RemoteME: Experiments in Thin-Client Mobile Computing

by

Vipul Delwadia

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2009

Abstract

Mobile phones are ubiquitous, however they are vastly underpowered compared to their desktop counterparts. We propose a technique to play potentially resource intensive games over a network, and provide a prototype system called RemoteME which implements this technique. We also explore the responsiveness requirement for systems of this nature, establish benchmarks for responsiveness via user studies. We evaluate our implementation by measuring its responsiveness and comparing it to this benchmark.

Acknowledgments

I would like to thank my parents and sister for their support throughout this Masters.

Many thanks to Stuart Marshall and Ian Welch for your excellent supervision. I would be nowhere without your guidance and wisdom, your help is much appreciated.

Thanks to my friends in Memphis, including the students from Honours 2008 and 2009. Thanks to Stephen and Paley for their help and support, always willing to provide assistance or insight.

Thanks to everyone in ELVIS for making me feel guilty, but more importantly thanks for giving me useful advice and feedback from time to time.

Thanks to the fourteen participants of the user experiment, your involvement was crucial.

Finally, thanks to my friends down town constantly reminding me how I should give up University and join "The Real World". I'm glad I ignored their advice.

Contents

Chapter 1

Introduction

People want to perform more complex tasks on their mobiles, and we are interested in collaborating with desktops and servers to bridge the gap. Mobile phones are now commonplace, and increasingly are being used for tasks other than human-to-human communication, and support tasks previously associated with more powerful desktop computers or servers. However, mobile devices in general are still underpowered compared to their desktop counterparts, so there are some tasks still out of reach.

One solution is to virtualise the entire interface, using technologies such as Virtual Network Computing (VNC) [42] and Remote Desktop Protocol (RDP) [25]. These solutions turn the client – the mobile device – into a simple display and input device by forwarding user interactions. However, this simple approach doesn't handle situations where one wishes to partition the functionality between client and server.

As an alternative, our technique for client/server computing is to use a specialised client designed to support a particular domain of applications. In this technique the client application can execute code as part of the remotely running program. The whole system is application specific, and allows for finer access controls. In this system the client can provide functionality such as locally storing files, and caching application resources across sessions. Server migration is also an advantage, allowing clients to

save state locally and resume it on any server.

1.1 Hypothesis & Contributions

We hypothesize that it is possible to build a system using our technique, with acceptable responsiveness for a domain such as gaming, that divides responsibility between mobile clients and a server.

This project has three main contributions:

- A study into acceptable responsiveness, by identifying the time constraints which a system needs to meet to support the required domain of gaming.
- The description and exploration of our technique, and determining whether or not it is feasible given the timing requirements.
- The proof-of-concept system RemoteME, a prototype created which utilises our technique. This system has acceptable responsiveness for games with simpler graphics, but does not perform well enough for games with completed graphics.

1.2 Applications

The first scenario is preserving digital content, specifically video games. Various organisations around the world, including the National Library of New Zealand, are interested in preserving games so that they can be played at a later time. Our technique can be used to provide access to the preserved digital content, for example in a museum setting, where an exhibit (the server) can run the games that the patrons (clients) wish to play.

The second scenario is customers test driving many applications – such as games – without requiring multiple installations. A consequence is that

the locus of control over the intellectual property is kept with the owners, making them more amenable to allow this kind of system where they wouldn't allow traditional systems. This can even be extended to allow games which require significantly more computationally expensive AI to be played on the underpowered mobile device.

1.3 Structure

The structure of the document is:

Chapter 2 discusses digital content preservation in more detail, and outlines related work in the remote control and responsiveness areas.

Chapter 3 describes our methodology as we address the questions raised by our hypothesis. It also details the pilot study performed to determine the effect of delay upon users.

Chapter 4 details the user study to determine if network delay has a negative effect on game play and to determine a network delay benchmark.

Chapter 5 provides system detail about our prototype remote control system called RemoteME.

Chapter 6 evaluates the performance of RemoteME against the benchmark from the user study, as well as comparing the performance of the RemoteME client against the native application.

Chapter 7 summarises the project. We also list our contributions, and suggest areas for future work.

Chapter 2

Background

This chapter covers the background of our research. First we look at the preservation of digital heritage, and previous work in this area, and then propose our remote control technique as a solution. We then cover work which looked at establishing and testing responsiveness, which is key in setting a benchmark to evaluate our technique. Finally we explore existing remote control solutions, and how they relate to our technique.

2.1 Preservation

Following on from our scenario about preserving digital content, this section explores work already conducted in this field.

UNESCO [43] describes the preservation of digital heritage as "an urgent issue of worldwide concern". Increasingly, cultural institutions are recognising that the preservation of and access to digital culture falls within their mandate. Internationally, a number of National Libraries are involved in projects to develop preservation solutions, such as Europe's PLANETS project [31] and the Library of Congress' "Preserving Virtual Worlds" [19].

The context for our research is interest in preserving access to New Zealand's early video games. The motivation for considering NZ games

was that from the mid-1970s the NZ software industry was prolific – partly in response to import restrictions accompanied by incentives for the local electronics industry. Locally produced video games were popular entertainment in the 1980s and provided an introduction to the digital age for New Zealanders. Many games contain distinctive NZ content [39]. While early games have been kept by local enthusiasts, their future is uncertain, as they are now unplayable due to technical obsolescence and hardware and software deterioration.

The two main technical approaches to preserving software are emulation and porting. With an emulation approach, the original hardware and software architecture of the original machine is emulated and the archived software runs directly on top. With a porting approach, the original software is either manually or automatically rewritten for a new hardware and software architecture. Emulation is widely seen as easier to achieve and less costly than porting [32]. Emulation has been a popular approach for hobbyists who want to maintain access to their favourite games as well as institutions. For example, the Netherlands National Library sponsored the development of IBM's Universal Virtual Computer [20] in order to try and preserve access to historic digital images and documents.

For historic games, one of the major barriers to archiving early video games is copyright [8]. Under the fair dealing provisions of the copyright act, researchers may not make more than one copy of a video game. A copy includes converting the program into a different computer language or code, otherwise than incidentally in the course of running the program. This is a major barrier to using migration to port the code for a video game to a new architecture. It is unclear if emulation is also an issue as the argument can be made that the existence of a version in memory at runtime is simply incidental.

Under an exemption under the National Library Act, the National Library is permitted to use such processes but only for video games that have been legally deposited, and there has only been a requirement to do

this since October 2006. Anything prior to this does not fall under this exemption so games developed before that date must be archived under the same constraints faced by researchers.

Providing access to these preserved games is a problem even if emulation is allowable under the law in New Zealand. Games are archived to allow future generations to access them and play them for both study and for pleasure. We would like to see people able to access preserved games in the context of the National Library or museums such as Te Papa.

There are three problems to be solved. First, how can we work around potentially only being able to have one copy of a game. Second, even when working with deposited games that potentially can be made available in more than one copy at a time, how do we prevent easy copying of these games and violation of copyright? Third, how do we make the process of accessing games lightweight so that specialised hardware is not required.

One solution to these problems is to use a thin-client approach to provide access on mobile devices to games running on servers. The games are instrumented to export graphics-related updates to the mobile device and the mobile device sends input to the server.

Portable devices such as netbooks and mobile phones are getting closer and closer to becoming our primary (and occasionally only) computing device. Despite the increasing device capability and network capacity, it is still challenging to provide certain classes of applications that require significant memory or computation. One such example application domain is that of games. Mobile devices are still underpowered compared to standard desktops widely available in the consumer market (Table 2.1). A remote gaming service could allow users to connect with their mobile devices using a small client application, say in a museum setting. Multiple users could gain access, one at a time, to a single copy of the game running on the server.

Our architecture addresses the three problems outlined above. First, the number of copies in use at any one time are controlled by controlling

	Nokia N95	Dell Optiplex 755
Processor	Dual 332 MHz (ARM11)	3.00 GHz (Intel x86)
Memory	160 MiB	2 GiB
Screen Size	2.6 in	17 in
Screen Resolution	240x320	1280x1024

Table 2.1: Comparison of a typical mobile device and computer, specifically the devices used in the later experiments.

execution at the server. Second, because users do not have direct access to the game, it is not downloaded to their device, so the copying of copyright material may be prevented. Third, the mobile devices don't need to be as powerful, because the server does the heavyweight computation, and the mobile device is only required to do graphics output and input capture.

2.2 Studies

In the previous section we looked at our technique for remote control, however without a measure of sufficient responsiveness the success of this technique is unknown. In this section we look at a number of studies towards setting a benchmark for responsiveness, starting with Miller's work on response times between users and computers [23].

Miller proposed a number of different minimum response times corresponding to various activities, based on his opinion as a behavourial scientist. Among the activities presented, the lowest (and most relevant) response time proposed is 0.1 seconds, i.e. 100 ms, for the response to control activation such as the click of a typewriter key.

While Miller's work is important in establishing a guideline very early in the computing industry, studies more recently have looked at the effects of latency on user performance. In these studies, the latency is the time difference between the server and client's version of the current status of the objects in the game. The architecture these studies explore is described

in Section 2.3, specifically Figure 2.2.

One study [34] looked at the effect in the real-time strategy game *WarCraft III* [5]. In this case they found that for long and semi-automatic tasks, such as constructing structures, there was no correlation between latency and performance. For more involved tasks, mainly combat, there is also no correlation for both unit scores and proportions of games won by a client affected by latencies ranging from 0–1600 ms. While these results are promising, it is difficult to be conclusive about the results as their study had a limited number of test subjects (2–4).

Another study [4] looked at the effect of both packet loss and latency on a first person shooter game, Unreal Tournament 2003 [10]. This study found that generally packet loss or latency has no noticeable impact on a player's performance. However, the one area where it is noticeable is in precision shooting, where the correlation between induced latency and hit fraction is very high, and the mean hit accuracy suddenly drops approximately 35%. While this study had a significant amount of data (over 200 experiments), this specific test only comprised of two test subjects doing a ten minute experiment three times each.

Unfortunately, neither of these studies established any guidelines for what might be acceptable latency for users. Moreover, both of these studies relate to multiplayer games where a central server only coordinates game play between game programs running on players' desktops. It is uncertain whether the results can be extended to our situation where the client does minimal processing and the game executes on a central server.

2.3 Client/Server Architectures

There are two common network-based architectures, thin client and traditional client/server, which we will discuss in this section. These architectures form the basis of our experiments described in the next chapter.

A thin client architecture [44] resembles Figure 2.1. The client is "thin" in the sense that it is simply providing input and output, and the server is responsible for all the application processing. Here the *latency* is the amount of time a single message takes to transmit from a client application to a server application (or vice versa). In the example, the latency is $t_L = t_1 - t_0$.

The *delay* is the amount of time from a request for an action to that action being initiated. The example has the request being made at time t_0 and the action initiated at t_1, so the delay is $t_D = t_1 - t_0$. In a thin client system the delay is quite likely to be the same as the latency, due to the architecture.

Response time is the amount of time from a request for an action to that action being performed. As in the example, if a request is made at t_0 on the client, and that action is performed at t_2 on the client, then the response time is said to be $t_R = t_2 - t_0$.

Figure 2.2 shows how these concepts relate in a traditional client/server architecture, where some of the application specific processing is shifted to the client. The client is now more than just input/output, and can in fact be a standalone application. Most multiplayer games are examples of a traditional client-server architecture.

2.3.1 Thin-client Computing

A potential solution is to remotely provide applications on a central server via thin client computing, not only for typical mobile games on a mobile device but also for playing fully-fledged, resource intensive games which cannot be played on a mobile device currently.

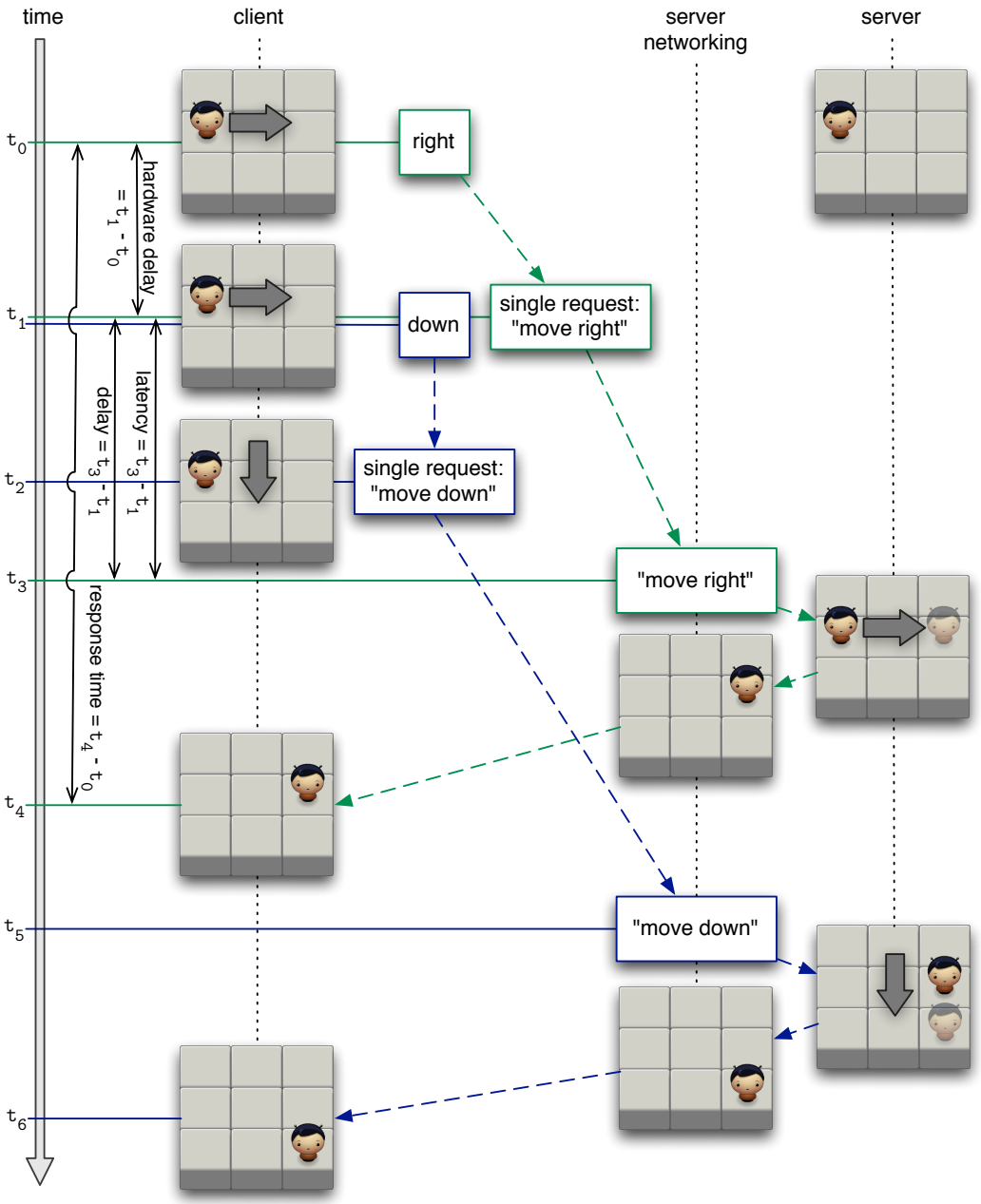


Figure 2.1: An example of request handling in a *thin client* architecture. The hardware button corresponding to *right* is pressed on the client at t_0. The client software receives this request from the hardware at t_1, and forwards it to the server who receives it at t_3. This gives the hardware delay as $t_1 - t_0$, and delay and latency are equal in this case at $t_3 - t_1$. The multithreaded server finishes its computation and sends the frame to the client, who receives it at t_4, while computing the next request. The response time is thus $t_4 - t_0$. Note that the client can forward more key presses while the first is being processed, for example *down* is received and forwarded by the client software at t_2.

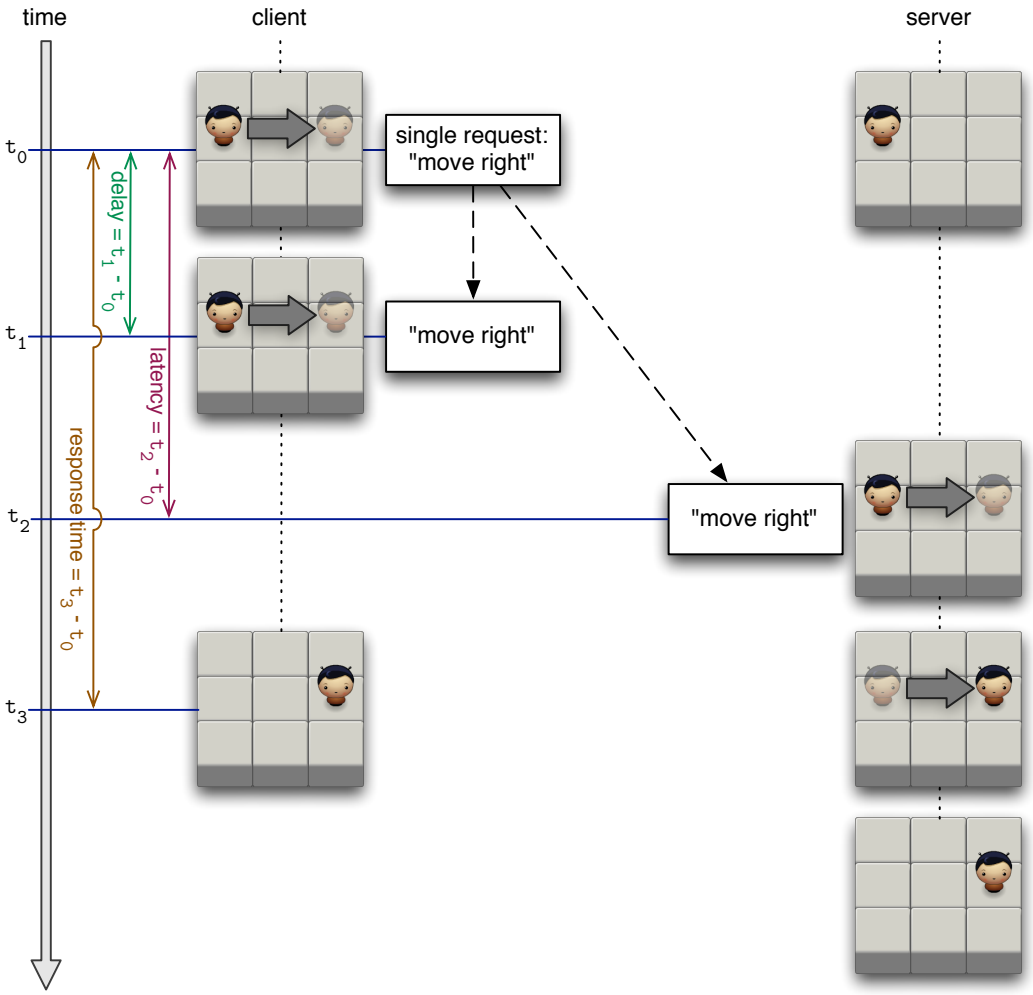


Figure 2.2: This example is of request handling in a traditional *client/server* architecture. On the client the "move right" request is at time t_0. This request is both processed locally and forwarded on to the server. At time t_1 the client processes the request, giving the delay as $t_1 - t_0$. The latency is $t_2 - t_0$, as the request reaches the server at time t_2. Once the client has performed the "move right" action, at time t_3, the response time is $t_3 - t_0$.

Thin client computing has been reborn as *cloud computing*. Applications such as office suites, photo editors and presentation tools are moving from running locally to running in the cloud with access via web browsers. This offers several advantages over running applications locally [2, 7, 14, 22] including reduced costs and increased reliability, scalability, security, and perhaps most importantly, mobility.

Additionally, it may also offer a compromise solution to the problem of protecting copyrighted digital artifacts (such as the executable binaries of historical and present-day applications) while still affording access to users. Storing game code on the server reduces barriers, allowing for a compromise between the two extremes and giving a more attractive solution to copyright holders. Copyright holders who are not willing to give up all their rights can use this solution, and can still give the ability to play the games.

Recently there has been interest in developing thin clients that can be used to access a range of games hosted on a server. For example, OnLive [28] or Games@Large [16] are thin clients used to access a range of games and aim to lessen the hardware requirements for running a game by offloading activities such as 3D graphics rendering to a server that is accessed over broadband.

While these services provide the capability for remote computing, the performance requirements for accessing remote applications via local user interfaces are not fully understood. Hosting games on a server and displaying them locally inevitably introduces network delay that will affect users' response times. This network delay is the time between user actions and the system's response, e.g. the time between pressing a button and the result of pressing a button which causes something to happen on the screen. Too much network delay and applications such as games may be unplayable, making all the other benefits of a thin client approach moot.

2.3.2 Remote Desktop Solutions

Remote desktop software allows a user of a less powerful machine to access their desktop that runs on a more powerful machine across a network. There are a large number of remote desktop software applications available for a range of platforms including Windows, Linux/Unix, Mac OS X, embedded devices and mobile devices. However, most use variations of one of the following protocols for communication: Remote Desktop Protocol (RDP), Remote Frame Buffer (RFB) and X11.

RDP [25] is the technology underlying Microsoft's remote desktop software for the Windows Server and Client operating systems. Once a connection is made between an RDP client and server, the server sends commands to render windows to the client, which are low-level graphics commands. User input via keyboard/mouse is sent back to the server from the client accordingly. The client updates the screen when the server requests an update. The client can cache *Glyphs* and bitmaps by size on memory (small) or disk (large) [46]. Note that this is much the same for Citrix's ICA, upon which RDP is built [40].

RFB is the protocol that is used by VNC (Virtual Network Computing). The RFB protocol operates in a client/server manner, but the server sends the display to the client as rectangular snapshots of the remote desktop (or frame buffer). The snapshots are simply bitmaps representing rectangular sections of the remote desktop. Instead of sending just the raw bitmaps, the rectangles can be encoded in a number of ways. The server pushes updates to the client, and the client can cache only when using Copyrect [40].

X [33] is a client-server system, where the server is the application rendering the window, and the client is the application requesting to be rendered. When a window is to be rendered, the X client application sends the X server a request to create a window with an id. Further, when the client wishes to fill that window with content, it refers to it by id. The content is sent as high-level graphics, and the server pushes updates to the client. Client caching is unlikely as it is application/toolkit specific [40].

