i^{j,k} \Rightarrow ST(T_i^j, C_k) = OT(T_i^j, C_k) + WT(T_i^j, C_k) \tag{6.11}$$

- *Waiting time*: The waiting time (WT) of task t on C_k is the time during which t is in C_k's buffer, equal to the sum of execution times of other tasks before t in the queue.

$$\begin{gathered}W(T_i^j, C_k) = \\ \sum_{t \in \{ts | S(ts,t')=C_k, 0 \leq t' < OT(T_i^j, C_k)\}} ET(t, C_k)\end{gathered} \tag{6.12}$$

- *Execution time*: The execution time (ET) of task T on cloudlet C_k is the time it takes for the job to complete execution on C_k. Denoting the CPU frequency of cloudlet C_k by $C_k.freq$, the execution time of the number of instructions of the task is thus given by

$$ET(T, C_k) = \frac{(T.num\_of\_inst.)}{(C_k.freq)} \tag{6.13}$$

This definition has been used in previous works, e.g., [111]. However, the actual execution time may be affected due to issues such as caching among others.

- *Finish time*: The finish time for the task T_i^j will depend on the type of dependency. There are two scenarios to consider, to model the finish time.

  The first scenario is the finish time of a (parent) task T_i^j:

$$\begin{gathered} T_i^{j,k} \wedge T_i^{l,m} \wedge (T_i^j, T_i^l) \in D_i \Rightarrow \\ FT(T_i^j, C_k) = ST(T_i^j, C_k) + \ ET(T_i^j, C_k) + \\ TT(T_i^j.result, C_k, C_m, T_i^l) \end{gathered} \tag{6.14}$$

  The second scenario is the finish time of a leaf task, where the final action is user U_i downloading the result:

$$\begin{gathered} T_i^{j,k} \wedge \forall t \in T_i, (T_i^j, t) \notin D_i \Rightarrow \\ FT(T_i^j, C_k) = ST(T_i^j, C_k) + \ ET(T_i^j, C_k) + \\ TT(T_i^j.res, C_k, C_m, U_i) + \ \frac{(T_i^j.res.size)}{(C_m.\ downrate)} \end{gathered} \tag{6.15}$$

- *InterCloud Result Transfer time*: The intercloud transfer time (TT) is the time it takes to transfer the result from one cloudlet to another.

$$TT(T.res, c_{src}, c_{dest}, t_{dest}) = \frac{(T.res.size)}{(Tx.rate(c_{src}, c_{dest}))} \tag{16}$$

6.2.4 Problem Complexity

The problem of scheduling of tasks with precedence constraints to satisfy a deadline is known to be NP-hard [67]. Such scheduling is a special case of the problem we pose here, making our problem intractable in the general case. In this chapter, we provide a distributed algorithm to minimise the finish time of jobs for every user.

6.3 A Fully Distributed Offloading Algorithm

In this section, we present a *fully* distributed greedy scheduling algorithm for task offloading that attempts to minimise the completion time of offloaded jobs. The user process (see Algorithm 1) and the cloudlet process (see Algorithm 2) will each execute their respective protocol to manage the interactions between them.

Each process is made up of a set of variables and a set of actions. Figure 6.2 shows the interleaving of actions from a client perspective with a cloudlet. Figure 6.2 shows a case where the user remains in the range of a cloudlet between the time it gets in its range and the time the cloudlet has completed execution of the task. As shown in algorithms 1 and 2, After initialization processes in both sides, the advertise presents in a time unit for a communication purpose between IoTD and the cloudlet. The client will send the IoTD-id as (i) and the cloudlet will give the cloudlet-id (J), cloudlet capabilities (J.CPU), and the cloudlet queue length (J.QL) as show in both sides. First, we will explain the details of the client process and then the cloudlet process.

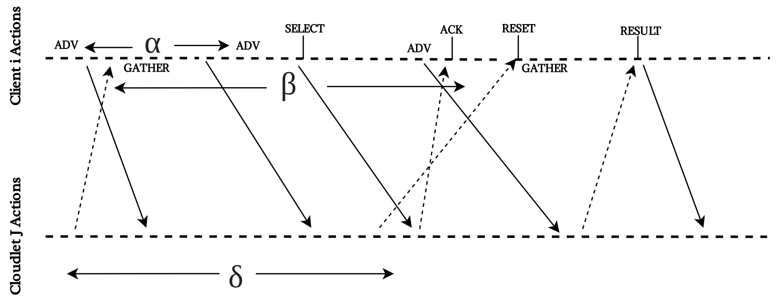


Figure 6.2: Client Interactions With Cloudlets

Algorithm 1 : *User Process i*

```
**Variables**:
$TaskL\ : DAG\ of\ tasks:\ t_1 \cdots t_{n_i}$   $\leftarrow$ Job Graph
$ExecTime[]$: Array of execution times.
$CurrentT$: Task
$CList$: List of cloudlets info $< id, cpu, ql >$
$Temp[]$: Array of unack'ed cloudlet assignment
$ADV, RESET$: Timers
$WaitFor[]$: Array of set of cloudlet ids init $\emptyset$
$allocT[]$: array of cloudlet id init $\perp$
$result$: RES $\leftarrow$ Assume result is of type RES
$THR$: Threshold $\leftarrow$ Maximum time of $ExecTime[]$ in selected cloudlets

*Action* **Adv**
**upon timeout** $< ADV,\ \alpha >\ expires$ **do**   $\leftarrow$ Advertise Presence
    **BCAST** $< HEARTBEAT,\ i >$;

*Action* **Reset**
**upon timeout** $< RESET,\ \beta >\ expires$ **do**   $\leftarrow$ Reset Cloudlet List
    $CList\ := \emptyset$ ;

*Action* **Gather**
**upon rcv** $< HEARTBEAT, (CLid, CLcpu, CLq) >$ **do** $\leftarrow$ EDGE Information (id, cpu, queue)
    $CList := CList \cup \{(CLid, CLcpu, CLq)\}$ ;

*Action* **Select**
**upon** $< TaskL \neq \emptyset >$ **do** {   $\leftarrow$ Choosing Cloudlet
**if** $CList \neq \emptyset$ **then**   {
    $currentT\ :=\ head\ (\ TaskL\ )$;
    $\forall\ C\ \in\ CList$   **do** {
        $execTime[C] := (\frac{size(currentT)}{C.cpu}) + C.ql\}$;
    **if** $execTime[C]\ <=\ THR$ **then**   { % do not delay offloading
        $temp[currentT]\ := argmin_C$ execTime[]   $\leftarrow$ cloudlet with lowest exec time
        $\forall\ p\ ,\ currentT\ \in\ p.children$ **do**{
            $waitFor[currentT]\ := waitFor[currentT] \cup\ \{allocT[p]\}$;
        **send** $< i, (currentT,\ waitFor[currentT]), temp[currentT] >$
        **set** $< timer\ , \beta >\}\}\}$;
    **fi;**
**fi;**

*Action* **Ack**
**upon rcv** $< ACK(\ k,\ i,\ t)\ \ i >$ **do**   $\leftarrow$ Confirmation Process
    $allocT[t] :=\ k$ ;
    $TaskL\ :=\ tail(TaskL)$;
    $execTime[]\ :=\ \infty$ ;

*Action* **Result**
**upon rcv** $< RESULT,\ (i, r) >$ **do** $\leftarrow$ Receiving Result from cloudlet
    $result\ :=\ r$;


```

Algorithm 2 : *Cloudlet Process* J

```
**Variables**
$src[]$: array of client id.
$allC := init\ \emptyset,\ src[\ ] := \perp$ ,
$QL :=\ Queue\ of\ tasks\ in\ Cloudlets,$   ← Initialization Process
$Dest$: set of (cloudlet id, client id, task id)
*Action* **C-Adv**
**upon** $< timeout,\ \partial >$ **do**   ← Advertise Presence
    **BCAST** $< HEARTBEAT,\ $(J, J.cpu, J.ql)$ >$;
*Action* **C-Gather**
**upon rcv** $< HEARTBEAT,\ k >$ **do**   ← Collect Clients List
    $allC\ := allC\ \cup \{k\}$ ;
*Action* **C-Reset**
**upon** $< timeout,\ \partial >$ **do**   ← Refresh Clients List
    $allC\ :=\ \emptyset$;
*Action* **C-Req**
**upon rcv** $< c,\ (t,\ pc),\ J >$ **do**{← Receive Client Request
    $QL\ :=\ QL\ \hat{}\ \langle (c,t) \rangle$ ; ← Add Tasks (t) to the Queue (FIFO)
    $src[t] := c$;
    $\forall\ d\ \in\ pc$ **do**{
        **send** $< DEPEND,\ < J,\ c,\ t >,\ d >$;← From (DAG)}
    **send** $< ACK,\ < J,\ c,\ t >,\ c >$ };
*Action* **C-Proc**
**upon** $< task\ (\tau)\ finish\ processing >$ **do**{  ←After Execution
**if** $Dest\ \neq\ \emptyset$ **then**
    $\forall c \in Dest$ **do**
        **send** <J , $(c.i, c.t, \tau.result), c >$;
    $proc := head(QL)$;
    $QL := tail(QL)$;
**else**
%send result to client; look for cloudlet which has client in its range
    **send** $< SEARCH,\ (src[\tau],\ \tau.id,\ \tau.result),\ \perp >$};
**fi;**

*Action* **C-Check**
**upon rcv** $< L$, cid, ct, res) , $J >$ **do**
$Res := Res \cup \{(cid, ct, res)\}$;
**if** $head(QL) \in Res$ **then**  ← Checking if result is available for task
    $Exec(head(QL))$;   ← Executing on processor
    $QL := tail(QL)$;
**fi;**

*Action* **C-Depend**
**upon rcv** $< DEPEND,$ ( L, $i$, t) , J> **do**
    $Dest := Dest \cup$ {(L,i,t)};

*Action* **C-Return**
**upon rcv** $< SEARCH,$ ( s, t, r) , $\perp$> **do**
**if** $s \in$ allC **then**
    **send** $< RESULT,\ (i,\ r) >$   ← Downloading Process
**fi;**
```

Algorithm 1 (for a user) has six actions after defining the variables (Line 1-12), explained below:

- Action **Adv**: This action sends $\langle HEARTBEAT \rangle$ messages to advertise the presence of a client (or device) i to cloudlets within its range (Line 13-15).

