Software Defined Architecture For Hybrid Edge Computing

Jude Vivek Joseph, B.E.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

(Future Networked Systems)

Supervisor: George Iosifidis

Co-Supervisor: Jeongho Kwak

September 2018

Declaration

I, ***Jude Vivek Joseph***, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Jude Vivek Joseph

August 30, 2018

Permission to Lend and/or Copy

I, ***Jude Vivek Joseph***, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Jude Vivek Joseph

August 30, 2018

Acknowledgments

I would first like to express my sincere gratitude to my supervisor ***Dr. Georgis Iosifidis*** of Trinity College Dublin, for all the help and resources he provided me throughout the course of this dissertation.

I would also like to thank my co-supervisor ***Dr. Jeongho Kwak*** of Trinity College Dublin, who continuously extended his support in sharing suggestions and guiding me throughout the research work with this expertise.

I would also like to acknowledge ***Dr. Marco Ruffini*** of Trinity College Dublin, as the second reader of this thesis and I am grateful for his valuable comments on this thesis.

I would like to thank my family for all the support and encouragement they gave me throughout the time I spent on this dissertation.

Finally, I would like to thank my classmates who were always there for offering their help for the duration of my time spent in college.

Jude Vivek Joseph

University of Dublin, Trinity College
September 2018

Software Defined Architecture For Hybrid Edge Computing

Jude Vivek Joseph, Master of Science in Computer Science
University of Dublin, Trinity College, 2018

Supervisor: George Iosifidis

ABSTRACT

The technological advancement in mobile devices has enabled the evolution of new resource-intensive mobile applications. However, the local execution of computationally demanding applications in the mobile devices are constrained by the limited battery power and energy consumption. By leveraging mobile cloud computing (MCC) which offloads the local workloads to the cloud for remote execution, the performance of the mobile computing can be significantly improved. Nevertheless, the cloud computing technology induces a significant delay to exchange data back and forth between the mobile devices and cloud servers. To cope with this network delay problem, mobile edge computing (MEC) paradigm brings the computation and storage to the network edge enabling to handle peak loads efficiently from the mobile devices. However, the edge devices are limited in storage and capacity, making it less reliable for massive workload computations.

Employing a hierarchical hybrid architecture of mobile, edge and cloud enables to manage the workloads across all the tiers effectively. This minimizes the processing delay

or energy consumption at the mobile devices by opportunistically deciding the tiers for the task execution. With the goal of achieving higher energy efficiency and minimal delay in workload processing, we suggest a novel algorithm which minimizes the energy consumption at a mobile device in a hybrid mobile-edge-cloud architecture. The proposed idea is fully implemented and validated in a real network environment using various network interfaces. i.e., WiFi/Cellular (4G). We optimize the CPU speed and the transmission delay by evaluating the energy consumption at the mobile devices. On top of these primary results, we propose an algorithm that effectively reduces energy consumption up to 47% by trading off minimal processing delay.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Resource-rich applications such as video encoding, multi-player chess game, and image recognition demand high mobile processing power and transmission energy. The smartphones are trying to match up with the demand for processing power and are continuously growing. e.g., the latest Qualcomm Snapdragon mobile processor has a CPU clock speed of 2.9 GHz (max). Also, considering the networking traffic at the smartphones, Cisco VNI Mobile forecasts the network traffic to grow from 92% to 99% by 2021 [1] which increases the average load a mobile would experience to carry out any computation.

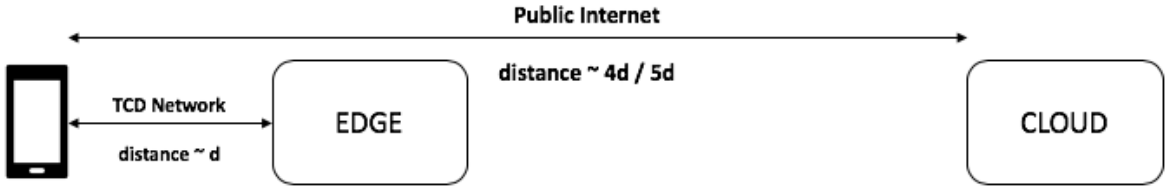

Figure 1.1: Distance comparison of Edge and Cloud from Mobile.

(Distance between mobile-edge components are much shorter when compared with the distance between the mobile-cloud components.)

However, processing all the computations locally in mobile at the maximum CPU clock speed has an adverse impact on the mobile energy consumption since mobile battery power is highly correlated with the mobile CPU clock frequency [2]. The idea of offloading the computation to the remote edge computer (Edge Computing) or cloud data center (Cloud computing) is one of the efficient ways of handling resource-intensive applications by saving mobile energy resources. With the increased use of cloud resources, industry leaders, e.g., Amazon Web Services (AWS) and Microsoft Azure are expanding their PaaS products by making them more user-friendly and cost-effective. Moreover, the latency in

transmitting the data back and forth between mobile devices and the cloud is a significant concern for applications that demand low network latency (e.g., autonomous cars, online multi-player games). Since the virtual machines (VM) managed by the cloud providers (such as AWS, Azure) can be located anywhere in the world, the response time from those servers is not guaranteed and can vary between 36.69 to 374.56 milliseconds[1] making it less reliable for low latency demanding applications.

Trying to address the fallbacks in mobile cloud computing, mobile edge computing has gained attraction in the recent times which aims to bring the computation and storage closer to the network edge. Figure 1.1 depicts the distance comparison between the mobile-edge and mobile-cloud components where it can be observed that the edge components are expected to be placed much closer to the mobile devices while compared with cloud components. Additionally, with Internet of Things (IoT) becoming a great buzzword in the industry and every city in the world having a grand vision of becoming a smart city, edge computing is expected to intrude the technology in no time through smart city applications [3]. Cisco global cloud index forecasts that the edge data generated by IoT devices and people would exceed 500 zettabytes by 2019 whereas the global cloud data would be reaching 10.4 zettabytes by the same time which will introduce massive network traffic [4]. Also, researchers expect the edge devices to carry out 45% of the generated workload computation by itself by 2019 [5]. However, given the minimal storage capacity, and its limited computational power, not every application can make efficient use of edge computing (e.g., edge devices may not be able to handle heavy loads at the given time) making the cloud computing inevitable. Moreover, edge computing would be a big boon for the applications that demand very low latency, which is the primary use case of edge computing. Additionally, we should consider the network energy[2] consumption at the mobile devices while offloading the computation to a remote server, whereas this is not applicable during local processing at the mobile CPU resources. 5G technology aimed at provisioning ultra-reliable low-latency communications is expected to support edge computing big-time to reap its maximum benefits [6].

With multiple technologies trying to address the mobile computation problem from various aspects, efficient mobile computation has become a prime area to explore in application development for better user engagement and effective use of mobile resources. One of the most crucial bottlenecks in mobile computation is to efficiently manage the smart-phone energy resources without compromising on application performance. This demands an efficient algorithm to strike a balance between the networking energy and the

[1] https://www.cloudping.co/

[2] Energy consumed by a mobile phone in handling the network traffic using the cellular or wireless network interface.

CPU energy consumption for completing a task in the mobile device.

1.2 Motivation and Contributions

Computational efficiency of any mobile workload depends on various factors namely (a) *the number of CPU clock cycles required to handle the application processing at any given time* (b) *the amount of battery power spent during the processing.* Since different applications demand various levels of computational power, the number of CPU clock cycles impacts the computational efficiency. Additionally, the mobile battery power is primarily influenced by CPU clock frequency where the battery consumption increases linearly as the CPU clock speed increases [2]. Applications such as video encoding demand high processing power, thus increasing the battery consumption in contrast to the applications demanding less processing power namely default video player [7].

Kalic *et al.* [8] have shown that mobile networking energy depends on choosing the better network interface for the network traffic. For instance, the WiFi network interface consumes less energy when compared to the cellular interface if the mobile is located within WiFi coverage area with high data rates. Choice of energy efficient interface based on the workload acts as one of the driving factors in handling the network traffic in the mobile devices. Previous works [7, 9, 10] have studied the behavior of computational offloading under single hierarchical architecture and static network environments. Considering the heterogeneous nature of mobile and network infrastructure, optimizing the energy under varying loads and network conditions is crucial. Additionally, exploring the possibilities of edge and cloud computing for the mobile environment paves the way for limitless opportunities for better performance.

Therefore, our research work proposes a hybrid software architecture *Hybrid Edge Computing (HEC)* using edge and cloud computing technologies that dynamically allocates the task resources in the mobile devices. Additionally, our architecture efficiently chooses the network and CPU resources for achieving the best computational performance thus saving energy in the mobile devices. Compared to the earlier works where only single layer offloading system was considered [10], our proposed HEC architecture is based on multi-level hierarchical architecture exploiting the edge and cloud computing paradigms. Additionally, HEC considers the varying impact of CPU clock speed on the transmission delay[3] which was not considered in the previous studies [11]. To the best of our knowledge, HEC is the first work to optimize the combined mobile-edge-cloud offloading policy and CPU/network speed scaling by considering the impact on network dynamics during code

[3]Time spent in offloading network traffic through network interfaces.

offloading process thereby saving mobile energy. This work is carried out with the strong motivation that if the computational speed at the edge or cloud setup is significantly faster than executing locally, it may be worth offloading the task while compromising on the energy and delay being spent.

This dissertation is mainly aimed at answering the following research questions.

(i) How to decide an optimal location of execution: local/edge/cloud?

(ii) How to optimize mobile CPU clock speed for local processing?

(iii) How to choose an optimal network interface: WiFi/Cellular?

(iv) How to optimize mobile CPU clock speed while offloading a workload to the edge?

Experiments were conducted using popular smartphones (*Samsung S8, OnePlus 2, Asus Zenofone*) under multiple environments to study the real measurements of (i) *the CPU energy impact under different CPU clock speeds and workloads* (ii) *data throughput values under various WiFi and cellular networks* and (iii) *the offloading energy and computational effects by using multiple edge and cloud configurations with different computational powers*. Based on the experimental values, the developed HEC algorithm was implemented on the Android platform mobile phones for the mobile end, a laptop serving as the edge device and the AWS EC2 instance serving as the cloud VM infrastructure forming the hierarchical architecture.

Key contributions of this research work are listed as follows.

- The proposed HEC algorithm is the first to study about the impact of mobile CPU clock speed on network transmission delay while offloading workload through network interfaces.
- Considering the energy-delay tradeoff, we propose a hierarchical architecture including Mobile-Edge-Cloud systems to efficiently schedule and handle the different set of workloads during the computation to address the limitations of mobile-cloud and mobile-edge architectures.
- HEC algorithm jointly considers the CPU clock speed, mobile battery power, and network interface selection to decide on the optimal computation location selection and CPU clock frequency thus resulting in nearly 47% reduction in energy consumption with a delay of just 12 min. on average.

1.3 Dissertation Outline

In the rest of this research work, we discuss the related work in Chapter 2. Chapter 3 details about the system model that serves as the base for this complete dissertation. The experimental setup for running the baseline tests is explained in Chapter 4. We evaluate and discuss the baseline results in Chapter 5. In Chapter 6, we design and evaluate the benefits of HEC algorithm and architecture by presenting the experimental results. Finally we conclude the research and possible future works in Chapter 7 and Chapter 8 respectively.

Chapter 2

Related Work

In this chapter, we study state of the art in detail related to this research work.

2.1 CPU clock speed and network selection

Various studies, e.g., [12, 13] explored the control of CPU clock speed in mobile devices along with the network selection in the device since the above two are the critical elements that affect the processing and network delay during any application processing or offloading.

2.1.1 Network Selection

Nicholson *et al.* [14] studied various methods of scanning and choosing the best access points in detail. Probing method to pick the best access point (AP) was used in this research to select the best network for the transmission. Additionally, this facilitates the sandbox check about the port restrictions and provides end to end security that is extensible for most of the architectures. Ra *et al.* [15] proposed an algorithm to effectively choose the best quality network between 3G and WiFi for the communication. The tail energy was not taken into consideration in this work whereas Shu *et al.* [13] studied the behavior and proposed a network model considering the tail energy. With the studies, it is evident that WiFi networks do not use tail energy to persist the communication channel, whereas cellular network makes use of the tail energy to avoid breaking it, which has a direct impact on the mobile battery power [13]. However, there is no precise definition about good or bad network strength, the industry leaders and experts define heuristic values based on the experimental results that may vary based upon the environments and the application that demands extensive usage of bandwidth [16]. Also, Nicholson *et al.* [17] observed that considering only the signal strength might not allow us to pick the

best AP. Choosing a valid AP at the shortest time would allow reducing the friction that mobile has to go through during the user mobility phase.

2.1.2 Energy Saving

Kwak *et al.* [12] studied a joint CPU clock speed control and network selection scheme to minimize the energy consumed by the CPU and the networking energy. However, the system model proposed by them did not consider the mobile cloud computing environments.

Further, Kwak *et al.* [11] proposed an efficient way to manage the CPU clock cycles by using a tradeoff mechanism for minimal delay tolerant applications in a cloud-only architecture. The study showed around 42% energy saving by efficiently controlling the network interface selection and controlling the CPU accordingly. Additionally, they also demonstrated that this joint optimization algorithm could be extended for use in case of real-time video streaming applications which can tolerate acceptable delays without impacting its performance drastically. The constant increase in mobile CPU clock speeds (maximum speed up to 2.8 GHz [18]) motivates to develop rich mobile applications providing the best user experience. With the increase in computation power at the mobile end, it directly impacts the mobile battery consumption in addition to the energy that is utilized by the network (4G/LTE/WiFi) resources. Their architecture only involved mobile-cloud resources and they considered the impact of CPU clock speed while offloading the computation to the cloud to be always constant under the given environment.

