Honors Project Report

Cloudlets: Supporting co-located interactions using the Raspberry Pi

BY

Zola Mahlaza

Supervisor:
Prof. Edwin Blake

Co-supervisors:
Pierre Benz
Thomas Reitmaier

	Category	Min	Max	Chosen
1	Requirement Analysis and Design	0	20	10
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	10
4	System Development and Implementation	0	15	10
5	Results, Findings and Conclusion	10	20	15
6	Aim Formulation and Background Work	10	15	15
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	0
Total Marks		8		80

Department of Computer Science
University of Cape Town
Cape Town, South Africa

October 2014

Cloudlets: Supporting co-located interactions using the Raspberry Pi

Abstract

The cloud (or more formally, cloud computing) can be viewed as the treatment of computing as a utility. Cachin et al[5] define cloud computing as being the flexible online data storage services, ranging from passive ones, such as online archiving, to active ones, such as collaboration and social networking [5, p. 81]. It is obvious to see that cloud computing platforms play an essential role to modern lives and the economy. As an example, the Google clould platform provides services to companies like Coca Cola, Khan Academy, Ubisoft and others. Unfortunately, cloud computing is well suited for devices connected to networks with fast Internet speeds. It faces challenges in mobile devices in a number of areas due to weak network signals and bandwidth-induced delays. It should be easy to notice that the technologies that power cloud computing can be used on a smaller scale through embedded systems. This results in the transformation of the cloud from a global panopticon to a hyper-localized and ad-hoc instantiation of the cloud. These instantiations are referred to as cloudlets. They address the issues of weak network signals and bandwidth-induced delays due to their proximity to the mobile device users. This report details an investigation of a Raspberry Pi for hosting a cloudlet. The use of a Raspberry Pi introduces the idea of affordable and portable cloudlets.

Contents

Listing of figures

Acknowledgments

I would like to thank Prof. Edwin Blake, Thomas Reitmaier and Pierre Benz for their guidance.

Thanks to Terry Tsen, Katlego Moukangwe and Gerard Nothnagel for the laughs at the lab.

Many thanks to my family.

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo.

Quoteauthor Lastname

1 Introduction

Research is about increasing the stock of knowledge, solving problems people face, devising ways to improve aspects of human lives, etc. As a researcher, it is sometimes helpful to be aware of distinctions between different geographical regions. There are numerous reasons why distinctions could exist, and some distinctions are consequences of other basic distinctions. As an example, the availability of infrastructure or lack thereof results in differences in the way technology is used in different areas. We have observed online media sharing become popular with the arrival of services such as Dropbox, Google+, Google Drive, etc. However, Walton et al[32] have shown that in places such as Khayelitsha (Cape Town, South Africa), co-located phone use surpassed online sharing. They observed that the media stored on the mobile devices became public personae, that is, it played the role of social media 'profiles'[32, p. 403].

The aforementioned services are great examples of cloud computing in use. The cloud (or more formally, cloud computing) can be viewed as the treatment of computing as a utility. Cachin et al[5] define cloud computing as being the flexible online data storage services, ranging from passive ones, such as online archiving, to active ones, such as collaboration and social networking [5, p. 81] Cloud computing platforms play an essential role to modern lives and the economy. The Google clould platform provides services to companies like Coca Cola, Khan Academy, Ubisoft and others. As far as computer science is concerned, the cloud is a useful computing paradigm from the perspective of a developer as it removes the costs associated with buying hardware. It also removes other complications associated with hosting one's own over-the-Internet services. It is not as attractive, however, from the perspective of a mobile device user as it has a number of faults. Essentially, cloud computing is well suited for devices connected to networks with fast Internet speeds. It still faces challenges in mobile devices in a number of areas due to weak network signals and bandwidth-induced delays. These challenges face mobile devices due to the distance of the cloud from the mobile device user. They are worth solving as they may deter some individuals from using cloud based applications.

The technologies that power cloud computing can be used on a smaller scale through embedded systems.

This results in the transformation of the cloud from a global panopticon to a hyper-localized and ad-hoc instantiation of the cloud. These instantiations are referred to as cloudlets. They address the issues of weak network signals and bandwidth-induced delays due to their proximity to the mobile device users. It is important to note that cloudlets have been explored through other technologies such as the HP-XW8200 workstation. Ling et al[16] used multiple HP-XW8200 workstations with two 3.2GHz Intel Xeonprocessors and 2GB RAM to evaluate the practicality of augmented reality cloudlets for mobile computing. They did find that their architecture was faster than the traditional approach of using the cloud.

This report, on the other hand, details the investigation of a Raspberry Pi for hosting a cloudlet. The use of a Raspberry Pi introduces the idea of affordable and portable cloudlets. Portability is important because among its benefits; it allows friends to have a channel which can support co-located interactions wherever they are. It need not be connected to the Internet therefore it can even support co-located interactions in areas where there is little to no mobile network coverage. The use of mobile device when co-located may seem counter-intuitive, however, Harper et al[11] (as cited by Reitmaier and Benz et al[22, p. 381]) has reported that people want to use mobile devices when co-located as means to enrich social interactions. It is then essential to understand and support co-located interactions.

The use of the Raspberry Pi provides new possibilities but it also introduces memory and computation limitations as compared to the cloud. The Raspberry Pi is a single board computer developed by the Raspberry Pi Foundation. It was designed for the purpose of being an affordable computer that will make it possible to teach computer science in high schools. There are three models which are available. These are model A, B and B+. They differ in specifications and those specifications can be found at http://www.raspberrypi.org/products/.

1.1 Problem statement and aims

There are numerous aspects the current cloud computing paradigm is lacking in, these limitations carry over to general cloudlets. The following are the limitations or aspects which the project will attempt to address:

- **Information Ownership, Control and Physical security**: Data centers for cloud computing service providers can be located in any country in the world. Users of cloud computing services may be concerned about where their information is stored due to privacy laws and government organizations in different countries. The proposed system should afford users the ability to control where the physical location of their data and who has access to it.
- **Latency**: Cloud computing is not suited for most areas in third world countries due to poor mobile network connectivity. In certain cases, cell phone providers have high charges. These two factors therefore limit the use of cloud services for mobile devices users. The cloudlet should reduce the latency associated with communication between the mobile device and the cloudlet.
- **Resource constraints**: The cloud has abundant storage available. This has the implication that cloud providers can make use of a variety of database management systems and algorithms. They can also be able store large quantities of data. The proposed cloudlet, however, will operate on a single board computer. This has the implication that the chosen algorithms and technologies need to use resources

effectively. Also, the files which are to be stored on the cloudlet will need to compressed to ensure optimal memory usage.

In essence, the project aims to (1) develop a lightweight communication model between clients and the cloudlet, (2) investigate suitable databases for a cloudlet instantiated using an embedded system, (3) investigate the practicality of using an inexpensive WiFi adapter for creating a cloudlet network, (4) investigate a combination of tools that effectively use computer resources and (5) create a robust cloudlet with a file service capable of handling frequent connections and disconnections.

1.2 Research Questions

- Can a single board computer be practical for creating cloudlets, taking in consideration battery consumption and network range?
  The primary users of cloudlets are mobile device users. Mobile device users are likely to go in and out of range. This means that the cloudlet should be able to quickly detect connections and disconnections. Also, the portability of the raspberry pi means that it should cover a relatively large range to cater for the mobility of its users. We aim to investigate if the battery will be drained quickly when certain tasks are performed and if the range of the cloudlet is practical. It is vital that we determine how long would a Raspberry Pi using a portable power source would last, what is the maximum distance covered by an inexpensive WiFi adapter and how is bandwidth affected.

1.3 Work allocation

The expected outcomes of the project are a cloudlet platform and an Android application that will interface with the file sharing service that will run on the cloudlet. The Android application explores the possibility of using the Raspberry Pi as a data sandbox. It is also used to determine which conceptual metaphors are appropriate for presenting ephemeral data to users. The platform capable of hosting numerous services and file sharing service are to be developed by Zola Mahlaza. The Android application which is a client for the file sharing service will be developed by Jarvis Mutakha.

1.4 Outline

Chapter 2 will discuss the existing literature on the challenges and success of cloudlets, technologies which are well suited for a cloudlet and ways to address the resource constraints faced by cloudlets hosted on embedded systems. Chapter 3 presents the design choices made and Chapter 4 resents the implementation of the cloudlet platform and the how a cloudlet service can be created. Chapter 5 presents the uses of cloudlets which will determine how we evaluate the cloudlet as being practical. Chapter 6 and chapter 7 reveal the test results and main results of the projects respectively.

2

Previous works

This chapter discusses and explores previous work related to the design of cloudlets. The system will run on an embedded system, therefore resources are constrained. Specifically, there is limited memory space and battery life should be maximized. In addition, it is of importance that the system have low latency as far as client-to-cloudlet communication is concerned. Also, it is essential the storage medium used is fast and does not result in large quantities of power consumed. In this chapter, we will discuss the following:

- Previous work on related to cloudlets, their uses and ways to ensure optimal memory and battery use.
- Databases for constrained environments.
- Dealing with the range of a possibly weak network signal.

There is existing literature on the concepts of cloudlets. A large portion of this existing work is addressing the offloading of computation from mobile device to a cloudlet to overcome the computation limitations of mobile devices. For instance, Ling et al[16] have proposed the use of cloudlets to improve augmented reality for mobile phones. The work done by Verbelen et al[30] is also of the same nature. This kind of research, albeit it presents great applications, makes the assumption that cloudlets have relatively abundant computing power. The interest of this project is the creation of a cloudlet that can operate effectively even when resources are constrained. The lack of literature pertaining to cloudlets hosted on embedded systems means that it necessary to look at work that presents cloudlets which do not have extreme memory and computation limitations and observe which aspects can be pruned to make a cloudlet suitable for an embedded system. We will also look at work that benchmarks various databases and data compression schemes, and provides results on which one is suitable for an embedded system, if any.

2.1 Related work

2.1.1 Practical co-located services

A fully functional cloudlet is useless without services. This is true even if the battery lasts for a year after charging it once or if the network signal can cover an entire city. A practical cloudlet is one with one or more services and can perform complex tasks without draining the battery quickly, one that is portable and one that uses memory efficiently. It is important that the services are practical, that is, they consider issues that arise in co-located interactions. In this cloudlet they should also handle mobility and possible simultaneous access to resources. As an example, if one where to develop a co-located file sharing service where the files can be edited simultaneously, it is important to introduce software lock mechanisms. Ah Kun and Marsden[1] refer to these mechanisms as floor control policies. There are numerous policies one could use to restrict modification of resources which have been accessed concurrently. However, the most appropriate methods are ad-hoc and three-second policies as mentioned by Ah Kun and Marsden[1].