- Action **Reset** causes a client to reset the list of cloudlets within its range. As the device moves, the set of cloudlets within the range of a client will change and the client will periodically reset its list to only keep current cloudlets (Line 17-19).

- Action **Gather** will enable a client to listen to heartbeats from cloudlets, to determine the cloudlets that it can communicate with. The information the client receives about a cloudlet is the following: (i) cloudlet id, (ii) cloudlet CPU capability and (iii) cloudlet (buffer) waiting time (Line 21-23).

- Action **Select** selects the cloudlet that returns the shortest completion time for a given task, among all cloudlets that are within range at that given time. This action is the one where the client offloads the task to the selected cloudlet. It also sends information about any task dependency to the selected cloudlet. A parameter Threshold THR is used to enable a user to decide whether it wants to postpone offloading. As such, the threshold parameter will enable edge nodes to share in task computation, thereby achieving load balancing (Line 25-38).

- Action **Ack** is an action that enables a client to receive an acknowledgement from a cloudlet that it has accepted an offload request, by queuing the task in its buffer. Once an Ack has been received for a task, then the client proceeds to schedule the next task in the list (Line 40-44).

- Action **Result** is an action that enables a client to receive the final result from the selected cloudlet (Line 46-50).

Similarly, Algorithm 2 shows the eight actions of a cloudlet after defining the variables (Line 1-5):

- Action **C-Adv**: This action causes the cloudlet to send a $\langle HEARTBEAT \rangle$ message to advertise its presence and capabilities to clients (or devices) that are within its communication range (Line 6-8).

- Action **C-Gather**: This action, similar to the client processes, allows cloudlets to collect the identities of clients that are in their neighbourhood (Line 9-11).

- Action **C-Reset**: This action, similar to the client processes, allow cloudlets to reset their list of clients in their vicinity. This needs to be done due to client mobility (Line 12-14).
- Action **C-Req**: This action queues the request of a client for task offloading and sends an Ack back when the task has been enqueued. It also tracks task dependencies (Line 15-21).
- Action **C-Proc**: This action executes when an task has finished executing, sending its results to dependents (Line 22-32).
- Action **C-Check**: This action checks whether results are available before a dependent task is executed. Else, the task is blocked (Line 34-40).
- Action **C-Depend**: This action allows a processor to send the result of a completed task to nodes that have tasks dependent on it (Line 42-44).
- Action **C-Return**: This action causes a cloudlet to push the result to a client (Line 46-52).

Table 6.2: Simulation Parameters in Chapter 6

Parameter	Value / Range
Map Area	0.5 x 0.5 km
Number of IoTD	10 to 50 Devices
Number of Edge Cloudlets	30 Cloudlets
Cloudlet uplink/downlink	40 Mbps / 40 Mbps
Cloudlet CPU frequency	2 GHz & CISC
Cloudlet coverage radius	40-50 meters
Central Cloudlet CPU frequency	2 GHz & RISC
CPU Cycles Required by One Task	2×10^9 ~ 2×10^{10}
Acceleration ratio between CISC & RISC	10~50
Input Data Size	1.0~4.0 MB
Mobility Model	WayPoint Mobility Model
IoTD Moving Speeds	0.5, 1.0, 1.5, 2 [m/s]

6.4 ns-3 Experimental Setup:

This section will first detail the setup for the simulation experiments conducted in ns-3 to evaluate the performance of the proposed distributed algorithm. We will subsequently present a sample of the results of the evaluation. The parameters of the experiments are shown in Table 6.2. To evaluate the performance of our proposed algorithm, we use the ns-3 network simulator [16]. We use parameters that have been used in related works and will be specified whenever required.

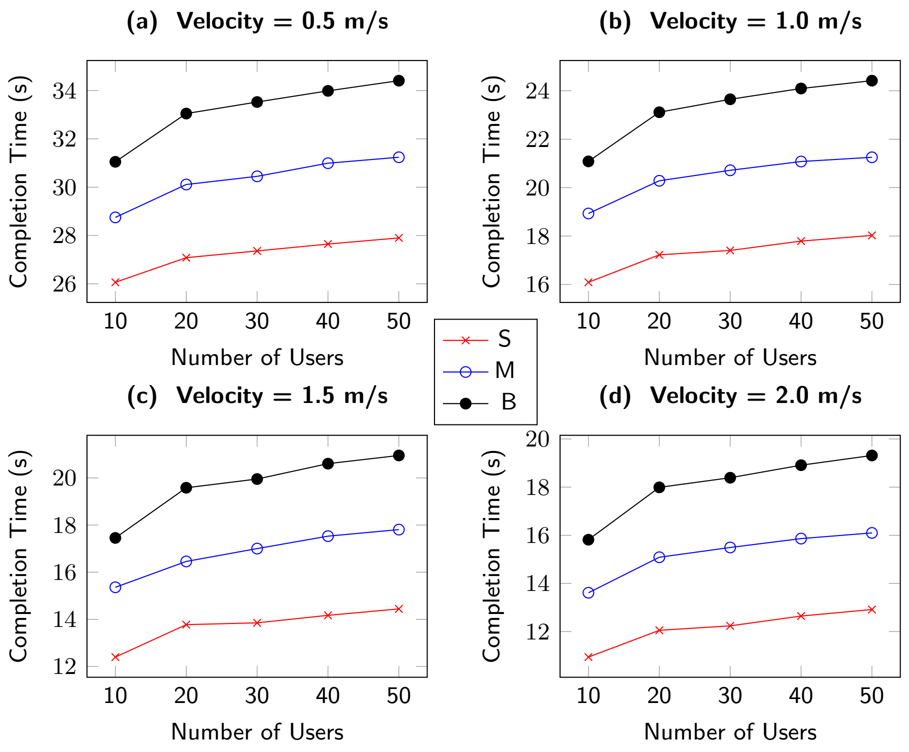


Figure 6.3: The Variation of Average Completion Times (a, b, c, d) with Small Threshold = $12s$

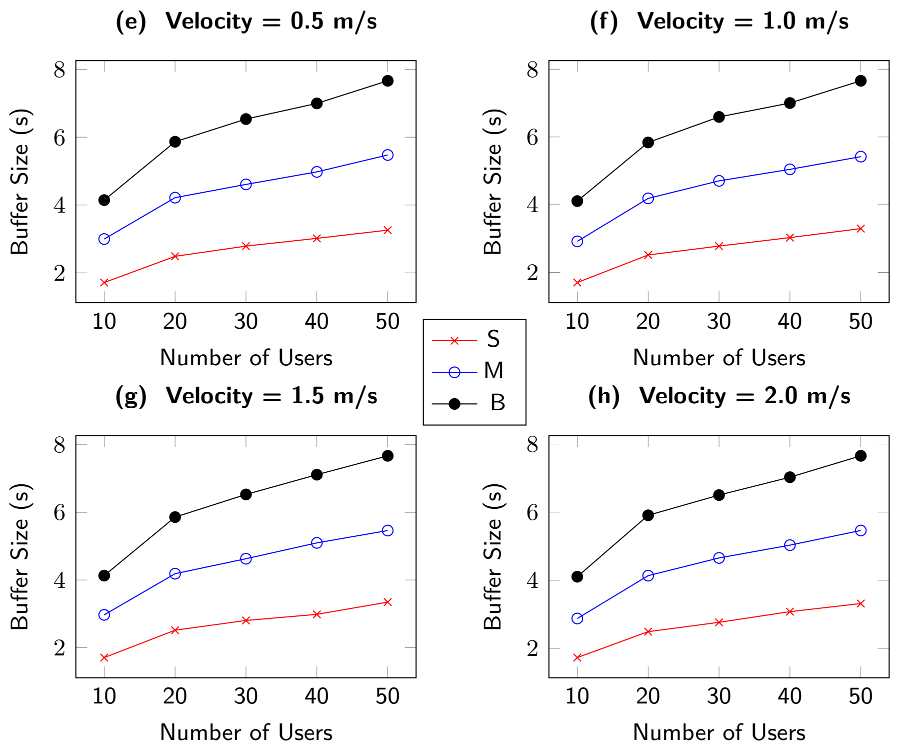


Figure 6.4: The Variation of Average of Buffer Size (e, f, g, h) with Small Threshold = $12s$