2.2 CPU Speed Scaling

Modern smartphones can operate at multiple CPU frequencies [Appendix A (iv)]. However, by default, the mobile devices operate at the high frequency to provide the rich user experience. CPU frequencies are configured through *CPU Governors* in Android and *interactive* is the default governor being used in mobile devices which sets the CPU clock speed to its maximum [19]. These governors work at the Android kernel level, and multiple governors work at different configurations [20]. *Userspace* is one particular governor that lets the user or an application to configure the clock speed dynamically which was used in this experimental dissertation setup which will be discussed later in Section 5. The apparent correlation is that when the CPU frequency decreases, the CPU usage increases correspondingly [19]. Additionally, at higher clock speeds, mobile device battery consumption is more which is so crucial in an energy constraint environment. Hu *et al.* [19] proposed an algorithm to dynamically control the CPU clock frequency

that samples the application process for 100 ms and makes a decision based on that to configure the CPU clock frequency at that instance. This level of controlling the CPU clock speed based on the application process density is crucial in this dissertation since this algorithm is not bound to any specific application and it is dynamic enough to be applied across different resource-intensive applications.

Controlling the CPU clock speed is mainly based on the DVFS (Dynamic Voltage and Frequency Scaling) and DPM (Dynamic Power Management) principles which allow us to tune the CPU clock speed and as well the number of CPU core usage in the mobile device [20]. We can take advantage of the various states of wireless network interfaces into consideration while tuning the CPU frequency. With the LTE wireless networks having four different interfaces such as *idle, promotion, data transmission, and tail* not every time we would need maximum frequency. In contrast to the WiFi network interface, the cellular network interface tail state is maintained for a period even after the data transmission was completed and this has an impact on the battery power of the mobile device. At this tail state, lower CPU frequency can reduce the load on the battery consumption of the device by saving the energy during download operation by altering the frequency [19]. However, they did not consider the current battery level of the mobile device which suffers a direct impact by tuning the CPU clock frequency frequently.

Additionally, various power management schemes namely *HotPlug*, *OnDemand* and *application-aware* were discussed by altering the CPU clock frequency [21]. However, the tuning approach was only based on the CPU utilization at the mobile end. Further, in Section 5.3 we observe that it may not be accurate that the CPU utilization metric would serve as the only criteria all the time depending on the application nature for altering the frequency values. Studies have proved that changing the CPU clock frequency at more frequent intervals results in poor energy management at the mobile end [19]. Volker seeker results [22] discusses that the energy spent on finishing a task early and staying idle until its deadline is more when compared to finishing a task at a slower pace closer to its deadline. Finding a stable CPU frequency to complete any process at the mobile end is turning out to be crucial to the rising trend in demand of computational power and energy.

2.3 Computation Offloading

Computational offloading in its core offloads the application execution logic partly or as a whole to the remote server for further execution, and the results are collected and displayed back at the mobile device with the primary aim of reducing the processing delay and to minimize the impact on the mobile battery power [23].

Chen *et al.* [24] discussed an opportunistic approach for offloading the computation in detail. Their approach considers offloading to the cloudlet only based on the application density at the given time along with the network choice. At a high level, the offloading decision can choose to execute the task locally or to offload the task either partially or completely [25]. To decide about full or partial offloading, various factors (static and dynamic) needs to be considered and it grows more complex as the scenario changes with every user and application based on its processing density [11]. Primary factors that would influence the decision would be the application nature, knowledge about the data to be processed and the dependency on the offloadable parts [25]. With the computation decision being the essence of this dissertation, in this section, we discuss various resources in this research area aimed at optimizing the choices across different computing paradigms. Here we consider only the full offloading decision and will not be covering the partial offloading since it is not under the scope of this research.

Liu *et al.* [26] aimed at minimizing the processing delay by considering the job queue and the energy at the mobile devices. Their optimal algorithm showed convincing results while compared with local execution delay, remote execution policy, and a greedy algorithm policy. Also, the research works [27, 28] pursued the extension of [26] using Lyapunov optimization and was able to observe 74% reduction in execution time. However, all the above works did not consider the dynamic nature of the mobile devices based on the battery power and the network strength variability. The energy loss on the mobile side for processing and offloading the task was not considered crucially in their algorithms.

Covering the setbacks in the above-discussed work, Kamoun *et al.* [29] proposed an optimization algorithm considering the energy savings during the computation offloading. The author suggested a deterministic and randomized offline strategy for offloading where the knowledge of the incoming application is known earlier, and it showed convincing results of nearly 80% in energy saving for offloading tasks. Additionally, the research works [30, 31, 32] enhanced the similar idea by improving the energy savings at the mobile end during the offloading process by considering a multi-user environment. However, the above work did not exploit the capability of tuning the CPU clock speed of the mobile device that can support in both local processing and during the offloading process from the mobile to the edge or remote cloud endpoints.

Guo *et al.* [33] discussed various methods to optimize the way how a task is handled at the edge and the cloud. Their work primarily focused on maximizing the amount of work performed at the edge and offloading to the cloud only when there is insufficient computational power at the edge to perform specific heavy computations. Other research studies, e.g., [34, 35] enhanced on the similar interest by considering the energy optimization at the edge devices and also to efficiently schedule the jobs across all the tiers.

2.4 Mobile Cloud Computing

Mobile Cloud Computing (MCC) platform is expected to generate revenue excess of $3.6 billion in the next four years [36]. Various factors need to be considered before offloading the task such as the network strength and the battery power. Inefficient decisions may result in poor performance for the user. Considering the dynamic factors effectively during the offloading decision is critical for the MCC technology to yield maximum benefits.

Recent trends in mobile application development are leveraging the processing power of the cloud resources to execute a task. Some of the resource-intensive mobile applications are offloaded to the cloud resources with higher configurations that are designed to process extremely resource-rich applications. However, the round trip networking delay from the mobile end to the cloud resource becomes a bottleneck if the network connectivity is bad between them which would cause the task completion to delay further. Also, energy consumption during this transmission is a critical factor to be observed. With cloud computing, even small applications can experience higher latency due to WAN delays [37]. Additionally, the cost of using cloud resources should be considered during the offloading process in place of using local CPU resources at no additional cost. As with every technology, various challenges and issues in using MCC were discussed by Ruay *et al.* [38] primarily on the aspects of latency, maintenance, and cost.

Many popular frameworks such as MAUI [7], Clone cloud [9], ThinkAir [10] and CDroid [39] uses MCC architecture and focus on more sophisticated algorithms to minimize the task computation time at the mobile considering many factors. Kwak *et al.* [11] formulated a matured energy trade-off algorithm to make the offloading decision based on various factors dynamically efficiently but leveraging the MCC architecture. Significant power savings were demonstrated with the MCC architecture by efficiently handling the CPU clock speed and the network interface selection to maximize the energy saving at the mobile end.

Though MCC provides lots of benefits, it was not expected to solve every problem in the communication domain with the primary factor being its high latency. Offloading the computation to the public cloud involves long latency during the exchange of data between them [40].

2.5 Mobile Edge Computing

Mobile Edge Computing (MEC) was introduced to bring the computation elements closer to the network edge. This enables a platform for offloading the resource-intensive

applications from the mobile devices to the edge for reducing networking latency. Additionally, the processing time can be still minimized due to its proximity nature thus providing significant benefits concerning time savings and energy.

Cloudlet architecture [23] is advantageous for non-trivial applications for the prime reason of close physical proximity, high bandwidth wireless access, and one-hop network latency. Trusted cloudlets/edge devices can be a single or cluster of resource-rich computers while compared to mobile processing power and are available for the nearby mobile devices. With recent developments in 5G technology where the core focus is to bring down the latency, 5G MEC (Multi-Access Edge computing) [25] is expected to bring down the average latency to 90% less than the average LTE network latency which could be a humongous advantage for resource-intensive life-saving applications.

Chen *et al.* [41] formulated the offloading decision in MEC environment among multiple users or tasks as a multi-user computation offloading game. Since the mobile usage and the mobility vary continuously among the users, the game theory approach suggested here serves as a promising framework for making an efficient offloading decision. Results from the experiments showed that the Nash equilibrium is maintained across the game by improving the offloading decision. Ha *et al.* [24] researched various real-time applications that are time sensitive and demonstrated nearly 80% reduction in time savings and 30-40% saving in terms of energy by exploiting the edge computing architecture.

Chapter 3

System Model

This section details the system model employed in this research work. We model the mobile end with an Android-powered smartphone and the edge component with a Laptop. AWS EC2 instance is modeled as the cloud component.

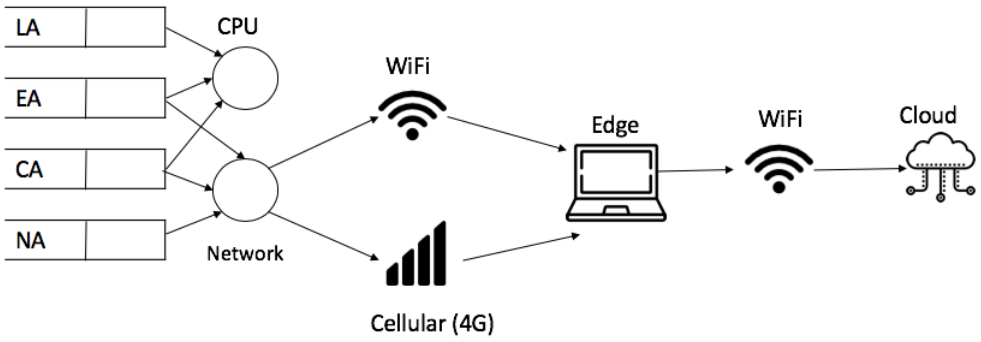

Figure 3.1: Framework for hierarchical mobile-edge-cloud code offloading.

(Proposed hierarchical system model involving mobile, edge and cloud components along with workload queuing model.)

3.1 Task Model

In this research work, the mobile phone tasks are classified into four different types such as local processing LA (non-offloadable) workloads, edge-offloadable (EA) workloads, cloud-offloadable (CA) workloads and legacy network traffic (NA) workloads. Offloadable workloads (OA) often refers to the combination EA and CA workloads as depicted in Figure 3.1

LA workloads can exploit only the mobile CPU resources since they are non-offload workloads. Whereas OA (EA and CA) workloads can make use of either the local CPU

resources or the network resources. OA resources utilize the network resources through the WiFi or cellular network interface during the offloading process to edge or cloud components[1]. We model all the tasks to be time bounded, such that at any given time slot t, the workload is completed before its maximum allowed time.

3.2 Processing Model

Throughout this research work, we consider only a single CPU core in the smartphones. The mobile phones used in the experiments has the latest configuration, and its processors CPU clock frequency can be tuned, which exhibits the capability of Dynamic Voltage Frequency Scaling (DVFS). Given the nature of various applications, each mobile workload requires different CPU clock cycles in processing the same amount data in bits which are influenced by the workload characteristics. This notion is denoted by *processing density* in the unit of cycles/bit. At any given time t, only either LA or OA workloads can exploit the local CPU resources given the consideration of single core CPU. The CPU clock speed influences LA workloads since it can exploit only the mobile CPU resources. Adding novelty to our work, we consider the impact of the mobile CPU clock speeds while processing the EA, CA and NA workloads using the network interfaces.

3.3 Computation Model

We model a hierarchical architecture to execute the computations at three different architectural entities namely on the local mobile device, edge component, and the cloud data center. We model the environment by installing the same computational code across all the three tiers. The application code remains the same across the tiers to achieve the same code performance without any difference in the application behavior.

$$\text{Computational power comparison: } P_m < P_e << P_c$$

Considering the computational capabilities, we assume that edge computational power (P_e) is always better when compared to the mobile processing power (P_m). Moreover, cloud data center processing power (P_c) is much higher than the processing power of the edge devices. It is safe to assume that cloud has infinite processing speed given the possibilities to exploit the cloud technologies. For instance, Lambda functions in AWS is one such example where AWS uses serverless computing enabling the auto-scaling feature. This eliminates the need for maintaining fixed computational power manually [42].

[1]Multi-homing technology is not configured in the smartphone to avoid transmitting data through cellular and WiFi networks simultaneously.

3.4 Networking and Energy Model

We configure the smartphone to have cellular connectivity, and we assume that the connectivity remains active all the time and WiFi connectivity is active at irregular intervals based on the environment. In our model, NA workloads can fully exploit the network resources whereas OA workloads can choose to use the network resources to offload the computation to the edge devices. Hence we model three different network interface states namely Cellular Network (C), WiFi (W) and No Network (N). A particular task (NA or OA) can choose to use any of these different states to complete its computation thus yielding results. Both the C and W interface speed varies spatiotemporally based on the network coverage and access points. At any given point in time, the smartphone can choose between {C, W, N} or {C, N} interfaces only because of the WiFi irregularity nature.

To obtain the realistic CPU parameters, we observe the energy consumption of CPU under various workloads in the Section 5. Also, we consider the energy consumed by the cellular and WiFi network interfaces during the process. This enables us to model the algorithm further to efficiently choose the optimum interface based on the correlation observed here.