- **Ad-hoc**: This is a policy whereby any user can control a resource at any time. Here, there are no software locks. A social protocol is expected to arise from the users[1, p. 282].
- **Three-second**: This is a policy whereby control is passed around among the users. A user acquires control and performs an action. When there is no action from the user on a given time limit (three seconds), control is passed to another user. A user will require control by attempting to perform an action[1, p. 282].

Practical services should not result in inconsistent data on the cloudlet. These inconsistencies can be caused by mobility of the client, simultaneous access to data, etc. All services should handle the mentioned issues and others which may arise due to the nature of the services being developed.

2.1.2 Resource constrained environments

It has been proposed that cloudlets can play a vital role in hostile environments[24]. These are environments where there is a little to no networking infrastructure or geographical regions where recovery is under-way after a natural disaster or terrorist attack, or simply an environment where there is a military presence. Satyanarayanan et al[24] highlight that cloudlets can be helpful because cloudlets offer services which are needed by mobile device users. These services are the overcoming of resource limitations, authoritative sourcing of data and synchronizing multi-user collaboration. Satyanarayanan et al[24] also propose the following scenarios where cloudlets can be of vital importance:

- **Disaster Recovery**
  When disasters such as earthquakes, bombings, etc have occurred, there is often loss of buildings and deformation of terrain. Useful tools such as maps and GPS become obsolete. A bottom-up approach to recreating repositories of information about the current state of the roads, buildings, etc is the most efficient. A crowd-sourced approach to imaging and geolocation by mobile users on the scene to a nearby cloudlet which will then upload the data to the cloud would assist rescue teams. This means that in order for cloudlets to be practical in a disaster recovery scenario, it should be able to handle a large number of clients. This also implies that storage of multimedia, be it destroyed roads or buildings, should be done efficiently.

- **Military operations**
  The US Department of Defense developed and tested a system named ”Land Warrior” around the 1990s[8]. The system comes with, among other things, a computer processor which is “fused with radios and a Global Positioning System locator. A hand grip wired to the pack and attached to the soldier’s chest acts as a computer mouse and also allows the wearer to change screens, key on the radio, change frequencies and send digital information”[8]. It is undoubtedly innovative but it is susceptible to wireless jamming and denial of service attacks as it needs to contact a distant cloud. Another challenge this system has is that of power. The department states that the “current batteries last about 150 minutes with all systems running”[8]. The sharing of computation among an operating army unit can increase battery life and possibly save the lives of some soldiers in their times of need. The cloudlet can be used to as a mediator for computation, computation can be sent to the cloudlet and it will deistribute computation among the unit. This may address, to some extent, the issue of power consumption. Unfortunately, cloudlets are also susceptible to wireless jamming and denial of service attacks like the cloud. In addition to these problems, the use of a portable cloudlet may create new problems for military units. Ither parties may attempt to gain access to the data stored on these devices. In order to prevent this, data on these devices may be protected using encryption. However, this creates the possibility that soldiers who are captured will be tortured for the decryption keys. This leads us to not include security and signal jamming prevention in our categorization of cloudlets as being “practical”. In other words, in this work we assume that the cloudlet will not be used in a military setting.

One can deduct from the aforementioned use cases that cloudlets have practical uses. The issues Satyanarayanan et al[24] identify which need to be addressed in cloudlets for the aforementioned environments is that of trust and cloudlet discovery, as that it could lead to wireless jamming, physical destruction, etc. Military operations will not be part of the environments where the proposed cloudlet will be practical. This is because it will not address wireless jamming, physical destruction of cloudlet and denial of service attacks.

2.1.3 Offloading computation

As mentioned before, the prominent use of cloudlet is computation offloading. It is known that mobile devices have limited computation power. There are numerous reasons why that is the case. For instance, in order for manufacturers to have portability and general usability of the mobile devices, they limit the computation ability of mobile devices. This is because an increase in computation power may cause, among other things, an increase in the heat discharged by the hardware. A solution to overcome limited computation power, as proposed by Giurgiu et al[10], is the use of the cloud to offload computation. This solution is effective, however, it is severely affected by the fact that clouds are located at a distance. The problem with it, as identified by Satyanarayanan et al[23] is that it suffers from bandwidth-induced delays and latency. The proposed solution had been to reduce the distance between the cloud and its users. Verbelen et al[30] have proposed, like a number of researchers, the use of cloudlets to offload computation from the mobile phone to a nearby cloudlet.

The cloudlets they propose are hosted on a laptop or computer in a wide area network. It is without a doubt that offloading of computation to cloudlet has great benefits to mobile device users as it can improve battery life. There have been attempts to test whether offloading computation to a cloudlet would have benefits in practise. An example of such is the study done by Verbelen and colleagues[30], they used augmented reality on mobile devices as a use case in order to evaluate their proposed system which offloads computation to cloudlets. Their

system puts emphasis on the decoupling of the augmented reality processing components. This is because only certain parts of the system are offloaded. The results they reported show that the cloud is not suited for an application that has real-time constraints as compared to a cloudlet for offloading computation. In essence, the cloudlet solves the problem of latency when communicating with a distant cloud. Their cloudlets perfomed better because latency was not big problem as the cloudlet was closer. It should be noted that embedded systems can also be used for the same task and will have the same advantage. In such cases, one could overcome network range limitations through the use WiFi Antenna. The problem, however, is the resulting increase in power usage due to the Antenna. The question at hand for embedded systems is understanding what tools to use to minimize power consumption but maintain a reasonable cloudlet.

2.1.4 PiCloud

A data center can be viewed as a facility "that houses and maintains back-end information technology (IT) systems and data stores—its mainframes, servers and databases"[9]. The infrastructure for a data center is expensive and this means that individuals without abundant finances are at a loss. Tso et al[28] have attempted to address this by constructing a "scale model of a data center composed of clusters of Raspberry Pi devices". This proposed model of a data center is named PiCloud. PiCloud is made up of 56 Model B raspberry Pis. It is important to note that PiCloud is cheaper than commercial servers and can effectively replicate cloud data center architectures. They also specify that they use Raspbian because it "comes with over 35,000 pre-compiled software packages". PiClould is reported to an effective replacement for commercial servers. Unfortunately, as one might expect, PiCloud is not suited for computation intensive tasks. The work done by Tso et al[28], even though it has implications to the proposed system, is very limited. PiCloud allows one to overcome the limitations of simulation systems such as iCanCloud and CloudSim. This advantage is impractical considering that 56 Model B raspberry Pis cost approximately 24,211 ZAR. Tso et al[28] have shown PiCloud can effectively replicate cloud data center architectures therefore there is a possibility that the Raspberry Pi can be a practical cloudlet. This means that a Raspberry Pi using a bulky operating system can effectively replicate cloud data center architectures. Therefore it is possible that the proposed system will perform better as it will use an optimized operating system.

2.2 Databases

A cloudlet, like a cloud, is a platform where the sharing of multimedia content is easy. It is possible to use a cloudlet as a server that maintains a list of users who want to share files. The exchange of files could be client-to-client. This approach of making the file sharing service means that the Raspberry Pi will only act as a broker. This is similar to how the Direct Connect (NMDC) protocol works[1]. NeoModus Direct Connect protocol is a peer-to-peer file sharing protocol. This approach, unfortunately, does not scale well when building a system that can support a number of services in addition to file sharing. It essentially means that cloudlet will an appliance capable of only file sharing. Services such as document collaboration would be difficult and inefficient when the files are stored on the mobile devices. This is largely because the client-server communication would be cumbersome. The solution to this, is to require files to be transferred to the cloudlet and then actions like

[1] http://nmdc.sourceforge.net/NMDC.html

sharing and editing be carried out. This requirement means that the cloudlet should accommodate for the storage of multimedia and it's metadata.

There exists numerous types of databases which can be used. Unfortunately, there is a lack of literature whose interest is the analysis of the performance of these various databases in a cloudlet scenario. As mentioned above, as a consequence of this, we will look at databases suitable for embedded systems. Moniruzzaman et al[19] states that NoSQL database management systems are useful when working with a huge quantity of data. This means that they are not of interest to this project. Other types of databases need to be considered.

It is essential to note that the type of database used affects data access control. The use of a relational database or a key-value pair database could mean that (1) users who have access to a specific multimedia object would be stored in the same database along the media in a relational database, (2) information about data access would be kept in the memory heap thus allowing frequent change, (3) information about data access, that is, file metadata would be kept as one string in the case of key-value pair databases. This should affect download and upload speeds.

2.2.1 Requirements

Olson[20] mentions that the problem working with embedded systems is that they have approximately the same traits as desktop and server systems, however, they have limited resources. The writer also states that in such cases, one must choose a database that best matches the required specific needs. This can be understood as the process of shedding databases which have less or more requirements than the necessary ones. The requirements for a database for this project are concurrency, speed and minimal power consumption. It may be tempting for one to "stand on the shoulders of giants" by copying the choices made by large companies and using the databases they use for embedded/mobile devices. Smart-phone operating systems such as Android[2] and iOS[3] ship with SQLite. One might assume that SQLite is a great choice. However, analysis of the SQLite website[12] shows that SQLite does not work well with high concurrency and client-server applications. This highlights the point that it is essential to consider all necessary requirements when choosing a database.

2.2.2 BerkelyDB

The Berkeley database is a descendant of the UNIX utility ndbm, it was developed by Margo Seltzer and Mike Olson[33]. Over the years it has grown to have functionality for concurrency, logging, transactions, and recovery[26]. It is suitable for embedded systems for a number of reasons. Berkeley DB is designed for rapid start-up, for instance, recovery happens automatically as part of system nitialization. This makes the Berkeley database suitable for system where a shut down could potentially happen without warning and restarted[26]. Seltzer[25] states that it's configuration is capable of being readily changed thus can support a number of different environments. The limitations of the Berkeley DB are that records in it are key-value pairs and the values are simply payloads.