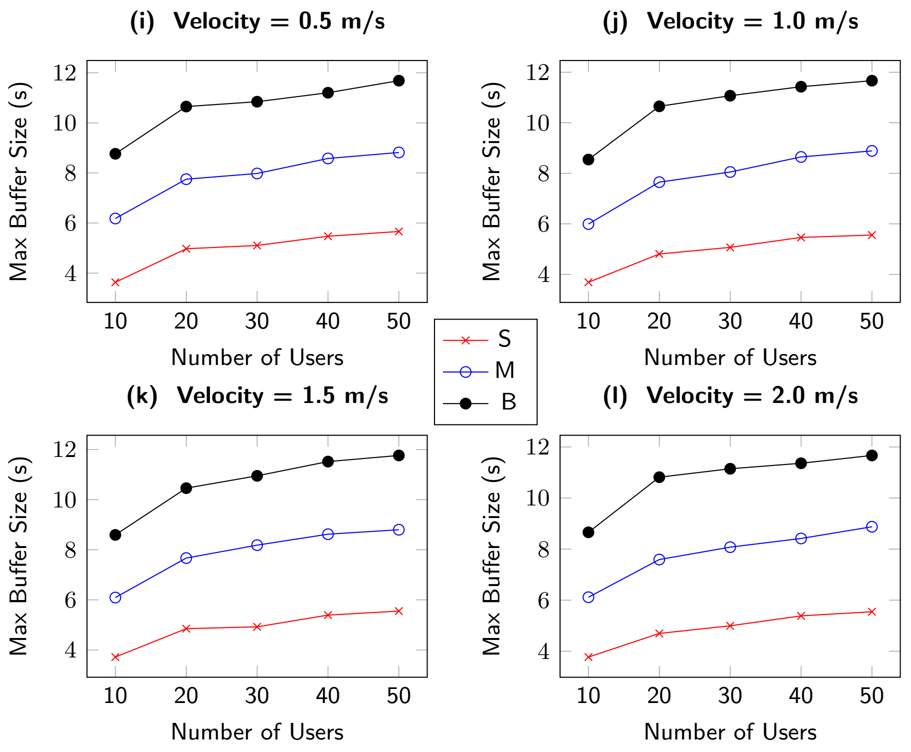


Figure 6.5: The Variation of Average of Maximum Buffer Size (i, j, k, l) with Small Threshold = $12s$

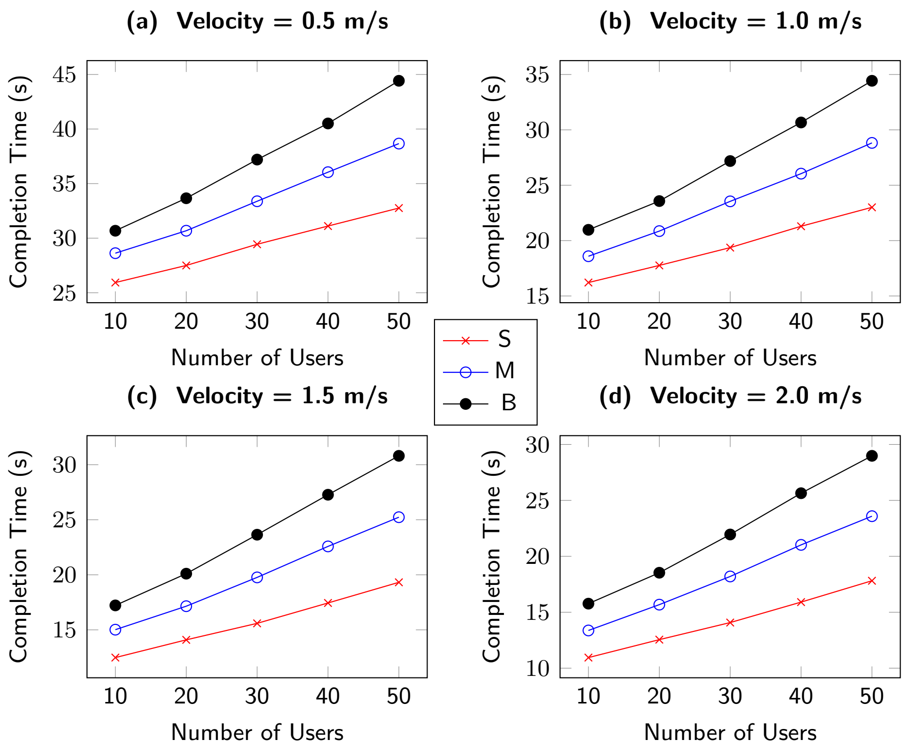


Figure 6.6: The Variation of Average Completion Times (a, b, c, d) with Large Threshold = $100s$

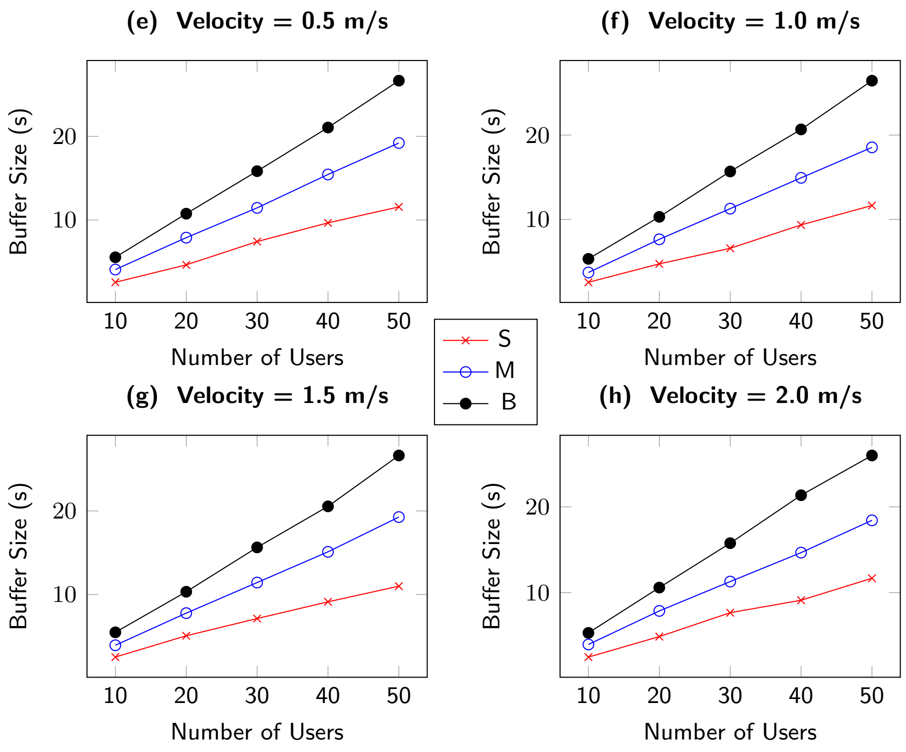


Figure 6.7: The Variation of Average of Buffer Size (e, f, g, h) with Large Threshold = $100s$

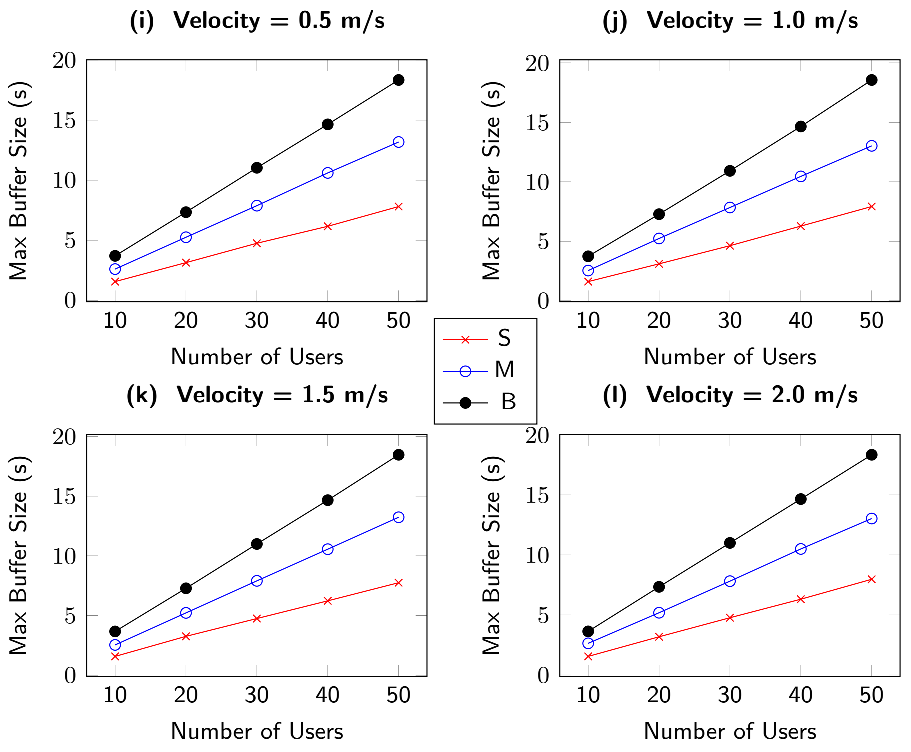


Figure 6.8: The Variation of Average of Maximum Buffer Size (i, j, k, l) with Large Threshold = $100s$

6.4.1 General:

We have used the WiFi-Phy 802.11-n standard configuration in ns-3 as we model the cloudlet network as a mobile edge network [46]. The WiFi frequency is set at 2.4 GHz. The protocol used for the communication between an IoTD and a cloudlet is UDP socket with Type-Id. The cloudlet transmission range is set to 40-50 meters. Communication between cloudlets happens over an LTE network [110]. The other parameters and values used in the simulation setup are summarized in Table 6.2.

6.4.2 Cloudlet and Edge Network Setup:

We consider an edge network with 30 heterogeneous cloudlets that are uniformly distributed over the setup area. All cloudlets have fixed positions in all the experiments. We divided the map into a left side and a right side. The model assumed is as depicted in Figure 6.1, where all cloudlets are directly connected to a base station. All cloudlets can communicate with each other via the base station via LTE. The reason for this is to study the impact of traffic communication on the cloudlet's buffer size. We run two types of experiments: (i) asymmetric mobility and (ii) symmetric mobility. In the asymmetric one, the IoTDs are split into two groups: 70% of users move from left to the right side and 30% move from right to left direction to induce higher traffic. The second experiment is symmetric mobility with 50% on both side of the map. For example, in an asymmetric experiment with 50 users, 35 users will be walking left to the right and the other 15 users will move the other way. We set up the starting point and the endpoint for all users to be in a straight direction during the mobility. The cloudlet has a task queue which works on first come first serve basis (FCFS) [65]. A newly uploaded task is added to the back of the queue.