Chapter 4

Baseline Experiment Setup

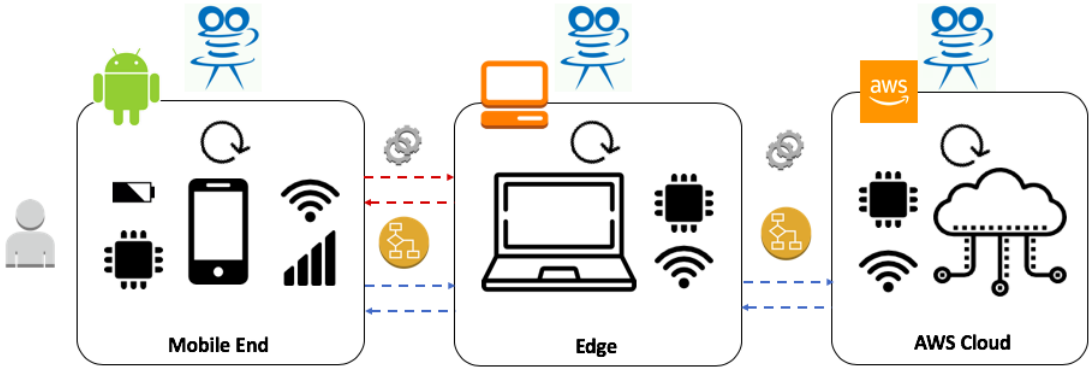

Figure 4.1: High-level architecture.

(Architecture design to execute baseline experiments across mobile, edge (laptop) and cloud (AWS) components.)

The experimental setup was planned with careful consideration of design choices to avoid any bias in architecture behavior.

4.1 Mobile Setup

The mobile phone being the critical component in the architecture where the workload is originally generated, the latest smartphones are chosen to conduct the series of experiments to observe the baseline values for the algorithm development. We used *Samsung S8, OnePlus 2* and *Asus Zenofone (X008DA)* smartphones with the latest Android versions (8.0.0, 7.0.0) during the experiments to study the behavior in various smartphone vendor configurations.

Mobile phones are rooted[1] in the experiment which enables to control the CPU clock

[1]Allows the user to configure the Android OS kernel-level configurations.

frequency based on our preference. This is possible in accordance with DVFS - the power and performance management technique inbuilt in most of the latest smartphones [27]. For exploiting the native Android OS kernel interface, we use TWRP package [43] that provides a shell interface to interact with native kernel commands. All the commands are executed as a *superuser* of the smartphone which provides complete control over the smartphone configurable parameters, and the developer options were enabled actively on the smartphones[2]. To carry out our experiments, we developed an Android native mobile application from scratch with Java as the primary programming language to monitor and configure the smartphone parameters during the experiments. Source code for the developed application is maintained in GitHub repository [44].

4.2 Network

All the smartphones are enabled with mobile internet using the active data package from network operator *Three*[3]. Also considering the varying spatiotemporal nature, smartphones are configured to connect to TCDwifi[4] (University Provided) and a private WiFi network (Virgin Media[5]) automatically. Average upload and download speeds are captured during every experiment. The outlier values observed during the experiments due to the unexpected network behavior are continuously monitored to avoid bias in the results. We perform the Boxplot analysis [45] to capture the distribution of values and to analyze the outliers.

4.3 Encoding Application

For this research work, we choose JCodec [46], a video encoding application whose tasks are modeled to be the primary workload in the experiments. We configure various sets of audio and video codec encoding operations as the workload to the mobile device. Since the processing density of a video encoding application varies between 200 to 1200 (cycles/bit), it enables to study the behavior of various mobile parameters at different configurations [11]. With careful attention in scheduling the workloads, the application was well researched before using them in the real experiments to avoid bias in the algorithm behavior. For instance, if the processing delay of an application is always higher or lower than the transmission delay, then the algorithm can behave biased and will result in

[2]http://www.vogella.com/tutorials/AndroidTools/article.html
[3]https://www.three.ie/
[4]https://www.tcd.ie/itservices/network/kb/wi-fi-coverage.php
[5]https://www.virginmedia.ie/

processing all the workloads locally (only in mobile), or all the workloads will be offloaded directly to the edge component. This behavior will defeat the purpose of this research work which is more focused on energy-delay savings by exploiting the hierarchical architecture with its opportunistic decisions. The observed correlation between the processing and networking delay for the JCodec application is detailed in the Section 5. Additionally, JCodec library supports the native Java and Android platform software development with platform agnostic Java classes. This enables us to implement the same application code logic across both the platforms making it portable across various environment settings.

4.3.1 JCodec - Mobile

We integrate the JCodec android library with the developed Android application to support video encoding. The frame count and the video file to be encoded are maintained as configurable dynamic parameters which allow us to reuse the same source code for running different sets of experiments under various loads. Inline with Android application development guidelines [47], Android threading model is used to handle the long-running process as a background process. The developed application requires access to the local storage to handle the video encoding operation, and full permissions to the network interface for workload offloading purpose[6].

4.3.2 JCodec - Edge/Cloud

To accommodate the application load at the edge and cloud components, a SpringBoot application[7] using Java is developed, and multiple endpoints are configured for running various experiments. The application is made portable by creating a flat executable jar, which can be shipped to the edge and cloud component similar to the code-clone technique [9] which are popular in mobile/pervasive computing.

JCodec native Java library is integrated with the developed application that performs the same operation as implemented at the mobile side since this library provides a consistent behavior of the application across different platforms. During the execution time, we configure the heap size of the Java Virtual Machine (JVM)[8] to accommodate large file uploads and downloads. Maximum heap size is configured using Java executable commands[9] and we configure to use the 64-bit processor during its process execution. These tuning parameters are configured to reflect during application startup at the edge and

[6]Android application prompts for permission approval during its initial startup.
[7]https://spring.io/projects/spring-boot
[8]Java Virtual Machine provides -Xms and -Xmx to limit the Heap memory usage.
[9]java -Xmx6144M -d64 -jar video-encoding.jar

cloud components. Performance impact and the memory footprint of SpringBoot applications are detailed in [48]. Source code for the developed application is maintained in the GitHub repository [49].

4.4 Edge Setup

Edge component in this Hybrid architecture is modeled using a MacBook Air laptop with the latest configuration. Configuration details of the edge component are listed in Appendix A (ii). We install Java in the edge and cloud components to allow installation of our developed application. The developed Java application integrated with JCodec (Java) is installed as a usual Java binary. This edge device is configured to connect to the WiFi network interface. Other system applications were suspended during the experiments to avoid impact on the application performance. Additionally, throughout this research work, we assume that the edge and cloud components are powered with infinite energy and the energy consumption at both edge and cloud is not studied in any measurements.

4.5 Cloud Setup

Many vendor solutions are providing various IaaS/PaaS resources in the marketplace namely Amazon Web Services (AWS), Microsoft Azure, Google Cloud Engine (GCE) and OpenStack. We choose AWS as the cloud component for this experiment considering its latest efficient rollouts and effective/flexible pricing model[10]. Two EC2 (Elastic Cloud Compute) instances with different configurations (Appendix A (iii))) are provisioned in the AWS. With Unix platform and the Java resources installed, the Cloud configuration resembles the edge component experiment setup apart from the fact that the computational speed and resources are exponentially high at the cloud end. Provisioned cloud resources are charged based on the type of the configuration and the usage. All the EC2 instances are allocated with a static IP address that can be used by the edge components during the offloading action. The developed Java executable binary is installed at the cloud side with the similar heap size tuning to serve high load requests.

4.6 Delay and Energy Measurement

Source code [49] includes the logic to capture the time spent in each operation across all the tiers. We capture the *Processing delay* - time delay for the processing the workload and

[10]https://aws.amazon.com/ec2/pricing/

the *Transmission delay* - time taken to offload the traffic between mobile and edge/cloud components separately to study the time spent on each section. All the measurements were calculated in seconds (secs) or minutes (min) for the ease of understanding and consistency. Also, the energy spent at the mobile end was captured using various methodologies.

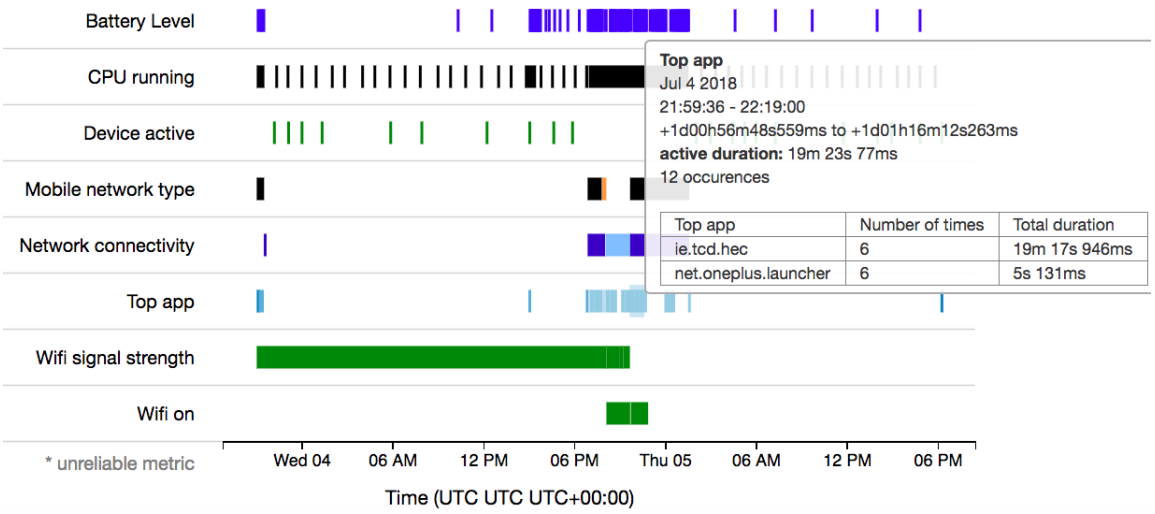

Figure 4.2: Battery Historian Interface.

(Battery historian tool interface to visualize the values such as network strength, battery power and WiFi status at different time slots for every application thread.)

(i) *Battery Historian* [50] is used along with the Batterystats to analyze the report generated by the Android device. As shown in Figure 4.2, this helps in observing the battery data at any given point of time, battery discharge rate, WiFi connectivity strength and the specific application that used the battery power during that time. Detailed information paves an additional way to capture the accurate results during the experiments.

(ii) *Accu Battery* [51], one of the widespread application to monitor the battery resources was used to measure the experimental results. This application provides segment-wise values of the battery being consumed even while the device is in sleep in addition to application wise energy consumption as depicted in Figure 4.3. We aggregate the results from both these methods to arrive at substantial numbers to depict the results.

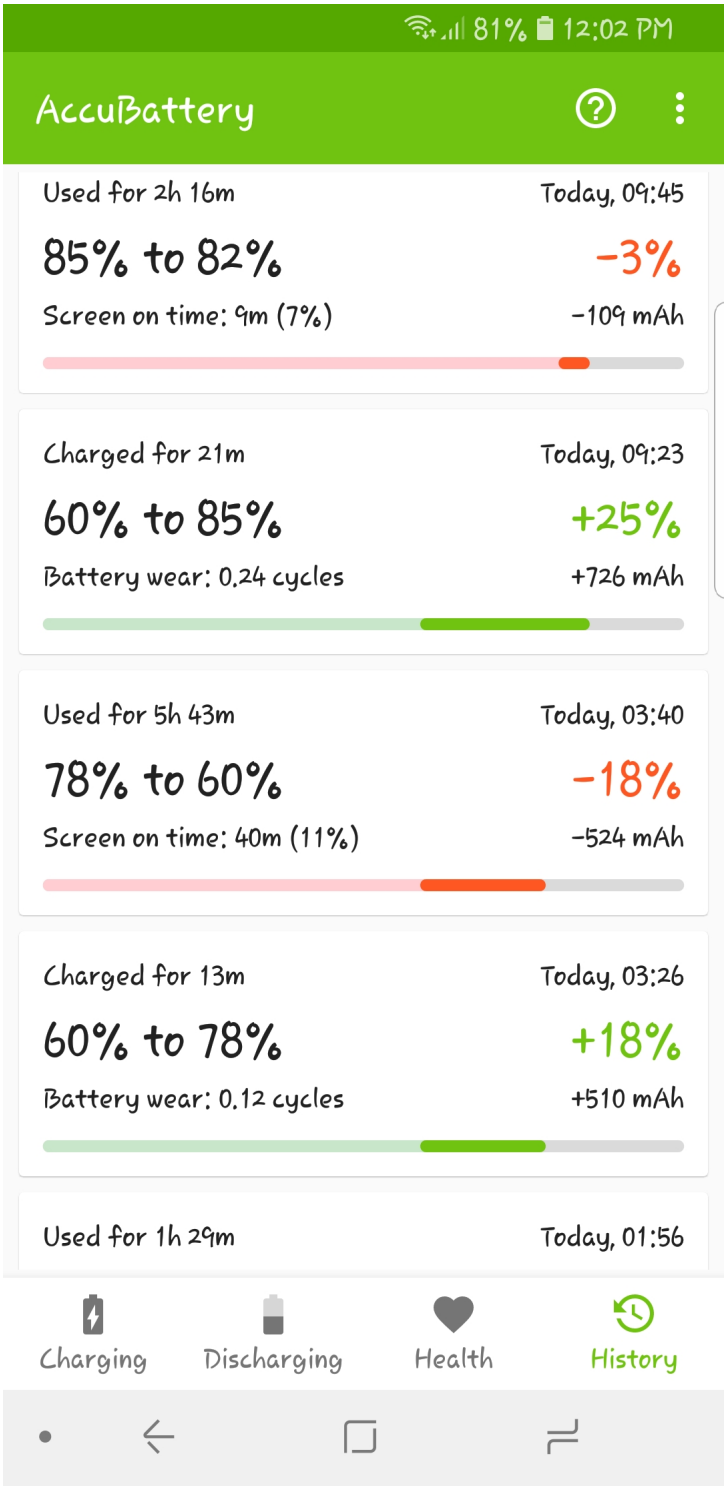

Figure 4.3: AccuBattery mobile application interface.