There have been at least two attempts at trying to reduce the amount of traffic required to use X across a network, namely NX and LBX. NoMachine's NX technology [27] utilises a number of different strategies for reducing the bandwidth of networked X sessions. One aspect is to compress the data sent with zlib [12]. In addition to compression, NX selectively caches data received to effectively reduce the need to re-send data which is static in the short-term. NX is considered a worthy alternative to standard network based X.

Low Bandwidth X (LBX) [45] is a proxy server (`libxproxy`) which caches often used data, such as window properties and font metrics. It also allows for compression of images and general stream data. `libxproxy` sat between the X clients and server, and didn't require changes to existing programs. Unfortunately, LBX provide small performance enhancements over slow networks, and more recent versions of X system have been optimized such that LBX's benefits were superseeded [30].

2.3.3 Streaming Solutions

Aside from the main remote desktop protocols discussed above, researchers are exploring the use of streaming solutions to remotely control a system.

Streaming Video

An example application is the model helicopters trial by the Merseyside Police in Britain. An operator controls the helicopter and receives feedback on the helicopter's surroundings via CCTV cameras mounted on the helicopter itself [3].

Zhuang and Wang [48] have developed an "IP-based real time video monitoring system with controllable platform". Their approach was to have a camera and microphone connected to an embedded system that compresses the incoming data and sends it to a server. When a client comes along, they request the stream from the server. Upon receiving the

compressed data, the client decompresses and displays the data as graphics. Cell phones, along with standard desktop machines, are supported as clients.

Games@Large

Games@Large [16] is a technology to provide interactive media streaming over a network. This technology, as indicated by the name, is intended to be applied to games. The Games@Large framework consists of a server, which is executing the game, and a client which displays the game. The data for the display is streamed via two different techniques.

The primary technique is to intercept calls to the host system's underlying 3D graphics architecture, either DirectX [26] or OpenGL [17], and forward these calls onto the client. The client is responsible for the rendering, and therefore is required to have a dedicated graphics processor.

The secondary technique, used when the client doesn't have a 3D renderer and dedicated graphics processor, is to render the calls on the server. The output of the rendering is then encoded and transmitted as a video stream.

The Games@Large experiments were conducted with two participants playing two games, Red Faction (a first person shooter game) and Sprill (a puzzle game). The server (game host) was connected to the clients (notebook computers) via 802.11g wireless. They measured the FPS[1] on the clients for both games, and the first-person shooter game had a mean FPS of 18.26 with standard deviation of 12.12. In comparison, the same game played natively on the server had a mean FPS of 59.23 and standard deviation 5.16. Further, they recorded the user observations/perceptions as they played the games. The participants gave the gaming experience between 4 and 5 (out of 5) in the Mean Opinion Score [15] scale, and that "there were some differences with the original game play but participants were not frustrated and could enjoy the game play".

[1] Frames per second, a measure of the render rate

Other Streaming Solutions

In addition to Games@Large above, there are a number of other services which specifically provide streaming gaming. OnLive is a streaming gaming service, where OnLive host the games, and they are played by the users on their TVs using set-top boxes, or computers via a web browser plugin [28]. The service aims to deliver 720p 60fps quality video over a 5 Mbps connection, or standard definition over a 1.5 Mbps connection. At this stage the service is only available to limited test participants, so the performance of this service is unknown.

A similar service from Gaikai, Streaming Worlds, provides streaming gaming in much the same manner, where the games are hosted on their servers, and played using a plugin in the client's web browser on a desktop computer [11]. Again, the service is not yet available to the general public so performance is unknown.

AMD in conjunction with OTOY have demonstrated a streaming solution similar to the previous two services, however their solution can serve many client types, such as smart phones and PDAs [1, 29]. Again, details are very limited so system requirements and performance are not known.

Finally, Spawn Labs have a set-top box which connects to existing console gaming systems and let the users connect over a network or the Internet to the console and play the games on their computer [35]. The client is the user's computer, and the server is the user's console gaming system which is located elsewhere. This system promises 720p video on a 2–5 Mbps connection, and standard definition video on a 500 kbps–1 Mbps connection. While performance is not known, their product is available for purchase at the time of writing this thesis.

2.4 Mobile Java

Our application area is mobile devices, and we have selected J2ME (Java Micro Edition) as the application platform. J2ME has widespread adoption among the mobile phone market, deployed on billions of devices [36].

J2ME is defined by two specifications, JSR118 – MIDP (Mobile Information Device Profile) and JSR139 – CLDC (Connected Limited Device Configuration). These two specifications provide an API which is similar to the standard Java API, and allows developers to create applications for devices which support J2ME. There are multiple versions of each specification, however we are interested in MIDP 2.1 and CLDC 1.1, and will assume these versions throughout this thesis. More detail on the API is contained in Chapter 5.

One of the issues with a traditional control system such as VNC and RDP is mapping from physical buttons to their on screen counterparts. J2ME abstracts this mapping, allowing applications to deal with commands rather than the physical buttons. Further, traditional control systems don't handle different screen sizes particularly well, whereas J2ME allows the application to easily adapt to the screen size and resolution. We can use these aspects of J2ME in our system to overcome the limitations of traditional control systems.

Chapter 3

Methodology

In this chapter we discuss our approach to tackling the three questions raised by our hypotheses. Section 3.2 provides the development methodology for this project.

Three experiments were carried out to determine if the prototype provided sufficiently responsive game play. Section 3.4 describes the pilot study conducted to determine the effect of delay upon the user experience. Section 3.1.2 describes the further study where users played games with artificial delays imposed. Section 3.1.3 describes an experiment to evaluate the performance of the RemoteME system when deployed on a local area network.

3.1 Overview

This project was conducted in three phases, described in the following sections.

3.1.1 Initial Development

The first phase of this project was to develop a prototype called RemoteME to demonstrate the feasibility of our thin client approach for playing 2D

games. We could not use any of the systems that were described in the background section such as OnLive because freely available implementations do not yet exist. At the end of this phase we had a working prototype to use to determine whether we could achieve acceptable response times.

3.1.2 Benchmark Creation

The second phase involved identifying the minimum delay for a usable remote gaming system as defined in Chapter 2. This involved two studies. The first was a pilot study with a small number of users playing a single game Asteroid Zone (Section 3.3) to enable us to iron out experimental issues. The details of this study can be found in Section 3.4.

The second study is larger, involving 14 users to determine statistically significant results. The results of the pilot study were inconclusive, thus a full user study was conducted, documented in Chapter 4. This study improves on the pilot study by testing volunteer participants, rather than ourselves, to alleviate the experimental issues. In addition, each participant plays a considerable number of games, and given the number of participants, this gives a much larger data set to perform analysis on, meaning that any correlations found have a greater statistical significance.

Further, a second game (in addition to Asteroid Zone) is tested, and issues with the Asteroid Zone, such as the death/re-spawn problem, are fixed in the user study.

Finally, the experiment is conducted entirely on the device, meaning that there are no network factors to consider. This also gives the ability to set the delay more precisely so that it is not subject to network issues such as jitter.

3.1.3 System Evaluation

The third phase consisted of further development and finally evaluation of the RemoteME prototype system. Performance of RemoteME for multiple

games was improved, and then evaluated against the benchmark range determined in the second phase, 75–150 ms. The system performed at the lower end of the range, achieving an average response time of approximately 77 ms for Asteroid Zone. Further optimisation has a high likelihood of improving this figure to below the lower threshold of 75 ms.

In addition to the main experiment with Asteroid Zone and RemoteME over WiFi, experiments with Bomber 2 were also conducted, as well as experiments over the 3G mobile Internet infrastructure. These experiments were disappointing giving no concrete results.

Finally, although our focus is on meeting the responsiveness requirement established by the experiments, RemoteME's performance in terms of device resources is still interesting, and is also explored in the evaluation.

Full details of the evaluation are in Chapter 6.

3.2 RemoteME Development Method

This project was conducted following a rapid development methodology. The major reason for this was the uncertainty associated with the requirements of the remote control system. Short iterative developments allowed us to adjust to the discovery of new requirements through experimentation. Using ourselves as unofficial "users", iterations were tested for speed and the user response fed back into the development.

3.3 Experiment Artifacts

Throughout the three experiments conducted during this project, there are a number of common artifacts used.

Asteroid Zone Asteroid Zone [9] is a simple shoot'em up space game. It consists of simplistic graphics, and a game play that requires the test subject to maneuver a space ship through an asteroid field (i.e. a set of moving asteroids), shooting the asteroids to score points. After a set of asteroids is destroyed, the player moves on to the next level – consisting of more numerous asteroids. If an asteroid hits the player's ship, then the ship is destroyed and re-spawns in the middle of the screen. The player starts with four lives, and is awarded another life every 500 points. Points are given for destroying asteroids, the smaller the asteroid the more points given.

Figure 3.1 shows a sample screen shot of the game. The game draws the ship and asteroids using simple polygons rather than special images.

Asteroid Zone was chosen specifically for its simplicity. The advantage of choosing such a game is that the nature of the game play requires the system to quickly respond to the player's actions, while at the same time the computation is sufficiently lightweight – and the information transmitted over the network sufficiently small – that the delay is a significant component of the overall response time.

The game has a simple set of controls, and players use the directional control to either spin left or right, or accelerate. A central button allows players to fire bullets. All objects wrap around the screen, when an object reaches an edge it appears on the opposite edge of the screen, with the same momentum and trajectory.

Mobile Phone The Nokia N95 was released in 2007 and represents a reasonably common yet sophisticated mobile phone currently on the market.

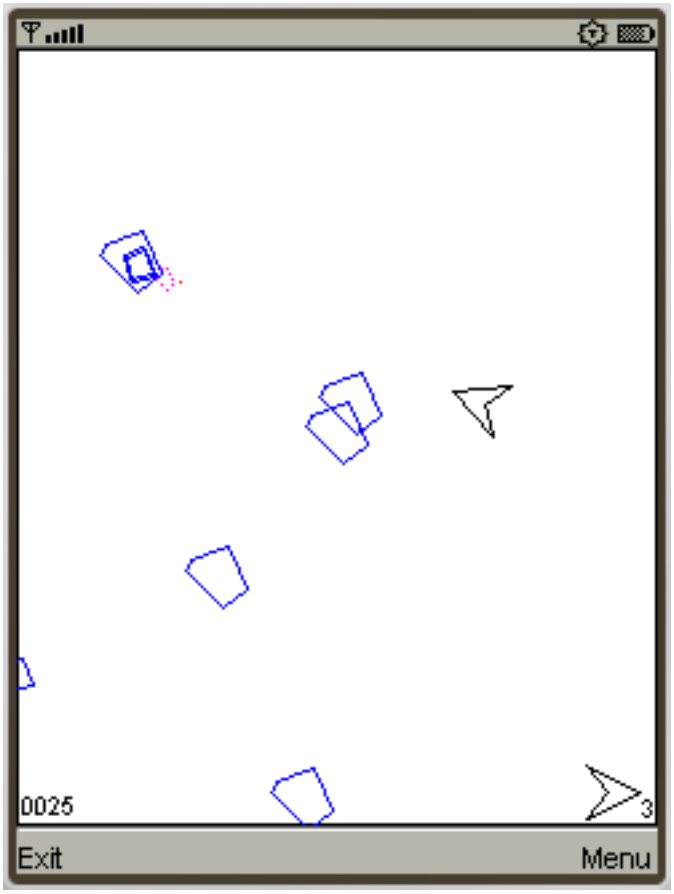


Figure 3.1: A screen shot of the Asteroid Zone game used during the pilot study. Despite the simplistic graphics and narrative, Asteroid Zone is a real-time game in the sense that noticeable lags between user input and the in-game response can detract from the game's playability.

Figure 3.2: The Nokia N95 mobile phone, with sliding keypad exposed. The Nokia N95 has a 12-key keypad, with additional soft keys and a four-way directional control surrounding a central button. This image was retrieved from the Wikimedia Commons and is licensed under the GNU Free Documentation License by the image's author: Asim18.

The phone has 64 MB of RAM and a 320x240 pixel display. The Nokia N95 has a 12-key keypad that can be hidden within the device and slid out as required. The phone also has additional soft keys and a four-way directional control surrounding a central button. Test subjects used the directional control and the central button to move and fire respectively.

Wireless Router Our experiments used a dedicated wireless router to avoid congestion and interference. The experiments were run in the same office as the wireless router, so distance was no more than a few metres and this was not varied during the experiment.

3.4 Pilot Study

We conducted a pilot study as a quick means of evaluating feasibility and discovering issues which may arise in subsequent experiments. This pilot study investigated the performance requirement for remote mobile gaming. The study used RemoteME, a prototype remote control system developed as part of the project, as the remote mobile platform (see Chapter 5 for details). The study experimented with introducing timing delays into RemoteME to observe the effect on the player experience. The pilot study's purpose was to identify how long the system could take in responding to a player's actions before the player's success (as measured by score) either was affected by the delay, or was perceived by the user as being affected by the delay.

3.4.1 Experiment

The experiment assessed how users are affected by various levels of delay which may be introduced by a remote control system. For this experiment a single 2D action game was chosen, because applications such as high action games will magnify the effect (if any) of user interface latency. The effect of delay on the user's performance was measured by both the user's score and perceived effect.

The experiment was carried out over the RemoteME system, where the network delays were added between the server and the WiFi router.

As a preliminary step, the delay variance of individual user action / system response pairs inherent in RemoteME, due to the wireless communication and packing/unpacking of commands, was measured. Over 1000 user action / system response pairs were monitored across 5 games. The resulting mean baseline RemoteME delay was 82 ms, with a standard deviation of 31 ms.

We analysed the fourteen participant's resulting data to see if there were correlations between the delays; decreases in the players' scores; and

the players' perceptions of whether the delay was noticeable and affected their game play.

Hypothesis

The hypothesis is that there is a correlation between the test subjects' scores and the delays, and between the responses to the following questions asked and the delay:

1. There was a consistent delay between my actions and the system's response.
2. The delay between my actions and the system's response negatively affected the playing experience.

The responses to these questions were measured using the Likert scale, where the response is between 1 (strongly disagree) and 5 (strongly agree).

Design

The experiment consisted of two participants playing two sets of the same game twelve games, each set separated by at least an hour of another activity.

For the pilot study, my two supervisors (Stuart Marshall and Ian Welch) were used as test subjects. Each subject played games of Asteroid Zone via RemoteME. A time delay of between 0 and 500 ms (in 50 ms increments) was randomly selected and added to the communication overhead between the RemoteME client and server via modifications to the network.

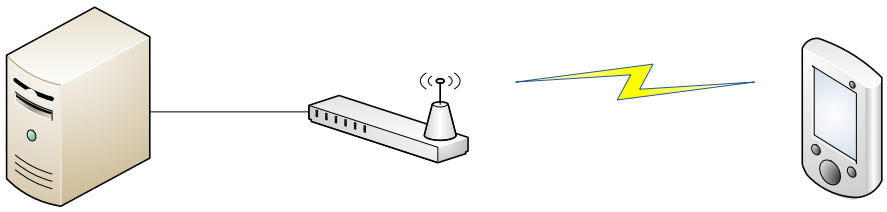


Figure 3.3: RemoteME architecture for the experiment. The server (a Dell Optiplex 755) running the application is connected on the wired network directly to a wireless access point, which the client (Nokia N95) is connected to via an unencrypted 802.11g WiFi connection.

The architecture of the system is given in Figure 3.3. The network delay is inserted using an application which can configure Traffic Control in the Linux kernel, `tc` [6]. The command to add the delay is:

```
tc qdisc replace dev eth0 root netem delay <x>ms
```

where `<x>` is the delay (in milliseconds) to be set. This command is executed on the server, and so the delay is imposed on the wired section of the network.

The player was not informed which time delay (if any) was used during any particular game. Instead, the players played the game to completion, at which point they were asked whether they thought any delay had a significant impact on their game play. The players' scores were also recorded.

Artifacts

The artifacts used for this experiment are described in Section 3.3, namely the game Asteroid Zone, the Nokia N95 mobile phone and the wireless router.

A RemoteME client was installed on the phone, and connected to a RemoteME server on a separate server machine via a single wireless access point.

In addition my two supervisors acted as test subjects for this pilot study. I was responsible for running the experiment (referred to hereafter as the *experimental controller*) and recording the scores, delays and perceptions of the two test subjects. This means that HEC approval was not required to conduct this experiment[1].

3.4.2 Procedure

While the main focus of the experiment was on how the controlled delay affected the two test subjects, it first needed to be identified if there was a significant *network* delay already present in RemoteME. This network delay is the amount of time the RemoteME client/server spends communicating over a wireless network and requiring some computation on each end to handle the traffic and instructions. Neither of these actions would be required if the game was running entirely locally on the mobile phone.

The experimental controller monitored RemoteME's performance via RemoteME's logging utility. This simple utility allows for measurements of the time difference between a single user action and the system's response, called the *response time* previously (Section 2.3). This response time was measured across all such interactions in five sample executions of Asteroid Zone. The experimental controller played these five games, and gathered these time differences without any controlled delay built into the wireless router.

This preliminary experiment differs from that with the test subjects since the test subjects experienced a constant delay added to all individual user actions in an entire game, whereas this experiment is identifying the variance in user action response times in a single game.

Once the RemoteME overhead was established we began the user experiment. Each test subject played two sets of twelve consecutive games of

[1]Human Ethics Committee (HEC) approval (`http://www.victoria.ac.nz/postgradlife/pages/pages_current_pg/ethics.html`) is required when involving other participants.

Asteroid Zone. Each set of twelve games was separated by at least an hour of another non-game-playing activity so that the test subjects didn't significantly improve their temporary playing ability between the first and last games. As well as this, before each of the two sets of twelve games, each test subject played three practice games natively on the mobile phone[2]. This was to ensure that the test subject's first few games in the set didn't suffer from the break in play by ensuring that the test subject had refamiliarised themselves with the game controls.

The sequence of actions during a single game are:

1. The experimental controller ran the wireless utility application to obtain and configure a random delay into the wireless router. The experimental controller recorded this delay without informing the test subject of the value. Since the random delay was selected from all possible values of the delay, it was possible for the test subject to encounter the same delay multiple times during a set of games, and to not encounter some delay values at all.

2. The test subject connected to the game server via the RemoteME client and proceeded to play a full game. Each game started out with four lives, although occasionally the test subject may play well enough to win an extra life.

3. At the completion of the game, the experimental controller decided whether to officially record the results of the game. We discovered this step was required after a practice game demonstrated a "flaw" in Asteroid Zone's play. When a ship dies, it re-spawns in the middle of the screen. However, if there is an asteroid in the middle of the screen, the ship automatically dies and re-spawns again. If the asteroid hasn't moved sufficiently far away, this triggers yet another death and the re-spawning cycle continues. We decided that if a

[2]This means that the game was not run via a RemoteME client/server; all computation was done entirely on the mobile phone without any network activity.

game involved more than one death via this auto-re-spawning death cycle, then we would discount the game since the score and perceptions would be significantly affected by an arbitrary event outside of the test subject's control.

4. If the experimental controller has decided to officially record the results of the game, then they record the final score next to the delay value, and ask the test subject to rate the following statements based on a five-point Likert scale (with the responses being: strongly disagree; disagree; neutral; agree; and strongly agree):
   - There was a consistent delay between my actions and the system's response.
   - The delay between my actions and the system's response negatively affected the playing experience.
5. If there were more games to play in this set, then the test subject and experimental controller went back to step one.

3.4.3 Experimental Issues

There are a number of issues with this experimental design that are improved in subsequent experiments.

Firstly, the choice of using my supervisors as test subjects may bias their answers to the Likert scale questions. We attempted to control any bias by two actions:

1. Ensuring that the test subject did not know of the specific delay values that they had encountered until after both sets of games were complete.
2. Ensuring that delay value for each encounter was selected randomly.
3. Ensuring that the author primarily responsible for RemoteME's coding was not a test subject.

Secondly, any such future experiment should significantly increase the number of games and sets so that each delay value has a sufficiently large population size to permit inter-population analysis. Currently, there aren't enough games for each delay value to allow for such in-depth analysis.

Finally, there could be potential issues with using a single mobile phone and a single computer, such as poor performance with either the phone or computer, or even that specific combination of computer and phone leading to adverse performance.

3.4.4 Results

This section presents the results of the pilot study. Correlations were calculated for the one independent variable (inserted delay) and three dependent variables (scores and player perceptions).

Combining the two games Asteroid Zone and Bomber 2 for the two test subjects together gives a Pearson correlation $r = -0.521$ between the controlled delay and the game score. This is shown in Figure 3.4. Splitting the games between the two test subjects gives r values of -0.630 and -0.411 respectively. While this indicates there is some negative correlation between controlled, inserted delay (hereafter referred to solely as *delay*) and the game score, this correlation is not as prominent as we had expected.

Similarly, combining the two test subjects' games for the purpose of calculating the Pearson correlation between the two Likert scale questions and the controlled delay gives a correlation $r = 0.686$ between Q1 and delay, and a correlation $r = 0.702$ between Q2 and delay (shown in Figure 3.5). Once again, splitting the games between the two test subjects gives correlations of $r = 0.674$ and $r = 0.689$ for Q1/delay, and correlations of $r = 0.716$ and $r = 0.700$ for Q2/delay.

This suggests there is some mild correlation between actual delay and the players' scores, and between actual delay and the players' perceptions of the effect of any perceived delay. However, the correlation was not as

strong as expected going into this pilot study. Future experiments can address this focusing on larger sample populations, investigate different levels of delay, different mechanisms for gauging the test subjects' impressions, or using confidence intervals to quantify the significance of the results.

Lastly, the mean delays and scores for various subsets of answers to the Likert scale questions were looked at. This data can be seen in tables 3.1 and 3.2. These tables divide up responses into "affirmative" and "not affirmative" categories, with the neutral answers falling into the latter category. Perhaps not surprisingly, games where test subjects' responded that they didn't affirm that the delay was noticeable or had an affect, had a higher mean score. Likewise the mean delay was less as well. However, it is not wise to read too much into these mean values due to the experimental limitations discussed earlier.

3.4.5 Summary

The purpose of this pilot study was to explore how delay affects the player's performance and perceptions, and to set a target which can be expanded upon in future. These early experiments suggest that there is not much room for permitting delays in the system (for computation and transportation), as small controlled delays deliberately inserted into the system did show some correlation with decreased player performance and perception.