[2] http://developer.android.com/guide/topics/data/data-storage.html
[3] https://developer.apple.com/technologies/ios/data-management.html

2.2.3 MySQL

MySQL is a popular open source relational database management system. It supports blobs (Binary Large objects). This means that both the multimedia and it's metadata can be stored within the database. This gives the advantage that files reside in the database and cannot be orphaned from it. Baumann et al[27] have benchmarked relational databases to see the performance of read and write operations for blobs. The databases they were benchmarking are PostgreSQL and MySql. They used three different storage engines; InnoDB, ARCHIVE and MyISAM. Their MySQL results results show that for the write operation: ARCHIVE and MyISAM show were fast and shown appeared almost linear whereas with InnoDb, frequent large spikes occurred at random times and heights. The read operation results revealed that ARCHIVE was better, particularly for large blobs. The "read times appear very regularly"[27], unlike the write operations. They reported that MySql showed the best result, and was very stable over the complete range as compared to PostgreSQL. Their recommendations for improving RDBMS performance when working with blobs are:

- "Logging should be turned off whenever feasible to avoid generation of undo and redo log entries for large objects.
- Choice of both database page size, buffer pools size, and BLOB structures on disk should be dictated by the size of the BLOBs to be handled. For small BLOBs, smaller page size avoid wasting disk space. For large BLOBs, a large page size is helpful, which usually is limited to 32K. As these parameters often are fixed during database creation some upfront reflexion [sic] on the expected BLOB sizes is feasible."

It is worth noting that the benchmarks were conducted on a 3.0 GHz Pentium 4 PC with with 512 MB main memory and 7,200 rpm IDE disks. It is possible that this results will not hold on an embedded system. The use of blobs will be beneficial only if it is faster and memory efficient to store the files in the database than storing the files independently on the filesystem and maintaining the metadata in the RDBMS.

2.3 Network range

There are numerous WLAN standards, 802.11n being the newest. 802.11n has the fastest maximum speed, best signal range and is more resistant to signal interference from outside sources[18]. Unfortunately, it is likely to interfere with near 802.11b/g based networks when multiple signals are used[18]. Antennas and adapters using the 802.11n WLAN standard have an outdoor range of approximately 243 meters[3]. This range, is greatly reduced indoors because of the walls. The range reduction is dependant on the materials that make up the wall. The Huawei WS320 Wi-Fi Repeater is an appliance that extends the coverage of WiFi network. It can increase a network by up to 100m[14]. Products such as this are relatively affordable.

The range of the wifi signal is limited. This has implications in dealing with mobile users. Users of cellphones are likely to go in and out of range. This and other issues concerning mobile computing are not new. A number of researchers including Evaggelia Pitoura and Bharat Bhargava presented their cases explaining the challenges as early as 1993. They presented their case in the paper "Dealing with mobility: issues and research challenges"[21]. "Non-mobile systems [...] can be seen have hosts which have two modes of operations. A host can either be fully connected to a network or its disconnected"[21, p. 4]. In mobile systems, as was foreseen by Pitoura and Bhargava[21], it makes sense to have varying degrees of modes of operation. These can range from

full disconnection, weak disconnection to string connection. These modes are determined by bandwidth.

The issues which face mobile systems as identified by Andrew Black and Jon Inouye[4] concern security, bandwidth and connectivity. They state that some applications make security decisions based on network topology. As an example, devices connected to the same subnet which is considered secure may choose not to use encryption when communicating. However, due to Mobile IP, the protocol that is designed to allow mobile devices to retain their IP addresses when moving across subnets, a security vulnerability might arise. Mobile devices could transfer unencrypted packets through an unreliable as they will be perceived as being in the same subnet. In the case of a cloudlet, there might be services which identify clients using the IP addresses. This has the implicit assumption that each IP identifies one mobile device. However, a mobile device may create a subnet thus an IP might identify a number of devices. This might create a security concern.

The overt issue with mobility which Black and Inouye[4] highlight is that mobile devices are likely to disconnect from the network. Also, even when devices are connected, the distance between the client and the cloudlet may vary thus "available bandwidth may change by three or four orders of magnitude". They state that this problem is not one "that will yield a technological fix"[4, p. 130]. Black and Inouye[4] suggest that in order to support mobility in information systems, applications and operating systems need to be modified. Early versions of email protocols required that mail be "pushed" to the receiver. This had the implication that receivers needed to always be connected to the same network. Current versions of email protocols now "pull" mail from a central location when needed. Services developed for the cloudlet at hand need to use the approach. Following this approach will do away with the possibility of data being sent and lost due to the receiver being disconnected from the network.

2.4 Discussion

The work done by Verbelen et al[30] on computation offloading operates under the assumption that cloudlets have abundant resources. This project will do away with the assumption that cloudlets have abundant resources and determine which technologies are suited for a resource constrained cloudlet, if any. The need for computation offloading is recognized and unlike the work done by Verbelen et al[30], we are also interested in allowing users to be able to offload computation to a secondary device wherever they are. This goal will be achieved through the portability of the proposed system. The battery life of the Raspberry Pi will present a challenge. The work done by Tso et al[28] shows that Model B of the Raspberry Pi has potential in replacing commercial systems. They have shown PiCloud can effectively replicate cloud data center architectures even though it does not use resources most effectively as it uses a bulky operating system. This has the implication that a more lightweight operating system may offer improvement. In the case where the cloudlet perfoms well, Satyanarayanan et al[24] have shown that these portable cloudlets can have important uses as they can even save lives. Other high level issues such as that of trust will need to be addressed. A pressing matter that our system will have to deal with is mobility. Pitoura and Bhargava[21] have shown that in services hosted in the our cloudlet that require little to no latency in detecting disconnections, it necessary varying degrees of modes of operation, that is, have varying modes of being "disconnected". Clients can be considered "disconnected" to a higher degree with decrease in bandwidth and can be considered "disconnected" to a lesser degree with an increase in bandwidth. Finally, the transfer of content from the cloudlet to the client should be initiated by the client. The cloudlet should not "push" content to the a client when it is available, it should wait for the client to

"pull" content. The client should be responsible to ensuring that it did receive the data. This guards against the loss of data to be sent to the client because it was disconnected.

3 Cloudlet design

3.1 Overview

This chapter presents the design choices that went into the creation of the cloudlet and the file service that accompanies it. It also explains the need for separation of the main cloudlet platform and the services. A brief description of the cloudlet platform and the actions it supports is provided.

3.2 Design goals and restrictions

The cloudlet should be robust, that is, it should be able to register and not crash in the case of frequent connections and disconnections. This is crucial as users of mobile devices are likely to move in and out or range. Also, the hardware should be used optimally to ensure maximum performance. Common functions such as searching should be optimal, however, there has to be a balance between computational efficiency and battery use. The components should not result in high battery consumption. The cloudlet will be used for supporting interactions between a relatively small number of users, therefore algorithms of $O(n)$ where n is the number of users, should be sufficient. The implementation of functions and/or software that already exists should minimal (if any). Third party libraries which may possibly be optimized, should be used to reduce the amount of work. This is a platform that should be able to support numerous services therefore is should be easy and quick to introduce new services.

Development time is limited therefore an abundance of alternative machine to machine communication mechanisms, algorithms, databases and services cannot be explored. Also, computation power, storage space and battery life are limited. This means that the services developed cannot be computation and memory intensive. The range of the WiFi adapter is not large, the signal is weak and as a result, it is interrupted by walls. A solution to this would be use the mobile devices as WiFi repeaters, however, this function will not be developed due to it's complexity and the existing time constraints. The chosen operating system for the Raspberry Pi is

Arch-Linux Arm. This choice is largely because Arch-Linux ARM is lightweight compared to the Linux distributions which are compatible with the Raspberry Pi. This choice, however, introduces the restriction that all packages need to be compatible with Arch-Linux ARM.

3.3 Communication

The communication between these the clients and the cloudlet is not done through raw sockets, that is, a custom protocol in not built on top of the TCP/IP stack. Instead, a messaging system that uses the publish-subscribe messaging pattern is used. Specificaly, client-cloudlet communication is done through the MQTT protocol. The reasons for using MQTT is that it has fast "response and throughput, and low battery and bandwidth usage"[13]. In addition, MQTT has a small code footprint[6]. Using an existing MQTT library allows us to reuse trusted software with a numerous features. The current MQTT broker that is used is Mosquitto, an open-source broker that implements version 3.1 of the MQTT protocol. ZeroMq has been investigated as an alternate technology. ZeroMq is a messaging library which not only supports the publish/subscribe pattern but also the push/pull, and router/dealer patterns. One of the problems discovered, however, was that ZeroMq is not easy to setup for receiving responses at the client side as compared to mosquitto. In addition, the documentation was found to contain the following:

```
There is one more important thing to know about PUB-SUB sockets:
you do not know precisely when a subscriber starts to get
messages. Even if you start a subscriber, wait a while, and
then start the publisher, the subscriber will always miss the
first messages that the publisher sends
```

In an attempt to avoid unexpected behaviour, ZeroMq was not pursued further. It is worth noting that even though ZeroMq exhibited strange behaviour, it had the advantage that it had support for various languages and had extensive documentation. Other technologies of a similar nature have not been explored. The reason is that MQTT and mosquitto fulfill the requirements. There is no time to explore alternatives which can only offer miniscule improvements.

3.4 User identification

The cloudlet requires a local network in order to facilitate for communication between the various devices. The network is brought into existence by creating a WiFi access point. The IP addresses in the network are dynamically allocated using a DHCP server (dnsmasq). However, the identification of users in the cloudlet using the network assigned IP address is not suitable. Consider the following scenario: There are three users A, B and C within a cloudlet and are identified with 10.10.0.1, 10.10.0.2 and 10.10.03 respectively.

- A (10.10.0.1) shares file with only B (10.10.0.2). C (10.10.03) does not have access to the file.
- Both B (10.10.0.2) and C (10.10.03) disconnect. The shared file is still on the cloudlet.
- B and C reconnect, however, B is allocated the address 10.10.03 and C, 10.10.0.2 by the DHCP server.

This will result in user C having access to the file shared by A with B. A solution to this is to identify users using the WiFi MAC addresses. This is vulnerable to MAC address spoofing. However, addressing MAC address

spoofing is beyond the scope of this project. Users are identified using their MAC address and chosen username. The two strings are concatenated and used as an identifier for the messaging broker, mosquitto. This approach means that devices are connected to the cloudlet if they are connected to the messaging broker. This makes it possible to detect connections and disconnections quickly as mqtt can detect connections and disconnections within 1oooms.

3.5 Programming Language

Time is limited, therefore a programming language already known by the system developers had to be used for development. This would ensure software quality and save time. The languages to choose from where Java, C++ and Python. The chosen language was python. Using python ensures that the code is readable which makes maintenance easier. Python code is also platform independent. There are also a large number of third party library in addition to the large standard library which can be used.