(AccuBattery mobile application interface showing various mobile battery values including the amount of battery discharge for a specific duration.)

Chapter 5

Baseline Results

This section discusses the summary of critical observations from the experiments. All the measurements discussed here are the average of minimum three to five test executions. Youtube dataset[1] video files were used during the encoding operations. To study the correlation between various parameters, the workload is scheduled at fixed intervals on the mobile with varying mobile CPU clock frequencies. We vary the load continuously, and the WiFi network is turned ON and OFF intermittently to analyze the network impact during the offloading process.

5.1 Load vs. Processing Time and Energy

Figure 5.1 depicts the impact of varying load on the processing time and the mobile energy consumption in different smartphones. The description of Figure 5.1(a) implies that as the workload density increases, the processing time in the mobile increases linearly. The difference in processing times at different smartphones are due to the fact that all smartphones have a variety of CPU clock speed. Impact of CPU clock speed on processing time is studied in Section 5.3. Additionally, Figure 5.1(b) depicts the impact on the battery power with varying processing time. The tight correlation between battery power and the processing time can be observed here where the battery consumption is more as the processing time increases. Since mobile CPU resources will be utilized for a longer duration, the energy dissipation is more as the processing duration increases.

[1]https://research.google.com/youtube8m/download.html

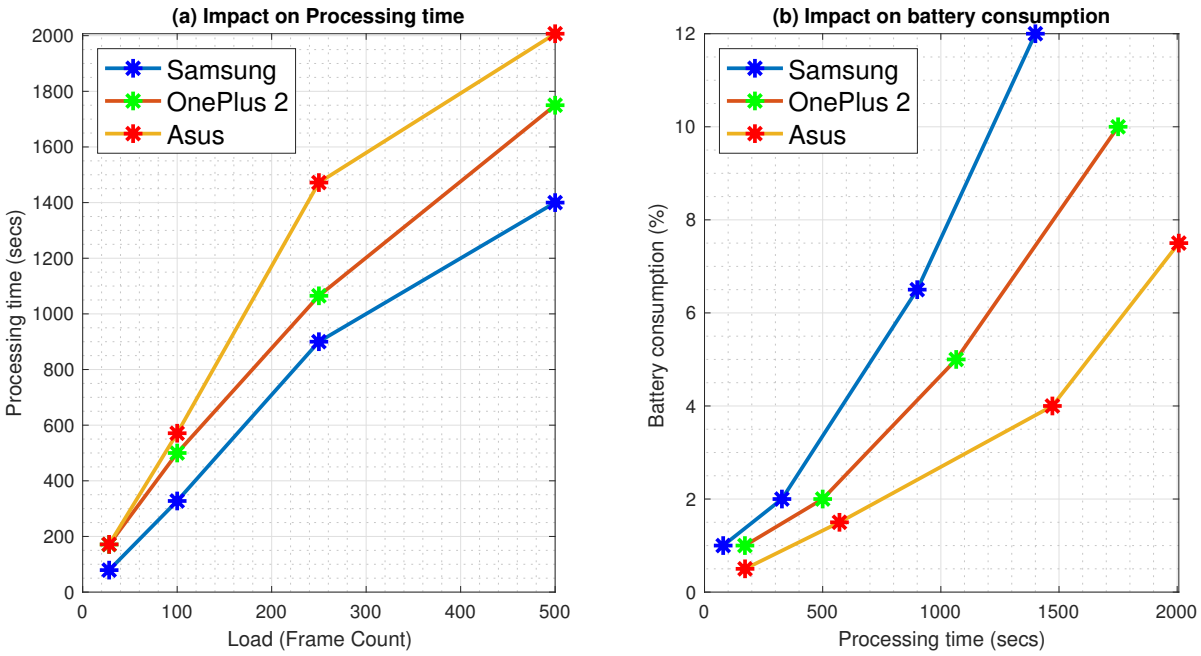

Figure 5.1: Comparison of processing time and energy consumption in different smartphones.

((a) Impact of varying load on the processing time.(b) Battery consumption impact due to the application processing time.)

5.2 Network Delay vs Energy

We conduct a series of experiments to analyze the impact of networking delay on the mobile energy consumption for the traffic offloading process. Various workload combinations were executed namely EA only, EA and CA combined and CA only workloads. The time spent in offloading the workload to the edge and cloud components and retrieving the computation results back on the mobile device is calculated as the networking delay or the transmission delay. The experimental results for varying loads are depicted in the Figure 5.2(a) that describes the transmission delay increases proportionally as the load is increased with an additional impact on the battery consumption.

Moreover, considering both Cellular and WiFi network interfaces, the impact of network selection on energy consumption is shown in 5.2(a). Two different WiFi networks are considered as discussed in the Section 4 and for the cellular network, a 4G connection is being used. As the WiFi coverage increases, the energy consumed by the NA and offloaded EA and CA workloads consume much lesser energy while compared with the energy spent on processing the same load using the cellular interface. As studied by Niranjan *et al.* [52], WiFi interface is more energy efficient since it does not consume the tail energy that the cellular interfaces make use of to persist the connectivity and

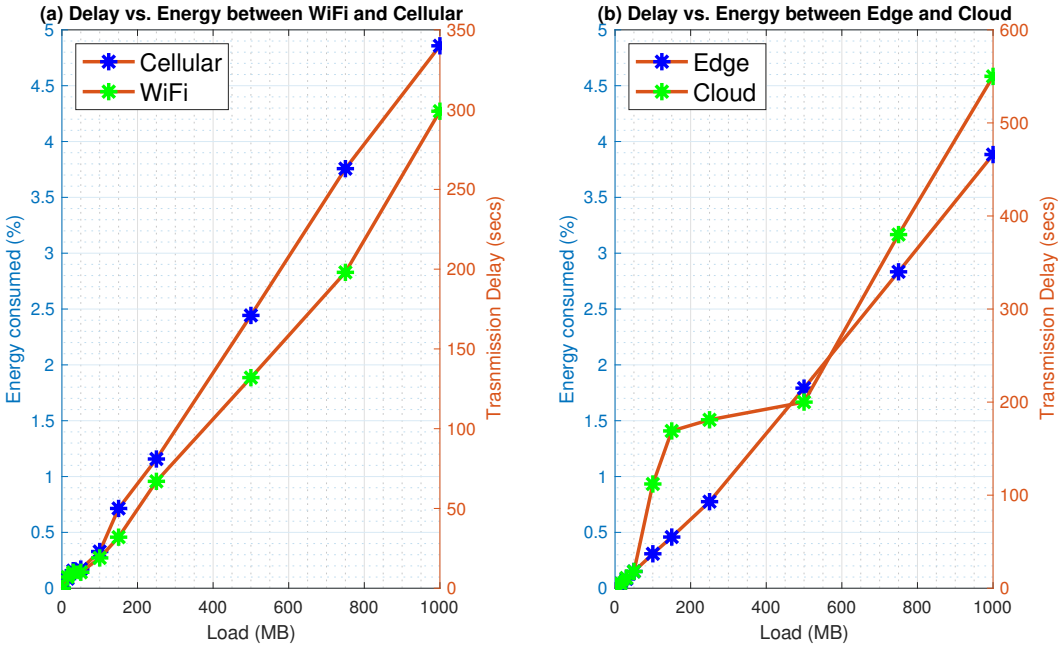

Figure 5.2: Impact of Transmission delay on energy consumption.

((a) Impact of network interface on energy consumption and transmission delay. (b) Impact of execution tier on energy and transmission delay with varying load.)

also because of the short coverage area for the WiFi service. It was observed that nearly 60% of the energy is wasted as the tail energy during the high-power state which has a direct impact on the energy being consumed by the mobile device [2]. For brevity, in this research, the impact of tail energy is not studied separately. Additionally, Figure 5.2(b) describes the variation in transmission delay between the edge and cloud components. The total round trip time between the mobile and edge is lesser when compared to the values between the mobile and cloud server. This is because of the proximity of the edge server and the variation in latency of the cloud servers.

5.3 CPU Impact on Processing and Networking Delay

Here we study the impact of mobile CPU clock speed on processing and networking delay. As depicted in Figure 5.3, we run the various workloads {LA, EA, CA} in their respective tiers. Figure 5.3(a) depicts the LA workload processing time variations under different mobile CPU frequencies. Processing time being referred here denotes the time spent only for processing the workload excluding the transmission delay. It is evident that there is a strong correlation between the CPU clock speed and the processing delay that an application can experience in a mobile device. As a result, we can observe that the higher the CPU clock frequency, the processing duration becomes smaller with varying

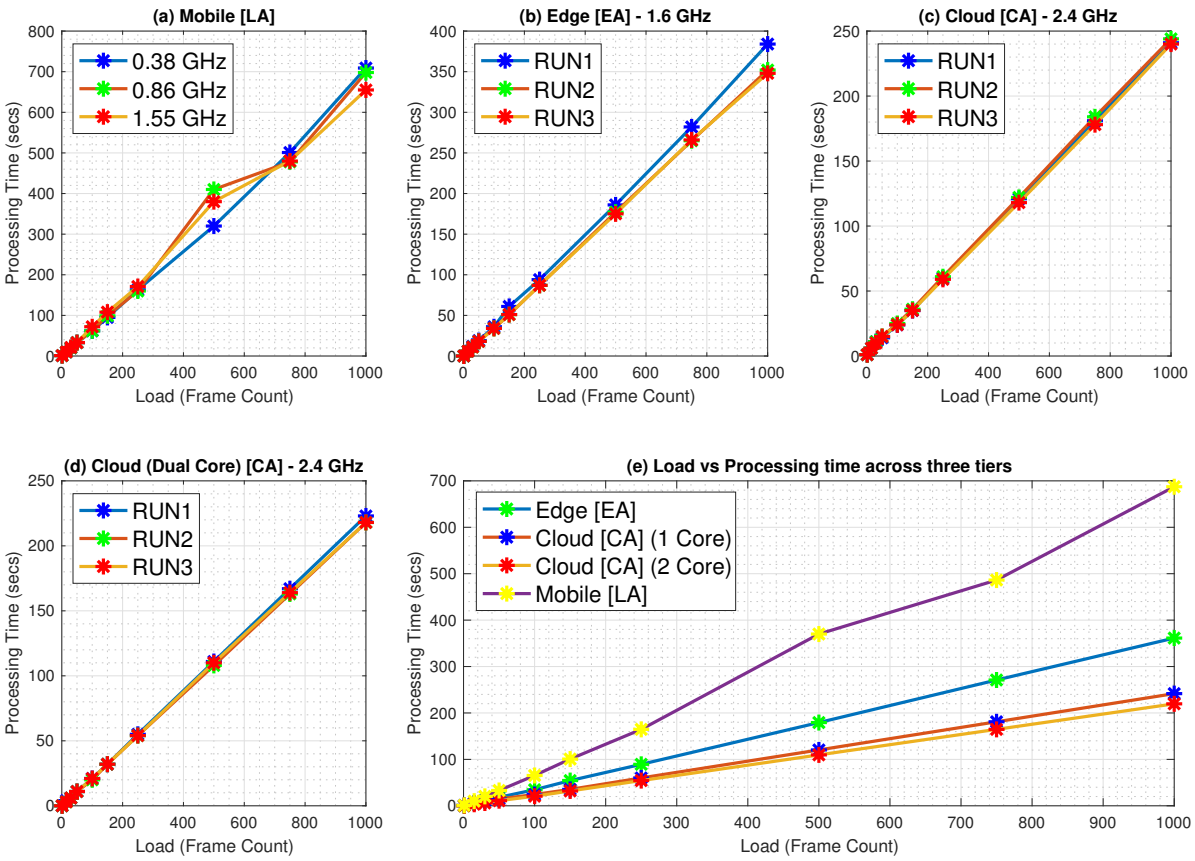

Figure 5.3: Processing time comparison for LA, EA, CA workloads across tiers.

((a) LA workload processing time for different mobile CPU clock frequencies.
(b) EA workload processing time at edge component for three test executions.
(c,d) CA workload processing time at cloud component for three test executions.
(e) Comparison of processing time at the maximum CPU clock cycles across all the tiers.)

load. Aligning with our research motivation, this observation enables us to control the CPU clock frequency within the mobile device to achieve delay reduction during process computation.