6.4.3 IoTD Device Setup:

We install WiFi module on each IoTD (client) device and we create $10 \ldots 50$ clients moving at a constant speed in a given run [46]. Based on previous research, e.g., [111], we set the input data size of each task to be in the region of 1 MB to 4 MB. The CPU cycles requirement ranges from 2×10^9 cycles per task (cpt) to $\sim 2 \times 10^{10}$ cpt. Besides the difference in CPU frequencies between the cloudlets and IoTDs, we additionally set the impact of acceleration rate on CISC and RISC processors as the same CPU settings in [111]. We consider users to travel at 4 different speeds, namely $0.5, 1.0, 1.5$ and 2.0 m/s during the experiments.

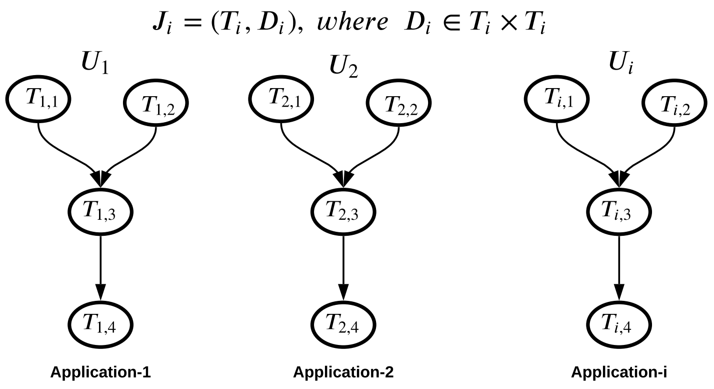


Figure 6.9: Job DAG for Gaussian Elimination [109] assumed in this work

6.4.4 Task graph:

As stated before, we model the set of tasks as a DAG due to dependencies as in Figure 6.9, e.g., [109]. We assume the job to be the Gaussian Elimination algorithm, whose DAG consists of 4 tasks [109]. We use Breadth-First Search algorithm (BFS) to order tasks in the DAG.The task parameters are set as indicated in Table 6.2, which were used in [111].

6.4.5 User Mobility:

For mobility, we use a waypoint mobility model, where each IoTD moves in straight lines between end points within a given time at a given user speed [61]. For each set of experiments, we set the offloading threshold to two values: (i) a large one, at 100s and (ii) a small one at 12s.

In Figures 6.14 and 6.12, a higher proportion (70%) of users are moving from left to right of the area. This will allow us to understand the impact of user movement on various performance metrics. Each experiment is repeated 10 times and averages are computed accordingly.

6.5 Evaluation and Results

In this section, we now present the results of our simulation experiments. We show the results that explain the impact of increasing the number of users with different mobility speeds with big/small threshold to analysis the variation of average completion times as shown in Figure 6.3, the variation of average of buffer size as shown in Figure 6.4, and the variation of average of maximum buffer size as shown in Figure 6.5. Due to reasons of space, we will present only a sample of the results. We use S, M and B to denote small, medium and big jobs respectively.

6.5.1 Offloading to a Centralised Cloudlet

As a base case, we consider the case where each user offloads their 4 tasks to a central cloudlet (see Figure 6.10). The X-axis represents the number of users as: (10, 20, 30, 40, 50), while the Y-axis represents the completion time. As can be observed, when the job is big (label B), the completion time increases. Also as can be expected, the completion time increases linearly with increasing number of users.

6.5.2 Average Completion Time

We analyse the average job completion times across all users (see Figures 6.3 and 6.6 (a, b, c, and d)). We make the following observations: (i) the average completion time increases with increasing job size, (ii) average completion time increases with increases number of users due to higher number of tasks to be executed and (iii) the average completion time decreases with increasing speed, due to the fact that a larger number of cloudlets become in range when users are moving faster, thereby achieving some form of load spreading. Thus, the possibility of offloading to a lightly loaded cloudlet becomes higher.

However, given that our algorithm has a threshold parameter to delay offloading, we conjecture that a large threshold means that delay is not needed. As such, a job may spend a longer time waiting in a buffer. On the other hand, a smaller threshold means a delay is preferred. As such, load spreading becomes possible and possibly leading to load balancing. This is corroborated by the observations we make, as the average completion times are lower when the threshold is smaller (see Figures 6.3 (a, b, c, and d) and 6.6 (a, b, c, and d) respectively) across all speeds and job sizes.

6.5.3 Average Buffer Size/Waiting queue size

We now investigate the impact on buffer size, which we measure in seconds. By buffer size, we mean the total execution time of the various tasks in the buffer, that captures the current total waiting time in the buffer. We believe that measuring buffer size in time units (i.e., seconds) is suitable due to our focus on mminimizing completion time.

In Figures (6.4 and 6.7 (e, f, g, and h)), we observe that the average buffer size (i) increases with job size, (ii) increases with the number of users (due to increasing queue length) and (iii) shows very little variation with increasing speed. We conjecture that this happens due to the fact that jobs are only offloaded onto a similar set of cloudlets, irrespective of the speed and that the set is dependent on the delay threshold. Also, since we conjecture that load

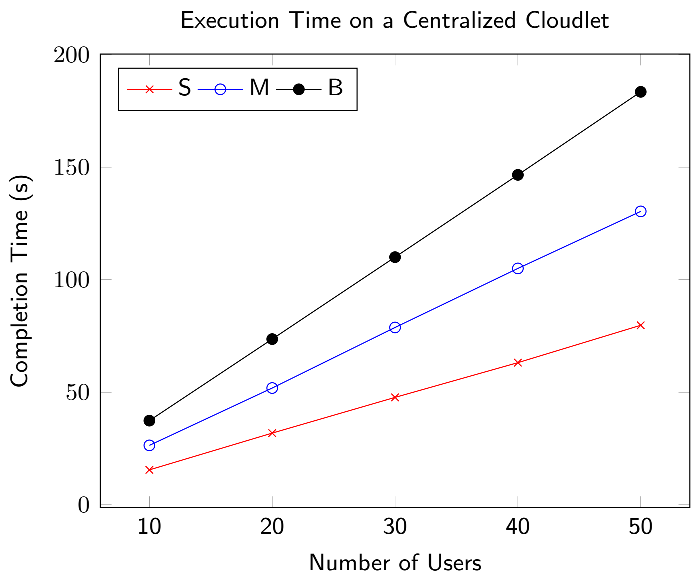


Figure 6.10: Completion Time for Multi-User on a Centralized Cloudlet

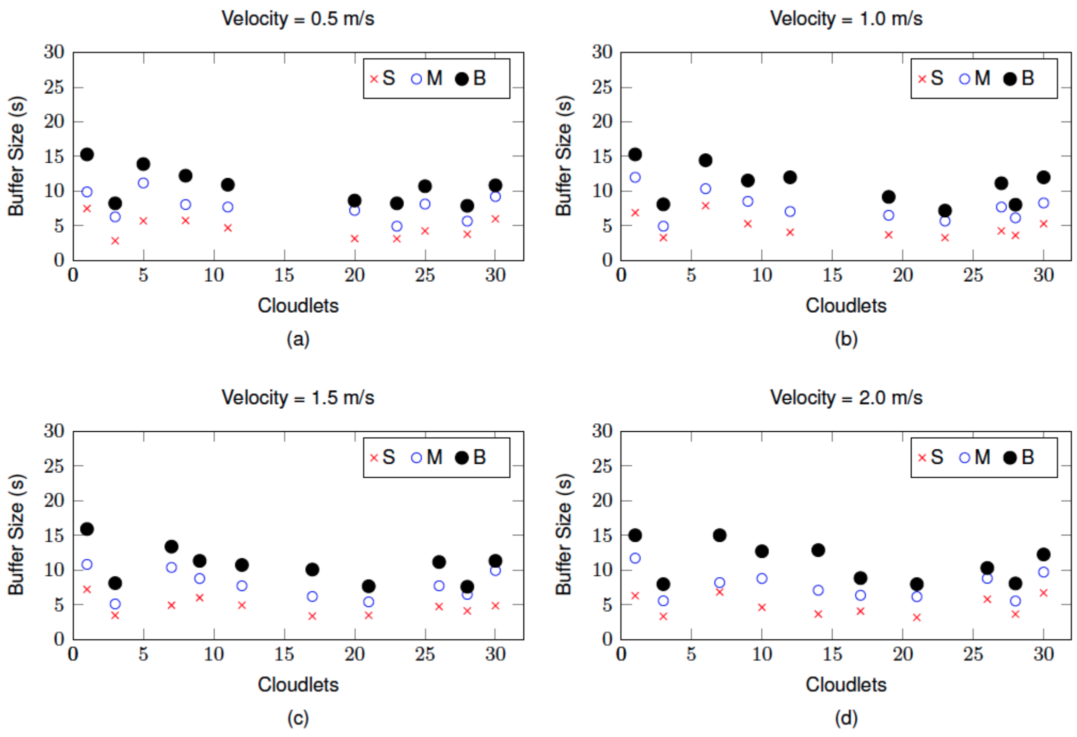


Figure 6.11: Symmetric Mobility with a Big Threshold Value:
Variation of Average Buffer Size