3.5.1 Tools

Model B of the Raspberry Pi is used to host the cloudlet. It was chosen because it has a better set of features compared to model A, these features include larger RAM. Model B+ was not used as it was not released at the time the project started. The differences between the three models can be found at the raspberry pi website http://www.raspberrypi.org/. Model B of the Raspberry Pi is affordable and has extensive support on the Internet compared to other single board computers thereby making it more attractive. The power source chosen is the RAVPower 3rd Gen Mini 3200mAh Portable Charger. It is affordable and can be recharged. The chosen WiFi adapter is a Realtek Semiconductor Corp. RTL8188CUS 802.11n WLAN adapter. The combination of the adapter, power source and the Model B is portable. The package which is used for for creating wireless access points is hostapd. The package is the standard for creating WiFi access points in Linux. A custom version which has been developed by Jens Segers is used. The source code can be found at https://github.com/jenssegers/RTL8188-hostapd. All other versions of hostapd do not work because the chipset on the Wi-Fi adapter is not compatible with the default WiFi drivers on a Linux system. The custom version of hostapd uses the rtl871xdrv driver. The package is released under the GPL v2 license. A DHCP server was needed for the created local network, dnsmasq was chosen. This is because it is small and it is designed for small local networks. The use of dnsmasq saves memory as other official dhcp servers such as isc-dhcp-server require more memory.

3.5.2 Starting cloudlet

The raspberry pi that hosts the cloudlet does not have a display. This makes it difficult to start and stop the cloudlet. The cloudlet has been made to start on boot when power is provided for the device. This approach requires little physical interaction from the owner thus a display is not necessary. This makes the proposed cloudlet more affordable. The 'systemd' package in ArchLinux is used to start the cloudlet process and it's various components when the Raspberry Pi boots. It is chosen because it is already installed by default and is used by other system tools thus cannot be uninstalled. Choosing the order in which the components start is as follows: (1) The network interface used on the Raspberry Pi is assigned an IP using 'ifconfig wlano 10.10.0.51', (2) the DHCP server is started using 'systemctl start dnsmasq', (3) the wifi access point is created using the custom hostapd as follows "hostapd /etc/hostapd/hostapd.conf" and the last step starts the cloudlet by starting

the controller on port 999 using ‘python2.7 controller.py -p 9999‘. All the above actions are carried out by a script which executes when the computer boots. The script is not executed until the network interfaces are ready. This order ensures that starting the cloudlet never fails as certain components are ready. The cloudlet has a fixed IP address therefore clients need not spend time discovering it.

4 Implementation

This chapter provides information on how the cloudlet is implemented. Information on how to create a service for the cloudlet is also included. We also provide information about the various classes which play a vital role in the cloudlet platform.

4.1 Component structure

Figure 4.1.1 shows the class hierarchy and components which make the cloudlet. The system has been segmented into numerous logical components. There is a management component, controller and communication & event handlers. The responsibility of the controller class, as it's name suggests, is to control the various parts of the cloudlet. It ensures that components of the cloudlet start running at the appropriate times. It is the entry point for the system. Its responsibility is to receive a port number and start the mosquitto event handler (labeled mqtt event handler in figure 4.1.1) and the communication handler on the specified port. The MQTT protocol is used for communication between the clients and the cloudlet. MQTT uses a publish-subscribe messaging pattern. This has the implications that communication between clients and cloudlet uses channels. Clients can subscribe to channels and receive data which is published by the cloudlet on those channels. The communication handler is responsible for subscribing to the appropriate channels for the cloudlet, these channels are used by clients t publish data or send requests to the cloudlet. These channels allow the following actions:

- Cloudlet sending connection status to client.
- Client requesting a service list.
- Client requesting a connected user list from the clients.

The service manager and the user manager form the management component. Their responsibility is to keep track of connected users, available services and perform other actions related to services and users. These

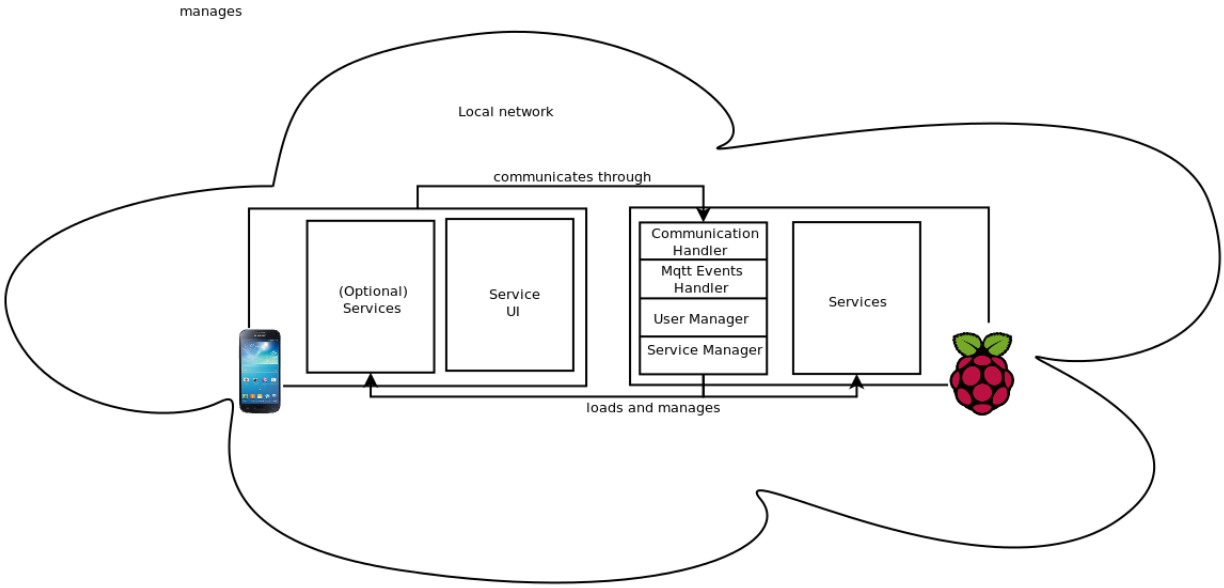

Figure 4.1.1 Overall view of the components of the cloudlet. The important relationships are shown through lines. (Images: Samsung Galaxy S4 Mini[29] and Raspberry Pi Logo[2]).

actions differ between the two components in that make up the cloudlet management system. For instance. the service manager is responsible for keeping track of available services, loading them and other management tasks for services. On the other hand, the user manager keeps track of online users and the services they are using. Cloudlet services can be physically located on the cloudlet and or the clients, and those clients can be mobile devices or laptops. Figure 4.1.1 shows a service as being in the intersection of front-end (clients) and the back-end (cloudlet on raspberry pi). Finally, the mosquitto event handler class is designed to extend mosquitto. It detects client connections and disconnections.

4.1.1 Mqtt Event Handler

Mosquitto, as stated above in section 3.3, is an open-source broker that implements version 3.1 of the MQTT protocol. Mosquito does not have a mechanism that allows applications to hook into such that they are alerted of connections and disconnections. However, the application writes messages on the standard error stream when there is a connection or disconnection. The mosquitto event handler is a class created to make it possible to automatically capture client connections and disconnections. This is done by allowing the mosquitto event handler class to start a mosquitto process and capture it's error stream. The data on that stream is used to alert the communication handler class of connections and disconnections. The readiness of the standard error stream, that is the availability of data on that stream, is determined by using the "select" system call which is available for all Unix-like and POSIX-compliant operating systems. It has been mentioned that the cloudlet is created to be capable of hosting multiple services. The design of the mosquitto event handler class takes this into account. It uses the observer pattern. This is the pattern where subscribers (services) are enabled to register with and receive notifications from a provider, the mqtt event handler class. Using this pattern makes it possible for services to register with the mosquitto event handler class and they will receive notifications in the case where a client connects or disconnects on the cloudlet.

4.1.2 Service

The service can be located on the raspberry pi or on the phone. This is illustrated in figure 4.1.1. Services are represented in the system through the service class. It contains information about a service such as it's name and description. The user manager and service manager are designed to grant or deny service requests. They have been implemented in this manner because future releases of the cloudlet should allow the cloudlet owners to ban certain users from using certain services. This functionality is not accessible with this release as the provision of such information to the cloudlet is not within the scope of this project of the frontend. The implementation of the services on the cloudlet requires communication between service and clients to not go through the communication handler. The only communication that goes through the communication handler are the actions clients can make which belong to the base cloudlet, that is, they belong to no service. These actions are listed in section 4.1. When clients request a service, they are allocated an IP and port to connect to using TCP/IP sockets. This is because services cannot use the mosquitto channels for communication as channels require the payload to not exceed 260MB. The mobile device has the ability to host services therefore it should advertise the hosted services such that cloudlet is aware of them. The names chosen for services need not be descriptive and because of this, each service has a string which acts as it's metadata. The string looks as follows:

```
``Name=somename;CloudletV=somenumber;
Description=somedescription.;authors=somename;
Copyright=Copyright someyear somename;
Website=https://example.com".
```

This information should be sufficient in the identification of services by humans. In addition, since this metadata is sent as one string. It removes the need for the client to request various fields which it needs.

4.2 Client connections and actions

The client is identified using a combination of the mac address and their chosen username. In order to join a cloudlet, the client must first connect to a WiFi access point named "CloudletX". The client should then connect to the mqtt broker, mosquitto. The address of the broker is fixed and it is "10.10.0.51:9999". The mqtt protocol requires clients to identify themselves and the clients should use a combination of the mac address and username. These should be formated in the form "<mac address>|<username>", and we shall refer to this combination as the identifier. After connecting to the MQTT broker, the client should then subscribe to the following channels:

- client/connecteduser/<username> - All messages received on this channel are identifiers of users connected to the cloudlet.
- client/service/<useraname> - All messages received on this channel are identifiers of services available on the cloudlet.
- client/serviceuserslist/<username> - All messages received on this channel are lists of users who use a service whose name was provided when the request for the list was made.
- server/login/<name> - This channel is used once, when the client attempts to connect to the cloudlet. The message received on this channel is the connection status.
- client/service_request/recvIP - This channel is used to provide clients with an ip and port to use to connect to the requested service.

- client/service_request/<identifier> - This channel is used for requests made by other clients for a service hosted on this client.

The cloudlet will provide the client with 2 seconds in order for it to subscribe to the channels. After which, a response will be received on the client side on the channel "server/login/<name>". The response can either be OK , UDUP or MDUP. OK means that the client is connected. UDUP means that the client has a duplicate username is will not be connected to the cloudlet. MDUP means that the client has a duplicate mac address and will not be connected to the cloudlet as a result.