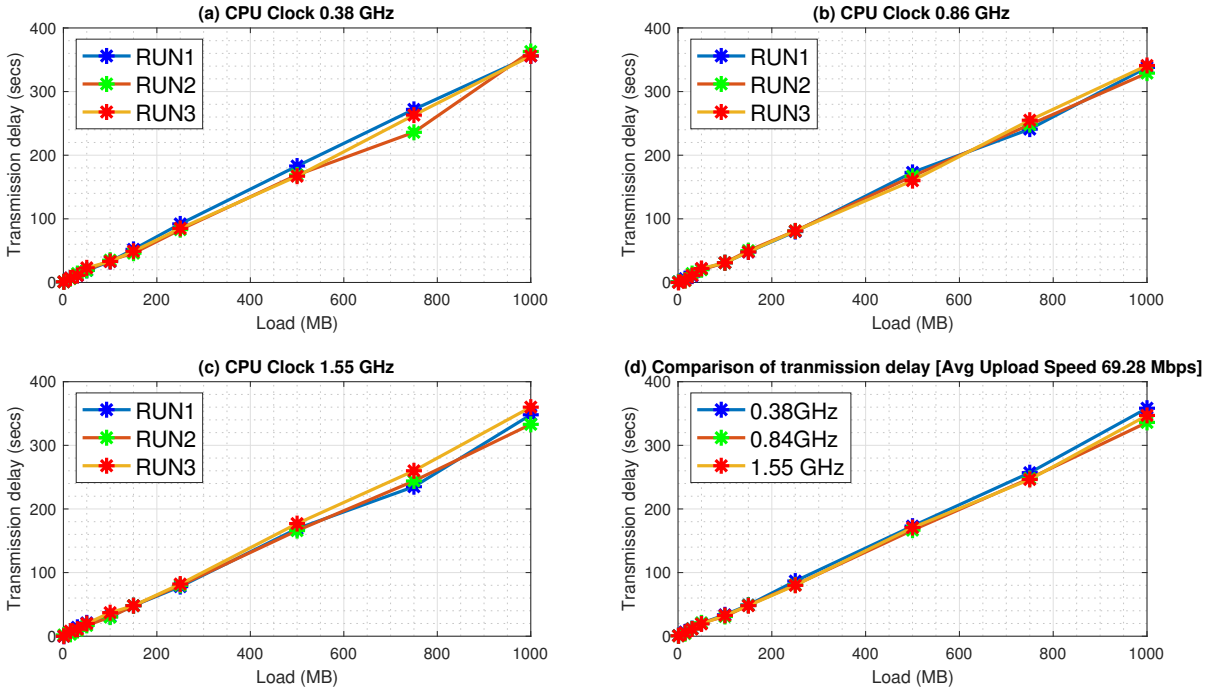

Figure 5.4: Impact of mobile CPU clock frequency on Transmission delay.

((a,b,c) Transmission delay during multiple runs for various mobile CPU clock frequencies. (d) Comparison of Transmission delay between various CPU clock frequencies.)

Additionally, Figure 5.3(b,c,d) describes the processing time across the edge and cloud components for multiple test executions. We also study the processing time experienced by CA workloads at single and dual-core cloud virtual machines[2]. We observe the processor power across the tiers, and it is safe to conclude that better the CPU clock speed at edge and cloud components, the processing time can be reduced further by offloading. The consolidated plot comparing the difference in processing time across all the three tiers is depicted in Figure 5.3(e). The LA workloads consume the maximum of the processing speed given its limited CPU frequency while compared to the EA and CA workloads where processing happens at the edge and cloud components with higher CPU clock frequency. Even though we observe that CA workloads consume minimal processing delay, as discussed in section 5.2, the latency factor between the cloud components introduces higher transmission delay while compared with the edge components. This demands an efficient method of trading off between energy and delay factors to achieve better efficiency.

[2]Negligible differences were observed between single and dual-core virtual machine during the experiments; hence we consider only single core virtual machine during rest of the experiments for brevity.

A crucial observation from this research work is depicted in Figure 5.4 where we study the impact of mobile CPU clock frequency on the transmission delay. We transmit files of varying sizes at fixed intervals during the experiment. This study demonstrates that as a function of mobile CPU clock frequency, the transmission delay of the offloadable workloads (EA, CA and NA) is impacted. To the best of our knowledge, we are the first to study about this impact of mobile CPU clock frequency on the network dynamics. Figure 5.4(a) describes the variation in transmission delay at the lowest CPU clock frequency at the mobile, whereas Figure 5.4 (c) depicts the variation at a higher frequency. As shown in Figure 5.4(d), higher CPU clock frequency minimizes the transmission delay.

Moreover, this impact is very active concerning the average load that gets offloaded across the tiers. For very low EA, CA, NA workloads, the impact of the CPU tuning is negligible regarding delay savings. Additionally, at very high loads, energy savings are negligible when the frequency gets altered. Nominal load sizes benefits hugely with this CPU frequency impact concerning both energy and delay savings. Therefore, this result observation adds a significant and unique contribution to this research work.

5.4 End to End Delay as a Function of Time

With the previous experiments studying about the impact of CPU and delay in terms of varying loads, Figure 5.5 shows the impact of the end to end delay as a function of CPU frequency under fixed load since it is essential to study the behavior of delay in the temporal space given the mobility of smartphones. With variation in time, the end to end delay across three level of CPU frequencies varies consistently. It is evident that the network interface strength is continuously changing with the spatiotemporal relation impacting the transmission delay. Outlier values were observed during the experiment, depicting a real-time environment setup where the coverage of the network interface may not be consistent always.

The description of Figure 5.5 (a,c,e) implies that the processing delay remains close to constant at various time intervals during the process. However, the transmission delay varies consistently at different time slots impacting the total end to end delay. Also, the box plots represented in Figure 5.5 (c, f, i) reflects on the range of values for processing and transmission delay and its mean values.

Figure 5.6 represents the variation, in end-to-end delay at different time intervals across all the three tiers across the different set of CPU clock frequencies. In the graph, it can be observed that the total end to end delay while processing only in the mobile is consistently lesser when compared to the delay at the other two tiers combined. Additionally, depending on the location of cloud VM resource connectivity, the latency between

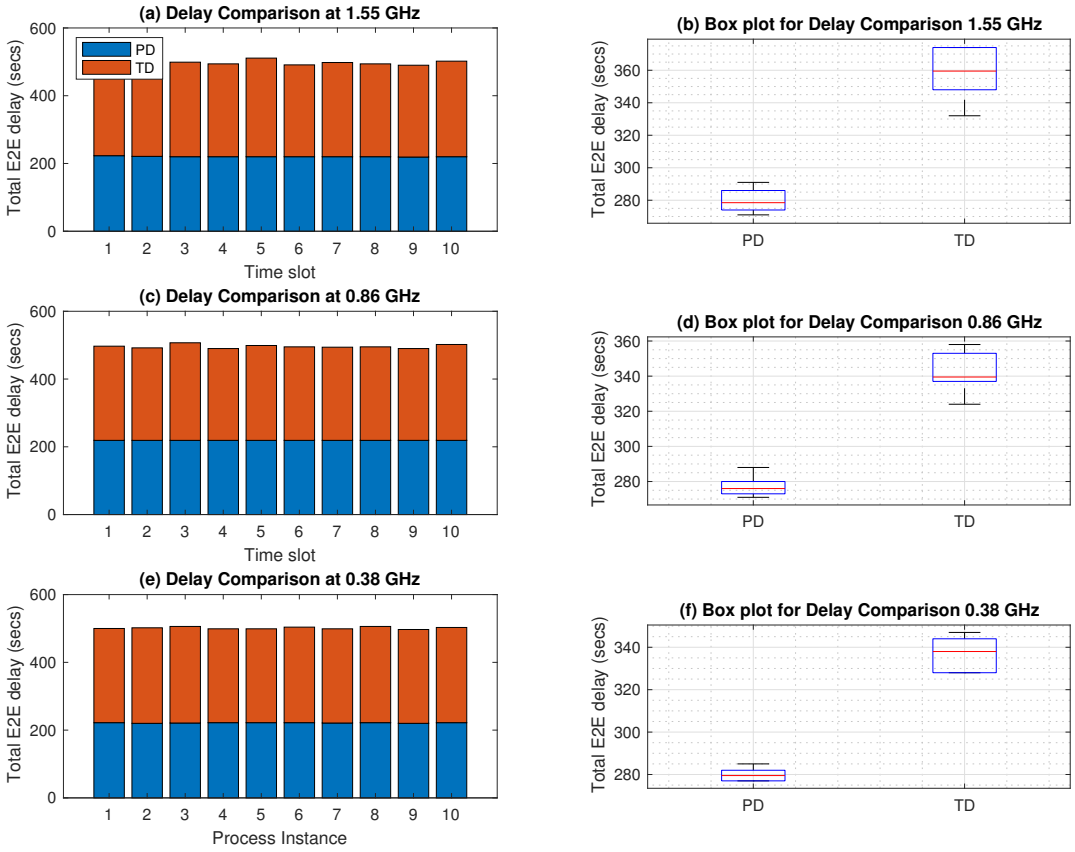

Figure 5.5: Impact on End to end delay based on time variations.

((a,c,e) Total end to end delay at various time slots for different CPU clock frequencies. (b,d,f) Boxplot depicting the mean and outlier values for processing and transmission delay; PD-Processing Delay, TD-Transmission Delay.)

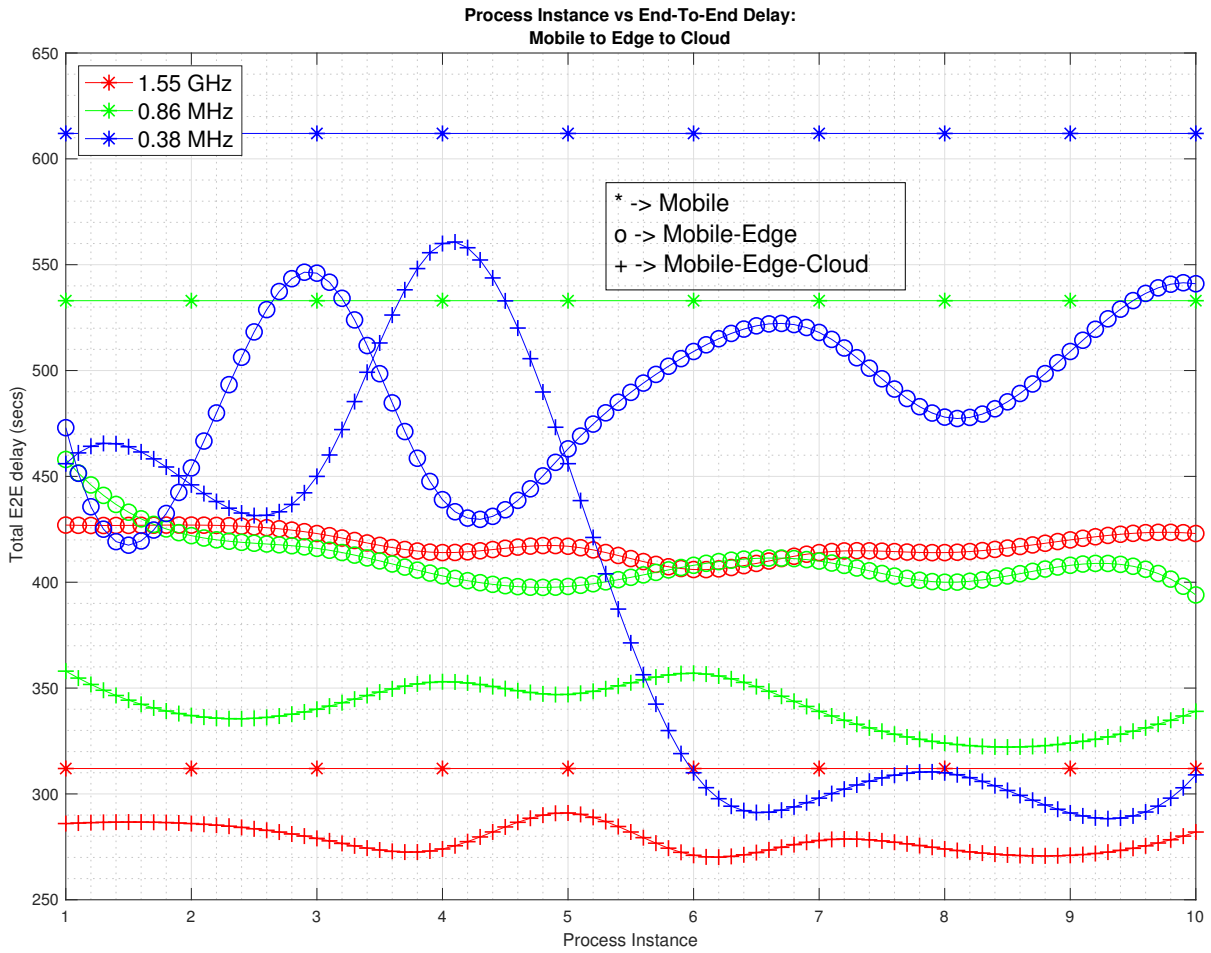

Figure 5.6: End to end delay as a function of fixed load for different types workloads.

(Comparison of end to end delay across various tiers and different CPU clock frequencies at different time slots.)

the edge and cloud shows some impact on the networking delay as discussed in the Section 5.2. This is one of the driving factors for this research work because offloading the workloads directly to the cloud in all cases may not be an optimal solution to maximize the energy savings.

Despite the impact of time variation on processing delay on the edge and cloud, the local processing (PA workloads) is not sensitive to time since the processing is entirely independent about the network coverage or the network interface selection. The processing delay at one particular mobile CPU frequency instance is consistent irrespective of time the experiment was executed. However, it is notable that the total end to end delay at the Mobile-Edge-Cloud architecture is comparable with the local processing delay at some instance of the experiment.

Chapter 6

Algorithm Design and Implementation

In this section, we develop and discuss an optimization function with the strong motivation of exploiting the Hybrid architecture by minimizing the energy consumption and the end to end delay. The algorithm design to address the research question is supported by the baseline results discussed in Section 5.