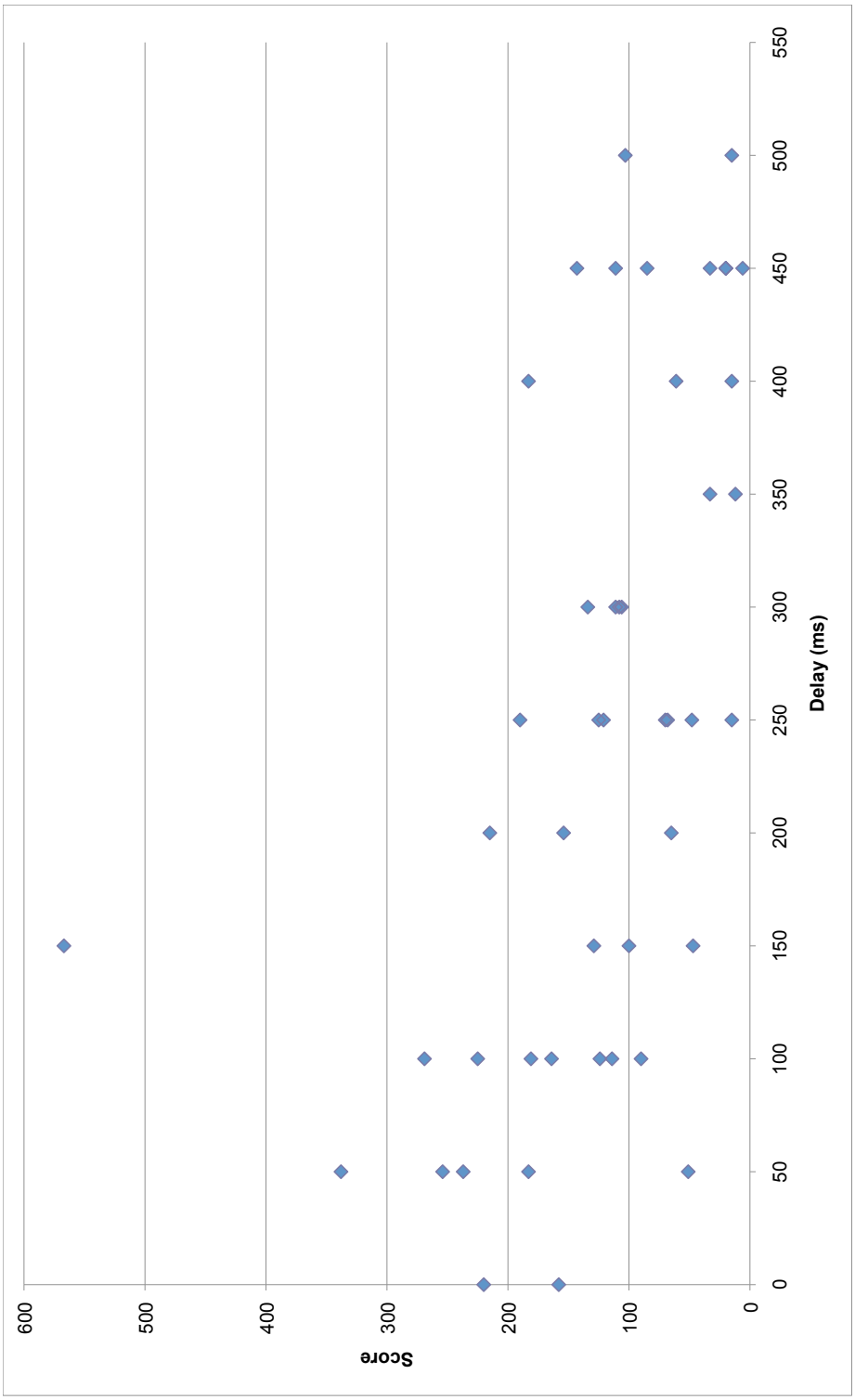


Figure 3.4: A scatter chart of the combined test subjects' scores and delays, showing mild correlation.

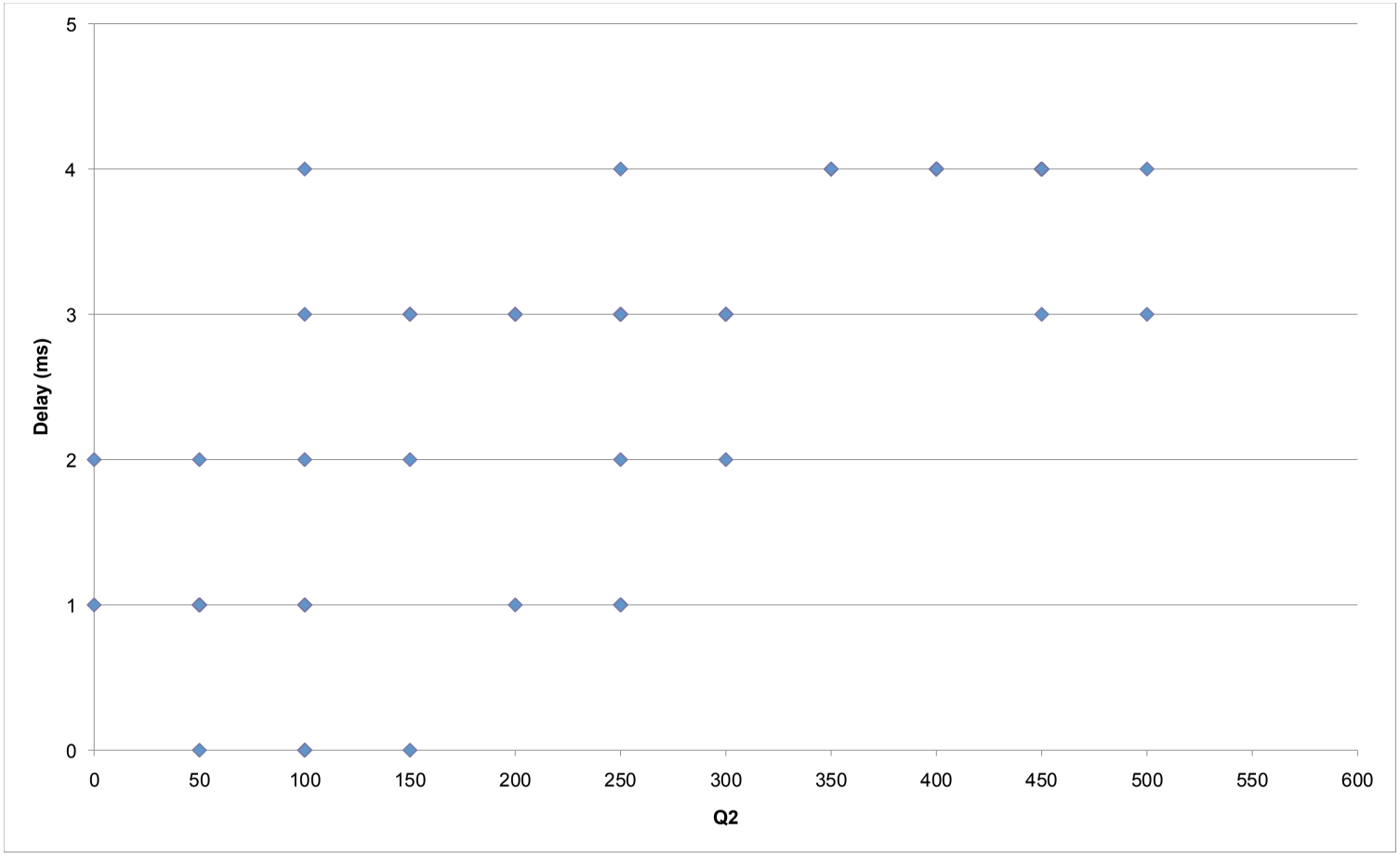


Figure 3.5: This is a scatter chart of the combined test subjects' Q2 responses and delays. Q2 was: "The delay between my actions and the system's response negatively affected the playing experience". On this scale: 0 is strongly disagree; 1 is disagree; 2 is neutral; 3 is agree; and 4 is strongly agree.

Q1	Q2	Delay (ms)	Score
0	0	100	124
0	0	100	225
1	0	150	567
1	0	50	338
1	1	50	183
1	1	100	269
1	1	100	181
1	1	0	220
1	1	50	237
1	1	50	254
1	1	250	15
1	2	250	70
2	1	200	215
2	1	250	190
2	2	150	100
2	2	0	158
2	2	50	51
	2	300	108
	2	100	114
111.8	121.1		
199.8	190.5		

Table 3.1: These are the delays for the games where a test subject answered with an "strongly disagree" (0) or "disagree" (1) or "neutral" (2) to one of the Likert scale questions. Sometimes a test subject may have given different responses to both questions for the same game, so a game may only be represented in one of these columns. The second to last value in each column is the *mean delay* for that question given this subset of answers. The last value in each column is the *mean score* for that question given this subset of answers.

Q1	Q2	Delay (ms)	Score
3		100	114
3		300	108
3	3	450	153
3	3	500	103
3	3	250	121
3	3	200	65
3	3	300	111
3	3	200	154
3	3	150	129
3	3	250	48
3	3	150	47
3	4	100	164
3	4	450	6
3	4	450	20
4	3	100	90
4	3	400	61
4	3	300	134
4	3	450	85
4	3	250	125
4	3	300	106
4	4	450	33
4	4	400	15
4	4	500	15
4	4	350	33
4	4	400	183
4	4	350	12
4	4	250	68
4	4	450	6
4	4	450	111
319.0	327.8		
83.4	81.4		

Table 3.2: These are the delays for the games where a test subject answered with an "agree" (3) or "strongly agree" (4) to one of the Likert scale questions. Sometimes a test subject may have given different responses to both questions for the same game, so a game may only be represented in one of these columns. The second to last value in each column is the *mean delay* for that question given this subset of answers. The last value in each column is the *mean score* for that question given this subset of answers.

Chapter 4

User Study

It is important to establish what delays are acceptable to users as this forms a non-functional requirement for any implementation. Subsequent to our pilot study, we carried out a more extensive study with usability experiments to determine whether network delay has a negative effect upon game play, and to determine a network delay benchmark for the particular types of games that were used for the experiments. The experiments into the effects of network delay were carried out by inserting delays into the response time of two different games running on a mobile device. The player scores were measured, along with level reached (for Asteroid Zone [9] only, as the other game, Bomber 2 [47], only has three levels), their reports on how noticeable the delay was, and their thoughts on whether they were satisfied or not with the game play experience.

4.1 Experiment

This section discusses the experiment setup into how test subjects using software systems are affected by the inclusion of delays into the user-action/system-response cycle.

2D action games were chosen for the experiment because such games require constant input and frequently update the display, and will mag-

nify the effects of network delay. Further, games by nature are competitive, so any delay has a negative effect on the player's "performance". Performance can be approximated by recording player score and (in the case of one of the two games) the level reached.

The network delay was simulated by artificially adding delays between the user's actions and the system's response (Figure 4.1), such that the player can press buttons consecutively without having to wait for the response. We measured the effect it has on the participants by asking them to rate the delay, as well as recording their score in the game. The inserted delay is hereinafter referred to as simply *delay*. The delay is set to one of three values, 75, 150 or 300 ms, and each value is repeated three times, for a total of nine games. Each game has a randomised order of the delays to control for learning effects. The games were implemented for a mobile phone (Nokia N95).

The use of volunteer participants rather than ourselves as per the previous study requires HEC approval[1]. HEC approval was sought and successfully obtained; the application and participant forms can be found in Appendix A.

4.1.1 Hypotheses

The main purpose of the experiment was to accept or reject the following hypotheses:

H_0 There is no correlation between a player's performance in terms of score or level reached and delay.

H_1 There is no correlation between a player's perception of delay or player experience.

[1]Human Ethics Committee (HEC) approval (`http://www.victoria.ac.nz/postgradlife/pages/pages_current_pg/ethics.html`) is required for this experiment.

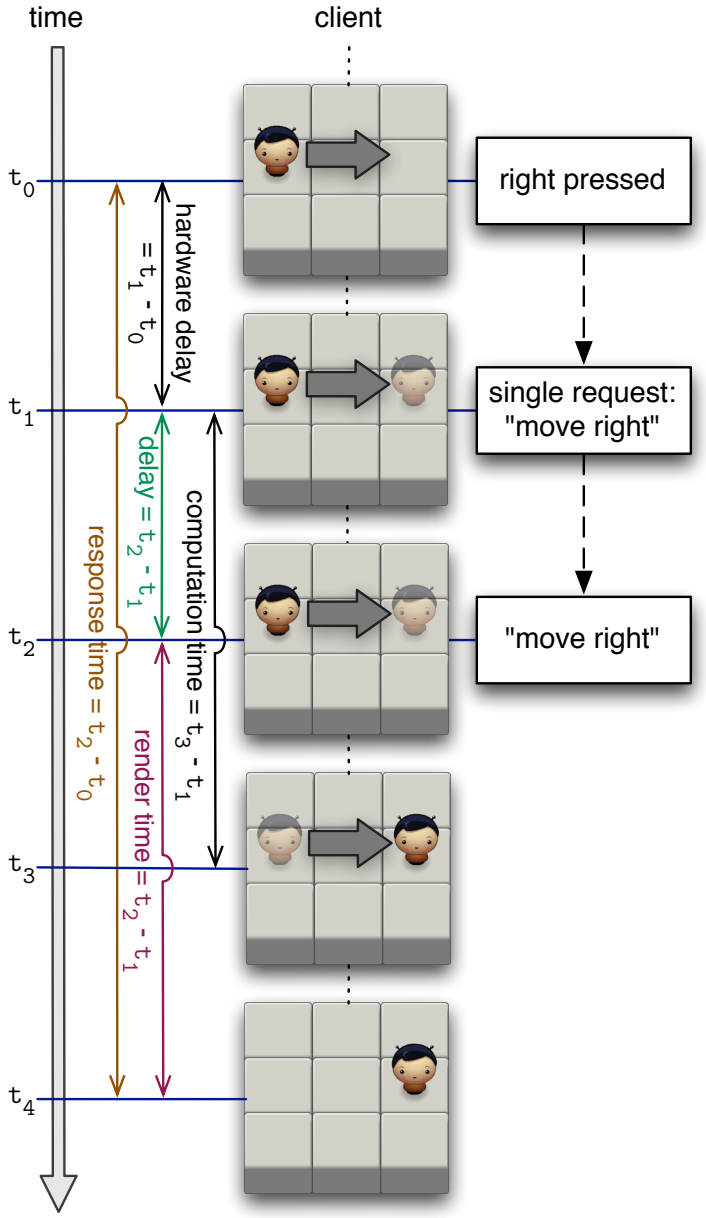


Figure 4.1: An example of inserting delay into the system. The right button is pressed at t_0, which then triggers the event handler in the game at t_1. The request to "move right" is generated on the client at t_2. The client begins processing this action at time t_2, thus the delay is $t_2 - t_1$. If this is the first session, then no delay is added and the request is sent through immediately. In the second session, this delay is set to one of 75, 150 or 300 ms. At time t_4 the rendering of the display is complete, and so the render time is $t_4 - t_2$. The computation required to move the character right is given by $t_3 - t_1$. The response time is the sum of hardware delay, delay and render times, alternatively given by $t_4 - t_0$.

Note that as Bomber has only three possible levels whereas Asteroid has five, H_0 was restricted for Bomber to measuring score rather than level.

In addition, to attempt to identify an acceptable range of delays, the following hypotheses are defined:

H_2 There is no difference in average performance in terms of score between no delay and 75 ms delay.

H_3 There is no difference in average performance in terms of score between no delay and 150 ms delay.

H_4 There is no difference in average performance in terms of score between no delay and 300 ms delay.

4.1.2 Design

This experiment has a within-subjects design to attempt to control for differences in skill between players. The experiment consisted of fourteen participants playing two games a set number of times in two sessions. The two sessions were separated by at least ten days. Each participant played games of Asteroid Zone and Bomber 2 natively on the phone.

In the first session each participant plays each game eight times. The games themselves are unmodified, and are running natively on the mobile phone. The players were accurately told that there was no delay in the games. The purpose was to allow users to become familiar with each game. In the second session, the participants play each game (with introduced delay) nine times. After each play through of each game, the participants were asked whether they noticed a delay and if it had a significant impact on their game play.

Across the sessions the participants' scores were recorded, along with other game play data such as accuracy. This data was used to perform some analysis to see if there were correlations between the delays; de-

creases in the players' scores (and accuracies); and the players' perceptions of the noticeability of the delay and its impact on their playing experience.

4.1.3 Artifacts

Both the system artifacts in the experiment were shoot'em up games. This was deliberately chosen as these games require quick responses. Both games were relatively simplistic examples of their domain, and this was also a deliberate decision as the Incorporated delays would form a significant portion of the overall computation time. Both games were also freely available under open source licenses, making modifications for the purposes of our experiment easy.

Asteroid Zone

The first application is Asteroid Zone, a simple shoot'em up space game. This is the same game as from the pilot study (Section 3.3), however there are things to note in addition to the basic game from the previous chapter. First is that when the player's ship spawns, it is give an invulnerability shield for a few seconds. This is to specifically combat the issue encountered in the previous study, where the ship can continuously spawn and die because an asteroid is in the position where the player's ship spawns.

In addition to the directional control and central button to control the ship, players may use the following keys: 4 and 6 to rotate left and right, 2 to accelerate and 5 to shoot.

Figure 4.2 shows a sample screenshot of the game with the invulnerability shield.

Bomber 2

The second application is Bomber 2, a side-scrolling action game. The graphics are more advanced than Asteroid Zone, with a scrolling background, animations for explosions and decoration like trees and clouds.

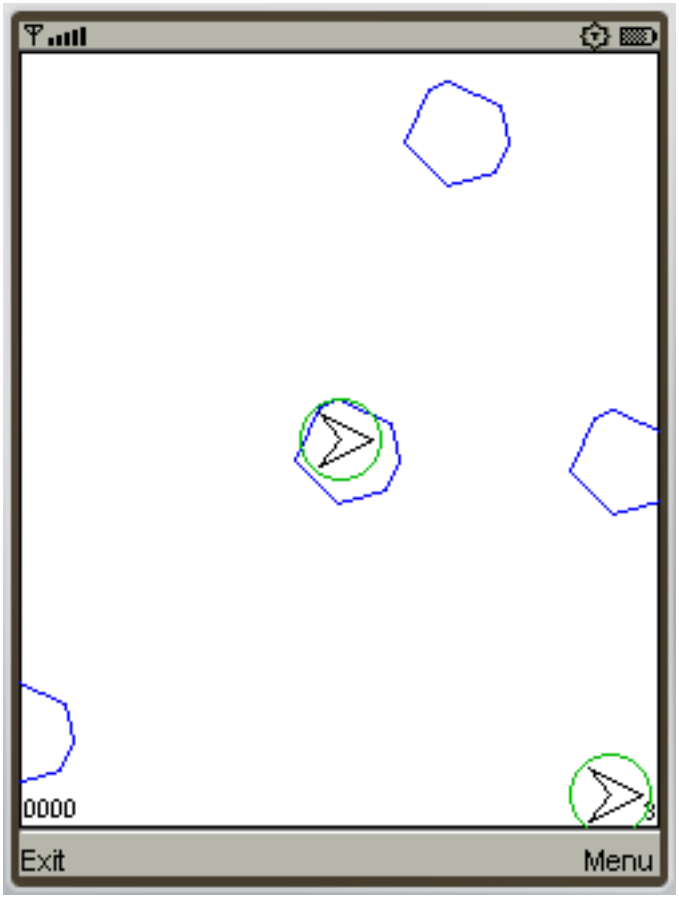


Figure 4.2: A screenshot of the Asteroid Zone game used during the user study. Here the player's ship has just spawned, and so there is a invulnerability shield (coloured green) around the ship. Even though the ship is touching an asteroid, both survive.

The objective is to destroy the targets (marked with yellow triangles) by shooting or bombing them with your fighter plane. When no targets remain the level is complete and the player moves on to the next level, with a different layout and arrangement of enemies and targets. If the player's ship is killed by enemies or collision, the player returns to the start of the map with a fresh set of bombs. The player has five lives per level, and six bombs per life. Lives are reset once a level is complete.

Figure 4.3 shows a sample screenshot of the game. This game draws the various objects using images loaded into memory, and the objects are drawn using the API directly onto the screen.

Controls for Bomber 2 are limited to 4 actions: 2 to rotate clockwise, 8 to rotate anti-clockwise, 5 to launch a bomb and either 1 or 3 to fire bullets.

Game Modifications

In order to reliably set the delay between the user actions and the system's response, modifications were made to both games to provide this functionality. The strategy is to suppress every user action for the desired period of time, and then trigger the action once that period has elapsed. In this study the period of time – the delay – was set to be one of three values: 75, 150 or 300 ms.

For both games the implementation is the same; a secondary thread keeps a queue of the user actions along with the new time they should be fired at, and approximately every 20 milliseconds the thread triggers actions in the queue that are due. Whenever a user presses a key it is added to the queue along with the time that it is due to be triggered. This is illustrated in Figure 4.4

In addition to providing delay functionality, logging capabilities were added to each game. For both games all user input is logged, i.e. movement and combat. Furthermore, Asteroid Zone now tracks when an asteroid is hit and logs this, as well as when the player dies. Once the game is quit, the log is saved to a memory card installed in the phone, which in-

Figure 4.3: An action shot of the other game used in the user study, Bomber 2. This game features modern graphics and complicated real-time game play, including enemies shooting back. These elements will be made more difficult when the user's input is delayed.