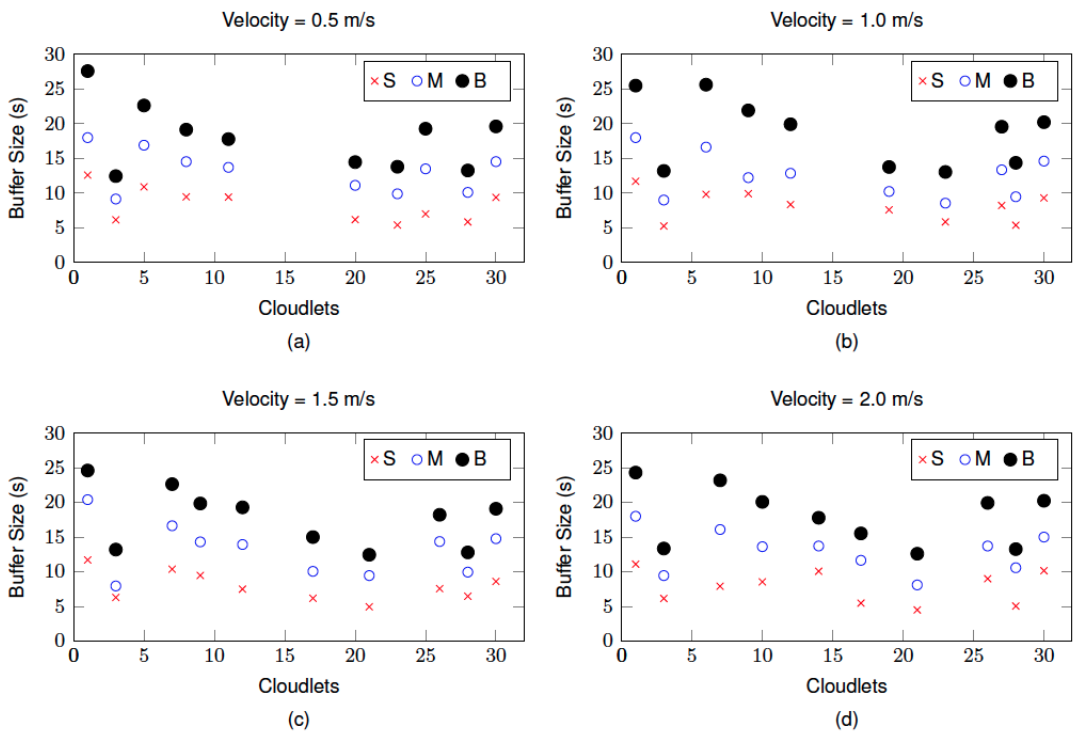


Figure 6.12: Asymmetric Mobility with a Big Threshold Value: Variation of Average Buffer Size

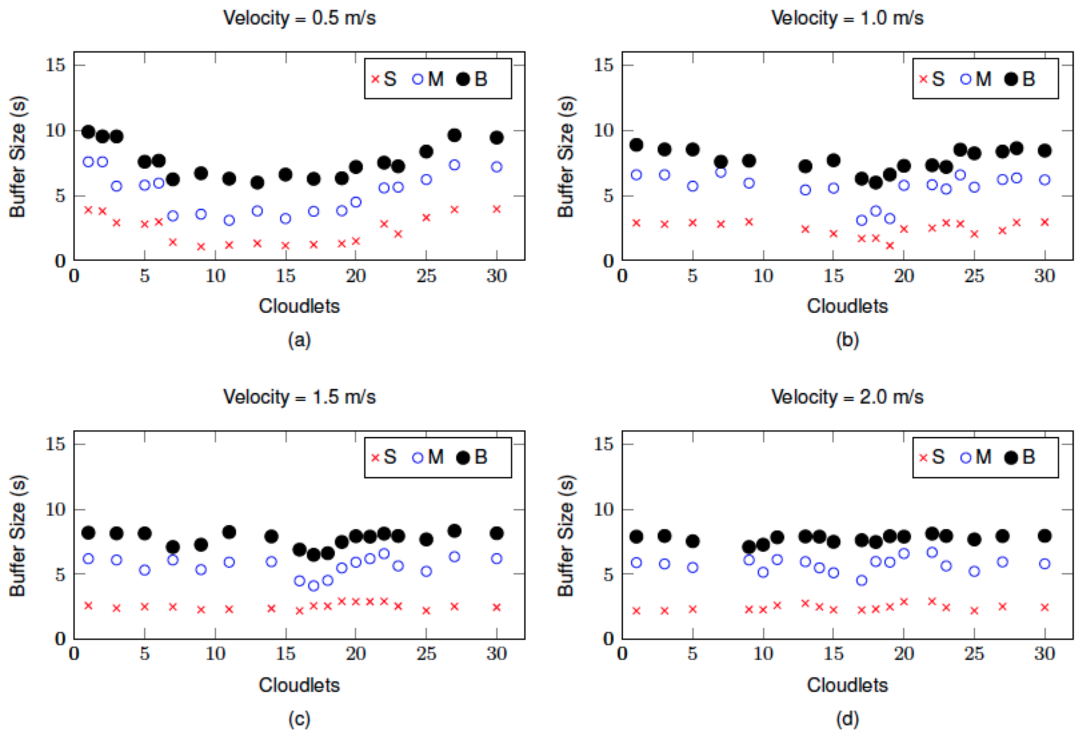


Figure 6.13: Symmetric Mobility with a Small Threshold Value: Variation of Average Buffer Size

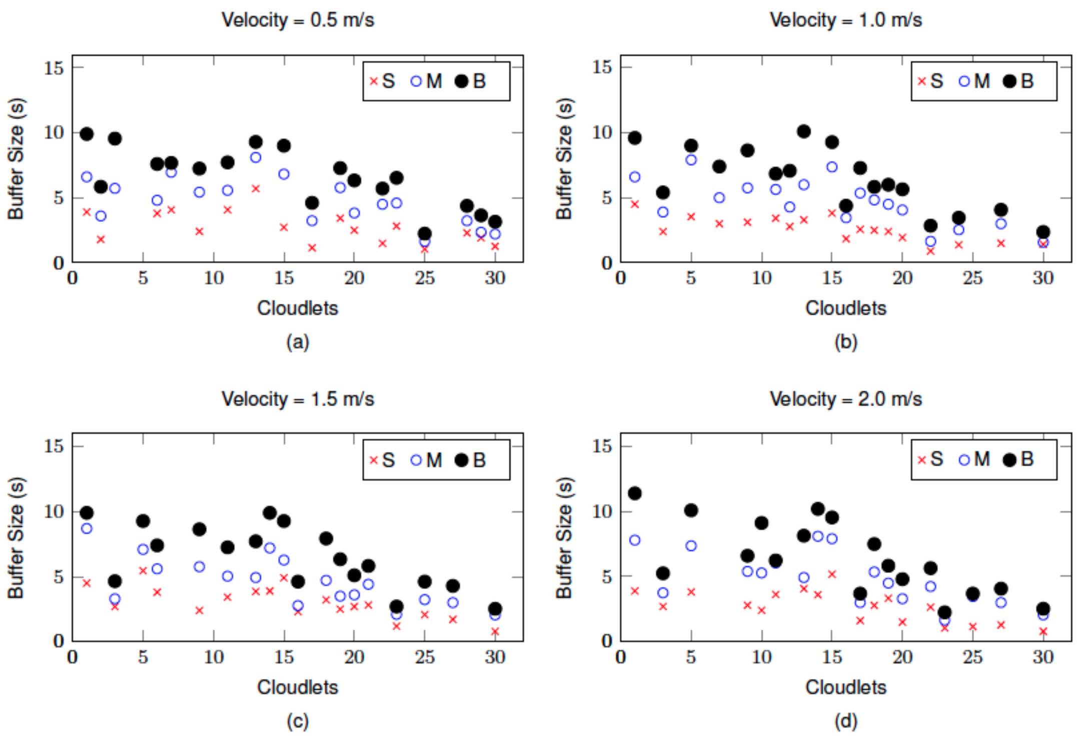


Figure 6.14: Asymmetric Mobility with a Small Threshold Value: Variation of Average Buffer Size

balancing occurs at lower thresholds, we also observe that the average buffer sizes decrease, thereby supporting our intuition (see Figures 6.4 (e, f, g, and h) and 6.7 (e, f, g, and h) respectively).

6.5.4 Average Maximum Buffer Size

The maximum buffer size is in terms of a maximum waiting time as seconds and we calculate the average of maximum waiting time of all offloadable tasks in edge node buffer. To address the last observation, we analyse the average maximum buffer size in the network in Figures (6.5 and 6.8 (i, j, k, and l) respectively). We observe that similar patterns concerning the impact of job sizes and number of users on maximum buffer sizes, when compared to average buffer sizes.

However, we also observe that the average maximum buffer size is larger than its corresponding average buffer size (see Figures 6.5 and 6.8 (i, j, k, and l)) respectively). For example, the average buffer size for a large job for 40 users moving at 0.5 m/s is around 15s, while the average maximum buffer size for the same large job for 40 users at the same speed is 20s. The same pattern is observed at various speeds and across job sizes. This is due to the fact that there are cloudlets that are receiving the "larger" proportion of tasks due to them being powerful and such cloudlets are located at the start of the trajectory.

However, such pattern is not observed in Figures 6.5 (i, j, k, and l), where the threshold is small. This is due to the fact that, irrespective of whether a cloudlet is powerful, a user will prefer to delay (low threshold) when a queue reaches a certain size. Thus, the maximum buffer size is reduced.

6.5.5 Buffer Size Profiling

Earlier, we observed that average buffer sizes were unchanged across various speeds, when the threshold is high. To investigate this, we profile the size of buffers in the network. The x-axis represent the distribution of the cloudlets from left to right.

Figures 6.13 and 6.11 show the buffer profiles under symmetric mobility with 50% of users starting from both right and left side while Figures 6.14 and 6.12 show the profiles under asymmetric mobility, where 70% of users start from the left to right of the map area. As can be noticed, under symmetric mobility and across all speeds, few cloudlets were used in the network, around 30%, when the threshold is high. Due to this, it means that, irrespective of speed, most jobs are offloaded onto the "early" nodes, which means speed does not have a noticeable impact on average buffer sizes. On the other hand, when the threshold is low and at higher speeds, it can be noticed that the buffer size at every node is roughly similar, supporting our intuition of load balancing at lower thresholds.