6.1 Optimization Function

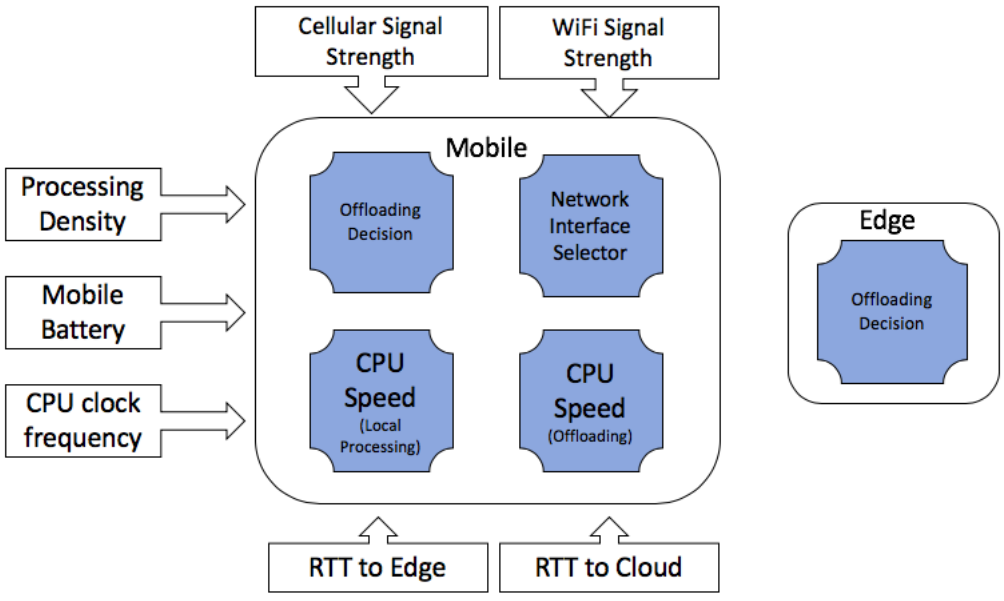

Figure 6.1: Control Knobs.

(Five control knobs in the architecture to tune performance.)

With the five control knobs as referred to in Figure 6.1, the optimization function aims to address the below decisions:

(i) *Offloading Decision (Mobile)*: Choose an optimal location to execute the process considering the process density and the energy throughput values.

(ii) *CPU Speed (Local Processing)*: Minimize the energy spent in processing LA and local EA, CA workloads by configuring with optimal CPU frequency.

(iii) *Network Interface Selector*: For offloadable EA, CA and NA workloads, choose the best (energy-delay efficient) network interface to offload the traffic.

(iv) *CPU Speed (Offloading)*: Configure the optimal CPU clock frequency during workload offloading thus minimizing the energy consumption.

(v) *Offloading Decision (Edge)*: Choose an optimal location to execute the process considering the process density and the network throughput values.

Also, the algorithm considers fairness while maintaining the workload queues to make sure that all the workloads are processed within the allowed time duration. Considering the experimental time slot t, the total energy consumption at the mobile device is given as $\mathrm{E}(t)$. Similarly, the total end to end delay at any given time t is represented as $\mathrm{D}(t)$.

At any given time slot (t), the objective function can be defined as

$$\min\ (\mathrm{E}(t) + V\ \mathrm{D}(t)) \rightarrow (1)$$

where V is the energy-delay tradeoff parameter.

With the aim of minimizing the total energy spent and the delay to complete a task as a function of time t, this is further detailed as

$$\min([E_L\ (r_L(t)) + E_N(r_O(t))] + V\ [Q_L(t) + Q_E(t) + Q_C(t) + Q_N(t)]) \rightarrow (2)$$

where E_L and E_N corresponds to the energy consumption values during the local processing and the network processing respectively. The CPU clock speed frequency during local processing is represented as r_L whereas r_O denotes the CPU clock speed during the offloading process. Queue lengths of LA, EA, CA and NA workloads are denoted as Q_L, Q_E, Q_C and Q_N. Additionally, Q_E and Q_C can be combined as Q_O representing the offloadable workload queue length. i.e., $Q_O = Q_E + Q_C$

In the algorithm design, we aim to minimize the function (2) dynamically based on the various control knobs at any given time slot t.

6.2 Algorithm Design

The core objective of this algorithm is to dynamically optimize and control the energy consumption and the CPU clock speed by reasonably maintaining the workload delay. Introducing the dynamicity into the algorithm makes it more efficient and eliminates the need of having prior knowledge about the network interface availability, workload arrival rate, and workload processing density.

To manage the five control knobs, for every time slot t, the algorithm at the smartphone is defined as follows.

Algorithm 1 Hybrid Edge Computing.

if $Q_L(t) > Q_O(t)$ **then**
  if $Q_O(t) >= Q_N(t)$ **then**
    Schedule LA and OA* as the primary workload
    $r_L{}^*(t) = \min ([E_L (r_L(t))] + V [Q_L(t)]) \rightarrow (11)$
    $l^*(t) = \min ([E_N (\mathrm{l}(t))] + V [Q_E(t) + Q_C(t)]) \rightarrow (12)$
    $r_O{}^*(t) = \min ([E_N (r_O(t))] + V [Q_E(t) + Q_C t)]) \rightarrow (13)$
  end
  else if $Q_O(t) < Q_N(t)$ **then**
    Schedule LA and NA with LA as the primary workload
    Configure CPU speed $r_L{}^*(t)$ as (11)
    $l^*(t) = \min ([E_N (\mathrm{l}(t))] + V [Q_N(t)]) \rightarrow (14)$
    $r_N{}^*(t) = \min ([E_N (r_N(t))] + V [Q_N(t)]) \rightarrow (15)$
end
else if $Q_L(t) <= Q_O(t)$ **then**
  if $Q_L(t) > Q_N(t)$ *and* $L_{PD} > N_{PD}$ **then**
    Schedule LA and OA with OA as the higher priority
    Configure CPU speed $r_L{}^*$(t) as (11)
    Select Network Interface $l^*(t)$ as (12)
    Configure CPU speed $r_O{}^*$(t) as (13)
  end
  else if $Q_L(t) <= Q_N(t)$ *and* $L_{PD} <= N_{PD}$ **then**
    Schedule OA and NA with OA as the higher priority
    Configure CPU speed $r_O{}^*$(t) as (13)
    Select Network Interface $l^*(t)$ as (12)
    Configure CPU speed $r_N{}^*$(t) as (15)

* OA = EA + CA;

where $l(t)$ is the network interface selection parameter that decides to choose between cellular and WiFi network interface.

Algorithm explained here considers deriving a new CPU clock speed $r_L{}^*(t)$ for workloads that gets scheduled locally and chooses the networking interface $l^*(t)$ for the offloadable workloads. Also as a significant contribution to this research work, the algorithm is designed to configure the new CPU clock speed $r_O{}^*(t)$ after choosing the networking interface for offloadable workloads. This is based on the impact of CPU clock speed on the network transmission delay as discussed in Section 5.

The algorithm is designed to allocate the jobs to CPU and network resources initially and then to tune the CPU clock speed and finally to make the network interface selection. The job scheduling scenarios are majorly classified into two scenarios as depicted in Figure 6.2.

6.2.1 Scenario 1

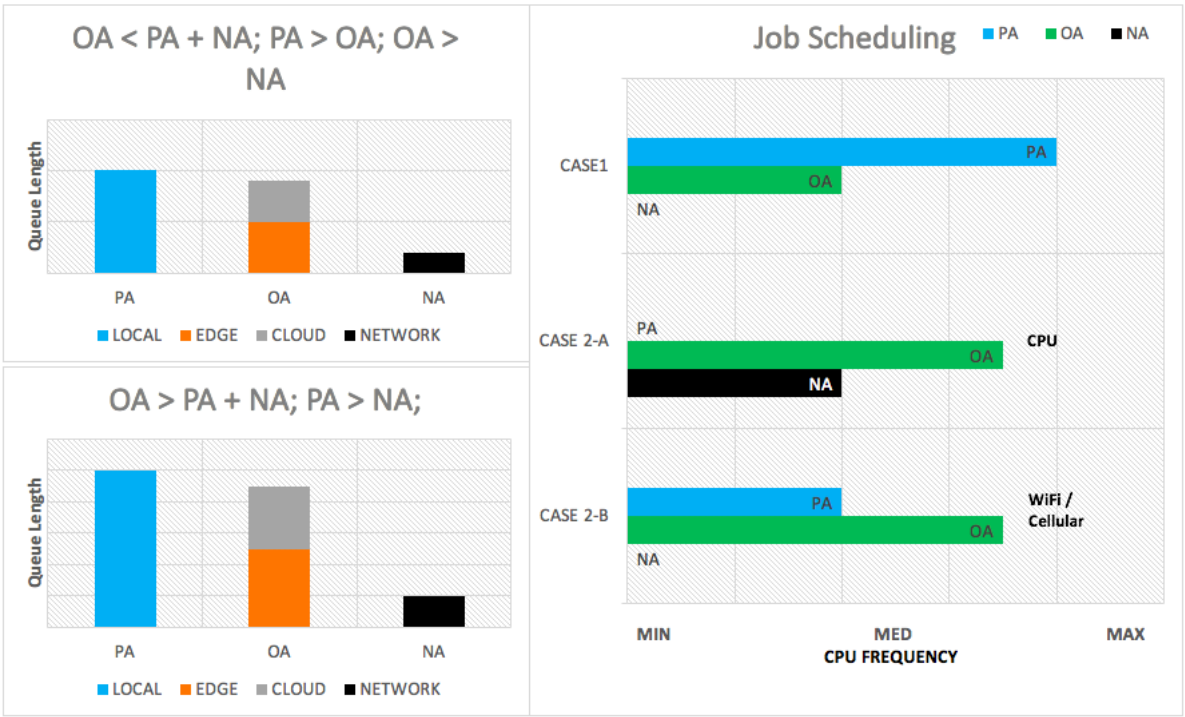

Figure 6.2: Algorithm Job Scheduling.

(Job scheduling priority for different workload queues and network interfaces.)

The primary scenario is where LA queue is greater than the OA queue; OA queue is greater than NA queue, and also OA queue is not greater than LA and NA queue combined which can be represented as,

$$Q_L\ (t) > Q_O(t)\ ;\ Q_o(t) > Q_N\ (t)\ ;\ Q_o(t) < Q_L(t) + Q_N(t)$$

At any given timeslot t, LA workloads can exploit only the CPU resources, and NA workloads can use the networking resources. However, with the case of OA (EA and CA) workloads, it can exploit both CPU and networking resources. In this algorithm, to maintain fairness in addressing the queue length and also to minimize excessive delay to avoid exceeding the maximum amount of processing time, urgent tasks take priority in the scheduling process. Therefore, when the LA queue is more than OA queue, the LA workloads will be scheduled with the highest priority on the CPU resources, and when CA queue is greater than NA queue, networking resources will be utilized by the CA resources as referred in Figure 6.2 case(1). In contrast, if the NA queue is greater than CA queue, NA resources utilize the networking resources for offloading the tasks to the edge component.

6.2.2 Scenario 2

Considering the other scenario where the EA + CA queue is more than LA + NA, offloadable workload EA and CA will be scheduled with higher priority.

$$Q_O\ (\mathrm{t}) > Q_L(\mathrm{t})\ ;\ Q_o(\mathrm{t}) > Q_N\ (\mathrm{t})\ ;\ Q_L(\mathrm{t}) > Q_N(\mathrm{t})$$

Given this situation, the OA (EA + CA) workloads can still exploit either of CPU or networking resources for completing the task. Therefore the algorithm decides on this factor based on the processing density and queue length of LA and NA queues where the output could result in only two variants such as {OA, LA} or {OA, NA} workloads getting scheduled.

When OA and LA workloads are scheduled, the OA workloads use the networking resources for offloading, and CPU resources are allocated to the LA workloads. Moreover, in the case of OA and NA tasks getting scheduled, OA will exploit the CPU resources, and NA uses the networking resources for processing.

6.2.3 Decision Algorithm - Edge

The decision algorithm at the edge component side is detailed as follows.

Algorithm 2 Hybrid Edge Computing Algorithm At Edge.

Schedule EA and CA simultaneously

if $Q_E >>$ *and* RTT_{EC} *is within acceptable values* **then**
| $Q_C[\mathrm{n}] = Q_E[\mathrm{n}]$
end

where RTT_{EC}[1] is the average Round Trip Time between the edge and cloud components. We refer to the values discussed by Tauber *et al.* for defining the acceptable values for WiFi strength and bandwidth values [53].

When the EA queue is very much higher, and the networking delay to the cloud is on the minimal side, partial workload from EA gets pushed to the CA queue. Since the processing delay at the cloud component is very minimal when compared with edge side processing delay as discussed in Section 5, the algorithm strives to minimize the queue length by offloading the workload to cloud. However, the energy factor is ignored in this edge side algorithm since the cloud, and edge components are considered to have infinite energy as referred to in Section 4. Additionally, WiFi is the default choice of the network interface for the offloading purpose between the second and third tier.