The are channels used for client outgoing traffic. These channels are used for the requests made by the clients to the cloudlet. These actions are the requesting of a service list, requesting a list of connected users from the cloudlet and requesting a list of users of a service. There are other actions such as advertising services and requesting a service. The following table shows the channels, the payload one must send for each action and their role.

Table 4.2.1 Channels for sending making requests from client to cloudlet, the expected payloads and roles explained.

Channel	Payload	Role
server/serviceusers	servicename + "|" + name	Used for requesting connected to cloudlet using the specified service
server/connectedusers	name	Used for requesting connected to cloudlet
server/servicelist	name	Used for requesting list of available services on cloudlet
server/useservice	identifier + ";" + servicename	Used to request a service.
server/service	services json string	Used to advertize services hosted on the client.

4.3 Creating a service

4.3.1 Cloudlet

The services and code are separated on the cloudlet, that is, the code for the base of the cloudlet and the services is not on the same folder. The services which reside in the cloudlet are located in the folder "services". This makes it easy to identify and be able to remove services when necessary. In order to create a service in the cloudlet, one must create a subfolder inside "services". The folder name should be the name of the service and have no spacing. Figure 4.3.1 shows the folder structure of a cloudlet with a service named "file_sharer". Only two files should be created in the service folder. The files are "__init__.py" and "description.txt". All service source code must be located in "__init__.py" and class that has the same name as the service has to be created as it is the entry point to the service. It is good practice to create all helper classes within "__init__.py".

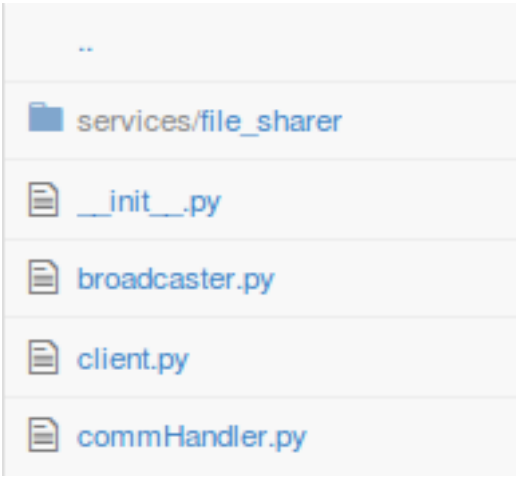

Figure 4.3.1 Folder structure of cloudlet hosting "file_sharer" service.

The description text file contains metadata of the services. These fields are listed and explained in table 4.3.1. The following is a snippet of the description file of the file sharing service:

Listing 4.1 Description of file sharing service

```
Name=file_sharer
Description=A service for sharing files with co-located friends.
Authors=Zola Mahlaza <adeebnqo@gmail.com>
Website=http://adeebnqo.github.io/cloudlets
CloudletV=1.4
Copyright=Copyright 2014 Zola Mahlaza
```

As mentioned, the fields in the above snippet are explained in table 4.3.1. These fields all serve a purpose.

Table 4.3.1 Service description fields.

Name	This is the name of the service. No spaces should be exist in the service name.
Description	This is a description of the service. There is currently no maximum length. The client UI should take this into account.
Authors	Creator(s) of the service.
Website	Url to the web page containing more information about the service.
CloudletV	Version of the cloudlet this service is compatible with.
Copyright	Copyright notice.

All services developed for the cloudlet have to abide by the interface illustrated in figure 4.3.2. The start method should start the service and return, it should not block. It is suggested that one use multiple threads. The "request_service" method provides the IP and an open port the service's server socket is listening on. These values are to be returned in the form "<IP>:<Port>". These values will be used by client when connecting to that service.

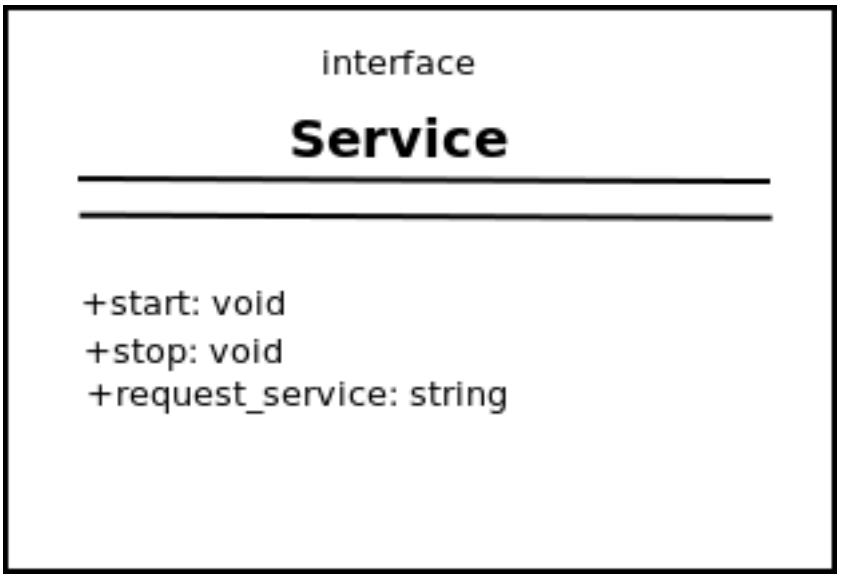

Figure 4.3.2 Interface for cloudlet services.

The following is an example of a service that implements the interface shown in figure 4.3.2. It should be noted that all other non-relevant methods and local class variables are not shown.

Listing 4.2 File Sharing service which implements the service interface.

```
class file_sharer():
        def request_service(self):
                (host,port) = self.sockt.getsockname()
                return '{0}:{1}'.format(host,port)
        def stop(self):
                self.sockt.close()
        def start(self):
                t = threading.Thread(target=self.wait_connections)
                t.daemon = True
                t.start()
```

4.3.2 Client hosted services

Services on the mobile follow a similar approach when it comes to development. The difference lies how the cloudlet discovers them. Services which are located on cloudlet are loaded into execution by the service manager. However, if services are located on the client side then it is necessary for them to be advertized so that the service manager can discover them. In order to advertise services, client should send the list of services they host. The list should contain a description of all the services the client hosts, listing 4.3 shows the format of the JSON string which is to be sent using the channel "server/service".

Listing 4.3 JSON string which is to be sent from client to cloudlet to advertize client hosted services.

```
{
        "username":<username>,
        "macaddress: <mac address>,
        "services": ["Name=<name>\nDescription=<description>.
                     \nAuthors=<author list>\nWebsite=<URL>\n
                    CloudletV=<version number>\nCopyright=<copyright notice>"]

}
```

4.4 Conclusion

The platform at hand uses a service-oriented architecture. The base of the platform supports a limited number of actions. The capabilities of the platform can be further expanded through developed services. Communication between the cloudlet and all clients is done through the MQTT protocol. This is a lightweight machine to machine messaging protocol for use on top of the TCP/IP protocol. The services which extend the cloudlet can be physically located on the mobile device or the device that hosts the cloudlet. Figure 4.1.1 shows the physical location of services in the cloudlet. The services which extend the cloudlet need to use raw sockets. That is, they should create a server using raw sockets and not use the MQTT protocol as it has limitations in the message payload size. When a user request a service, they should be provided an IP and port. Those details will be used to communicate. The implementation of the cloudlet allows quick loading and removal of services. It allows services to be hosted on the Raspberry Pi and clients. It is designed to quickly detect connections and disconnections. It ensures that humans can be able to tell what a service does and provides them with information where they can get more information about the service. It uses a well established messaging system to ensure that communication between client and cloudlet is easy and well separated as client can perform numerous actions.

5 Use cases

5.1 Introduction

This chapter presents the examples uses of the proposed system. We also present the literature which supports the need for such services. The role of this chapter is to present services which will be used in setting up tests that will address the research questions.

5.2 M-novels in libraries

Stephen King, in his memoirs as a writer, writes that "Books are portable magic"[15, p. 96]. A number of people cannot experience this magic unfortunately. This might be due to under resources libraries, no Internet access to access free literature or perhaps unfamiliarity with the culture of reading. The last problem, unfortunately, cannot be easily solved and this project will not address it. This project will focus on allowing people to have access to literature. The Shuttleworth Foundation's m4lit project explored the potential of mobile devices for publishing and authoring books. It made use of an m-novel written by Sam Wilson, entitled Kontax. The goal of the m4Lit research project was to investigate how South African teens responded to Kontax. Walton's[31] details the observations and results of the project. Walton[31] uses the success of Kontax to conclude that "teen mobile literacies and school literacies might be bridged to some extent for urban teens through the provision of mobile-accessible texts"[31, p. VII]. We propose a use case, motivated by the m4lit, where m-novels will be be hosted on the cloudlet and mobile devices users can download and be able to consume them. This m-novel distribution service will be characterized by the following:

- A large number of clients connected to the service at one time. It is then necessary for us to see if there will be client who cannot be catered for when the number of users increase.
- There will be, on average, a large time in between downloads per user. It is highly unlikely that users will download numerous without pausing for at least 50 seconds in between each download. We will view how this affects battery consumption.

This use can be generalized such that these m-novels can be distributed in a number of locations. These locations include taxis, informal and formal convenient stores, etc. Portability of the cloudlet becomes necessary for locations such as taxis.

5.3 Ad-hoc co-located collaborative work

Mobile computing systems have become an integral part of people lives. Cellphones, a form of a mobile computing system, have had helpful applications in education, citizen journalism, etc. Cell phones have been used to improve health services and delivery of health care, for instance Praekelt Foundation's TxtAlert has assisted in the reduction of the Lost To Follow Up rate, that is, the rate of "patients that do not attend clinic appointments regularly"[1]. We We present a use case, originally presented by Luyten et al[17], which focuses on different area which is also of great importance. The area is co-located mobile collaborations. Consider the scenario where a number of people working on an architectural project meet at the the construction site to to discuss the preliminary sketches. The goal of the meeting is to "establish a consensus about the final style of the building for this construction site and collaboratively edit and annotate the current draft sketches"[17, p. 507]. Further suppose that the project leader has sketches on a laptop. The project leader's goal is to present his ideas to the colleagues and require them make annotations and suggestions to improve the design of the sketches. Using traditional methods, a single person can be delegated to capture suggestions from people and write them down. The second approach would be to rotate the laptop among the parties involved, each person adds annotations and passes the device to the next party. One can attempt to improve this by allowing all participants to access the sketches via a cloud based application. This is a far more effective approach, however, it raises issues with file ownership and might not be applicable in areas where there is no network coverage. We propose the use of the cloudlet by people working on an architectural project meeting at a construction site to discuss building sketches. The cloudlet will create a network that will allow clients to connect, make annotations and perform any other action necessary actions. The data will be centralized on the cloudlet and can be further migrated to a different storage medium such as a laptop. This service will be characterized by the following:

- Small number of users.
- The time between actions taken by each client is small. A large number of requests can be made by one client in a short period of time. This is because users are likely to have a number of suggestions, these interactions are likely to be short and the time between action made by a client will be small.