Under asymmetric mobility, from Figures 6.14 and 6.12, under a low threshold, it can be observed that most nodes are used for offloading, though load balancing is not achieved, as in the symmetric case. But, as speed increases, it can be observed that the middle nodes have higher buffer sizes. This is because, as mobility is asymmetric, nodes from the left get filled quickly, forcing later users towards the central nodes. For high threshold, the pattern is different as users tend to adopt a "bin-packing" pattern, making central nodes less busy.

6.6 Summary

In this chapter, we have formalised the problem of offloading of mobile and dependent tasks onto an edge network. We have presented the first fully distributed algorithm for such task offloading, using a delay threshold parameter to encourage load balancing, thereby reducing completion times. We have conducted extensive simulation using a standard network simulator, ns-3. We have shown that our offloading algorithm provides significantly improves performance compared to the base case of offloading to a central cloudlet, sometimes by up to a factor of 10. We have also shown the impact of mobility patterns on the performance of the algorithm. Since low threshold values yield good load balancing, there is a tradeoff to be made, whereby lower thresholds may result in some jobs not completing as users keep looking for better options, to ultimately run out of options. In all of our experiments, all jobs were completed.

Chapter 7

Conclusions and Future Work

This thesis has presented work on the development of algorithms for task offloading in mobile edge network. However, as in most works, there are assumptions that have been made and limitations exist in our approach.

In this chapter, we will discuss some of the limitations of the work and provide avenues for future work as part of the conclusion.

7.1 Thesis Limitations

Computational offloading with dependent tasks and user mobility may be considered from a variety of perspectives. Primarily this thesis is concerned with issues that affect the offloading during the user mobility. The analysis of these issues is conducted mainly in the user mobility mechanism and the edge nodes connection with the user without failure.

Our user mobility model followed the assumption that the client is mobile and its travel path is known apriori, i.e., waypoint model with the start and endpoints known. Though this may be viewed as a limitation, such an assumption is not unrealistic for the considered application scenario. Essentially, the user is interested in getting the results of the tasks along the way; the results may trigger a change of travel plan (in this case, this will be a considered a "different user"), and thus would not affect the viability of the solution. It is also worth mentioning that our solution can be still applied even if the travel path is not known; in such a case only the current location and expected intermediate destination (rendezvous point) will be specified along with the tasks. The random waypoint model could be considered a more realistic model, to be able moving in different directions. However we have not use random waypoint model because it makes analysis of the results more difficult and impossible in some scenarios.

Another assumption we have made is that any user can only offload a single task to a cloudlet, on the assumption that resources should be shared across users. This in effect rules out users hogging resources. However, there can be cases where offloading more tasks onto the same cloudlet can be beneficial, especially if is expected that the will be lots of communication between the tasks. We have not addressed this scenario in our work and this is an area for future research.

Our work using a central delay server was used to be able to mimic the fact that there are multiple users in the network, thereby allow speeding up of the simulation experiments. This simplifying assumption was removed in our distributed solution where each user had to compute the delay at each server. In effect, the delay server provided network-wide information to users, allowing them to know whether to defer an offload or not. On the other hand, knowledge of users are only local and it is very difficult for a user to know if deferring a task offload is going to be beneficial. We showed, through the use of threshold, how the process of deferring offloading can be achieved without knowing network-wide information and we showed that the load is more evenly balanced when task deferring becomes possible.

7.2 Conclusion

This thesis has focused on developing offloading algorithms to present the novel problem of offloading and scheduling dependent tasks during the user mobility on edge networks. The aim is to minimise the completion time of the job during the task offloading process. Current offloading works in MCC/MEC suffer from at least one of these limitations: (i) assume independent tasks, (ii) assume static users (without mobility), (iii) assume a central server that performs the offloading on behalf of users or (iv) focus on a single user. In this thesis, we address all these challenges together by developing heuristic offloading algorithm and distributed offloading algorithm. Initially we outline problems associated currently with offloading dependent tasks during the user mobility on MEN. Next, we present a summary of the previous work performed in developing techniques for computational offloading in mobile cloud computing (MCC) and mobile edge computing (MEC). In chapter 3, the information theoretic models focused on practice when the offloading algorithms are simulated in ns-3 simulator, CPLEX with ILP model, and the real deployment. Also, we have analyzed and defined the statement problem of the offloading dependent tasks with explanation of the current offloading challenges by using a case study. We developed a heuristic offloading algorithm to offload a job that consists of dependent tasks on a set of cloudlets involving user mobility on the edge. Given that the problem is in general intractable, we provided a heuristic that returns a schedule that will potentially reduce the computation time in the cloudlet. We developed an ILP model with the optimal solution and implemented it in CPLEX solver with the same datasets using a ns-3 simulation. We conducted a range of simulation experiments using ns-3 with a range of job sizes, mobility speed and delay distributions. Finally, we ran a real deployment of the heuristic with a Face Recognition Application and Flask Server.

Our results show that, under certain conditions, it is impossible for the heuristic to return a proper schedule. We have studied the relationship between delay, speed and job size and shown that the completion time gain increases with the size of the job. Next, we have formalised the problem of offloading of mobile and dependent tasks onto an edge network with multi-users. We have presented the fully distributed algorithm for such task offloading, using a delay threshold parameter to encourage load balancing, thereby reducing completion times. We have conducted extensive simulation using an ns-3 simulator. We have shown that our offloading algorithm provides significantly improves performance compared to the base case of offloading to a central cloudlet, sometimes by up to a factor of 10. We have also shown the impact of mobility patterns on the performance of the algorithm. Since low threshold values yield good load balancing, there is a tradeoff to be made, whereby lower thresholds may result in some jobs not completing as users keep looking for better options, ultimately leading to such users running out of options. In all of our experiments, all jobs were completed. In summary, we developed the offloading algorithms for mobile devices hosting dependent tasks, where a dependent task cannot start until its parent has completed, with the aim of reducing completion latency. We ran extensive simulations using ns-3, ILP model with the optimal solution in CPLEX and deployment. Our results show that our offloading algorithm provides significant improvement compared to offloading in current works.

7.3 Directions for Further Work

There are a number of future avenues of work that could be undertaken by expanding on ideas presented in this thesis. This section will begin by describing immediate ideas through which the algorithms presented could be explored further, and then present a new area regarding the task offloading in edge network.

7.3.1 Expand the Heuristic Algorithm with Offloading Failure

In Chapter 5, we formulate the offloading problem of dependent tasks in edge as a constraint satisfaction problem and we provide a heuristic that attempts to reduce the computation time of dependent tasks. This model does not consider the offloading failure that could happen in connection or in the edge node during the offloading process. Offloading connection failure that could happen during the communication between the edge users and the edge nodes will affect the offloading process. Some connection issues that could prevent the access to the edge nodes and cause connection failure would be, for example, user mobility with high speed or weaker bandwidth of wireless connection. Also, edge node failure or crashes during the offloading process. Edge nodes may sometimes be unreachable due either to node failure or loss of service due to being out of communication range. These issues affect the performance of computational offloading. It is beneficial to extend our heuristic offloading algorithm by considering solutions for the offloading failures. We need to consider a waypoint mobility model for the mobility model during task offloading. We need to use a dynamic programming technique to divide the problem under consideration into sub problems in order to define the toleration techniques for the failure assumptions. We will extend the current heuristic and formulate the problem as an optimization problem involving (i) task offloading process, (ii) task scheduling process, (iii) mobility mechanism, and (iv) fault-recovery management as (failure manager). The failure manager will trace and monitor the failure whether due to the communication failure or node failure. All failure decisions should be reported in the extension part of the fault recovery algorithm.

7.3.2 Expand the Distributed Algorithm with Job Priority

In Chapter 6, we developed a distributed offloading algorithm for mobile devices hosting dependent tasks, where a dependent task cannot start until its parent has completed, with the aim of reducing completion latency. This model does not consider the task where a particular job requires priority completion under time constraints during the offloading process. We can extend the distributed offloading algorithm by considering two aspects of the priority job: (i) considering the level of prioritisation set for urgent tasks with non-preemptive priority queues in the cloudlet, (ii) in the process of task offloading to the cloudlet, the waiting buffer is allocated on the basis of priority with updating for new tasks. The system model with the extension of priority job will consist of edge network model, priority and dependency application model, computational offloading model and user mobility model. Priority and dependency application model will consist of a set of tasks with dependency constraints and application deadline. Each application can be divided into multiple dependency tasks. We need to model the application A_i (application of user i) as a directed acyclic graph DAG with entry tasks $T_i^{ent.}$ (as independent tasks at the root level) and exit tasks $T_i^{ext.}$ (as last tasks at the last level). Entry tasks $T_i^{ent.}$ do not have any immediate predecessor tasks and exit tasks $T_i^{ext.}$ do not have any immediate successor tasks. We model a set of tasks in an application with dependency constraints $T_i = \{T_i^1 \ldots T_i^j\}$ between entry tasks and exit tasks. As such, we will model the application as a directed acyclic graph $A_i = (T_i, D_i)$ where $D_i \subseteq T_i \times T_i$ and $(T_i^j, T_i^l) \in D_i$ means that task T_i^l depends on task T_i^j. Entry tasks $T_i^{ent.}$ and exit tasks $T_i^{ext.} \subseteq\ T_i$. A task will be associated with the following meta-data, at least: $\langle$the size of input data, number of instructions of input data, the maximum delay allowed to complete the task$\rangle$. Each application has a maximum delay time T_i^{max} i.e. the completion time of the set of all tasks $T = \bigcup_{i=1}^{n} T_i$ in application(i) $<=$ T_i^{max}. The deadline of the exit tasks $T_i^{ext.}$ in application(i) will be equal or less than the T_i^{max}. We will assume tasks belonging to different applications are independent of each other i.e. $T_i^1 \in A_i^1 \neq T_i^2 \in A_i^2$, where 1 and 2 are two different applications with the same user (i). We will assume tasks belonging to the same application will have dependency requirements. Each job can be described in these terms as T_i= $\{(T_i^j.Size), (T_i^j.n), (T_i^j.r), (T_i^{j(max)})\},\ T_i^j \in T_i$ represents one application i, where $(T_i^j.Size)$ is the size of input data for a task j in application T_i, $(T_i^j.n)$ is the number of instructions for a task j, (T˙i˜j.r) is the task result, and $(T_i^{j(max)})$ is the maximum deadline to complete the application T_i. The maximum deadline for applications that do not have priority is : $(T_i^{j(max)} = \infty)$ to be ordered on the normal queue (NQ) as FCFS. The exit task of the application should be completed before the application deadline:

$$(T_i^{ext.} \leq T_i^{max})\ \wedge\ (T_i^{max} = T_i^{j(max)})$$

Bibliography

[1] Number of smartphone users from 2016 to 2021. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, . Accessed: 2021-08-16.