6.3 Energy-Delay Tradeoff

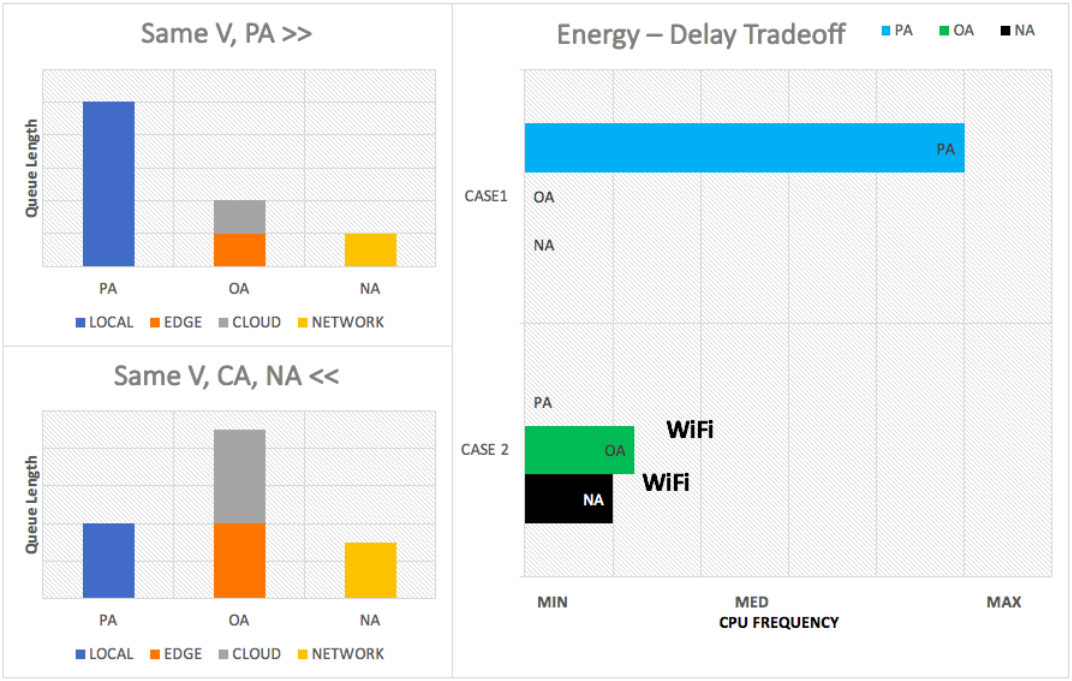

Figure 6.3: Energy-Delay Tradeoff.

(Depicts the algorithm decisions for different values of energy-delay tradeoff value V.)

As referred in the minimization function (1), the energy-delay tradeoff parameter is referred as V. To arrive at an optimal value in the algorithm by balancing the delay and energy consumption accordingly the value of V can be tweaked accordingly.

[1]We sample this by uploading and downloading a sample file (1MB) between edge and cloud components.

With reference to the detailed function (11) and (13), the factor without V refers to the term that strives to minimize the energy during the processing. For the same value of V in (11) and (13) if the queue length is longer, then the algorithm from equation (11) (13) (15) thrives to choose a higher CPU clock speed which results in the reduction of queue lengths. Similarly, the network interface selection can also be influenced using the tradeoff parameter V as depicted in Figure 6.3 case (1). In contrast, as referred in Figure 6.3 case (2), for the same value of V, if the processing delay is relatively shorter, the algorithm equation (12) and (14) tries to minimize the energy consumption factor. This is possible by reducing the CPU clock frequency and also choosing energy efficient network interface for OA and NA workloads. Additionally, if the delay is the primary bottleneck to be addressed, the algorithm equation (12) and (13) choose delay efficient interface between WiFi and cellular network at the given time slot t.

The developed algorithm is compared and listed in the table 6.1 along with its various characteristics.

	Hybrid Edge Computing	**Single Layer Dynamic Policy**	**Single Layer Dynamic Policy with Hierarchy**	**Static**
Applications	LA, EA, CA, NA	LA, CA, NA	LA, EA, CA, NA	LA,CA
Algorithm Type	Dynamic	Dynamic	Dynamic	Static
CPU clock control (For Processing)	Yes	Yes	Yes	No
Network Selection	Yes	Yes	Yes	No
CPU clock control (For Offloading)	**Yes**	No	No	No
N-Tier Architecture	**3 Tier**	2 Tier	3 Tier	No

Table 6.1: Algorithm comparison with its characteristics.

As referred in Table 6.1, while comparing with the single layer dynamic policy, the proposed HEC algorithm considers the hierarchical architecture of 3 tiers. This provides more options for the decision algorithm to choose an optimum location for execution. Additionally, the knob for controlling the CPU clock speed during offloading is a crucial feature of this algorithm when compared with the existing dynamic and static algorithms.

6.4 Algorithm Implementation

To evaluate the proposed algorithm performance, we implement and execute the algorithm as discussed below.

Experiment Setup: This section explains the implementation of the algorithm discussed in Section 6. Similar experimental setup as detailed in Section 4 is used here. Algorithm 1 is implemented in a Samsung Galaxy S8 smartphone. Also, the algorithm 2 is implemented as a web application at the edge component and it is accessible through a public static IP address. Both the algorithms are developed using Java as the programming language. Cloud component is designed under the AWS system where an EC2 instance is deployed, and the developed web application is configured with public IP address.

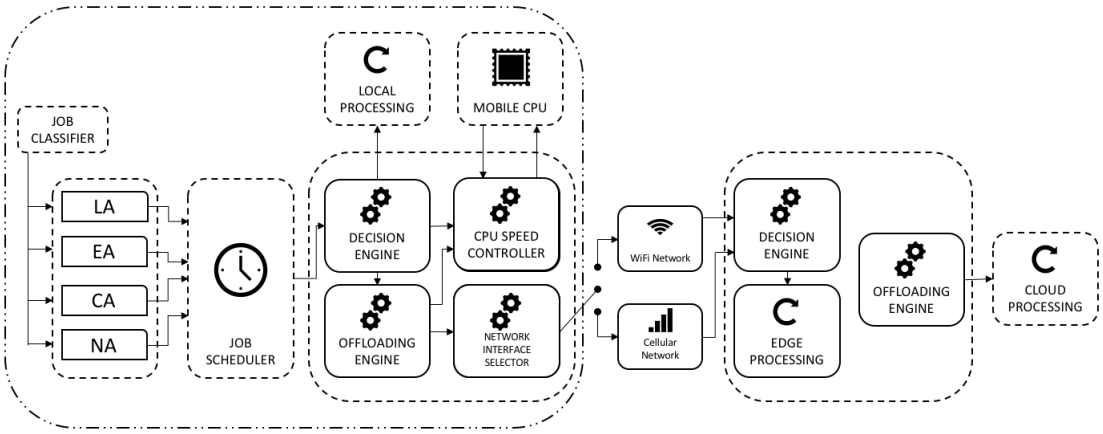

Figure 6.4: Algorithm Implementation.
(HEC algorithm implementation internal details.)

Implementation: Detailed algorithm structure is depicted in Figure 6.4. This Hybrid Edge Computing architecture enables to implement the algorithm under the mobile-edge-cloud setup. Major control components of this architecture are discussed below.

(i) Workload Manager that includes the *Job Classifier* and the *Job Scheduler* is responsible for handling the classification and the scheduling of the incoming workload arrivals. Based on the processing density and the current workload queue lengths, the classifier categorizes the workload into LA, EA, CA and NA types. The job scheduler is majorly responsible for scheduling the next workload ready for processing from the available queues.

(ii) *Decision Engine* acts as the heart of architecture which chooses optimal values that serve as the input factors to the other components. This decision engine receives

the parameter values from the other components that help in making the decision at the given time slot t. Decision engine at the mobile side controls the CPU speed controller and the offloading engine on the mobile side. Edge side decision engine is intact with the offloading engine at the edge side.

(iii) *CPU speed controller* is one of the main components of this architecture which is responsible for altering the smartphone CPU clock speed based on the decision from the algorithm output. This component is designed based DVFS techniques, where the software gains root access to the Android OS and configures the mobile to the desired CPU frequency as demanded by the algorithm decision engine. This component stays active for almost the entire life cycle of the algorithm completion since the local workload processing and the offloadable workloads benefit from the varying CPU clock at any given time slot t.

(iv) *Offloading Engine* on the mobile side makes a crucial decision based on the throughput values between mobile and the edge component. This is applicable for EA, CA and NA workloads which are offloadable workloads. This engine controls the input being fed into the network interface selector and the CPU speed controller since the transmission delay is impacted by the CPU clock speed as referred in Section 5. The offloading engine at the edge side is primarily for offloading the delayed workload to the cloud component where infinite processing power is considered.

(v) *Network Interface Selector* component chooses the optimal interface based on the need of the algorithm. It chooses either of the WiFi or the cellular network interface for the traffic offloading purpose. This interface selection is mainly based on the energy consumption and network strength of the interfaces. The bandwidth availability and the latency factors are also considered which influences the decision.

Along with the Hybrid Edge Computing algorithm, other baseline algorithm designs were evaluated during this process namely (a) *Single Layer Dynamic Policy* - Architecture involves only Mobile and Cloud resources (b) *Multi Layer Dynamic Policy* - Architecture involving mobile, edge and cloud architecture without controlling the CPU clock speed during network transmission (c) *Static Policy* - Classifies and offloads the LA, EA, CA and NA workloads on round-robin fashion by controlling the CPU clock frequency randomly. Other algorithm experiment values serve as the baseline values to be compared with for analyzing the HEC algorithm benefits.

Additionally, during the experiment, the smartphone display is switched off to avoid its impact on the battery power measurements. Energy consumption at the mobile device is measured using Battery Historian and Accu Battery application where their aggregate

values are considered for presenting the results. The smartphone is configured with the ability to connect to the cellular (4G) and WiFi network. The WiFi is turned ON and OFF in a fixed interval for intermittent availability and to avoid one interface bias choice (no one specific interface is chosen every time). With this settings, the mobile can transfer the computation to the edge server. By scheduling the workload and executing the algorithm, the energy and time delay values are monitored continuously at every time slot for all the workloads.

6.5 Evaluation

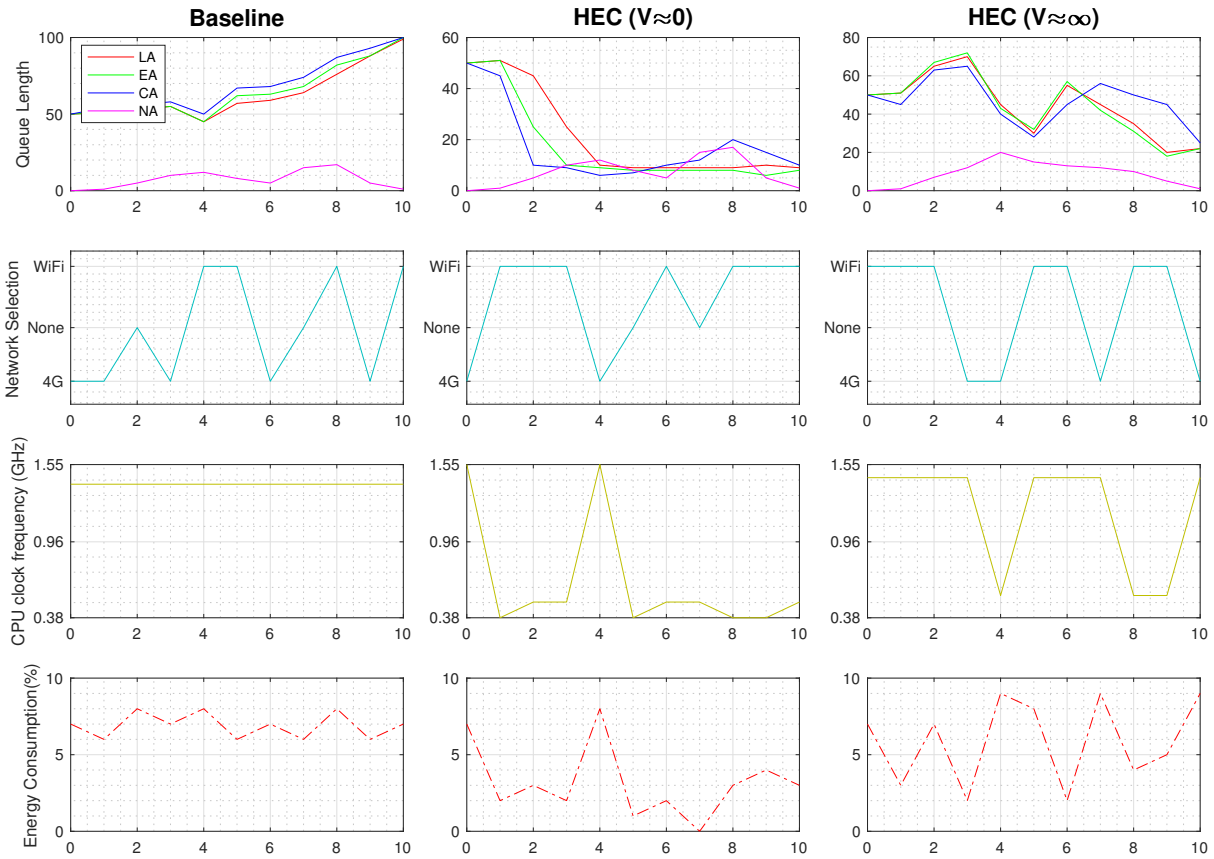

Figure 6.5: HEC algorithm implementation results.

(Comparison of variation in queue lengths, network selection, CPU speed scaling and energy consumption between baseline and HEC algorithm for different energy-delay tradeoff values.)

Figure 6.5 illustrates the critical observations of the algorithm implementation describing the queue lengths, network selection, CPU speed scaling and the energy consumption for different V values compared with the baseline values.

(i) HEC algorithm outperforms the baseline algorithm results in terms of maintaining

workload queue fairness, minimizing the power consumption and also exploiting the CPU and network resources effectively for any values of V.

(ii) As V gets smaller ($V \approx 0$), the average queue lengths decreases and the network selection happens less frequently allowing energy efficient interface to be selected. Additionally, the CPU clock speed gets lower demonstrating minimal power consumption.