5.4 Conclusion

We have presented two use cases that will determine our tests. That is, the characteristics of each use case make up the majority of the variables which will be considered in the testing. These variables are (1) Number of clients, (2) Time between action made by each client and (3) distance between client and cloudlet.

[1] http://www.praekeltfoundation.org/txtalert-partners.html

6

Results and discussion

6.1 Databases

The use two use cases presented in chapter 5 require the ability to store data in a database. Two databases are tested, these databases are MariaDB and the BerkerlyDB as discussed in chapter 2, section 2.2. MariaDB is a drop-in replacement for MySQL[1]. The most important database actions for this project are insert, retrieve and delete. We have also tested the actions available on the file sharing services in an attempt to observe how they affected by the database used. This is important because timing of the file sharing service actions is done on the client. This allows to us to see if the database speeds are not compromised by the time it takes to transfer the data back to the client from the cloudlet. The actions of the file sharing service, in addition to the aforementioned database actions, are update, retrieve publicly accessible files, retrieve files shared with user and delete all data shared by user. We have tested all actions on the Raspberry Pi. The work done by Baumann et alRef Works:106 has already revealed that among the storage engines they tested InnoDB performed the worst. They also reported that MyISAM was among the best performers, therefore we have used MyISAM as the storage engine for the MySQL based database, MariaDB. Time is determined using CProfiler and in doing so, we obtain the CPU time and not the wall-clock time. The actions are executed 1000 times each. This has lead us to use a relatively small file of about 521 kb. This is because the tests were conducted in an environment that had to be monitored and an increase in file size resulted in an increase in test times that made it difficult to monitor the tests. Figure 6.1.1 shows the average operation time over 1000 runs for the labeled databases. Figure 6.1.2 shows the average operation time over 1000 runs for the file sharing operations using the labeled databases.

[1] https://mariadb.org/en/about/

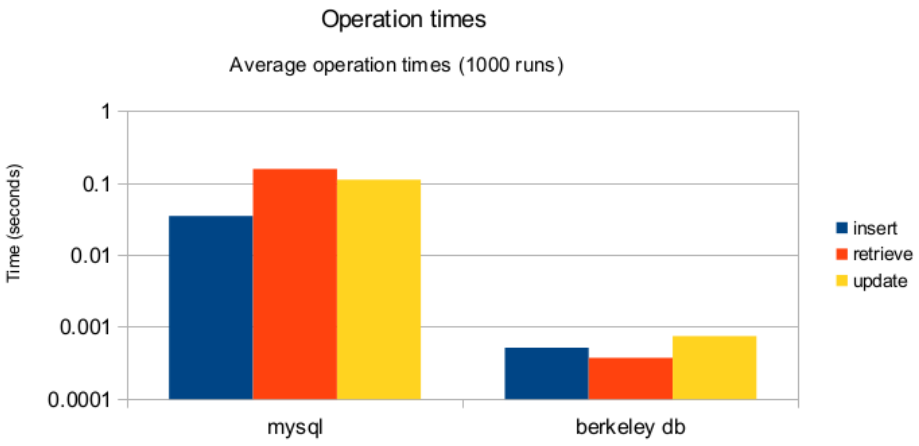

Figure 6.1.1

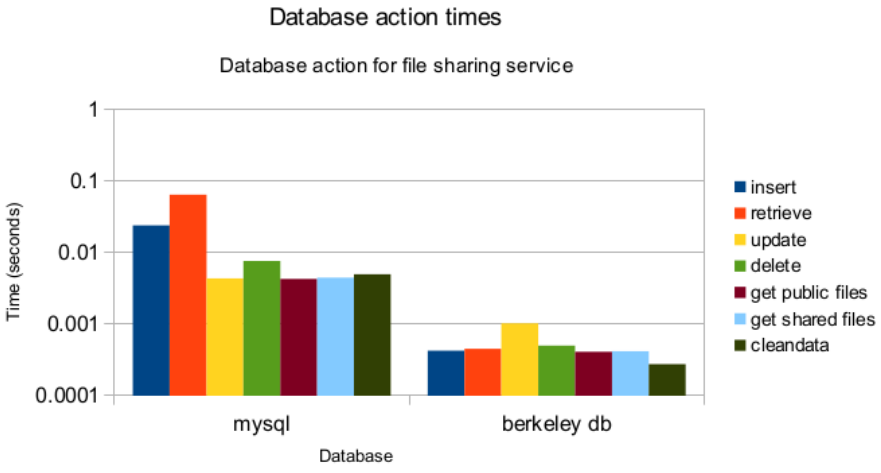

Figure 6.1.2

The key-value pair database performs than the relational database on all actions. The retrieve action, is faster in when a key-value pair database is used. However, in a relational database the opposite is true. This means that if one uses a key-value pair database, it might worth using a cache to avoid the increased time that is associated with the retrieve function. BerkeleyDB performs better than MariaDB and the transfer of data from the cloudlet to the client does not diminish the difference.

6.2 Bandwidth

The bandwidth decreases with an increase in distance between client and cloudlet. This fact cannot be changed, however, we can try to observe the extent to which the bandwidth is affected. The experiment simulates a single client. This client will upload and remove two files from the cloudlet. There will be no pause between these actions. The actions are each performed 50 times. The files used and their sizes are as follows:

- blue.jpg - 384,5 kB
- audio.mp3 - 2,0 MB

There are three tests conducted per action. The variable that is changed is the distance between the client and the cloudlet. Test 1 occurs at 2.87 meters, test 2 at 4.92 meters and test 3 at 4.51 meters. This can be visualized using the ad-hoc co-located collaborative work use case as presented in chapter 5. Our goal is to see whether or not removing and adding annotations, possibly accompanied by multimedia, would not take a long time.

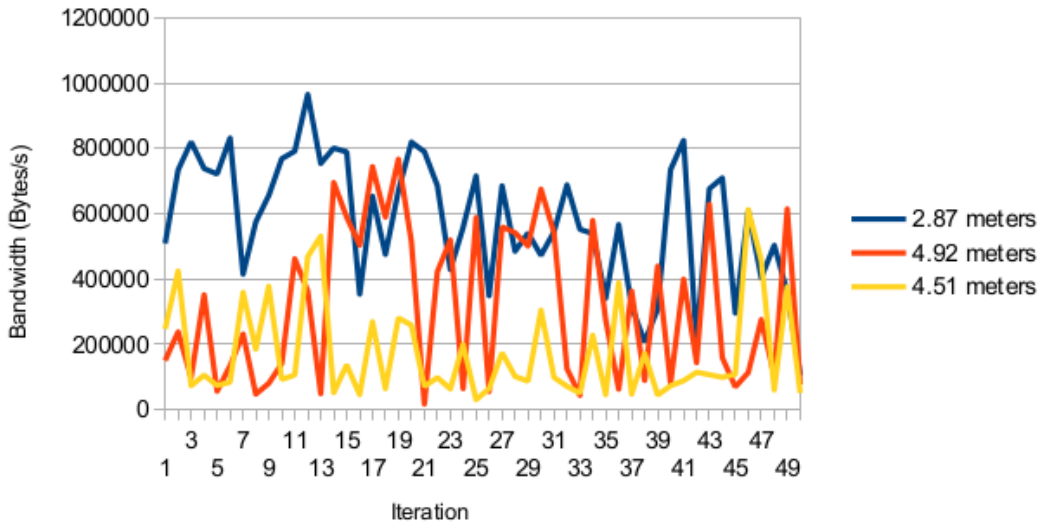

Figure 6.2.1 Bandwidth changes for the upload action over 50 iterations

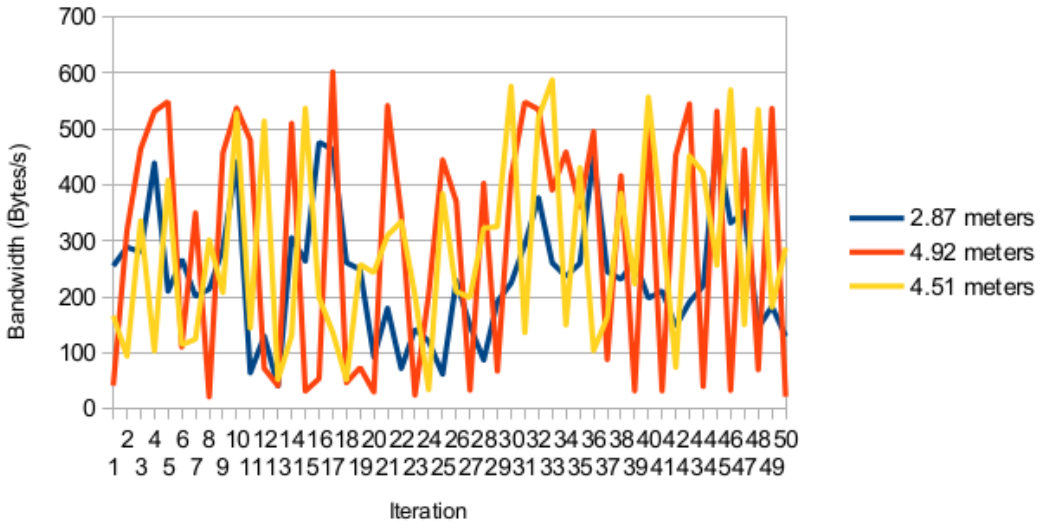

Figure 6.2.2 Bandwidth changes for the download action over 50 iterations

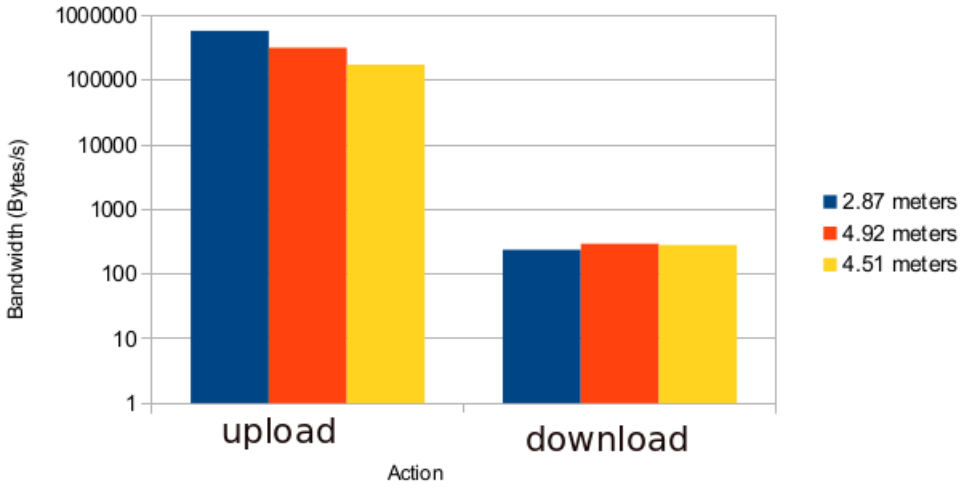

Figure 6.2.3 Average bandwidth for the download and upload action over 50 iterations at different distances.