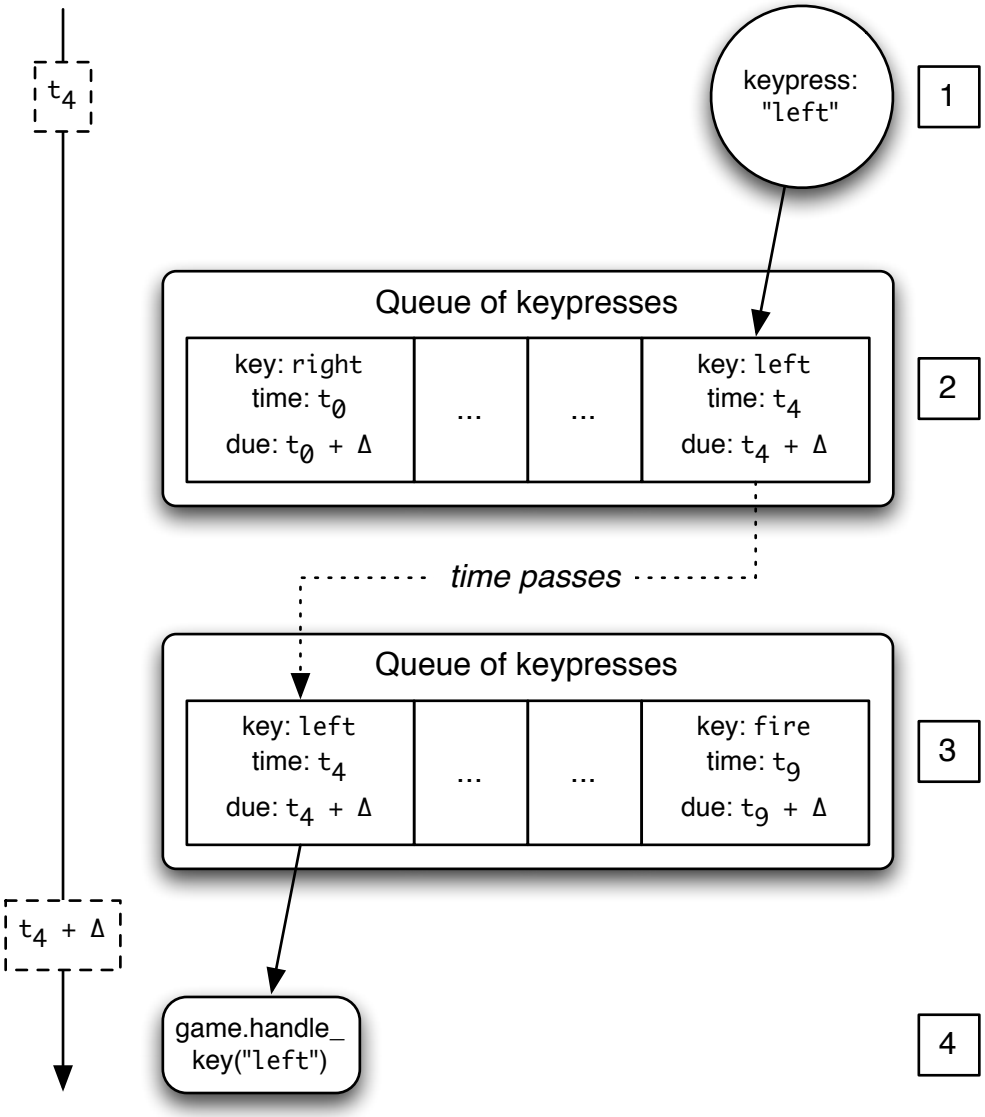


Figure 4.4: An example of how the delay is imposed. The key "left" is pressed at time t_4 (1) and is added to the end of the queue (2). Once the current time has passed the due time, $t_4 + \Delta$, (3), the key is sent to the game as a key press (4).

cludes the actions, as well as the final score, the number of shots fired and landed, and value that the delay was set to. Bomber 2 tracks when each type of ammunition is fired and when it hits. Further, the type of object hit is recorded, both in terms of target/non-target and house/tank/etc.

These game modifications were possible to be made because both games were open source with freely available code – one of the reasons for choosing these games over other J2ME games (Section 2.4).

Mobile Phone

All the participants used the same Nokia N95 mobile phone during the experiment described in Section 3.3. While the phone has a directional pad, participants used the keypad with the appropriate number keys as described previously because the directional pad is close to special phone buttons which are very easy to press, resulting in a failed run as the game exits upon pressing one of those.

Test Subjects

The test subjects were varied in age, and were mostly staff and students of the School of Engineering and Computer Science at Victoria University. There were 11 males and 3 females for a total of fourteen participants. It was not asked if any of the subjects had used that particular mobile phone before, or if they had played either of the games before, or even their general experience with games, because this does not affect the results as we are comparing the participants with themselves. Overall all the participants were familiar with computers, and all have a computer science background.

4.2 Procedure

Each participant played each game eight times in the first session and nine in the second session. Each session was separated by a period of at least ten days, so that the participants didn't improve their playing ability significantly between the sessions. The participants were allowed to play a couple of practice games before each session and for each game, these were not delayed at all. This was to allow the participants to (re-)acquaint themselves with the controls and gameplay, and to prevent their first games from suffering. Each game was played following this sequence of actions:

1. The experiment controller looked up the delay value to be set for this play through, and set the game's delay to that value. If it was the first session, then the delay was set to 0 ms. The delay values are pre-generated, and each delay level is repeated three times. The order of delays is randomized for each participant.

2. The controller then completes the initialization for the game, and then passes the phone to the participant. The participant then begins the game by pressing the designated button.

3. Once the play through is complete (either by the participant exhausting all their lives or finishing the final level to be played), the participant passes the phone back to the controller.

4. If it was the second session (with delay), the controller asked the participant to rate the following statements from 1 – 5, where 1 is strongly disagree and 5 is strongly agree:

    - There was a noticeable delay between my actions and the system's response.
    - The delay between my actions and the system's response negatively affected the playing experience.

5. If there were remaining games to play, the participant and controller return to step one.

4.3 Experimental Issues

There were several design issues discovered during the experiment.

The first issue is that the participants were much better at the game than anticipated, and so the total playtime was limited to the natural "end" of the play through. Specifically, for Asteroid Zone, rather than waiting for the player to exhaust all their lives, the game was stopped after the end of level 5, the "last" level in the game. For Bomber 2, the game was stopped after the end of level 3 (the "last" level is level 4). Even with these adjustments, the sessions lasted for an unreasonably long time (over four hours), and so the number of plays of each game was reduced from 16 to 8, for both Bomber 2 and Asteroid Zone, bringing the majority of sessions down to under two hours.

The second issue is an extra command in Asteroid Zone, *teleport*, that places the player's ship in a new, random position on the screen. While this seems like a useful command, it results in unpredictable game play. There is no limit on the number of times you can use teleport, nor is there a minimum time between uses. Furthermore, it makes no guarantees on the placement of the ship and the player may end up next to or in an asteroid. For these reasons the user wasn't told about teleport unless they asked, at which point we advised the user not to use the command. Overall, teleport was pressed 18 times out of a total of over 230,000 commands, thus the data with teleport was left in.

The third issue was somewhat beyond our control. The manner in which Bomber 2 is typically played results in periods where the player doesn't need to press any buttons. The mobile phone is configured to dim the display after a period of inactivity, and so the display dims while the game is still being played. When this occurred during the user's play, they

Delay (ms)	Mean Score	SD	df	t-Test (p)	Mean Runtime (s)
0	484	229	124	n/a	284
75	549	253	40	0.128	198
150	475	227	41	0.825	197
300	347	219	41	0.001	162

Table 4.1: Various statistics for Asteroid Zone at each delay level. The t-Tests for each delay are against no delay, and are two-sampled, equal variance, and two-tailed.

were advised to press an unbound key to wake the display without affecting the game.

Finally, due to an issue with one of the participant's play data, discovered after the conclusion of the experiment, Asteroid Zone has marginally fewer data points than Bomber 2.

4.4 Results

Here we present the results of the user study. The main calculations are correlations for one independent variable and three dependent variables for each game. The independent variable is the delay between the user's actions and the system's response, and the dependent variables are the user's answers to the two questions asked, as well as their score.

4.4.1 Asteroid Zone

Taking all the participants' data together, the Pearson correlation for the delay vs score is $r = -0.201$ (see Figure 4.6). This is a slightly negative correlation, but it is not as strong as we expected. However, it is statistically significant at a 95% level. Taking delay vs level the correlation coefficient is -0.217, and is graphed in Figure 4.7.

Looking at the scores for each delay value gives mean scores and student's t-Test values as given in Table 4.1. The null hypothesis is accepted

Delay	User Response				
(ms)	1	2	3	4	5
75	25	13	1	2	0
150	16	17	5	2	1
300	6	8	14	8	5

(a) Question 1

Delay	User Response				
(ms)	1	2	3	4	5
75	30	7	3	0	1
150	22	13	4	1	1
300	7	14	14	6	0

(b) Question 2

Table 4.2: Frequencies of user response values at each delay for Asteroid Zone for the two questions – "There was a noticeable delay between my actions and the system's response" (4.2a) and "The delay between my actions and the system's response negatively affected the playing experience" (4.2b).

for the first two t-Tests, 75 ms vs 0 ms (H_2) and 150 ms vs 0 ms (H_3), because the t-Test value is above 0.05, but the null hypothesis for 300 ms vs 0 ms (H_4) is rejected. This indicates that while delays of 75 ms and 150 ms may be acceptable, a 300 ms delay for Asteroid Zone gives a lower mean than no delay, and so it negatively impacts the player's experience.

4.4.2 Bomber 2

Bomber 2 has weak results. Again, taking all the participants' data, the correlation coefficient for the delay vs score is $r = 0.017$. A coefficient this low suggests that there is no correlation at all, and it is not statistically significant. The mean scores and t-Tests for each delay level (vs no delay) are given in Table 4.3.

When all delays are considered the null hypothesis H_0 for Bomber is accepted. However, when the other hypotheses are explored, the null hypothesis H_2 is rejected for the lowest delay value, 75 ms, but accepted for the two higher values (H_3 and H_4). This is a highly unusual result, somehow suggesting that delaying the input by 75 ms may give a better score than not at all! It could be that players are overcompensating at the higher levels or that delaying the user input slightly makes the game easier to play.

Delay (ms)	Mean Score	SD	df	t-Test (p)	Mean Runtime (s)
0	313	92	108	n/a	436
75	346	89	41	0.047	376
150	325	99	41	0.487	366
300	317	83	41	0.789	434

Table 4.3: Various statistics for Bomber 2 at each delay level. The t-Tests for each delay are against no delay, and are two-sampled, equal variance, and two-tailed.

Delay (ms)	User Response 1	2	3	4	5
75	31	10	1	0	0
150	25	12	5	0	0
300	14	13	12	2	1

(a) Question 1

Delay (ms)	User Response 1	2	3	4	5
75	38	2	2	0	0
150	34	6	1	1	0
300	22	13	5	2	0

(b) Question 2

Table 4.4: Frequencies of user response values at each delay for Bomber 2 for the two questions – "There was a noticeable delay between my actions and the system's response" (4.4a) and "The delay between my actions and the system's response negatively affected the playing experience" (4.4b).

As for the questions asked of the users, the frequencies are shown in Tables 4.4a and 4.4b. Again, calculating Spearman's rank correlation coefficient results in 0.480 and 0.518 for the first and second questions respectively. While these are significant values, they are not as large as for Asteroid Zone. Further, the frequency tables show that while the users on the whole gave higher ranks to the larger delays for the first question ("there was a noticeable delay ..."), they did not give as high values for the second question ("the delay ...negatively impacted my playing experience") for the higher delays. Perhaps the gameplay for Bomber 2 is such that the users can compensate for delay, to the point where it doesn't affect their gameplay or affects it very little compared to Asteroid Zone.

4.5 Discussion

The aim of this user study was to determine the point at which the delay became unacceptable. From the results, we can say that for Asteroid Zone, while 75 ms and 150 ms may be acceptable delays, at 300 ms the user's experience is negatively affected. The specific point is not known, but it is assumed to be between 150 ms and 300 ms delay. One user, while playing at 150 ms, said that "any delay under this is good", and while playing at 300 ms, "if it's this delay I'd stop playing". Another user commented that "delay is a lot harder when you're moving around".

For Bomber 2, we can't make any judgments on which delays are unacceptable. The results were inconclusive, suggesting that some delay (75 ms) is better than no delay! It may be the case that more data is required for Bomber 2 at the higher end, with delays greater than 300 ms. On the other hand Bomber 2's game play may be such that large amounts of delay are easier to compensate for, but they still affect the player's experience negatively. One user commented that "this is more noticeable than asteroids", however another said that the delay on Bomber 2 was "way easier to compensate" and that "any delay is acceptable".

One trend apparent in both Asteroid Zone and Bomber 2 is the increase in average score between 0 and 75 ms delay. Further, the scores for Bomber 2 are higher with delay than without, and Asteroid Zone's scores are also surprising. Figure 4.5 shows the average scores relative to the scores at 0 ms delay, to account for the differences in score between Asteroid Zone and Bomber 2.

We initially thought that this was due to the users learning the games, and thus performing better in the second (delayed) session than in the first, however we counter this by separating the sessions by at least ten days. Further, correlation calculations for score vs run gave mixed results, and the average scores excluding the first two runs don't vary significantly from the all-inclusive average scores, leading us to believe that

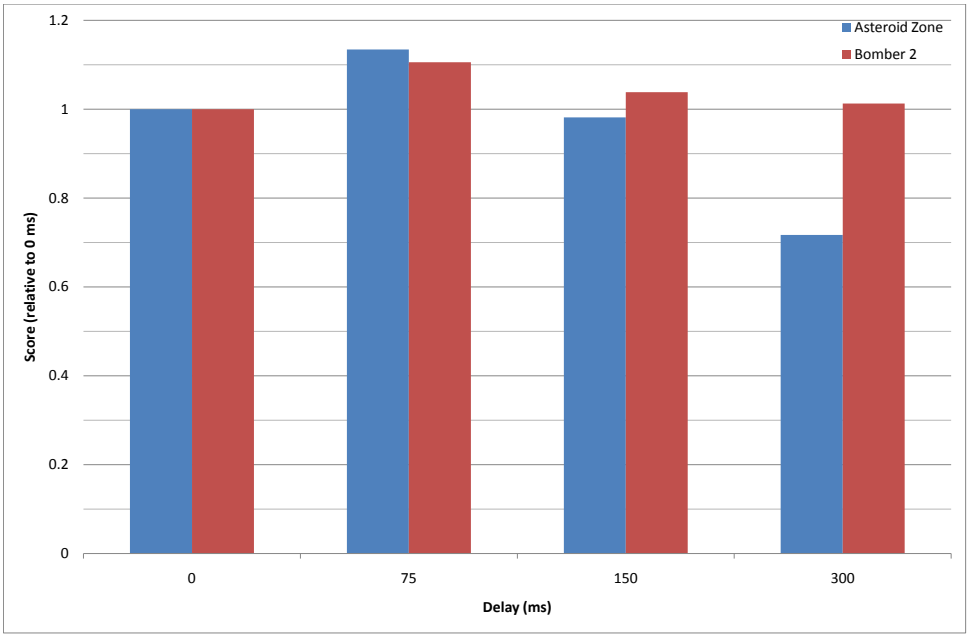


Figure 4.5: Average scores for the two games, relative to the average scores for 0 ms delay.

game learning is not like to be the cause of this anomaly.

There were other interesting results from the data which we didn't run any advanced analysis on, such as the average playing time for each delay level. For Asteroid Zone, the average run time is 284 seconds with no delay, and this consistently decreases as delay increases until 161 seconds at a 300 ms delay. For Bomber 2 the numbers are quite different, starting at 436 seconds at no delay, and then 376, 365 and 434 seconds for 75 ms, 150 ms and 300 ms delays respectively.

We also performed analysis on sequences of actions just for Asteroid Zone, based on a user's comment that we should "look for overturning, where I go past the point and then turn back".

Consider a sequence of actions: *right right right*. With no delay, the probability of the next action being *right* is 0.672, and *left* is 0.035. At 300 ms delay, the probability that the next action is right is 0.762, a slight increase, however the probability that the next action is left doubles to 0.070. Going

the other way, where the sequence of actions is *left left left* and the next action is *left* or *right*, the probabilities barely change between no delay and 300 ms delay. These numbers suggest that while users may adjust their game play when delay is increased, it is not consistent between sequences.

4.6 Conclusion

The interest here is in identifying how delays built into the user action / system response cycle affect user performance and perception. In this chapter the results of an experiment conducted with 14 participants and 2 mobile computer games were presented. Based on the results, we can not conclusively give a single delay value as the minimum requirement; rather we propose that a range of values, 75 ms to 150 ms as the requirement a remote thin-client system will need to meet to be acceptable for 2D single player shooter and side scroller games. This result has implications for thin-client systems as it provides a measure against which to assess timing performance for a system's various sub-components.

There are several ways to advance this work. One limitation of the study is that all participants had some background in computer science. It would be interesting to see whether the results also hold for other demographics. There are also application domains other than games to explore (and also other types of games, such as those requiring extensive use of 3D graphics libraries), and a more refined experiment would also include a finer-grained range of delay levels between 75 ms and 150 ms.

Figure 4.6: A scatter chart of the score vs delay for all participants; Asteroid Zone.

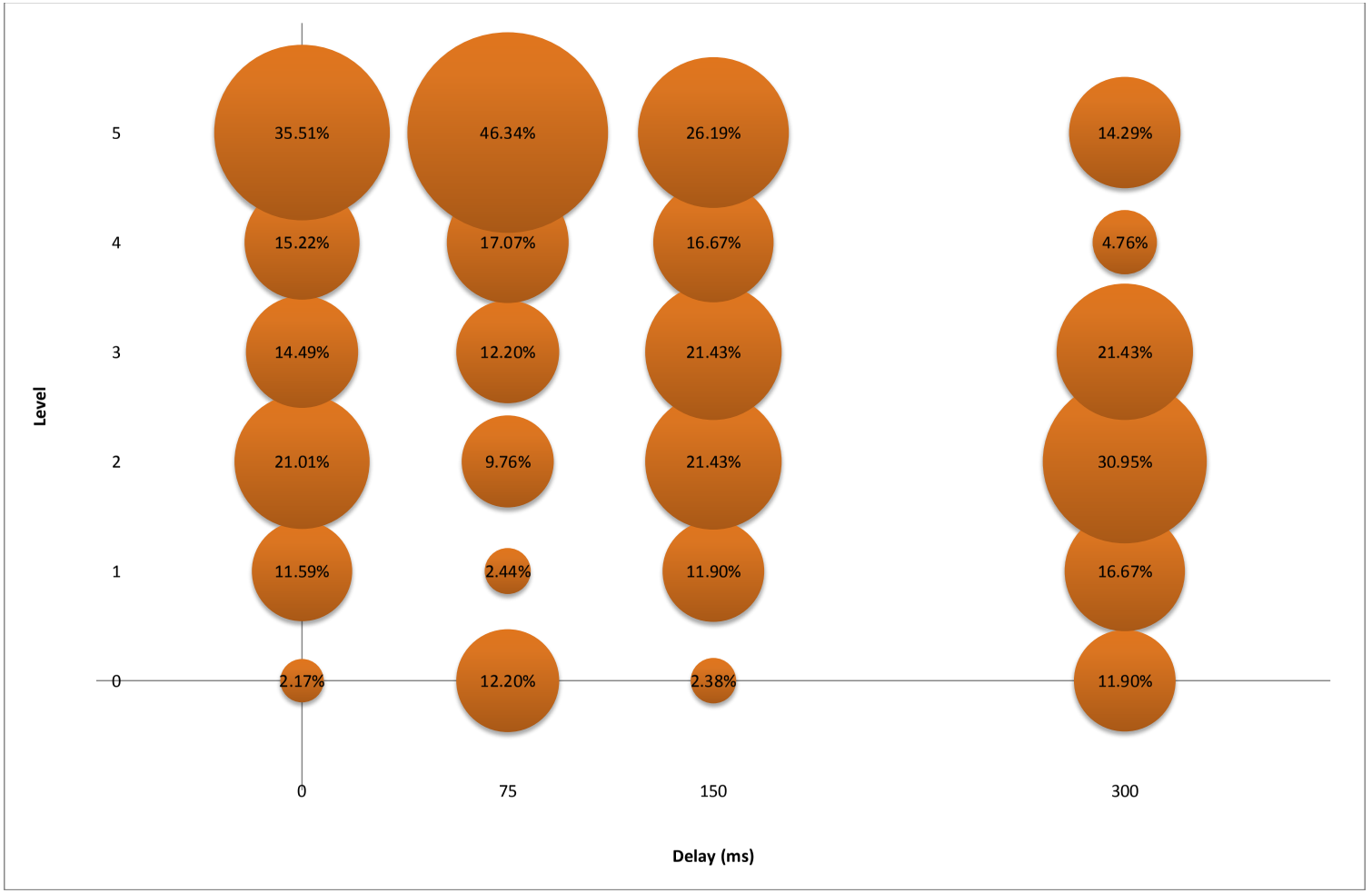


Figure 4.7: A bubble chart for Asteroid Zone, showing the level vs delay for all participants, where the size of the bubble represents the proportion of data at that level.