[2] Connected cars worldwide - statistics facts. https://www.statista.com/topics/1918/connected-cars/#dossierKeyfigures, . Accessed: 2021-12-08.

[3] Luca Aceto, Andrea Morichetta, and Francesco Tiezzi. Decision support for mobile cloud computing applications via model checking. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 199–204. IEEE, 2015.

[4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[5] LS Ashiho. Mobile technology: Evolution from 1g to 4g. *Electronics for you*, pages 94–98, 2003.

[6] Sergio Barbarossa, Stefania Sardellitti, and Paolo Di Lorenzo. Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *Signal Processing Advances in Wireless Communications (SPAWC), 2013 IEEE 14th Workshop on*, pages 26–30. IEEE, 2013.

[7] Marco V Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 1285–1293. IEEE, 2013.

[8] Pronaya Bhattacharya, Sudeep Tanwar, Rushabh Shah, and Akhilesh Ladha. Mobile edge computing-enabled blockchain framework?a survey. In *Proceedings of ICRIC 2019*, pages 797–809. Springer, 2020.

[9] Jing Bi, Haitao Yuan, Shuaifei Duanmu, Meng Chu Zhou, and Abdullah Abusorrah. Energy-optimized partial computation offloading in mobile edge computing with genetic simulated-annealing-based particle swarm optimization. *IEEE Internet of Things Journal*, 2020.

[10] Suzhi Bi, Liang Huang, and Ying-Jun Angela Zhang. Joint optimization of service caching placement and computation offloading in mobile edge computing systems. *IEEE Transactions on Wireless Communications*, 19(7):4947–4963, 2020.

[11] S Binitha, S Siva Sathya, et al. A survey of bio inspired optimization algorithms. *International journal of soft computing and engineering*, 2 (2):137–151, 2012.

[12] Andrew D Birrell, Roy Levin, Michael D Schroeder, and Roger M Needham. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, 1982.

[13] Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. Solving mixed-integer quadratic programming problems with ibm-cplex: a progress report. In *Proceedings of the twenty-sixth RAMP symposium*, pages 16–17, 2014.

[14] Xinchen Cai, Hongyu Kuang, Hao Hu, Wei Song, and Jian Lü. Response time aware operator placement for complex event processing in edge computing. In *International Conference on Service-Oriented Computing*, pages 264–278. Springer, 2018.

[15] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless communications and mobile computing*, 2(5):483–502, 2002.

[16] Gustavo Carneiro. Ns-3: Network simulator 3. In *UTM Lab Meeting April*, volume 20, pages 4–5, 2010.

[17] Zheng Chang, Liqing Liu, Xijuan Guo, and Quan Sheng. Dynamic resource allocation and computation offloading for iot fog computing system. *IEEE Transactions on Industrial Informatics*, 2020.

[18] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[19] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2016.

[20] Siyao Cheng, Zhenyue Chen, Jianzhong Li, and Hong Gao. Task assignment algorithms in data shared mobile edge computing systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 997–1006. IEEE, 2019.

[21] Byung-Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *HotOS*, volume 9, pages 8–11, 2009.

[22] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[23] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[24] Alexandre da Silva Veith, Marcos Dias de Assuncao, and Laurent Lefevre. Latency-aware placement of data stream analytics on edge computing. In *International Conference on Service-Oriented Computing*, pages 215–229. Springer, 2018.

[25] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of things*, pages 61–75. Elsevier, 2016.

[26] Debashis De. *Mobile cloud computing: architectures, algorithms and applications.* Chapman and Hall/CRC, 2016.

[27] Maofei Deng, Hui Tian, and Bo Fan. Fine-granularity based application offloading policy in cloud-enhanced small cell networks. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 638–643. IEEE, 2016.

[28] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony QS Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[29] Jakub Dolezal, Zdenek Becvar, and Tomas Zeman. Performance evaluation of computation offloading from mobile device to the edge of mobile network. In *Standards for Communications and Networking (CSCN), 2016 IEEE Conference on*, pages 1–7. IEEE, 2016.

[30] Luobing Dong, Meghana N Satpute, Junyuan Shan, Baoqi Liu, Yang Yu, and Tihua Yan. Computation offloading for mobile-edge computing with multi-user. In *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*, pages 841–850. IEEE, 2019.

[31] Nur Idawati Md Enzai and Maolin Tang. A taxonomy of computation offloading in mobile cloud computing. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 19–28. IEEE, 2014.

[32] S Erana Veerappa Dinesh and K Valarmathi. A novel energy estimation model for constraint based task offloading in mobile cloud computing. *Journal of Ambient Intelligence and Humanized Computing*, 11:5477–5486, 2020.

[33] Melike Erol-Kantarci and Sukhmani Sukhmani. Caching and computing at the edge for mobile augmented reality and virtual reality (ar/vr) in 5g. In *Ad Hoc Networks*, pages 169–177. Springer, 2018.

[34] Fariba Farahbakhsh, Ali Shahidinejad, and Mostafa Ghobaei-Arani. Context-aware computation offloading for mobile edge computing. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2021.

[35] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future generation computer systems*, 29(1):84–106, 2013.

[36] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Srirama, and Rajkumar Buyya. Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine*, 53(3):80–88, 2015.

[37] Ying Gao, Wenlu Hu, Kiryong Ha, Brandon Amos, Padmanabhan Pillai, and Mahadev Satyanarayanan. Are cloudlets necessary? *School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-15-139*, 2015.

[38] Yang Ge, Yukan Zhang, Qinru Qiu, and Yung-Hsiang Lu. A game theoretic resource allocation for overall energy minimization in mobile cloud computing system. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 279–284. ACM, 2012.

[39] Miguel Grinberg. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[40] William Gropp, Ewing Lusk, and Anthony Skjellum. Using mpi: portable parallel programming with the messagepassing interface. 1999, 1994.

[41] Zhiqiang Gu, Ryuichi Takahashi, and Yoshiaki Fukazawa. Real-time resources allocation framework for multi-task offloading in mobile cloud computing. In *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5. IEEE, 2019.

[42] Songtao Guo, Bin Xiao, Yuanyuan Yang, and Yang Yang. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[43] Songtao Guo, Menggang Chen, Kai Liu, Xiaofeng Liao, and Bin Xiao. Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing. *IEEE Transactions on Mobile Computing*, 2020.

[44] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219, 2011.

[45] William E Hart, Carl D Laird, Jean-Paul Watson, David L Woodruff, Gabriel A Hackebeil, Bethany L Nicholson, and John D Siirola. *Pyomo-optimization modeling in python*, volume 67. Springer, 2017.

[46] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[47] Vu Huy Hoang, Tai Manh Ho, and Long Bao Le. Mobility-aware computation offloading in mec-based vehicular wireless networks. *IEEE Communications Letters*, 24(2):466–469, 2019.

[48] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing?a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[49] Pei-Qiu Huang, Yong Wang, Kezhi Wang, and Zhi-Zhong Liu. A bilevel optimization approach for joint offloading decision and resource allocation in cooperative mobile edge computing. *IEEE transactions on cybernetics*, 2019.

[50] Lei Jiao, Roy Friedman, Xiaoming Fu, Stefano Secci, Zbigniew Smoreda, and Hannes Tschofenig. Cloud-based computation offloading for mobile devices: State of the art, challenges and opportunities. In *Future Network*

and Mobile Summit (FutureNetworkSummit), 2013, pages 1–11. IEEE, 2013.

[51] Caihong Kai, Hao Zhou, Yibo Yi, and Wei Huang. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Transactions on Cognitive Communications and Networking*, 2020.

[52] Yi-Hsuan Kao, Bhaskar Krishnamachari, Moo-Ryong Ra, and Fan Bai. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Transactions on Mobile Computing*, 16(11):3056–3069, 2017.

[53] Parmeet Kaur and Shikha Mehta. Efficient computation offloading using grey wolf optimization algorithm. In *AIP Conference Proceedings*, volume 2061, page 020011. AIP Publishing LLC, 2019.

[54] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *International Conference on Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2010.

[55] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Infocom, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[56] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[57] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[58] Yu-Kwong Kwok and Ishfaq Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47(1):58–77, 1997.

[59] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[60] Wael Labidi, Mireille Sarkiss, and Mohamed Kamoun. Energy-optimal resource scheduling and computation offloading in small cell networks. In *Telecommunications (ICT), 2015 22nd International Conference on*, pages 313–318. IEEE, 2015.

[61] Daniel Lertpratchya. Wi-fi module in ns-3, 2014.

[62] Chao Li, Junjuan Xia, Fagui Liu, Dong Li, Lisheng Fan, George K Karagiannidis, and Arumugam Nallanathan. Dynamic offloading for multiuser muti-cap mec networks: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 2021.

[63] Zhi Li and Qi Zhu. Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing. *Information*, 11(2): 83, 2020.

[64] Bowen Liu, Xiaolong Xu, Lianyong Qi, Qiang Ni, and Wanchun Dou. Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user mec environment. *Journal of Systems Architecture*, 114:101970, 2021.

[65] Chen-Feng Liu, Mehdi Bennis, and H Vincent Poor. Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In *2017 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7. IEEE, 2017.