The increase in distance greatly reduces the bandwidth when performing the upload function. The observed change is that bandwidth dropped from 0.58Mb/s to 0.17Mb/s. This shows that at a close distance, data transfers will be fast. The download actions differently. There is no drop, however, there is an unexpected rise in average bandwidth. This rise is minuscule, this suggests that for the download function will not be affected very much by change in distance.

6.3 Power consumption

The use cases presented in Chapter 5 determined our testing variables. We have used simulations to determine the current being drawn by the raspberry Pi. The clients are simulated using an 64bit computer running Xubuntu Linux. The computer has Intel Core i5-370 CPU × 4 and 4 GiB RAM. The number of clients and time between action is set multiple times and the power consumption is measured using an Ammeter. The actions each client can make are:

- getaccessiblefiles - Retrieves all files to which the requester has access to.
- upload - Upload a file with it's metadata to the cloudlet.
- remove - Removes a file with it's metadata from the cloudlet.
- download - Download file from cloudlet.
- requestusersoffilesharer - Request users who are connected to the cloudlet
- checknewfiles - Check if there are new files which requester has access to in the file sharing service

Figure 6.3.1 shows how the battery consumption changes when there is no pause between actions taken by the clients, that is, client make continuous requests. Figure 6.3.2, however, reports a small number of clients who have long pauses between actions affect power consumption. Figure 6.3.3 shows power consumption where there many clients and the time between actions is also increased. Finally, figure 6.3.4 shows power consumption where there are small clients connected, making actions and the time between actions is very small.

The power consumption never exceeds 0.5 A. The used power supply is the RAVPower 3rd Gen Mini 3200mAh Portable Charger. The time can be approximated as follows:

$$t \approx \frac{3200mAh}{500mA} \tag{6.1}$$

$$\approx 6.4h \tag{6.2}$$

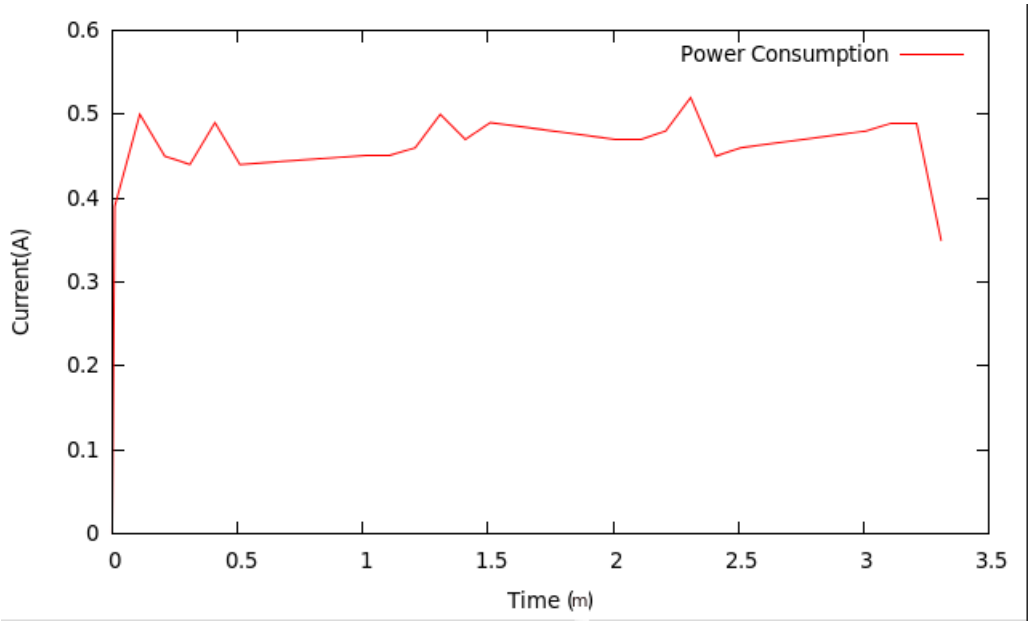

Figure 6.3.1 Power consumption when one client continuously performs the file sharing services' upload and remove actions.

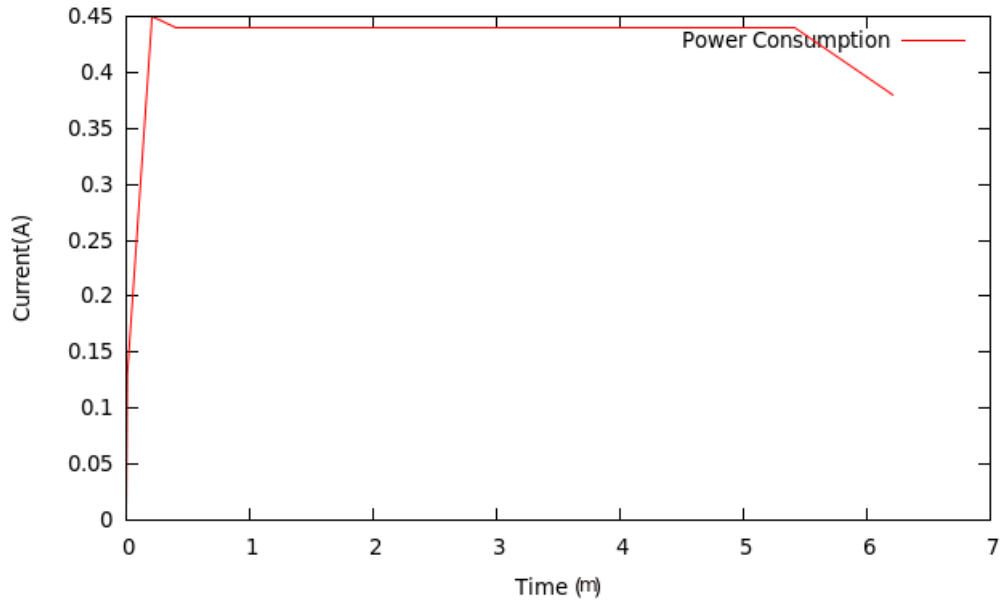

Figure 6.3.2 Power consumption when five clients who are performing each of the file sharing service's actions in parallel with a pause time of 1m:30 between actions.

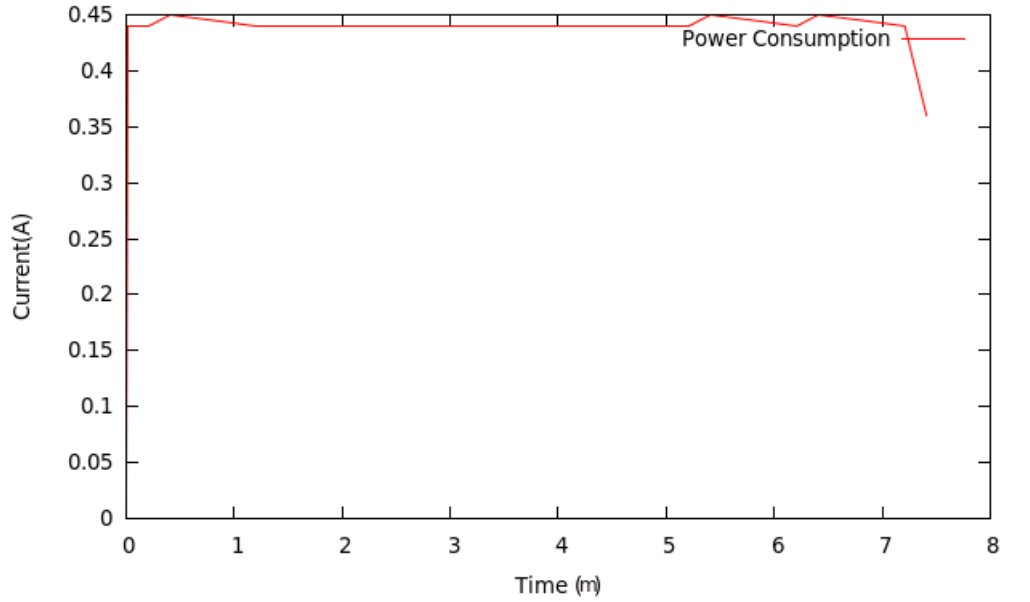

Figure 6.3.3 Power consumption when 20 clients who are performing each of the file sharing service's actions in parallel with a pause time of 2m between actions.

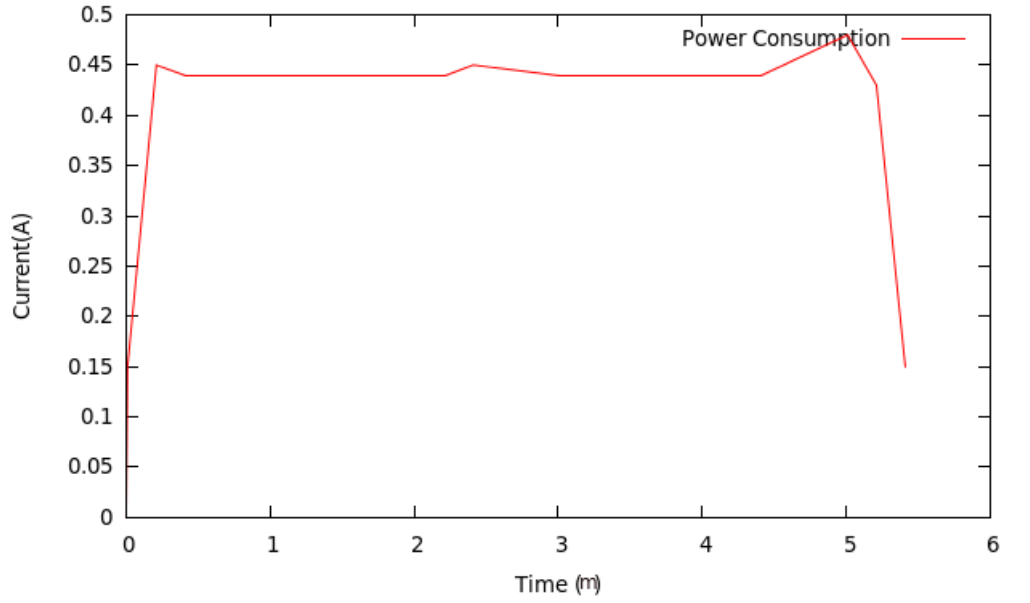

Figure 6.3.4 Power consumption when two clients who are performing each of the file sharing service's actions in parallel with a pause time of 15seconds between actions.