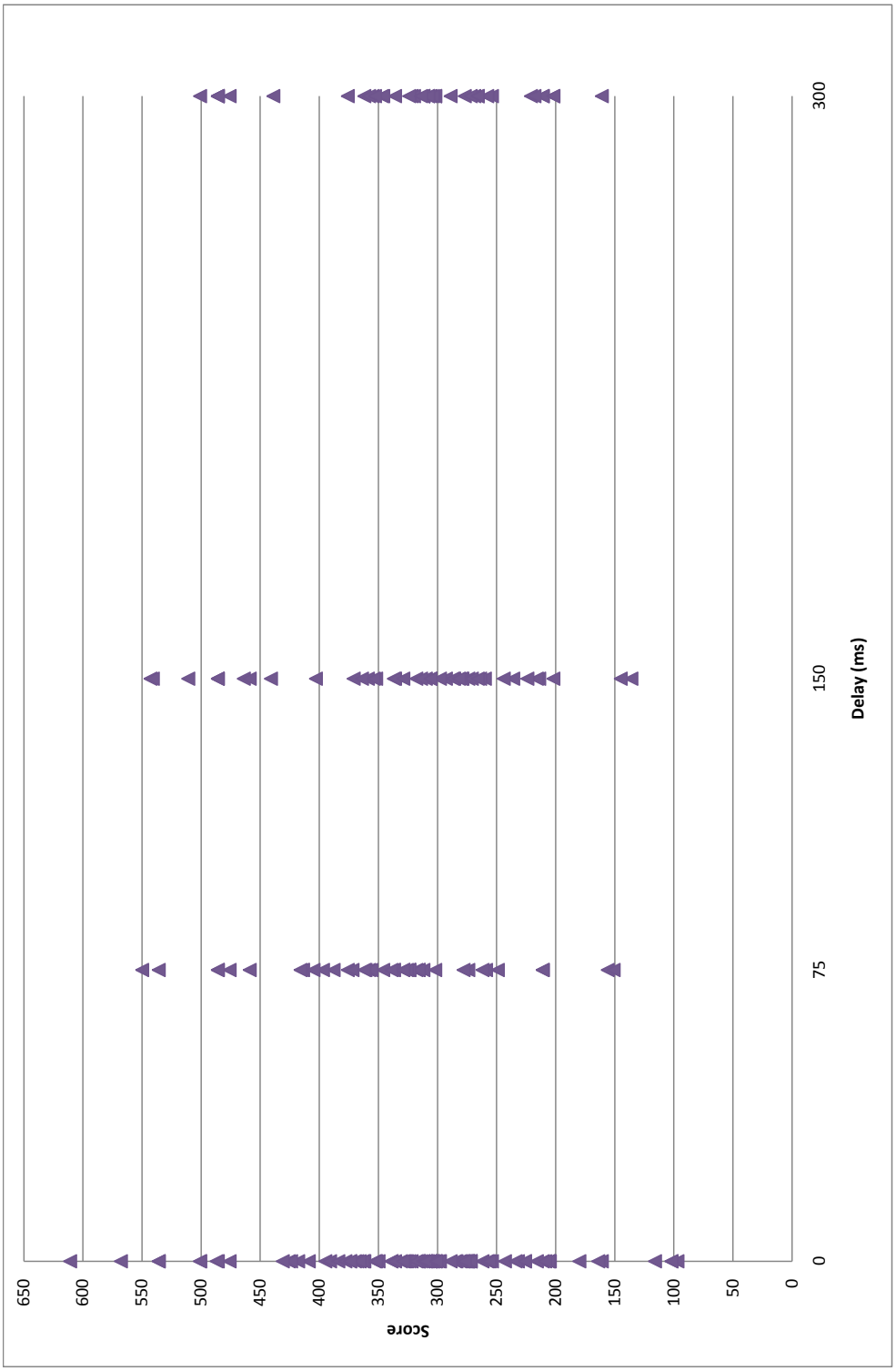


Figure 4.8: A scatter chart of the scores vs delays for all participants; Bomber 2.

Chapter 5

RemoteME

RemoteME is our client/server system upon which applications are controlled remotely. RemoteME lets users control applications which are executing on the server using the client. Applications which are supported by MIDP 2.1 and CLDC 1.1 are supported by RemoteME. Section 5.1 describes the low level workings of the system, and Section 5.2 details the performance logging system in RemoteME. Finally Section 5.3 highlights some of the implementation issues encountered during development.

5.1 Architecture

We now describe the different components of the RemoteME architecture. The architecture diagram is shown in Figure 5.1.

5.1.1 Client

The RemoteME client application executes within a virtual machine running on the mobile device.

The RemoteME Client is fully self contained, and does not rely upon any APIs other than the ones provided by CLDC 1.1 and MIDP 2.1 (see

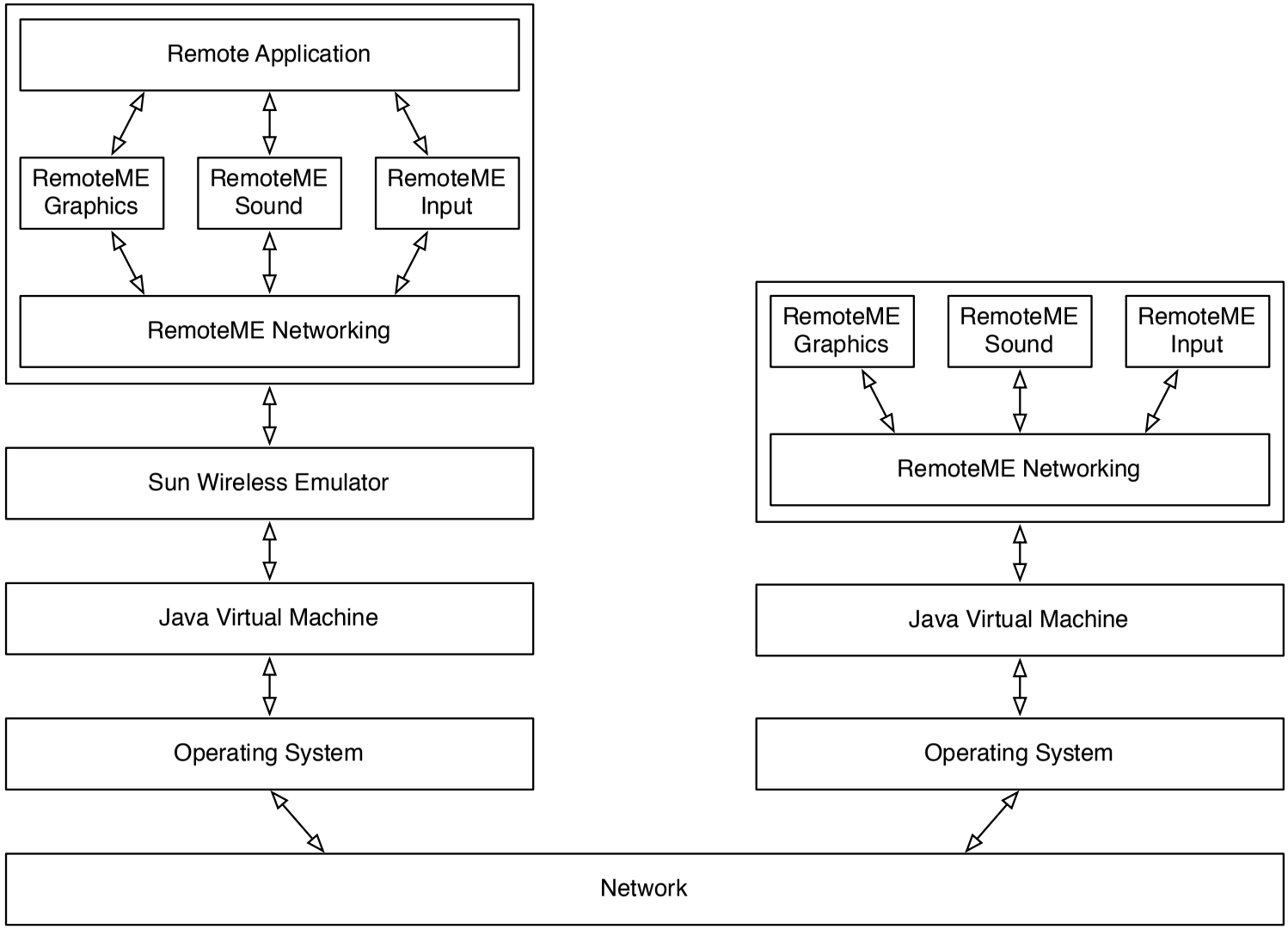


Figure 5.1: The RemoteME stack. On the left is the server, where the remote application is executing on top of the server component of RemoteME. The client is on the right, and is typically executing on a mobile device.

Section 2.4). Further, the client software does not require any 3rd party libraries and has very minimal code.

5.1.2 Server

The RemoteME server application runs in a Java virtual machine running as a server. In the implementation the software stack consists of a wireless emulator and the modified application being exported to the client. The emulator is the Sun Wireless Toolkit [37], as distributed by Sun Microsystems [38] without any modifications. The application that is to be run remotely is modified by weaving the code with the RemoteME AspectJ aspects. The code which is woven in handles the graphics, sound, network and user input to and from the RemoteME client.

5.1.3 System

RemoteME is a traditional client/server system, with the stack shown in Figure 5.1. The RemoteME system is server-heavy, in that only the server executes the application code, and the client displays the application and forwards input to the server. The RemoteME client is purely event driven, and events are either received from the RemoteME server or from the mobile device itself. Events from the server can be one of two types: object creation and method invocation.

Object Creation

To reduce complexity on the client, a simple object model is used. For every instance of an API class created on the server, a corresponding instance is created on the client. For example, if a `Sprite` object is instantiated on the server, the constructor's parameters are sent and a corresponding instance is created on the client. To keep track of objects, a `Map` is kept on both the client and server (see Figure 5.2). When objects are instantiated

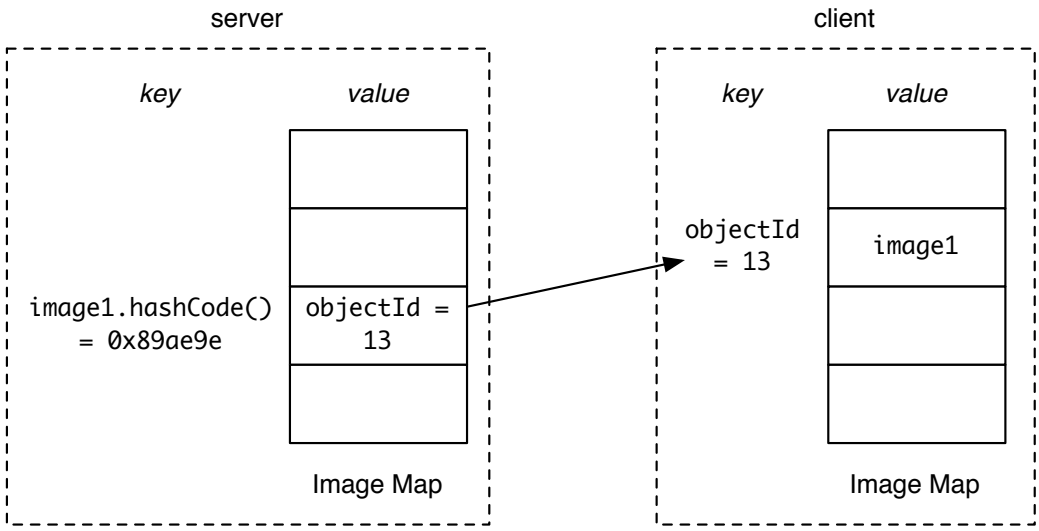


Figure 5.2: Example object map for images. On the server `image1` indexes to the integer 13 in the object map. This value is then used to index the object map on the client, hence `image1` is stored.

on the server, they are added to the map, indexed by the value given by the `hashCode()` method on the object, and the value is an incrementing integer, called the *objectId*. That *objectId* is sent along with the constructor call to the client. Upon receiving the request to construct the object on the client, the object is added to the map, indexed by the sent *objectId* and the value is the newly created object.

Method Invocation

Method invocation is the other type of event sent by the server. In this case, there are two additional pieces of information sent along with the method parameters. One piece of data sent is the object's id that is the target of the method invocation. The *objectId* indexes into the map of objects, and is used retrieve the correct instance to invoke the method on.

The other piece of information is the method that is being invoked. Methods are identified by a unique id, which is an integer sent by the server. However since the set of methods and classes do not change in

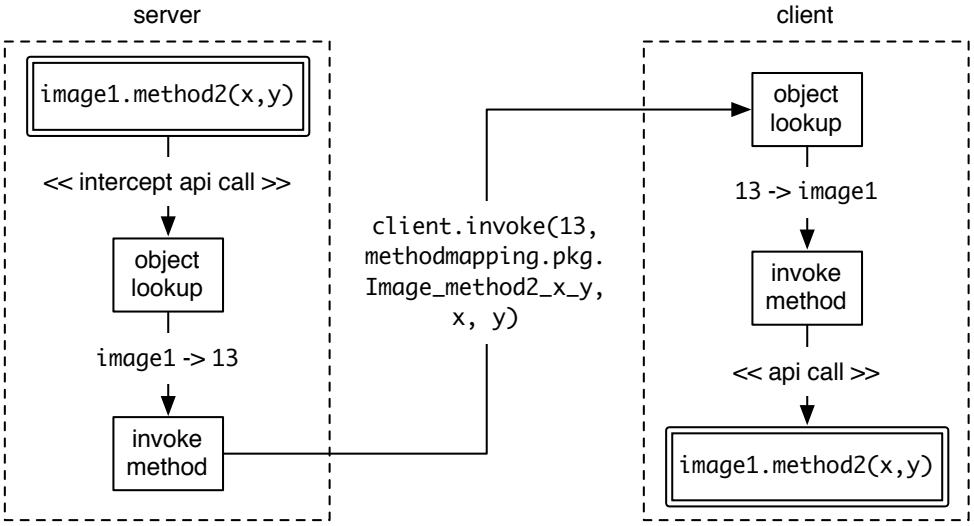


Figure 5.3: RemoteME method invocation. `method2` is called on `image1` on the server with parameters `x` and `y`. RemoteME intercepts the method call, looks up `image2`'s ID in the map, and then tells the client to call the method named `method2` on the `Image` object referenced by `13` with the parameters `x` and `y`.

the J2ME API, they are numbered statically at compile time, and so the lookup on the client is very fast. A consequence of this is that the ids will need to be recreated if the API changes. A single class holds all the method ids, numbered in order according to the minimal amount of information that is required to distinguish each method. This is the concatenation of package name, class name, method name, and parameter types.

Furthermore, since only a few of the supported classes can be instantiated, there is a Map for each class to speed up the lookup process. Thus the method invocation procedure is given in Figure 5.3.

Finally, as with the object creation, method invocation only applies to API classes. For the case of subclasses, only the final dispatch to an API method will be forwarded onto the client, all other method calls will be executed only on the server.

5.1.4 RemoteME Application

The remote application is the application that is exported from the server to the client. The source code of the application is not required to be modified manually. Instead, the application's compiled code, along with various generated and constant classes, is woven with RemoteME's AspectJ [18] aspects. This woven code is then packaged as a RemoteME Application, ready to be run on the standard, unmodified Sun Wireless Emulator.

The RemoteME aspects provide a very specific set of cross-cutting concerns. These concerns pointcut any method call to the API present in J2ME, where the method has side effects[1]. Essentially, any method which modifies J2ME API instances will be pointcut.

On exception to this strategy is pointcutting specific constructors, such as the constructors for the `Sprite` class. Here AspectJ cannot pointcut the constructor because the code would need to be woven into the API, which is not feasible. In these cases, a helper utility generates a proxy class, which wraps all the original class' methods including the constructor. The utility also replaces all references to the original class with the proxy class, in essence replacing the original class with the proxy class in the remote application.

5.1.5 RemoteME graphics

Graphics is one of the three components of the stack which sit between the remote application and RemoteME's networking. The graphics component, comprised primarily of the graphics aspect, pointcuts the graphics API, including the some of the `java.microedition.lcdui` package and all of the `java.microedition.lcdui.game` package.

[1] That is, any method which changes state. Methods which do not alter state, such as getter methods, have no side effects. Each method is inspected manually to determine if it has side effects.

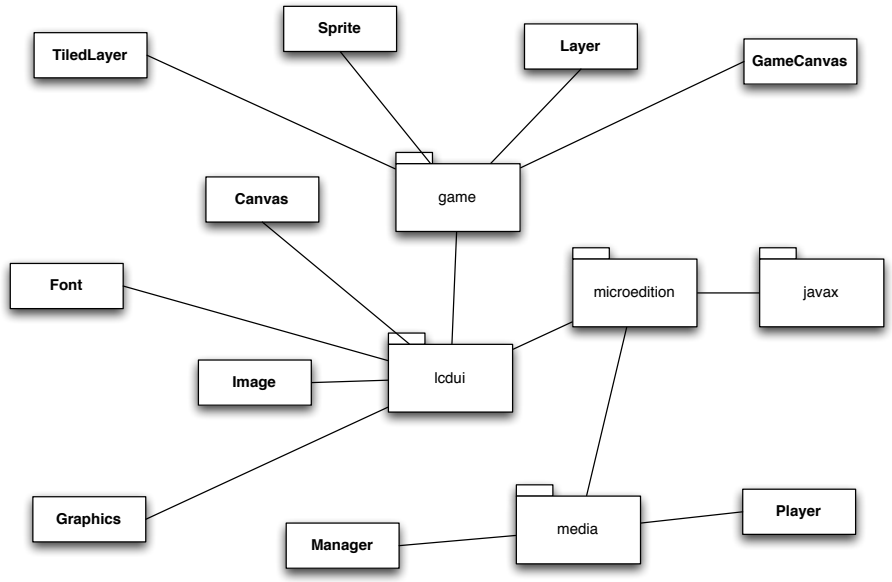


Figure 5.4: Package diagram showing J2ME API classes which are implemented in RemoteME.

From the `java.microedition.lcdui` package, RemoteME supports the `Canvas`, `Font`, `Graphics` and `Image` classes.

`Canvas`

The `Canvas` class handles low-level events and issues graphics calls for drawing to the display. In RemoteME, there is a single `Canvas` which is kept consistent across the client and server. When a new `Canvas` object is created it replaces the current one, and from that point on all operations on the canvas occur on the new canvas.

`Font`

The `Font` class represents fonts and font metrics. Since `Font`s cannot be instantiated, and they can be represented by three integers (face, style and size), RemoteME simulates font support by substituting the `Font` ob-

ject with those three integers across the network when `Font` is used as a method parameter.

`Graphics`

The `Graphics` class provides 2D rendering capability for drawing primitives such as lines and rectangles, as well as images. RemoteME supports the class fully, providing the same rendering capability.

Rendering of primitives such as lines and rectangles is rather straightforward, requiring no more than method invocations with the parameters as is. Image handling is more complex, and is described next.

`Image`

`Image` objects are handled using the map strategy described earlier. `Image` objects are stored in the map upon creation, and then retrieved from the map when they are used as parameters or when methods are invoked upon them. `Image`s can occupy large amounts of memory, so this object store doubles as an efficient cache of images.

`java.microedition.lcdui`

In addition to the classes described above, there are numerous classes which are not supported by RemoteME. The primary reason for this is that RemoteME is a prototype, proof-of-concept system rather than a fully fledged implementation. Adding support for the missing classes is straight forward, however, requiring only three additions: (1) an aspect which pointcuts all the required methods in the class is added to the RemoteME server; (2) a data model (object map), if necessary, is added to the client and server in the same manner as `Layer`, `Sprite`, or `Image`; and (3) the appropriate command handlers are added to the client to support the new class.

Support for the user interface classes, such as `Alert`, `Form`, or `List` will require that the implementation for the `Display` class be completed, however the procedure is much the same as described above. The implementations for these classes was omitted because the vast majority of games for J2ME use the `Graphics` API rather than `Form`s.

java.microedition.lcdui.game

In this package the classes which are fully supported in RemoteME are `GameCanvas`, `Layer`, `Sprite` and `TiledLayer`.

GameCanvas

`GameCanvas` extends `Canvas`, and is used when greater control is needed over input handling and painting. The implementation of `Canvas` in RemoteME allows `GameCanvas` to be treated as a `Canvas` without any issues. Typically when applications use `GameCanvas` they don't have a `paint()` method and simply call the `flushGraphics()` method, however since canvases are implemented using `GameCanvas` on the client, both approaches to painting are supported.

The other difference of note is the key press handling mechanism, where typically applications poll for key presses rather than having callback events. Again, the implementation on the RemoteME client supports both the event-based and polling methods fully.

Layer

`Layer` is an abstract class representing a visual element in the application. RemoteME handles layers using the object map technique described earlier, and since `Layer` is the common supertype of both `Sprite` and `TiledLayer`, those classes are also in the object map.

Sprite

A `Sprite` is a `Layer` with the ability to be animated, among other features. Since `Sprite` extends `Layer`, RemoteME simply treats it like any other non-API class, and object creation/reference is handled by the `Layer` handling mechanism.

TiledLayer

A `TiledLayer` is a grid of tile images, typically used to create large scrolling backgrounds. `TiledLayer`s are handled in exactly the same way as `Sprite` objects.

5.1.6 RemoteME sound

Sound is another component of the RemoteME layer of the stack, and while it is not as important as graphics for the prototype, nor is the implementation in RemoteME as fully fledged, there is basic support for playing sounds in RemoteME.

The relevant package for sound is `javax.microedition.media`, and from this package the sound support is in the `Manager` and `Player` classes.

Manager

The `Manager` class provides the means to create a `Player` object. This class is pointcut only on the method which instantiates a `Player`. `Player`s are created from an `InputStream` which is a stream of bytes representing the sound.

Player

The `Player` class provides playback of sounds. Players can be constructed from a number of sources including capture devices and http streams.

`Player` objects are handled using the map technique as per the graphics classes.

"Sound"

There is no sound class in the J2ME API, so a pseudo sound class was created in RemoteME on both the client and server, to facilitate efficient caching of the bytes that represent sound, much like what occurs with `Image` objects on the graphics side.

5.2 RemoteME Logging System

An important part of this thesis is evaluating whether our implementation's performance falls within the boundaries of a reasonable response time. Therefore the RemoteME client application includes an option to record the performance of the system, specifically the response time as mentioned in Section 2.3. To minimise any affect the logging system may have on the network traffic, the logged data is kept on the client in memory until it is requested to be sent to the logging server (not necessarily the RemoteME server).

The data is recorded in the following manner:

1. When a key is pressed on the client, the time in milliseconds as provided by the Java API is recorded, and sent to the server along with the key as the unique ID for that key press. The timestamp is recorded at the earliest possible opportunity.

2. When the server receives a key command, it stores this unique ID as it's latest ID.

3. The server signals the end of each screen's drawing commands with a call to the paint method. If there is a recorded key press ID, then this is sent with the paint command.

4. Finally, when the client receives a paint command with an attached ID, it checks that the ID matches the ID it was expecting – the last ID sent – and records the current time in milliseconds if they match. The latter time is subtracted from the former to determine the elapsed time between a key press and the related graphics update. The timestamp is recorded at the latest possible opportunity – the next operation is the drawing of the buffered image to the screen.

5. The elapsed time is stored in a standard Java data structure. This data structure is enumerated and sent to the logging server upon request.

5.3 Implementation Issues

During the implementation of RemoteME we discovered two issues relating to: the server runtime; and limitations of AspectJ.

5.3.1 Server Runtime

A key implementation issued was the choice of server runtime. We focused on J2ME 2.4 to constrain the scope of the project, reasoning that developing a protocol with J2ME at both sides would simplify development. In addition, our functional requirements include that the server part of the system be a computer (rather than a mobile phone), and that the applications we execute on the system are J2ME applications. This leaves us with three server runtime options to choose from: extending a third-party J2ME emulator; creating a runtime from scratch; or the Sun Wireless Toolkit with cross-cutting aspects.

There are a few different projects attempting to emulate J2ME, however only two have source code available and thus are suitable for use: MicroEmulator [21] and ME4SE [13]. MicroEmulator is a fully-fledged, cross-platform J2ME emulator, and the idea is to integrate the RemoteME

server components directly into the emulator at the appropriate places. This would allow for easier development, as the system can then be debugged with the standard Java debugging tools (such as the Eclipse debugger), as well as greater performance.