[66] Juan Liu, Yuyi Mao, Jun Zhang, and Khaled B Letaief. Delay-optimal computation task scheduling for mobile-edge computing systems. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 1451–1455. IEEE, 2016.

[67] Liuyan Liu, Haisheng Tan, Shaofeng H-C Jiang, Zhenhua Han, Xiang-Yang Li, and Hong Huang. Dependent task placement and scheduling with function configuration in edge computing. In *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2019.

[68] Tong Liu, Lu Fang, Yanmin Zhu, Weiqin Tong, and Yuanyuan Yang. Latency-minimized and energy-efficient online task offloading for mobile edge computing with stochastic heterogeneous tasks. *IEEE Transactions on Mobile Computing*, 2020.

[69] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[70] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[71] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[72] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*, pages 1–6. IEEE, 2017.

[73] Mohammed Maray, Arshad Jhumka, Adam Chester, and Mohamed Younis. Scheduling dependent tasks in edge networks. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pages 1–4. IEEE, 2019.

[74] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in engineering software*, 95:51–67, 2016.

[75] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in engineering software*, 69:46–61, 2014.

[76] Stuart Mitchell. An introduction to pulp for python programmers. *Python Papers Monograph*, 1(14), 2009.

[77] Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, 2011.

[78] Olga Munoz, Antonio Pascual-Iserte, and Josep Vidal. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology*, 64(10):4738–4755, 2015.

[79] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity.* Courier Corporation, 1998.

[80] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015.

[81] Milan Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[82] Hua Peng, Wu-Shao Wen, Ming-Lang Tseng, and Ling-Ling Li. Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment. *Applied Soft Computing*, 80: 534–545, 2019.

[83] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang, and Zhiguo Ding. A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, 8:116974–117017, 2020.

[84] Jan Plachy, Zdenek Becvar, and Pavel Mach. Path selection enabling user mobility and efficient distribution of data for computation at the edge of mobile network. *Computer Networks*, 108:357–370, 2016.

[85] Michael L Powell and Barton P Miller. *Process migration in DEMOS/MP*, volume 17. ACM, 1983.

[86] Amir Masoud Rahmani, Mokhtar Mohammadi, Adil Hussein Mohammed, Sarkhel H Taher Karim, Mohammed Kamal Majeed, Mohammed Masdari, and Mehdi Hosseinzadeh. Towards data and computation offloading in mobile cloud computing: Taxonomy, overview, and future directions. *Wireless Personal Communications*, pages 1–39, 2021.

[87] Jennifer S Raj. Improved response time and energy management for mobile cloud computing using computational offloading. *Journal of ISMAC*, 2(01):38–49, 2020.

[88] Shima Rashidi and Saeed Sharifian. A hybrid heuristic queue based algorithm for task assignment in mobile cloud. *Future Generation Computer Systems*, 68:331–345, 2017.

[89] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[90] Heungsoon Rim, Seonggun Kim, Youil Kim, and Hwansoo Han. Transparent method offloading for slim execution. In *Wireless Pervasive Computing, 2006 1st International Symposium on*, pages 1–6. IEEE, 2006.

[91] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[92] Umber Saleem, Yu Liu, Sobia Jangsher, Xiaoming Tao, and Yong Li. Latency minimization for d2d-enabled partial computation offloading in mobile edge computing. *IEEE Transactions on Vehicular Technology*, 69 (4):4472–4486, 2020.

[93] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, 2014.

[94] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[95] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[96] Ali Shakarami, Mostafa Ghobaei-Arani, Mohammad Masdari, and Mehdi Hosseinzadeh. A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective. *Journal of Grid Computing*, pages 1–33, 2020.

[97] Nanliang Shan, Yu Li, and Xiaolong Cui. A multilevel optimization framework for computation offloading in mobile edge computing. *Mathematical Problems in Engineering*, 2020, 2020.

[98] Chang Shu, Zhiwei Zhao, Yunpeng Han, and Geyong Min. Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks. In *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9. IEEE, 2019.

[99] Chang Shu, Zhiwei Zhao, Yunpeng Han, Geyong Min, and Hancong Duan. Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach. *IEEE Internet of Things Journal*, 7(3): 1678–1689, 2019.

[100] Neil Smyth. *Kotlin/Android Studio 3.0 Development Essentials-Android 8 Edition.* eBookFrenzy, 2017.

[101] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *2012 IEEE symposium on computers and communications (ISCC)*, pages 000059–000066. IEEE, 2012.

[102] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile web service provisioning. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conferenceon*, pages 120–120. IEEE, 2006.

[103] Wen Sun, Jiajia Liu, and Haibin Zhang. When smart wearables meet intelligent vehicles: Challenges and future directions. *IEEE wireless communications*, 24(3):58–65, 2017.

[104] Yang Sun, Tingting Wei, Huixin Li, Yanhua Zhang, and Wenjun Wu. Energy-efficient multimedia task assignment and computing offloading for mobile edge computing networks. *IEEE Access*, 8:36702–36713, 2020.

[105] Sowndarya Sundar and Ben Liang. Offloading dependent tasks with communication delay and deadline constraint. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 37–45. IEEE, 2018.

[106] Vinu Sundararaj. Optimal task assignment in mobile cloud computing by queue based ant-bee algorithm. *Wireless Personal Communications*, 104(1):173–197, 2019.

[107] Ling Tang and Shibo He. Multi-user computation offloading in mobile edge computing: A behavioral perspective. *IEEE Network*, 32(1):48–53, 2018.

[108] Shanmuganathan Thananjeyan, Chien Aun Chan, Elaine Wong, and Ampalavanapillai Nirmalathas. Mobility-aware energy optimization in hosts selection for computation offloading in multi-access edge computing. *IEEE Open Journal of the Communications Society*, 1:1056–1065, 2020.

[109] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[110] Dongyu Wang, Zhaolin Liu, Xiaoxiang Wang, and Yanwen Lan. Mobility-aware task offloading and migration schemes in fog computing networks. *IEEE Access*, 7:43356–43368, 2019.

[111] Zi Wang, Zhiwei Zhao, Geyong Min, Xinyuan Huang, Qiang Ni, and Rong Wang. User mobility aware task assignment for mobile edge computing. *Future Generation Computer Systems*, 85:1–8, 2018.

[112] Muhammad Waqas, Yong Niu, Manzoor Ahmed, Yong Li, Depeng Jin, and Zhu Han. Mobility-aware fog computing in dynamic environments: Understandings and implementation. *IEEE Access*, 7:38867–38879, 2018.

[113] Klaus Wehrle, Mesut Günes, and James Gross. *Modeling and tools for network simulation*. Springer Science & Business Media, 2010.

[114] Huaming Wu, Katinka Wolter, Pengfei Jiao, Yingjun Deng, Yubin Zhao, and Minxian Xu. Eedto: An energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing. *IEEE Internet of Things Journal*, 2020.

[115] Chao Yang, Yi Liu, Xin Chen, Weifeng Zhong, and Shengli Xie. Efficient mobility-aware task offloading for vehicular edge computing networks. *IEEE Access*, 7:26652–26664, 2019.

[116] Jiachen Yang, Bin Jiang, Zhihan Lv, and Kim-Kwang Raymond Choo. A task scheduling algorithm considering game theory designed for energy management in cloud computing. *Future Generation Computer Systems*, 2017.

[117] Kun Yang, Shumao Ou, and Hsiao-Hwa Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE communications magazine*, 46(1), 2008.

[118] Lei Yang, Jiannong Cao, Shaojie Tang, Di Han, and Neeraj Suri. Run time application repartitioning in dynamic mobile cloud environments. *IEEE Transactions on Cloud Computing*, 4(3):336–348, 2014.

[119] Lei Yang, Jiannong Cao, Shaojie Tang, Di Han, and Neeraj Suri. Run time application repartitioning in dynamic mobile cloud environments. *IEEE Transactions on Cloud Computing*, 4(3):336–348, 2016.

[120] Lei Yang, Bo Liu, Jiannong Cao, Yuvraj Sahni, and Zhenyu Wang. Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds. *IEEE Transactions on Services Computing*, 2019.

[121] Lei Yang, Changyi Zhong, Qiuhui Yang, Wanrong Zou, and Ahmed Fathalla. Task offloading for directed acyclic graph applications based on edge computing in industrial internet. *Information Sciences*, 540:51–68, 2020.

[122] Lichao Yang, Heli Zhang, Ming Li, Jun Guo, and Hong Ji. Mobile edge computing empowered energy efficient task offloading in 5g. *IEEE Transactions on Vehicular Technology*, 67(7):6398–6409, 2018.

[123] Changsheng You, Kaibin Huang, and Hyukjin Chae. Energy efficient mobile cloud computing powered by wireless energy transfer. *IEEE Journal on Selected Areas in Communications*, 34(5):1757–1771, 2016.

[124] Changsheng You, Kaibin Huang, Hyukjin Chae, and Byoung-Hoon Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2017.

[125] Belen Cruz Zapata. *Android studio application development*. Packt Publ., 2013.

[126] Haoyu Zhang, Brian Cho, Ergin Seyfe, Avery Ching, and Michael J Freedman. Riffle: optimized shuffle service for large-scale data analytics. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[127] Jianmin Zhang, Weiliang Xie, Fengyi Yang, and Qi Bi. Mobile edge computing and field trial results for 5g low latency scenario. *China Communications*, 13(2):174–182, 2016.

[128] Jing Zhang, Weiwei Xia, Yueyue Zhang, Qian Zou, Bonan Huang, Feng Yan, and Lianfeng Shen. Joint offloading and resource allocation optimization for mobile edge computing. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[129] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.

[130] Bowen Zhou and Rajkumar Buyya. Augmentation techniques for mobile cloud computing: A taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)*, 51(1):13, 2018.