7 Conclusion

Our results reveal that the cloudlet will be able to last approximately six hours even when in constant use, that is, serving multiple requests from clients without pause. This means that the cloudlet is practical for the use cases identified in chapter 5 as far as power is concerned. We state say that because, firstly, co-located mobile meetings in outdoor locations cannot last for more than 2 hours. Therefore, a cloudlet charged once can be able to support multiple mobile co-located outdoor meetings. Also, since South African libraries are open from 08h30 to 18h00. The hosting and distribution of m-novels could be possible during that time, that is, the battery life will last slightly longer than 6 hours as it is less likely to be always serving requests from clients. In our power consumption tests, we simulated clients on a laptop computer. These clients could make all actions available on the file sharing service that has been developed. The avaliable actions are listed in section 6.3. Each client randomly chose an action it had to perform out of that list. It is possible that some clients only performed the actions which result in less power consumption. It should be noted that is unlikely, albeit possible. We also conclude that the well suited database for a cloudlet is the BerkeleyDB. This is because it performs the basic database operations (insert, update and delete) faster than MariaDb. It also requires small disk space for installation. It has been made clear that memory storage is not abundant on the Raspberry Pi. The use of external disks is not ideal. This is mostly because they require a power supply and may greatly increase battery consumption of the whole system. On the other hand, the data can be stored in compressed form thus a large SD card with negligible power consumption can be used. The raspberry PI can be used with an SD-card of size of up to 32GB[7] thus if data is compressed, large quantaties of data can fit into 32GB. The implemented approach requires the clients to compress files before transferring them to the cloudlet. This shows that the issue of small available memory can be addressed without an increase in the amount of power drawn. However, since portability of the cloudet is not always necessary, for instance, in the use case of m-novel distribution in libraries it can be sacrificed. The practical ways of dealing with small amount of memory available by sacrificing portability are without encryption:

- One can make use of a large USB stick or USB hard drive for storage. Large USB sticks or USB hard

drives require a large amount of power therefore the Raspberry Pi can be powered using larger non-portable power sources.

- One can use USB hard drives which have their own power source..

The maximum range that can be covered by the inexpensive WiFi adapter used is approximately 5 meters. The data transfer rates drop significantly with increase in distance between clients and the cloudlet. The drop will not have much of an impact to most interactions. However, it poses a threat to the transfer of large multimedia files. Co-located interactions are sometimes short and in such situations, it is impractical to transfer large files to and from the cloudlet from a distance close to 5 meters. The relatively slow speeds might have been caused by WiFi interference which is due to outdoor microwave links, cordless phones, Bluetooth devices, wireless video cameras, fluorescent lights, WiMAX devices or even bad electrical connections. This study did not sweep for these likely sources of interference. In this study, we attempted to use powertop, the package designed to measure a computer's electrical power consumption, to measure power consumption. The reports generated by powertop were found to not include power consumption but other diagnostics information. This is because ArchLinux ARM does not have the Advanced Configuration and Power Interface (ACPI). ACPI contains device drivers which create the possibility for the operating system to be able that monitor devices such as the battery. One should avoid relying on powertop for power consumption.

7.1 Future work

In this study, we observed that BerkeleyDB perform better than MariaDB. It might useful to determine if all key-value pair databases perform better than relational database in a cloudlet. If so, which database's queries result in lesser power consumption. We also did consider doing the compression on the Raspberry Pi. Implementing compression on the cloudlet side has implications on the power consumption. It might be useful to determine which compression schemes are fast on the Raspberry Pi and how do they affect battery consumption. Also, the biggest problem that is study reveals is that the increase in distance greatly reduces transfer speeds. Determining affordable ways to improve this might be helpful. Finally, this study did not address security in cloudlets. The introduction of encryption and other security mechanisms could broaden the areas to which cloudlets have uses.

References

[1] Leonard M Ah Kun and Gary Marsden. Co-present photo sharing on mobile devices. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 277–284. ACM, 2007.

[2] Paul Beech. Raspberry pi logo. Retrieved from http://goo.gl/cKHfTn. [Online; accessed 28 October 2014].

[3] Phil Belanger. 802.11n delivers better range. Retrieved from http://www.wi-fiplanet.com/tutorials/article.php/3680781, May 31, 2007. [Online; accessed 14 August 2014].

[4] Andrew Black and Jon Inouye. System support for mobility. In *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*, pages 129–132. ACM, 1996.

[5] Christian Cachin, Idit Keidar, and Alexander Shraer. Trusting the cloud. *Acm Sigact News*, 40(2):81–86, 2009. doi: 10.1145/1556154.1556173. URL http://dx.doi.org/10.1145/1556154.1556173.

[6] Organization for the Advancement of Structured Information Standards (OASIS). Mqtt. Retrieved from http://mqtt.org/. [Online; accessed 28 August 2014].

[7] Raspbery Pi Foundation. Frequently asked questions. Retrieved from http://www.raspberrypi.org/help/faqs/#sdMax. [Online; accessed 28 August 2014].

[8] Jim Garamone. American forces press service. army tests land warrior for 21st century soldiers. Retrieved from http://www.defense.gov/news/newsarticle.aspx?id=42267. [Online; accessed 22 August 2014].

[9] Gartner. Data center. Retrieved from http://www.gartner.com/it-glossary/data-center/. [Online; accessed 28 October 2014].

[10] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Middleware 2009*, pages 83–102. Springer, 2009.

[11] Richard Harper, Tim Regan, Shahram Izadi, Kharsim Al Mosawi, Mark Rouncefield, and Simon Rubens. Trafficking: design for the viral exchange of tv content on mobile phones. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 249–256. ACM, 2007. doi: 10.1145/1377999.1378015. URL http://dx.doi.org/10.1145/1377999.1378015.

[12] D. Richard Hipp. Appropriate uses for sqlite. Retrieved from http://www.sqlite.org/whentouse.html. [Online; accessed 28 October 2014].

[13] Kathleen Holm. Using mqtt protocol advantages over http in mobile application development. Retrieved from https://www.ibm.com/developerworks/community/blogs/sowhatfordevs/entry/using_mqtt_protocol_advantages_over_http_in_mobile_application_development5?lang=en. [Online; accessed 28 August 2014].

[14] Huawei. Wifi repeater. Retrieved from http://www.huaweidevice.co.uk/devices/huawei-ws320/. [Online; accessed 28 October 2014].

[15] Stephen King. *On Writing: A Memoir of the Craft*. Simon and Schuster, 2000.

[16] Chen Ling, Ming Chen, Wenjun Zhang, and Feng Tian. Ar cloudlets for mobile computing. *International Journal of Digital Content Technology and its Applications, AICIT (JDCTA)*, 5(12):162–169, 2011.

[17] Kris Luyten, Kristof Verpoorten, and Karin Coninx. Ad-hoc co-located collaborative work with mobile devices. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 507–514. ACM, 2007.

[18] Bradley Mitchell. Wireless standards 802.11 b, 802.11 a, 802.11 g and 802.11 n: The 802.11 family explained. Retrieved from http://goo.gl/m8XRS8, 2010. [Online; accessed 14 August 2014].

[19] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. 2013. URL http://arxiv.org/abs/1307.0191.

[20] Michael A. Olson. Selecting and implementing an embedded database system. *Computer*, 33(9):27–34, 2000.

[21] Evaggelia Pitoura and Bharat Bhargava. Dealing with mobility: Issues and research challenges. page 4, 1993.

[22] Thomas Reitmaier, Pierre Benz, and Gary Marsden. Designing and theorizing co-located interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 381–390. ACM, 2013. doi: 10.1145/2470654.2470709. URL http://dx.doi.org/10.1145/2470654.2470709.

[23] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[24] Mahadev Satyanarayanan, Grace A. Lewis, Edwin J. Morris, Soumya Simanta, Jeff Boleng, and Kiryong Ha. The role of cloudlets in hostile environments. *IEEE Pervasive Computing*, 12(4):40–49, 2013.

[25] Margo Seltzer. Berkeley db: A retrospective. Retrieved from http://sites.computer.org/debull/A07sept/seltzer.pdf. [Online; accessed 13 August 2014].

[26] Margo I. Seltzer and Michael Olson. Challenges in embedded database system administration. In *Proceeding of the Embedded System Workshop*, pages 29–31, 1999.

[27] Sorin Stancu-Mara and Peter Baumann. A comparative benchmark of large objects in relational databases. In *Proceedings of the 2008 international symposium on Database engineering & applications*, pages 277–284. ACM, 2008.

[28] Fung Po Tso, David R White, Simon Jouet, Jeremy Singer, and Dimitrios P Pezaros. The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures. In *Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on*, pages 108–112. IEEE, 2013.

[29] Unkown. Samsung galaxy s4 mini. Retrieved from http://goo.gl/l6FIyG. [Online; accessed 28 October 2014].

[30] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: Bringing the cloud to the mobile user. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pages 29–36. ACM, 2012. doi: 10.1145/2307849.2307858. URL http://dx.doi.org/10.1145/2307849.2307858.

[31] Marion Walton. Mobile literacies and south african teens: Leisure reading, writing, and mxit chatting for teens in langa and guguletu. *Shuttleworth Foundation*, 2010.

[32] Marion Walton, Gary Marsden, Silke Haßreiter, and Sena Allen. Degrees of sharing: Proximate media sharing and messaging by young people in khayelitsha. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*, pages 403–412. ACM, 2012. doi: 10.1145/2371574.2371636. URL http://dx.doi.org/10.1145/2371574.2371636.

[33] Himanshu Yadava and Mike Olson. *The Berkeley DB Book*, volume 79. Springer, 2007.