However, the implementation has numerous bugs, including an issue with the threading system rendering Asteroid Zone unplayable. While these bugs could have been fixed during the development of RemoteME, this is not the focus of this project. In addition, the ME4SE project is still quite young, and so the implementation is incomplete, meaning that it too is unsuitable for use as the runtime server.

The second option, create runtime from scratch, has some advantages. The runtime can be built piecemeal just like the rest of RemoteME, implementing features as necessary to keep the system light and fast. Further, the two options discussed thus far are not constrained by the limitations of AspectJ.

This option has the same issue as the previous one in that the project is not focused on developing a J2ME emulator. While it is not necessary to develop a complete implementation of the specification, task is not trivial evidenced by a mature project such as MicroEmulator having showstopping bugs.

The option which we chose is the AspectJ technique. Building on top of the official emulator allows us to assume that games we expect to execute on RemoteME are highly likely to run correctly. Further, while the system is more difficult to debug, building on top of a fully-featured J2ME environment allows for a much simpler server component.

5.3.2 AspectJ Limitations

AspectJ [18] is a mature project with but only has limited support for the J2ME runtime [41]. For example, key features of AspectJ such as the

`thisJoinPoint` field are unavailable for J2ME[2]. This required that support for API methods needed to be added individually, leading to code duplication on the server. Further, if new methods are added to the API, support for them needs to be added to RemoteME manually.

As mentioned earlier (Section 5.1.4), there is also a limitation in pointcutting constructors, meaning that they cannot be pointcut individually for API classes. This is an issue for a number of the J2ME classes, however it only arose for one class for RemoteME – `Sprite`. A workaround was created involving class rewriting, which is sufficient for a prototype, however this is not a great solution for a fully fledged system.

[2] `http://www.eclipse.org/aspectj/doc/released/progguide/language-thisJoinPoint.html`

Chapter 6

Performance Evaluation

In this section we evaluate the responsiveness of RemoteME based upon the findings in Chapter 4. Section 6.1 presents the study into the responsiveness of RemoteME when executing Asteroid Zone over a wireless network. Section 6.2 explores how RemoteME compares against a native version of Asteroid Zone in terms of CPU, memory, battery life and power consumption.

6.1 Measurement Study

This study carried out an experiment to determine whether the responsiveness of RemoteME falls within the range of acceptable response times. This range corresponds to the delay range determined in the previous chapter to be 75-150 ms for a 2D action game such as Asteroid Zone (Section 3.3). We conducted the experiments over a WiFi network, and also made attempts on 2.5 and 3G mobile networks.

6.1.1 Experiment

This experiment evaluated the performance of RemoteME as a remote control system for 2D action games. We evaluated the system's performance

by measuring the response times over 30 executions of the game Asteroid Zone (see Section 4.1.3). We attempted to evaluate Bomber 2 also, but these attempts failed (see Sections 6.1.3 and 6.1.4).

Response time is measured in this study, but in the previous study (Chapter 4), the benchmark was set using delay, and this includes the render time on the mobile phone. Therefore, the render time of the game needs to be determined so that we can accurately evaluate RemoteME. We played Asteroid Zone locally on the mobile phone and recorded the time taken to render each frame. Several thousand times were recorded and the render time was either 0 or 1 ms for 98% of the measured times. This time is much smaller than the range determined in the previous study, and thus can be ignored, giving an acceptable upper bound of 150 ms.

The game Asteroid Zone was played over the RemoteME system 30 times while every action's response time was measured using RemoteME's logging utility. Analysis of the measured response times was performed to determine if RemoteME's performance is acceptable as determined by the user study in Chapter 4.

Hypothesis

The hypothesis is that RemoteME can provide response within the 75 ms to 150 ms range. The null hypothesis (H_0) is:

> The mean response times for RemoteME will be greater than the benchmark 150 ms response time (with a 90% confidence interval).

Design

The experiment had a single player play 30 games of Asteroid Zone on the RemoteME system over a wireless network. The games were played in two groups of 15 games each, with a two hour break between the groups. The client was a mobile phone, and the server a desktop machine.

Artifacts

Asteroid Zone The game used, Asteroid Zone, is the same game from Section 4.1.3.

Mobile Phone The mobile phone used is the same as the previous study (Section 3.3), a Nokia N95. The player used the numerical keys to control the ship during the experiment.

Wireless Router The wireless router used is the same as the pilot study (Section 3.3), and the mobile phone was the only client for the router.

6.1.2 Procedure

The 30 games of Asteroid Zone were played by myself, thus HEC approval was not required. Each game was played following this sequence:

1. A new instance of the RemoteME server is started, as well as the logging server.
2. The player connected to the server using the RemoteME client on the phone, and played a full game, exhausting all lives or completing the "final" level (level five). This endpoint is consistent with the previous study, detailed in Section 4.3.
3. Once the player has finished that game, they activated the logging function of RemoteME, sending the complete log of response times to the logging server.
4. If there were more games to play, then the player went back to step one.

6.1.3 Experimental Issues

The major issue with this experiment is the selection of game. While we will see that RemoteME performed very well with Asteroid Zone in the next section, generalising these results to other game types is unpredictable. We can however say with high likelihood that similar games which are shape based (such as Asteroid Zone) rather than image based (such as Bomber 2) will have similar performance. The related issue is that we only have performance results for a single game. A combination of factors including the mobile phone prevented us from collecting results for Bomber 2, more details can be found in the next section (Section 6.1.4).

Brief testing showed that there was little variation in RemoteME performance between machines selected for the RemoteME server, the only consequence being increased hardware resource consumption on lower end hardware.

Similarly, while the selection of wireless router may be a factor, RemoteME performance was not affected when switching to another wireless router. In addition, congestion in the system is accounted for by executing multiple plays over a significant period of time, and given the simplicity of the network infrastructure, and that only one client is connected to the wireless router, the results in Figure 6.1 are as expected.

Another limitation of the experiment is that we were restricted to a single mobile phone. Based on the findings in the Bomber 2 experiment later in the chapter, as well as the above discussion, the mobile phone plays a key part in the overall performance of RemoteME, and that we need to test the system with other phones.

Finally, it is worth noting that the response times gathered for RemoteME can be reduced by changing the server component. Currently the server reproduces the display of the client, increasing the workload on the server. Removing this facet of the server may lead to improved response times, as well as reduced resource consumption on the server. We believe that this change will make a slight improvement but need to test this to confirm.

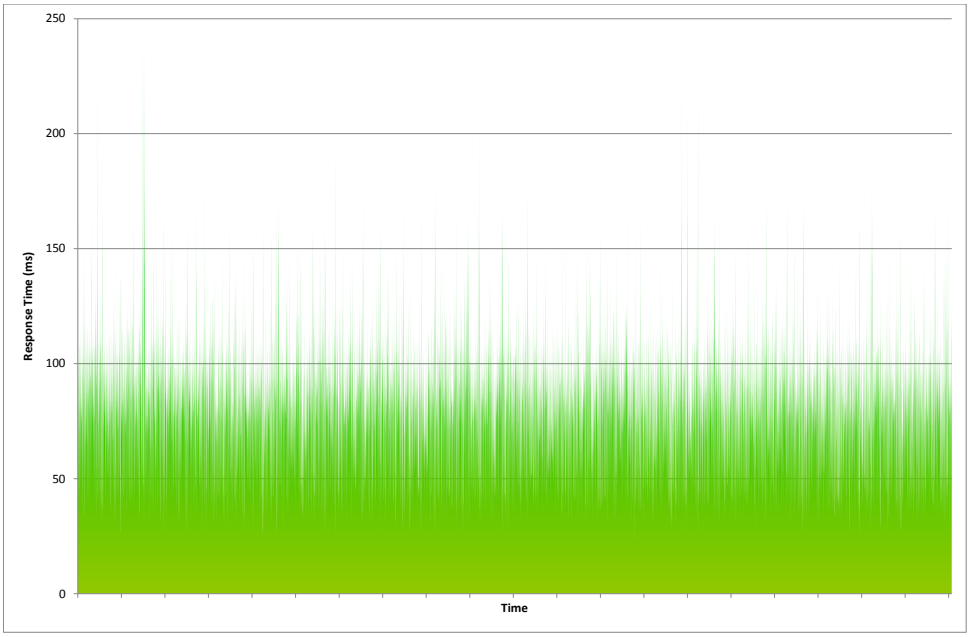


Figure 6.1: All response times for Asteroid Zone on RemoteME over WiFi.

6.1.4 Technical Issues

Multiple attempts were made to execute Bomber 2, the game from the user study (Section 4.1.3), on RemoteME. However the responsiveness of the game was very poor, with response times exceeding 1000 ms, and the logging system recording too few data points to perform statistical analysis. The most likely explanation for this is the mobile phone, specifically the capabilities of it. From the screenshot of Bomber 2 (Figure 4.3) we can see that it is visually complex, so much so that there are a significant number of updates sent to the client for each frame that is rendered. As a comparison, over 200 packets are sent for a typical frame of Bomber 2, whereas fewer than 30 packets are sent for Asteroid Zone.

In order to determine whether the bottleneck is the network or graphics on the client, the RemoteME client was configured to receive all updates but not perform any graphics operations. In this mode the response times

were considerably lower, at approximately 500-600 ms. This indicates that both the client's networking and graphics systems are being taxed, and we suspect that the phone cannot keep up with the number of packets being received, and that once it has passed on those packets to the RemoteME client, the client cannot render the frame before the next frame has arrived.

Finally, to test the potential of RemoteME over the Internet, we attempted to play Asteroid Zone on RemoteME using Vodafone's 3G mobile network as the communications medium. Unfortunately, an issue with the phone caused the 3G connection to drop immediately after the game was started, so no data could be collected. Further attempts to play over Telecom 2.5G were more successful, however it was not responsive enough to record data.

6.1.5 Results

In total 30 executions of Asteroid Zone over RemoteME were conducted, giving over 20,000 data points. The mean response time is 77.3 ms, with a standard deviation of 31.8 ms. The entire data set is shown in Figure 6.1, note that the response times and time do not seem to be related, suggesting no temporal correlation (by visual inspection).

At a 90% confidence level the interval for Asteroid Zone on RemoteME is 76.9 to 77.7 ms. This is within the 75-150 ms range for acceptable responsiveness. Further, RemoteME's response time logging may record times greater than they actually are (Section 5.2), meaning that the confidence interval may be lower than the lower bound of the acceptable range.

Figure 6.2 gives the distribution of response times between 0 and 200 ms, grouped by execution. This chart shows that every execution has two distinct peaks in the distribution. Taking the data in aggregate gives the distribution graph in Figure 6.3. To determine the cause of the twin peaks, some investigation is required.

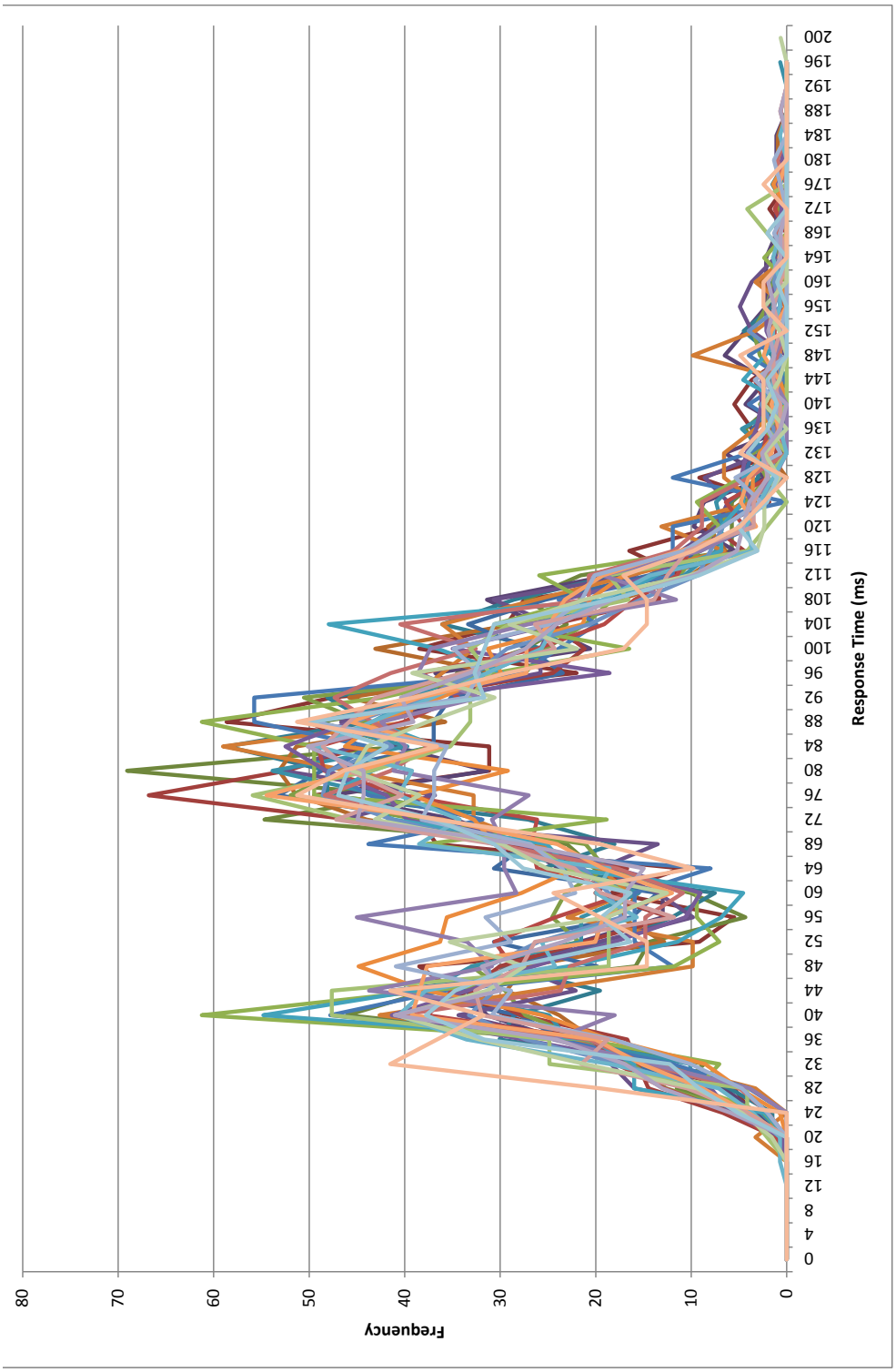


Figure 6.2: Response times for Asteroid Zone on RemoteME over WiFi. Each coloured line corresponds to a single execution. The response times are collected into buckets of size 4, and the frequency is the number of response times in that bucket. Response times over 200 ms are insignificant and thus ignored.

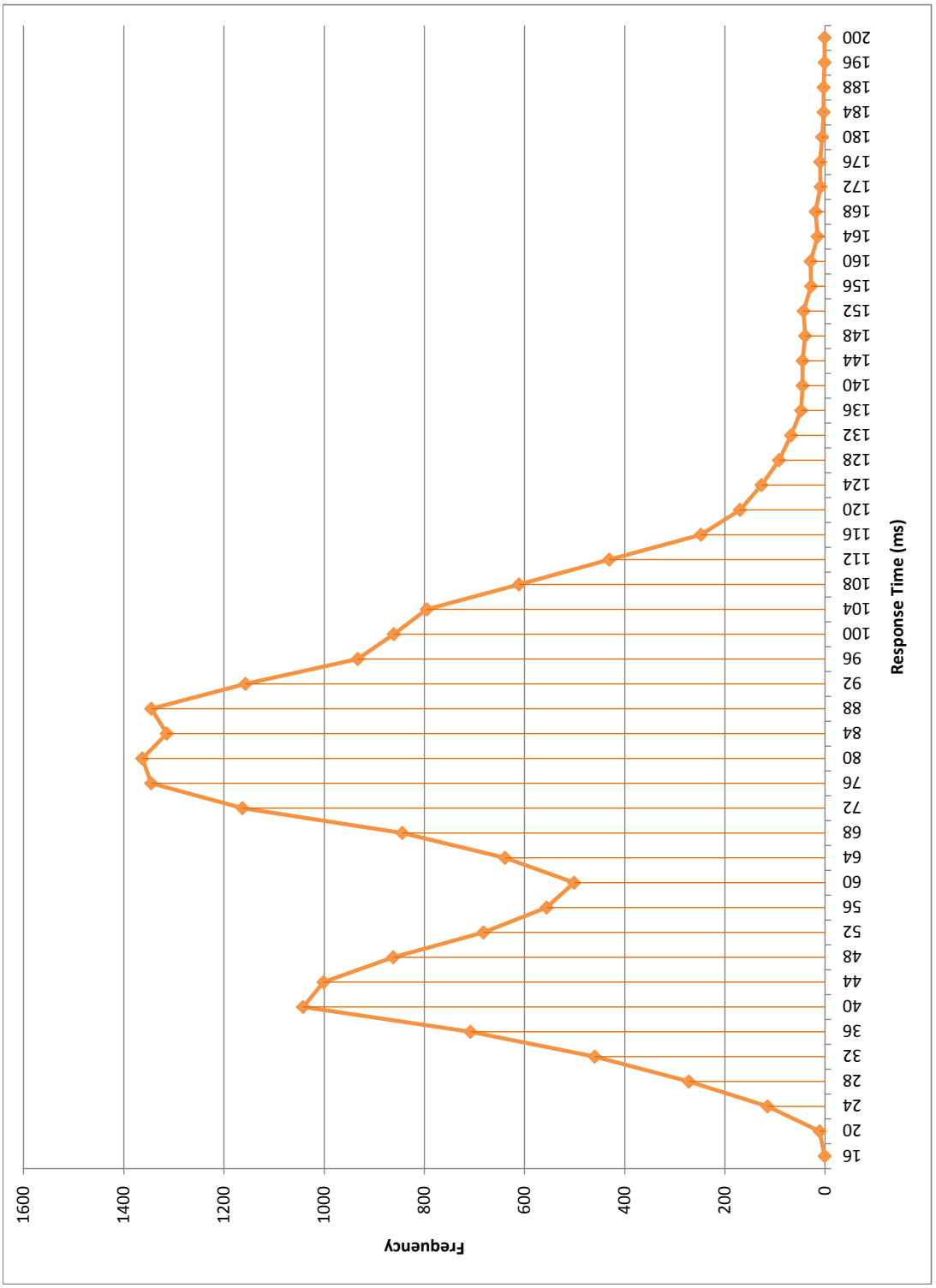


Figure 6.3: Distribution of response times for Asteroid Zone on RemoteME over WiFi. The response times are collected into buckets of size 4, and the frequency is the number of response times in that bucket. Response times over 200 ms are insignificant and thus ignored.

Looking at the source code for Asteroid Zone, the main program loop of the game consists of four primary operations:

```
1: loop
2:     MoveOtherObjects()
3:     ProcessKeys()
4:     MovePlayerShip()
5:     PaintScreen()
6:     Sleep(max(50 - looptime, 20))
7: end loop
```

Line 6 is the game pausing to regulate the frame rate. The pause lasts between 20 and 50 ms, depending on how long that iteration of the main loop took to execute. The time at the start of each loop iteration was recorded for one execution of Asteroid Zone on RemoteME, which we can take to be the same as the elapsed time for an iteration, and from the distribution in Figure 6.4 we can assume that it is 50 ms.

Based on this finding, the two peaks can be best explained by key presses being received on the server after it has already processed keys in that iteration, i.e. the key press is received after line 3 has executed. Thus, setting aside variations in network transfer speeds, there are two scenarios: if the key press arrives on the server before it has processed keys (line 3), then the response time will be 40–48 ms, i.e. the first peak from Figure 6.3; otherwise if the key press arrives after the server has processed keys, then the server will process it after waiting for up to 50 ms, giving the second peak of 76–92 ms.

Another key piece of data is the proportion of response time spent between processing keys (line 3) and sending the paint command to the client, which happens at the end of `PaintScreen` (line 6). This is shown in Figure 6.5, with a mean of 4.38 ms and standard deviation 2.02 ms.

Finally, the remainder of the response time is allocated to network transfer. The time to send a single packet from a mobile client to a desktop server and back was also recorded to a file on the mobile phone. Here the

client had two threads running, one sending the unique timestamp to the server, and the other receiving those timestamps. The time between that timestamp and the time at which the client received it is the round trip time, and one execution of this setup recorded over 400,000 values, shown in Figure 6.6.

The mean round-trip time is 11.79 ms (rounded to 12 ms), and the standard deviation 17.32 ms. If we assume that going in a single direction takes approximately half the time, then we get 6 ms for the time taken to send key presses in RemoteME.

Based on these results, we can divide the response time $t_{response} = t_{send} + t_{server} + t_{receive}$, with the send time 6 ms, server time 4 ms or 54 ms, and receive time the remainder.

6.1.6 Conclusion

This study evaluated the responsiveness of RemoteME on a WiFi network with a 2D action game, Asteroid Zone. We determined that the acceptable range of response times is 75-150 ms, and the results showed that RemoteME is within that range at 90% confidence. Further we analysed how the response times are divided among the different stages of the system, and attributed the bulk of the response time to the transferring of the graphics updates. We suggested that performance can be improved by reconfiguring RemoteME so that the server does not duplicate the application display. Moreover, the failed results for Bomber 2 suggest that performance may be hindered by the mobile phone used for the experiment, and that a more capable mobile phone may lead to greater responsiveness.