(iii) For larger values of V ($V \approx \infty$), the delay efficient network interface is frequently selected to reduce the queue lengths faster. The CPU is clocked at higher frequencies, resulting in more energy consumption but benefits with faster processing time.

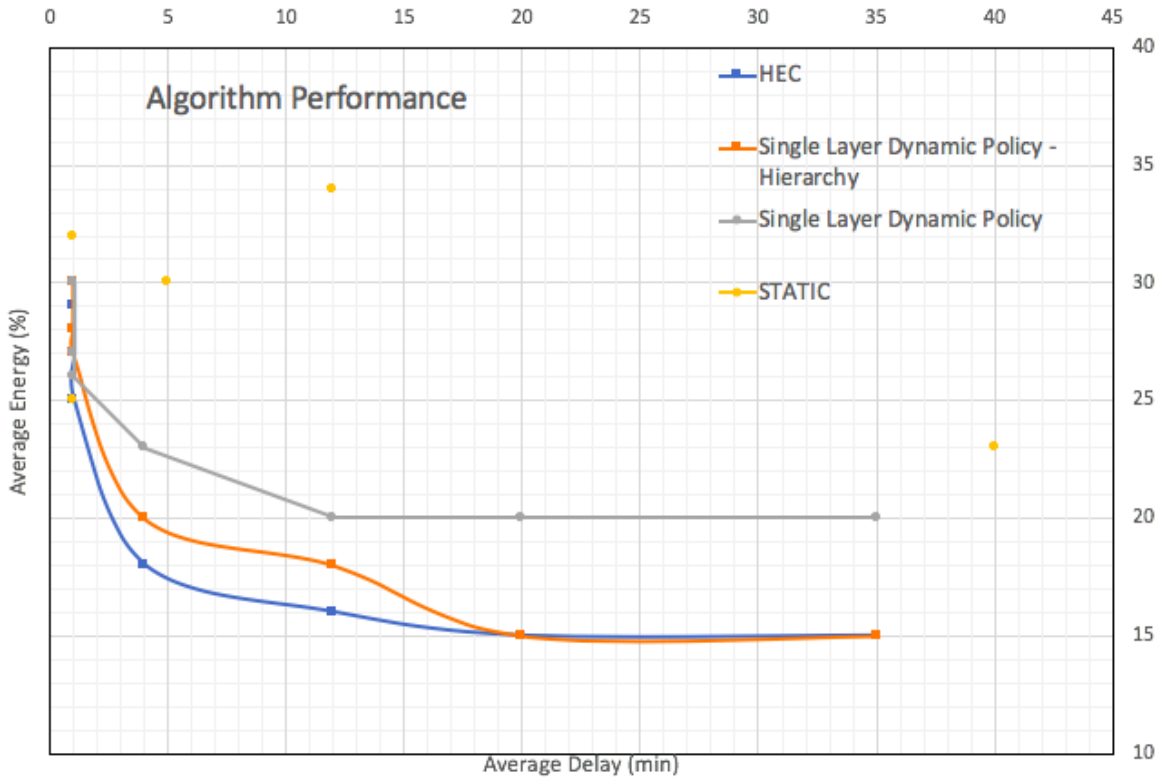

Figure 6.6: HEC algorithm evaluation.

(Energy-Delay savings comparison between HEC algorithm, single layer dynamic policy (with and without hierarchy) and static algorithm.)

The experiment results are summarized and compared against other algorithms in Figure 6.6. This graph illustrates the average end-to-end delay against the average energy consumption in the mobile side including the application processing delay and the network transmission delay. It can be observed that the HEC algorithm value outperforms the other algorithms by a significant margin. This is majorly due to the reason that HEC

algorithm dynamically exploits the CPU and network resources in addition to maintaining the fairness with the workload queue lengths. Observed values of HEC algorithm in Figure 6.6 describes that the average energy consumption is reduced by introducing a small amount of delay. Primarily, we observe that the average energy consumption dropped around 30% with a small increment in delay of 12 mins to the workload computation. This is nearly 47% of energy saving on the smartphone by trading off energy-delay by the HEC algorithm.

Chapter 7

Conclusion

To minimize the overall energy consumption and the processing delay at the mobile phones, we studied the energy efficient mobile-edge-cloud offloading systems. Also, we studied the correlation between various smartphone parameters and the results were analyzed in detail. Based on the experimental results, we proposed a dynamic Hybrid Edge Computing (HEC) algorithm which considers the various type of workloads (non-offloadable, offloadable and network workloads) generated from a mobile phone. As a novel contribution, we presented an analysis regarding the impact of mobile CPU clock frequency on the network transmission delay. We also proposed a hybrid architecture involving mobile, edge and cloud components and demonstrated the benefits of the proposed architecture and the algorithm by implementing on an Android and Java platform. Implementation results were compared against baseline values, and the HEC algorithm outperformed them in all aspects depicting energy savings up to 47% by adding minimal delay. Although the hybrid architecture demands high energy when compared to local processing, it is compensated by the effective decisions of HEC algorithm by choosing energy efficient interfaces for the network transmission and an optimum execution tier for processing. We firmly believe that the proposed algorithm and the architecture would be imperative in terms of saving energy and minimizing the delay by exploiting the computation offloading techniques as the edge and cloud computing paradigm is gaining more attention in the mobile communication arena.

Chapter 8

Future Work

With the advancements in mobile technologies and the recent improvements in the edge and cloud computing arena, below are the few possible extensions to our work.

(i) The similar optimization problem can be addressed by considering various code offloading policy in which only partial code is offloaded for remote execution in contrast to the computation offloading where all the computation is offloaded. Then we can imagine a situation where a heavy workload will be split into chunks of computation and gets offloaded across the tiers for completion.

(ii) A sophisticated process flow can be designed where the communication between the edge and cloud components is asynchronous. In which, the edge device submits the request to cloud, and the cloud can respond with the results to the same or different edge device that has more bandwidth at that given point of time which would make practical use of proximity edge devices serving more loads effectively in its idle time.

(iii) One another possible addition to this work, would be to consider the energy and cost factor at the edge and cloud data center. This makes the offloading decision algorithm at the edge to be more complicated since these additional energy factors will majorly influence the offloading decision.

Bibliography

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021 white paper," Mar 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[2] M. H. Memon, M. Hunain, A. Khan, R. A. Shaikh, and I. Khan, "Power management for Android platform by Set CPU," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2016, pp. 3953–3958.

[3] "IDC Forecasts Smart Cities Spending to Reach $158 Billion in 2022, with Singapore, Tokyo, and New York City Among Top Spenders." [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS44159418

[4] "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[6] "Key note on edge comptuing." [Online]. Available: http://acm-ieee-sec.org/2017/Edge+Computing+SEC+Keynote+Oct+2017+Mahadev+Satya.pdf

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62.

[8] G. Kalic, I. Bojic, and M. Kusek, "Energy consumption in android phones when using wireless communication technologies," in *2012 Proceedings of the 35th International Convention MIPRO*, May 2012, pp. 754–759.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, Mar. 2012, pp. 945–953.

[11] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, Dec 2015.

[12] J. Kwak, O. Choi, S. Chong, and P. Mohapatra, "Processor-Network Speed Scaling for Energy-Delay Tradeoff in Smartphone Applications," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1647–1660, Jun. 2016.

[13] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "etime: Energy-efficient transmission between cloud and mobile devices," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 195–199.

[14] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall, "Improved access point selection," in *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, ser. MobiSys '06. New York, NY, USA: ACM, 2006, pp. 233–245. [Online]. Available: http://doi.acm.org/10.1145/1134680.1134705

[15] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely, "Energy-delay tradeoffs in smartphone applications," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 255–270. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814459

[16] "Wifi lessons." [Online]. Available: https://www.metageek.com/training/resources/wifi-signal-strength-basics.html

[17] A. J. Nicholson, J. Han, D. Watson, and B. D. Noble, "Exploiting mobility for key establishment," in *Seventh IEEE Workshop on Mobile Computing Systems Applications (WMCSA'06 Supplement)*, Aug 2006, pp. 61–68.

[18] "Snapdragon Platforms Series Comparison," Jun. 2014. [Online]. Available: https://www.qualcomm.com/snapdragon/processors/comparison

[19] W. Hu and G. Cao, "Energy-aware cpu frequency scaling for mobile video streaming," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2314–2321.

[20] "Linux kernel," 2017. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[21] H. B. Jang, J. M. Kim, H.-J. Lee, S. W. Chung, Y. Shin, and J. C. Son, "Intelligent governor for low-power mobile application processors," in *2013 International Soc Design Conference (ISOCC)*, Nov 2013, pp. 206–207.

[22] "User Experience Driven CPU Frequency Scaling On Mobile Devices Towards Better Energy Efficiency," Mar. 2017. [Online]. Available: http://www.volkerseeker.com/publication/2017_phd

[23] K. Akherfi, M. Gerndt, and H. Harroud, "Mobile cloud computing for computation offloading: Issues and challenges," *Applied Computing and Informatics*, vol. 14, no. 1, pp. 1–16, 2018.

[24] M. Chen, Y. Hao, Y. Li, C. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: architecture and service modes," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 18–24, June 2015.

[25] "5g mobile edge computing will empower ot in manufacturing," 2018. [Online]. Available: https://www.abiresearch.com/market-research/product/1028885-5g-mobile-edge-computing-will-empower-ot-i

[26] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 1451–1455.

[27] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Dec 2016.

[28] S. Ulukus, A. Yener, E. Erkip, O. Simeone, M. Zorzi, P. Grover, and K. Huang, "Energy harvesting wireless communications: A review of recent advances," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 3, pp. 360–381, March 2015.

[29] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5529–5534.

[30] W. Labidi, M. Sarkiss, and M. Kamoun, "Energy-optimal resource scheduling and computation offloading in small cell networks," in *2015 22nd International Conference on Telecommunications (ICT)*, April 2015, pp. 313–318.

[31] W. Labidi, M. Sarkiss, and N. Kamoun, "Joint multi-user resource scheduling and computation offloading in small cell networks," in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2015, pp. 794–801.

[32] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[33] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "A Cooperative Scheduling Scheme of Local Cloud and Internet Cloud for Delay-Aware Mobile Cloud Computing," *arXiv:1511.08540 [cs]*, Nov. 2015.

[34] X. Guo, R. Singh, T. Zhao, and Z. Niu, "An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–7.

[35] V. D. Valerio and F. L. Presti, "Optimal virtual machines allocation in mobile femto-cloud computing: An mdp approach," in *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, April 2014, pp. 7–11.

[36] H. Atre, K. Razdan, and R. K. Sagar, "A review of mobile cloud computing," in *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, Jan 2016, pp. 199–202.

[37] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[38] Ruay-Shiung-Chang, J. Gao, V. Gruhn, J. He, G. Roussos, and W. Tsai, "Mobile cloud computing research - issues, challenges and needs," in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, March 2013, pp. 442–453.

[39] M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, and J. Stefa, "CDroid: Towards a cloud-integrated mobile operating system," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2013, pp. 47–48.

[40] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2016, pp. 1–8.

[41] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[42] "AWS Developer guide - AWS Lambda." [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[43] "Team win recovery project TWRP." [Online]. Available: https://dl.twrp.me/

[44] "Repository for HEC Android Application." [Online]. Available: https://github.com/Jude-iamgame/Hybrid-Edge-Computing

[45] "Box plot - MATLAB boxplot - MathWorks United Kingdom," https://uk.mathworks.com/help/stats/boxplot.html.

[46] "JCodec - a pure java implementation of video/audio codecs." [Online]. Available: http://jcodec.org/

[47] "AsyncTask," https://developer.android.com/reference/android/os/AsyncTask.

[48] "Spring Boot Memory Performance." [Online]. Available: https://spring.io/blog/2015/12/10/spring-boot-memory-performance

[49] "Repository for HEC Java application." [Online]. Available: https://github.com/Jude-iamgame/HEC-Java

[50] "Profile battery usage with Batterystats and Battery Historian." [Online]. Available: https://developer.android.com/studio/profile/battery-historian

[51] "Track your mobile battery performance." [Online]. Available: https://www.accubatteryapp.com

[52] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications," in *IMC*, Jan. 2009, pp. 280–293.

[53] M. Tauber and S. N. Bhatti, "Low RSSI in WLANs: Impact on application-level performance," in *2013 International Conference on Computing, Networking and Communications (ICNC)*, Jan. 2013, pp. 123–129.

Appendix A

(i) Mobile Configuration

Model	**Samsung S8**
Model Number	**SM-G950F**
Android Version	**8.0.0**
Kernel Version	**4.4.111mat**
QUALCOMM build	**183c040, Iff84fb1103**
Build Number	**G950FXXU3CRGH**

(ii) Edge Configuration

Processor	1.6 GHz Intel Core i5
Memory	8 GB 1600 MHz DDR3
Make	**Apple Inc.,**
Model	**MacBook Air**

(iii) Cloud Configuration

	Instance #1	Instance #2
Provider	**Amazon Web Services (AWS)**	**Amazon Web Services (AWS)**
Instance Type	**t2.large**	**t2.small**
No.of vCPU	**2**	**1**
Memory (GiB)	**8**	**2**
Processor	**2.3 GHz**	**2.4 GHz**

(iv) Mobile CPU clock frequencies

Available Scaling Governors frequencies (Hz) in the smartphone used in the experiments. **384000** 460800 600000 672000 768000 **864000** 960000 1248000 1344000 1478400 **1555200**