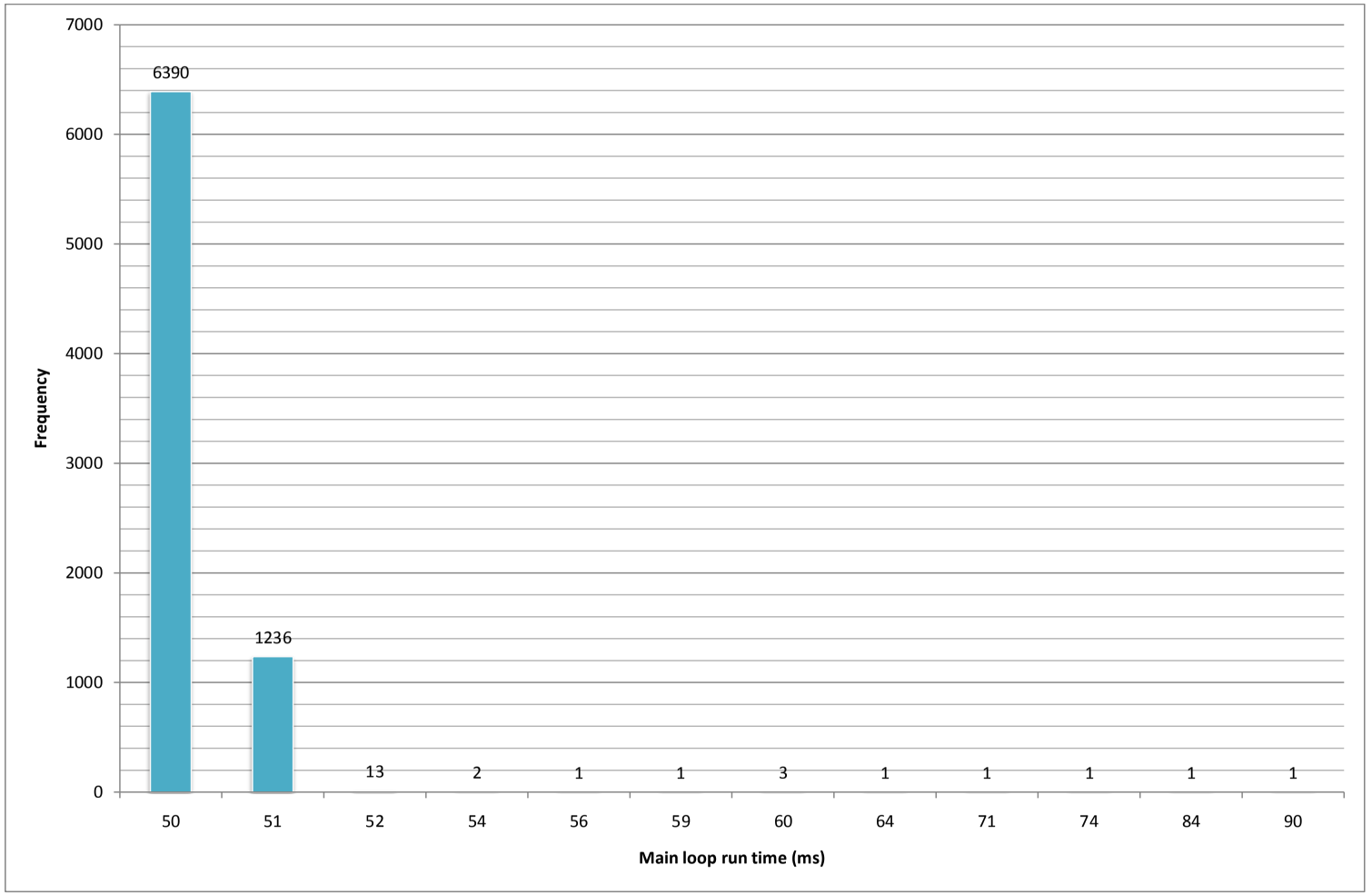


Figure 6.4: Main loop execution time for RemoteME running Asteroid Zone. The frequency is the number of occurrences of that response time in the data – omitted times have 0 frequency.

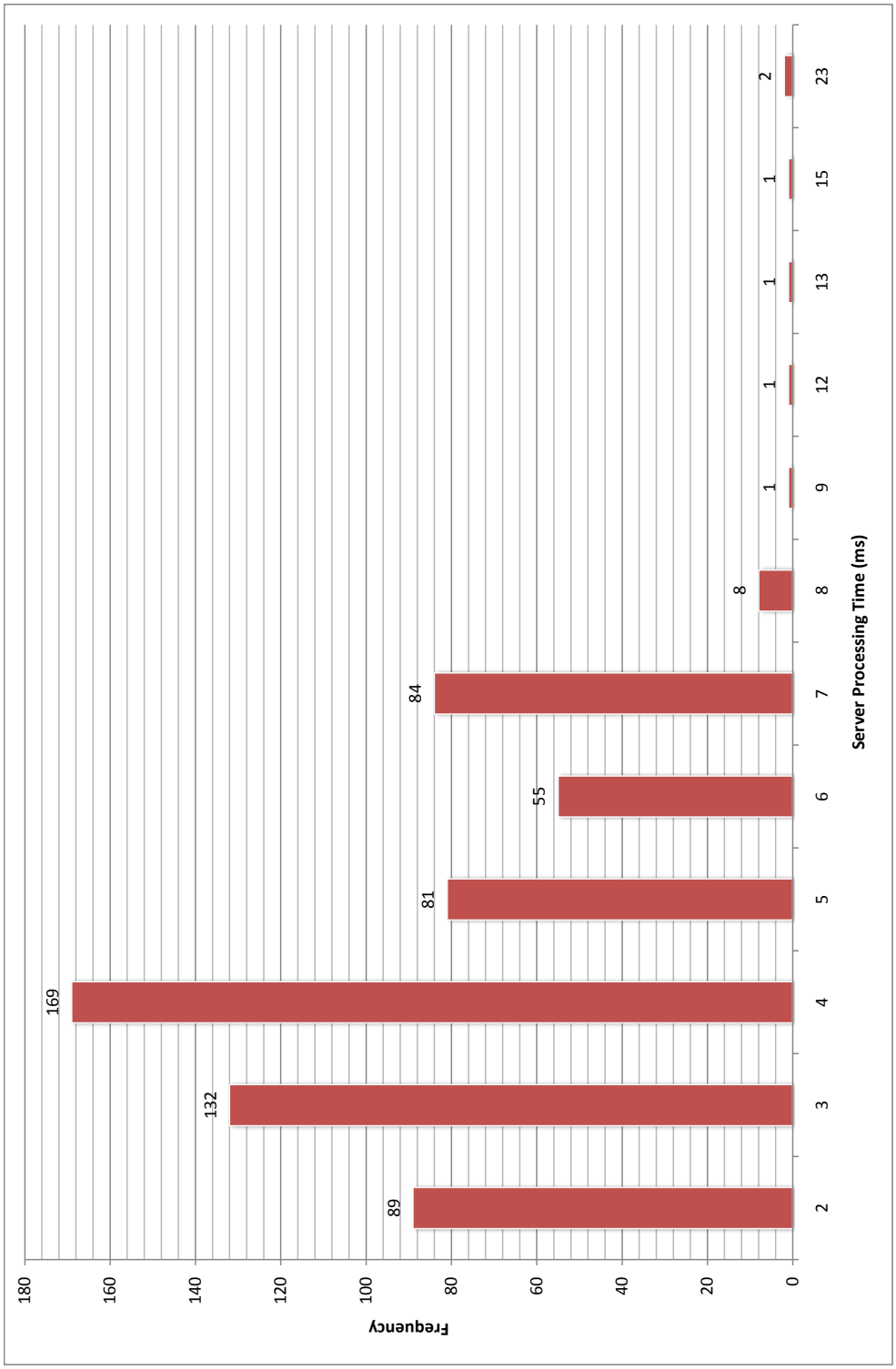


Figure 6.5: Server processing time for RemoteME running Asteroid Zone. The frequency is the number of occurrences of that time in the data – omitted times have 0 frequency.

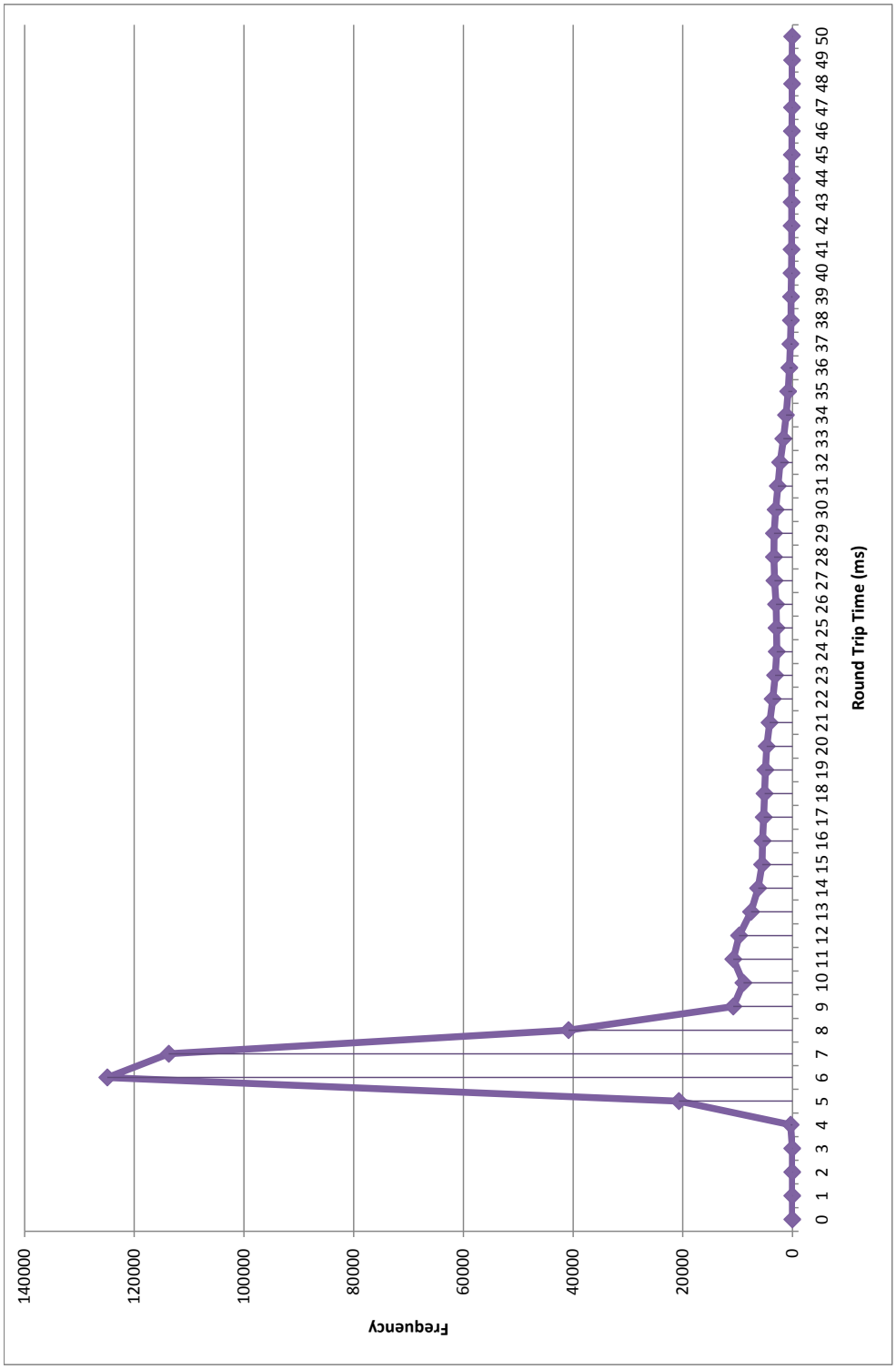


Figure 6.6: Round-trip times for a simple client/server. The frequency is the number of occurrences of that response time in the data, and times over 50 ms are insignificant and thus ignored.

6.2 RemoteME Device Statistics

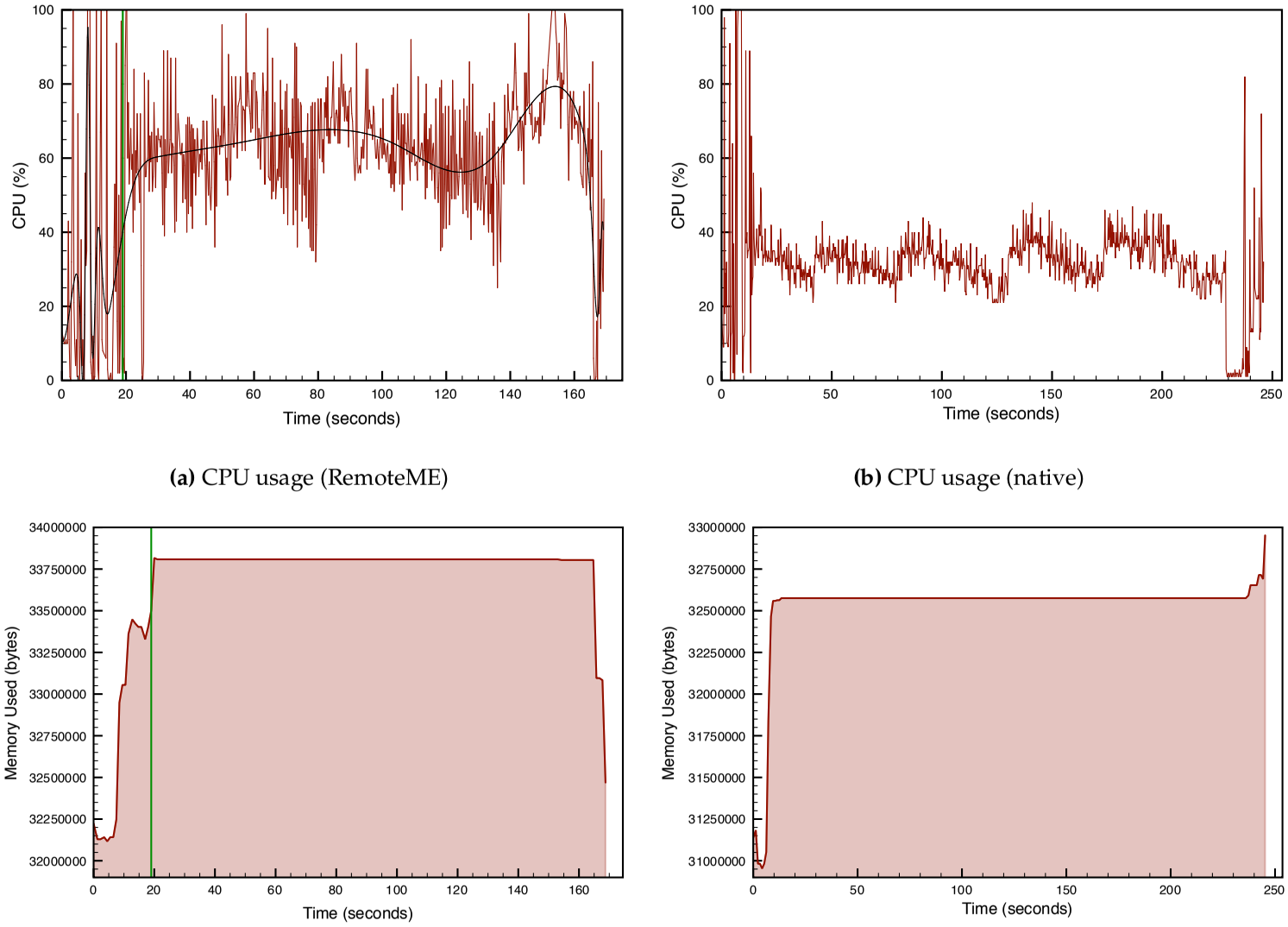
As discussed in the last section the typical response time for Asteroid Zone on RemoteME over WiFi is 77 ms. In addition to this performance statistic, there are various other statistics which were gathered for a single run of Asteroid Zone on RemoteME over WiFi, and for just Asteroid Zone natively on the phone. All the following were gathered in the same run, on the same Nokia N95 mobile phone as discussed previously. The data was gathered using a third-party program for the N95's operating system, Nokia Energy Profiler [24].

6.2.1 CPU

Figure 6.7a shows the CPU usage for RemoteME. The connection to the server is made at time 19 (indicated by the vertical line). While the CPU usage varies quite strongly, it is centered around the 65% mark; indeed, the mean is 59%. By comparison, the native run of Asteroid Zone is in Figure 6.7b, and has a much lower CPU usage overall, averaging 31%. Further, the native CPU usage is somewhat proportional to the number of objects on the screen, with clear CPU spikes at the start of a new level.

6.2.2 Memory

The memory usage for RemoteME is quite stable, as shown by Figure 6.7c. Once the connection is made at time 19, the amount of memory used on the phone is unchanged until the application exits. Likewise, the native Asteroid Zone memory usage is also stable (Figure 6.7d). Overall, the maximum memory usage for RemoteME and native is 1,699,840 bytes and 1,998,848 bytes respectively.



(a) CPU usage (RemoteME)

(b) CPU usage (native)

(c) Memory usage (RemoteME)

(d) Memory usage (native)

Figure 6.7: CPU and Memory usage of Asteroid Zone on RemoteME and natively.

6.2.3 Power and Battery

There are two statistics to consider here, RemoteME's power consumption and the change in battery level. The first is shown in Figure 6.8a as quite stable, averaging 1.55 Watts. In comparison, native Asteroid Zone (Figure 6.8b) has a mean of 0.43 Watts. This is a very large difference between the two, and can be attributed to RemoteME heavily using the power consuming WiFi radio.

The battery level is highly strange, however. In both RemoteME (Figure 6.8c) and natively (Figure 6.8d) the charge varies rather than just constantly decreasing. Note that in both cases the phone was not plugged in to power and was running solely off the internal battery; indeed, the monitoring application requires that this is the case.

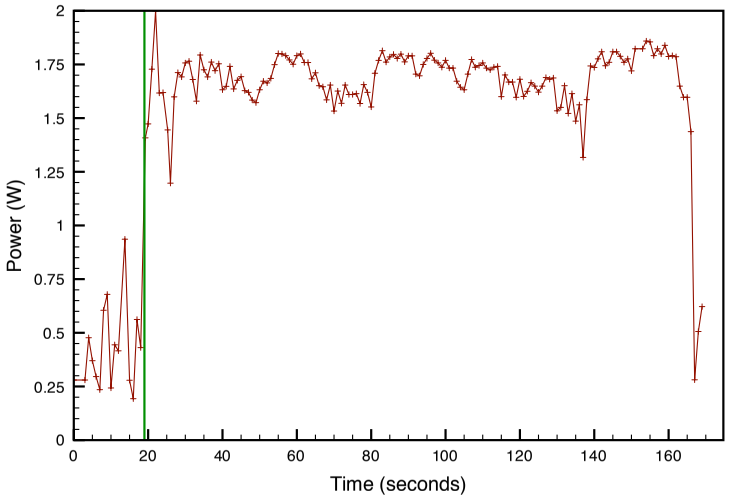
6.2.4 Network traffic

The final statistic is the network traffic. Note that this data is RemoteME only, as the native application does not need make external connections. The network traffic rate is continuously changing rather than being a constant rate. The mean rate is 28,370 bytes/sec and peaks at 55,879 bytes/sec.

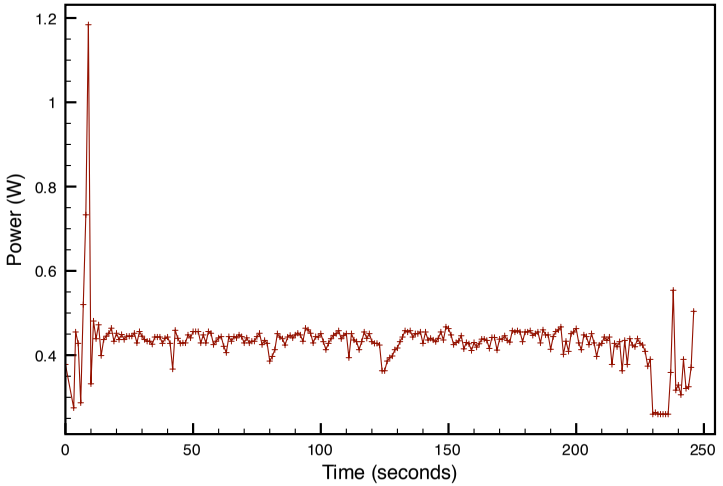
6.2.5 Conclusion

Overall, considering the various performance factors, RemoteME uses greater resources than running the game Asteroid Zone natively. While RemoteME's memory usage seems reasonable, the CPU usage is quite high. However, the more important factors when on a mobile device are the power consumption and battery drain, and RemoteME uses considerably more power than the native game. The battery also drains at a greater rate for RemoteME.

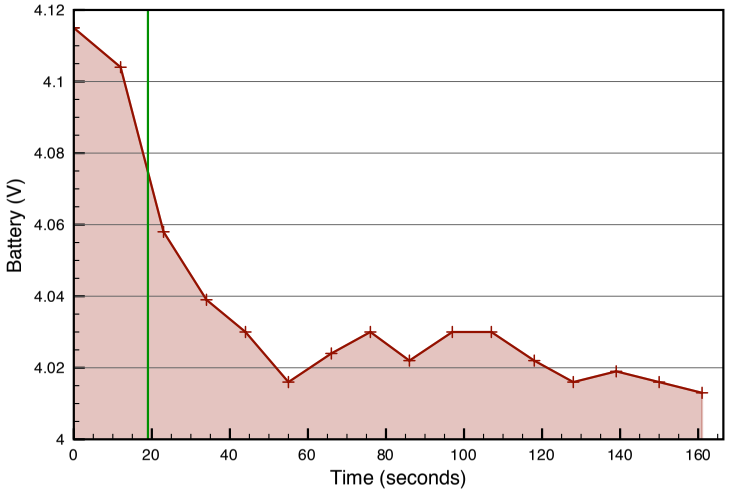
While RemoteME has a larger footprint, this footprint is likely to be



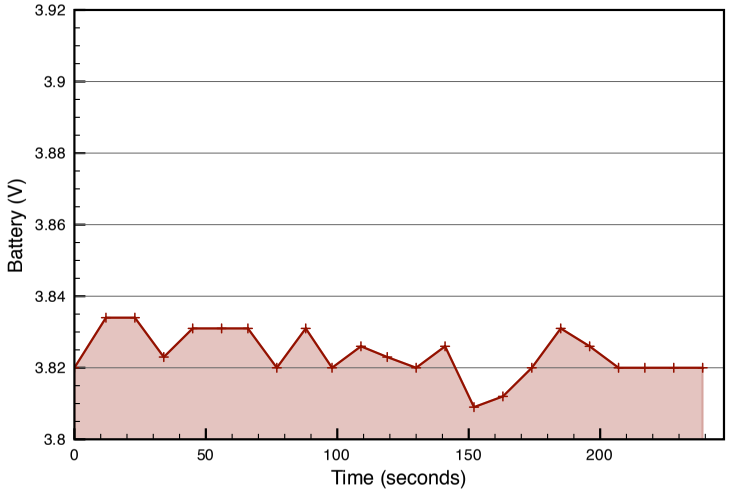
(a) Power consumption (RemoteME)



(b) Power consumption (native)



(c) Battery level (RemoteME)



(d) Battery level (native)

Figure 6.8: Power consumption and battery level of Asteroid Zone on RemoteME and natively.

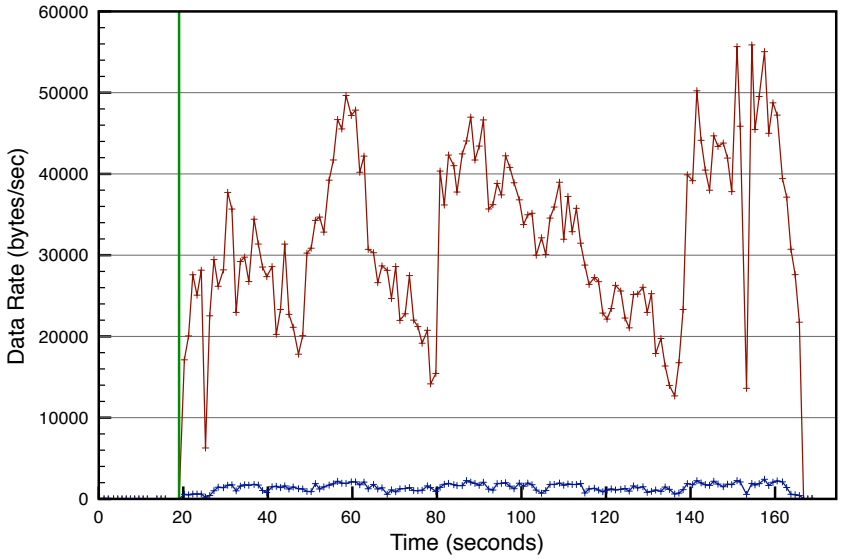


Figure 6.9: Network traffic during Asteroid Zone on RemoteME. The green vertical line marks the start of the connection, and in this case the start of the data. Inbound traffic is the upper plot (in red), and outbound the lower plot (in blue).

stable regardless of the game. The memory usage will vary based on the number of objects in the game, but the primary factor will be the pattern of updates. The number of updates will affect the network traffic, which will affect the WiFi usage. Ultimately the biggest issue will be managing the power consumption and battery drain because of the use of the WiFi radio.

Chapter 7

Conclusions

The goal of this thesis is to explore a new technique for mobile computing, remote control using a specialised client designed to support a particular domain of applications.

In Chapter 2 we detailed the motivation for the work, including the preservation of digital heritage, and games in particular. We also looked at existing solutions for remote control, and explored recent developments in the remote gaming market.

Chapter 3 discussed the three phases of the project: initial development of the prototype system called RemoteME; establishing the acceptable responsiveness requirements for architectures such as our own; and evaluation of the prototype system against the responsiveness requirements. We also described in detail the pilot study which first attempted to determine responsiveness requirement.

Chapter 4 discussed in detail the larger user study, which gave statistically significant results. The outcome of this study established that the acceptable responsiveness for 2D action games lies within the 75 ms to 150 ms range at a 90% confidence level, and that further studies are required to pinpoint the value within this range.

In Chapter 5 we presented RemoteME, our client/server system supporting thin-client mobile gaming. Here we discussed the implementation

of RemoteME, including a per-package completion status. We also highlighted key implementation issues encountered during the development of RemoteME.

Finally, Chapter 6 presented the evaluation of RemoteME against the responsiveness requirement we established. We showed that RemoteME is at the lower end of this range for Asteroid Zone, and that this performance can be improved upon. We also compared the RemoteME client footprint with the native game, and hypothesised that this footprint may more stable across similar games than their native counterparts.

7.1 Contributions

The contributions of this thesis are:

1. **Establishing a responsiveness requirement** – we have defined what is meant by acceptable responsiveness, and have determined the requirement on this falls within the 75 ms to 150 ms range of response times.

2. **Development of a remote control technique** – we proposed a technique for a remote control system and proved that it is feasible within the responsiveness requirement.

3. **Prototyping the approach through RemoteME** – we have developed a prototype system which implements our remote control technique, and evaluated the system against the responsiveness requirement.

7.2 Future Work

We suggest a number of areas for future work:

Improving performance for all games RemoteME can be modified such that performance for complicated games (such as Bomber 2) may improve without changing the client device.

Complete the API implementation While RemoteME's API coverage is sufficient for our evaluation purposes, we would like to see the whole J2ME API implemented. For example, completing the non-graphics UI system would allow for a set of applications outside the gaming space to execute on RemoteME, as well as completing the support for games which use J2ME's forms.

Standalone server runtime Currently the RemoteME server runtime is limited by the boundaries imposed by both AspectJ and J2ME (Section 5.3). A standard Java server runtime would allow for more freedom, and would improve the responsiveness of the system as a whole, for example by eliminating the server-side display "echo" which currently occurs.

Porting the client The RemoteME client requires a J2ME stack, however we would like to port the client to other platforms such as Apple's iPhone and even a native Java version.

Wider range of software/hardware The RemoteME protocol is designed for J2ME applications, however we would love to see it brought to other software/hardware platforms, for example Apple's Cocoa-Touch or Google Android's UI system.

Appendix A

HEC Documents

A.1 HEC Application Form

The following document is the application form for HEC approval for the user study.

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

HUMAN ETHICS COMMITTEE
Application for Approval of Research Projects

Please write legibly or type if possible. **Applications must be signed by supervisor (for student projects) and Head of School**

Note: The Human Ethics Committee attempts to have all applications approved within three weeks but a longer period may be necessary if applications require substantial revision.

1 NATURE OF PROPOSED RESEARCH:

(a) Staff Research	Student Research ✔		(tick one)	
(b) If Student Research	Degree	MSc	Course Code	**COMP591**
(c) Project Title:	Evaluation of Player Performance in Remote Mobile Gaming			

2 INVESTIGATORS:

(a) Principal Investigator	
Name	Vipul Delwadia
e-mail address	vipul@ecs.vuw.ac.nz
School/Dept/Group	ECS

(b) Other Researchers Name	Position
Stuart Marshall	Lecturer
Ian Welch	Senior Lecturer

(c) Supervisor (in the case of student research projects)	
Stuart Marshall, Ian Welch	

3 DURATION OF RESEARCH

(a) Proposed starting date for data collection 01/06/2009

(Note: that NO part of the research requiring ethical approval may commence prior to approval being given)

(b) Proposed date of completion of project as a whole 19/08/2009

4 PROPOSED SOURCE/S OF FUNDING AND OTHER ETHICAL CONSIDERATIONS

(a) Sources of funding for the project

Please indicate any ethical issues or conflicts of interest that may arise because of sources of funding e.g. restrictions on publication of results

Small research grant held by Dr Stuart Marshall in ECS paid from consultancy.

(b) Is any professional code of ethics to be followed **Y**

Association of Computing Machinery (ACM)

(c) Is ethical approval required from any other body **N**

If yes, name and indicate when/if approval will be given

5 DETAILS OF PROJECT

Briefly Outline:

(a) The objectives of the project

Identify how player performance in remote mobile games is affected by network and display response times.

b) Method of data collection

Subjects will play multiple runs of games on the provided mobile phone, and then:

(a) we will record their game scores and
(b) the subject will answer a post-experience written questionnaire

The experiment will run in two sessions (where each session is identical except for the amount of network and display delay). All subjects will participate in both sessions. The sessions will be at least three weeks apart, and each session will be four hours long for each subject.

(c) The benefits and scientific value of the project

This experiment will establish the bounds on reasonable delays in network and display transmission and computation for remote mobile games. These bounds will be informed by both player satisfaction and performance, and will create a benchmark against which tools in this area can be measured against.

(d) Characteristics of the participants

Participants are computer literate members of the current or past University community, and will have some experience of using mobile phones and having had played computer games.

(e) Method of recruitment

Word of mouth, advertisement

(f) Payments that are to be made/expenses to be reimbursed to participants

We will have a draw for two $50 vouchers that all test subjects will have an equal chance of winning.

(g) Other assistance (e.g. meals, transport) that is to be given to participants

None

(h) Any special hazards and/or inconvenience (including deception) that participants will encounter

None

(i) State whether consent is for:

(i)	the collection of data	**Y**
(ii)	attribution of opinions or information	**Y**
(iii)	release of data to others	**N**
(iv)	use for a conference report or a publication	**Y**
(v)	use for some particular purpose (specify)	**Y**

Data will be used to report and support findings for an MSc thesis.

Attach a copy of any questionnaire or interview schedule to the application

(j) How is informed consent to be obtained (see sections 4.1, 4.5(d) and 4.8(g) of the Human Ethics Policy)

(i) the research is strictly anonymous, an information sheet is supplied and informed consent is implied by voluntary participation in filling out a questionnaire for example (include a copy of the information sheet) **N**

(ii) the research is not anonymous but is confidential and informed consent will be obtained through a signed consent form (include a copy of the consent form and information sheet) **Y**

(iii) the research is neither anonymous or confidential and informed consent will be obtained through a signed consent form (include a copy of the consent form and information sheet) **N**

(iv) informed consent will be obtained by some other method (please specify and provide details) **N**

With the exception of anonymous research as in (i), if it is proposed that written consent will not be obtained, please explain why

(k) If the research will not be conducted on a strictly anonymous basis state how issues of confidentiality of participants are to be ensured if this is intended. (See section 4..1(e) of the Human Ethics Policy). (e.g. who will listen to tapes, see questionnaires or have access to data). Please ensure that you distinguish clearly between anonymity and confidentiality. Indicate which of these are applicable.

(i) access to the research data will be restricted to the investigator **N**

(ii) access to the research data will be restricted to the investigator and their supervisor (student research) **Y**

(iii) all opinions and data will be reported in aggregated form in such a way that individual persons or organisations are not identifiable **Y**

(iv) Other (please specify)

(l) Procedure for the storage of, access to and disposal of data, both during and at the conclusion of the research. (see section 4.12 of the Human Ethics Policy). Indicate which are applicable:

(i) all written material (questionnaires, interview notes, etc) will be kept in a locked file and access is restricted to the investigator **Y**

(ii) all electronic information will be kept in a password-protected file and access will be restricted to the investigator **Y**

(iii) all questionnaires, interview notes and similar materials will be destroyed:

(a) at the conclusion of the research **N**

or (b) 2 years after the conclusion of the research **Y**

(iv) any audio or video recordings will be returned to participants and/or electronically wiped **N/A**

(v) other procedures (please specify):

If data and material are not to be destroyed please indicate why and the procedures envisaged for ongoing storage and security

(m) Feedback procedures (See section 7 of Appendix 1 of the Human Ethics Policy). You should indicate whether feedback will be provided to participants and in what form. If feedback will not be given, indicate the reasons why.

Test subjects will be invited to provide their email address, to which an electronic copy of the MSc thesis will be sent to on the acceptance of the thesis.

(n) Reporting and publication of results. Please indicate which of the following are appropriate. The proposed form of publications should be indicated on the information sheet and/or consent form.

(i) publication in academic or professional journals **Y**
(ii) dissemination at academic or professional conferences **Y**
(iii) deposit of the research paper or thesis in the University Library (student research) **Y**
(iv) other (please specify)

Signature of investigators as listed on page 1 **(including supervisors) and Head of School.**

NB: <u>All investigators and the Head of School must sign before an application is submitted for approval</u>

	Date	
	Date	
	Date	

Head of School:

	Date	

APPLICATIONS FOR HUMAN ETHICS APPROVAL

CHECKLIST

- Have you read the Human Ethics Policy?
- Is ethical approval required for your project?
- Have you established whether informed consent needs to be obtained for your project?
- In the case of student projects, have you consulted your supervisor about any human ethics implications of your research?
- Has your supervisor read and signed the application?

- Have you included an information sheet for participants which explains the nature and purpose of your research, the proposed use of the material collected, who will have access to it, whether the data will be kept confidential to you, how anonymity or confidentiality is to be guaranteed?

- Have you included a written consent form?
- If not, have you explained on the application form why you do not need to get written consent?
- Are you asking participants to give consent to:

    - collect data from them
    - attribute information to them
    - release that information to others
    - use the data for particular purposes

- Have you indicated clearly to participants on the information sheet or consent form how they will be able to get feedback on the research from you (e.g. they may tick a box on the consent form indicating that they would like to be sent a summary), and how the data will be stored or disposed of at the conclusion of the research?

- Have you included a copy of any questionnaire or interview checklist you propose using?

- Has your application been seen by the head of your school or department (or the person given responsibility to consider applications on behalf of the head (see section 4.5(b) of the Human Ethics Policy).

PLEASE FORWARD YOUR COMPLETED APPLICATION FORM TO THE SECRETARY, HUMAN ETHICS COMMITTEE OR, IN THE CASE OF APPLICATIONS FROM SCHOOLS OR DEPARTMENTS WITH AN APPROVED ETHICS SUB-COMMITTEE, TO THE CONVENER OF THAT SUB-COMMITTEE

A.2 Information Sheet

This sheet was given to all participants of the user study.

Participant Information Sheet for a Study of Response Times

Researcher: Vipul Delwadia, School of Engineering and Computer Science, Victoria University of Wellington

I am a Masters student in Computer Science at Victoria University of Wellington, and my research is experimenting with remote technologies on a mobile phone. As part of my research, I am evaluating the effects of delaying the response to an individual's input on their performance in computer games, specifically on mobile phones. Both their game score and satisfaction are of interest.

The study has no requirements on the characteristics of the participants. The participants are required to play games on a provided mobile phone in two sessions. During each session they will answer a questionnaire and their scores will be recorded. There is a break of approximately two weeks between the sessions, and the total time required of the participants should not exceed four hours.

The responses collected will contribute to my research and will be put into a written report in an aggregated form to preserve confidentiality. There will be no personally identifiable material, as the responses will be presented as a group or quoted against a moniker. The only people that will see the responses are myself and my supervisors, Stuart Marshall and Ian Welch. The thesis will be submitted for marking to the School of Engineering and Computer Science, and deposited in the University Library. It is intended that one or more articles will be submitted for publication in scholarly journals.

All data collected will be stored securely in a locked file, and two years after the conclusion of the research will be destroyed. Participants can receive an electronic copy of the thesis upon acceptance by providing an email address. Participants can withdraw from the study until two weeks prior to submission of the thesis.

There will be a prize draw for two $50 vouchers, and all participants have an equal chance of winning.

If you have any questions or would like to receive further information about the project, please contact me at vipul@ecs.vuw.ac.nz or my supervisors, Stuart Marshall and Ian Welch, at the School of Engineering and Computer Science at Victoria University, P O Box 600, Wellington.

A.3 Consent Form

All participants signed the following form before commencing the user study.

VICTORIA UNIVERSITY OF WELLINGTON

CONSENT TO PARTICIPATION IN RESEARCH

Evaluation of Player Performance in Remote Mobile Gaming:
I have been given and have understood an explanation of this research project. I have had an opportunity to ask questions and have them answered to my satisfaction. I understand that I may withdraw myself (or any information I have provided) from this project (up to two weeks prior to submission of the thesis) without having to give reasons.

I understand that any information I provide will be kept confidential to the researcher and the supervisors, the published results will not use my name, and that no opinions will be attributed to me in any way that will identify me.

Signed:

_____

Name of participant (please print clearly):

_____

Email (if you wish to receive an electronic copy of the thesis):

_____

Bibliography

[1] ADVANCED MICRO DEVICES. Advanced Micro Devices. http://www.amd.com/, September 2009.

[2] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the Clouds: A Berkeley View of Cloud Computing. Tech. rep., UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009.

[3] BBC. Pilotless police drone takes off. http://news.bbc.co.uk/2/hi/uk_news/england/merseyside/6676809.stm, May 2007.

[4] BEIGBEDER, T., COUGHLAN, R., LUSHER, C., PLUNKETT, J., AGU, E., AND CLAYPOOL, M. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003®. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games* (New York, NY, USA, 2004), ACM, pp. 144–151.

[5] BLIZZARD ENTERTAINMENT. WarCraft III. http://www.blizzard.com/us/war3/, August 2009.

[6] BROWN, M. A. Components of Linux Traffic Control. http://linux-ip.net/articles/Traffic-Control-HOWTO/components.html, September 2009.

[7] BUYYA, R., YEO, C. S., VENUGOPAL, S., BROBERG, J., AND BRANDIC, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems 25*, 6 (2009), 599 – 616.

[8] CORBETT, S. Digital Heritage: Legal Barriers to Conserving New Zealand's Early Video Games. *New Zealand Business Law Quarterly 13*, 5 (April 2007), 48–71.

[9] DOUE, J.-F. Asteroid Zone. `http://jfdoue.free.fr/index.html`, September 2008.

[10] EPIC GAMES, INC. Unreal Tournament 2003. `http://www.unrealtournament2003.com/ut2003/index.html`, August 2009.

[11] GAIKAI. Gaikai. `http://www.gaikai.com`, September 2009.

[12] GREG ROELOFS, J.-L. G., AND ADLER, M. zlib. `http://www.zlib.net/`, September 2008.

[13] HAUSTEIN, S., KROLL, M., AND PLEUMANN, J. ME4SE. `http://kobjects.sourceforge.net/me4se/`, October 2009.

[14] HAYES, B. Cloud computing. *Commun. ACM 51*, 7 (2008), 9–11.

[15] INTERNATIONAL TELECOMMUNICATION UNION. P.800 : Methods for subjective determination of transmission quality. Tech. rep., International Telecommunication Union, August 2006.

[16] JURGELIONIS, A., FECHTELER, P., EISERT, P., BELLOTTI, F., DAVID, H., LAULAJAINEN, J. P., CARMICHAEL, R., POULOPOULOS, V., LAIKARI, A., PERÄLÄ, P., GLORIA, A. D., AND BOURAS, C. Platform for distributed 3d gaming. *International Journal of Computer Games Technology 2009* (2009), 15.

[17] KHRONOS GROUP. Opengl. `http://www.opengl.org/`, July 2009.

[18] KICZALES, G., HILSDALE, E., HUGUNIN, J., KERSTEN, M., PALM, J., AND GRISWOLD, W. G. An Overview of AspectJ. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming* (London, UK, 2001), Springer-Verlag, pp. 327–353.

[19] LIBRARY OF CONGRESS. 'Preserving Virtual Worlds' project. `http://pvw.illinois.edu/pvw/`, January 2009.

[20] LORIE, R. A. Long term preservation of digital information. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries* (New York, NY, USA, 2001), ACM, pp. 346–352.

[21] MICROEMULATOR TEAM. Microemulator. `http://www.microemu.org/`, October 2009.

[22] MILLER, M. *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. Que Publishing Company, 2008.

[23] MILLER, R. B. Response time in man-computer conversational transactions. In *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (New York, NY, USA, 1968), ACM, pp. 267–277.

[24] MOBILE NETWORK. Nokia Energy Profiler. `http://www.symbian-freeware.com/download-nokia-energy-profiler.html`, October 2009.

[25] MSDN. RDP. `http://msdn.microsoft.com/en-us/library/cc240445.aspx`, September 2008.

[26] MSDN. DirectX. `http://msdn.microsoft.com/en-us/directx/default.aspx`, July 2009.

[27] NOMACHINE. NX. http://www.nomachine.com/documents/NX-XProtocolCompression.php, September 2008.

[28] ONLIVE. OnLive: The Future of Video Games. http://www.onlive.com/, August 2009.

[29] OTOY. Otoy. http://www.otoy.com/, September 2009.

[30] PACKARD, K., AND GETTYS, J. X Window System Network Performance. In *USENIX 2003: Proceedings for the Annual Technical Conference, FREENIX Track* (2003), Cambridge Research Laboratory, HP Labs, pp. 207–218.

[31] PLANETS. PLANETS (Preservation and Long-term Access through Networked Services) project. http://www.planets-project.eu/, January 2009.

[32] ROTHENBERG, J. Ensuring the longevity of digital documents. *Scientific American 272*, 1 (1995), 42–47.

[33] SCHEIFLER, R. W., AND GETTYS, J. *X Window system: the complete reference to Xlib, X protocol, ICCCM, XLFD*. Digital Press, Newton, MA, USA, 1990.

[34] SHELDON, N., GIRARD, E., BORG, S., CLAYPOOL, M., AND AGU, E. The effect of latency on user performance in Warcraft III. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games* (New York, NY, USA, 2003), ACM, pp. 3–14.

[35] SPAWN LABS, INC. Spawn labs. http://www.spawnlabs.com, September 2009.

[36] SUN MICROSYSTEMS. Java ME. http://java.sun.com/javame/index.jsp, October 2009.

[37] SUN MICROSYSTEMS. Sun Java Wireless Toolkit for CLDC. `http://java.sun.com/products/sjwtoolkit/`, October 2009.

[38] SUN MICROSYSTEMS. Sun Microsystems. `http://www.sun.com/`, October 2009.

[39] SWALWELL, M., AND DAVIDSON, M. "Malzak". Accepted for publication, Forthcoming.

[40] TECHGENIX. Overview of Terminal Services. `http://www.windowsnetworking.com/articles_tutorials/Overview-Terminal-Services.html`, September 2009.

[41] THE ECLIPSE FOUNDATION. AspectJ Frequently Asked Questions. `http://www.eclipse.org/aspectj/doc/released/faq.php#q:aspectjandj2me`, October 2009.

[42] TIGHTVNC. TightVNC. `http://www.tightvnc.com/`, September 2008.

[43] UNESCO. *Charter on the Preservation of the Digitial Heritage, adopted at the 32nd session of the General Conference of UNESCO*. UNESCO, 17 October 2003.

[44] WEBMEDIABRANDS. What is thin client? `http://www.webopedia.com/TERM/t/thin_client.html`, September 2009.

[45] XFREE86 PROJECT, INC. LBX. `http://www.xfree86.org/current/lbxproxy.1.html`, September 2008.

[46] YANG, S. J., NIEH, J., SELSKY, M., AND TIWARI, N. The Performance of Remote Display Mechanisms for Thin-Client Computing. In *ATEC '02: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2002), USENIX Association, pp. 131–146.

[47] YANK, K. Bomber2. `http://j2mebomber.sourceforge.net/`, September 2008.

[48] ZHUANG, H., AND WANG, Z. IP-based real time video monitoring system with controllable platform. *Mechatronic and Embedded Systems and Applications, Proceedings of the 2nd IEEE/ASME International Conference on* (Aug. 2006), 1–4.