NTNU

Norwegian University of
Science and Technology

A Networked Service for the Remote Execution of Interactive Multimedia Applications

Ola Mogstad

Master of Science in Communication Technology
Submission date: July 2008
Supervisor: Leif Arne Rønningen, ITEM
Co-supervisor: Ole-Ivar Holthe, Geelix/ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Over recent years, the bandwidth and latency properties of public networks have progressed. In effect, it is now realistic to transfer high quality video streams over the Internet. Combined with the increased popularity of lighter and more mobile computers, this encourages the use of computer intensive applications executing remotely.

This assignment will attempt to develop a system that allows end users to play computer games that are remotely executed on high end servers and controlled through media streams. The work will take part under the development of an extension to the Geelix project, which allows computer gamers to share their gaming experiences with each other.

Assignment given: 15. January 2008
Supervisor: Leif Arne Rønningen, ITEM

Abstract

In an era of mobile computing where the latest of electronic consumer devices are cheaper, smaller, and lighter than ever before, the best of computer games require the very opposite. Whereas the newest in mobile phones now handle web browsing, word processing, and music and video playback with ease, the most popular game titles still demand heavy desktop computer configurations with one or more dedicated graphical processing units in order to run well and provide the user with the intended gaming experience. On this premise, a service where video games are played remotely in thin-client fashion seems to have more potential than ever. The objectives of this thesis is to design such a service and to realise it to the extent it is possible.

In this thesis the objectives were approached by first reviewing the theory of thin-client computing and the nature of video games. Based on the work of similar projects it was decided to create the service by streaming video games as live audio and video streams over IP-enabled networks. By comparing it to similar services in use today, such as traditional video streaming and online multiplayer games, an assessment was made to identify its biggest challenges. Latency and media quality were identified as the two key technical challenges to the perceived user experience in that they necessarily degrade the content from its original state. A third user experience factor was found in the content, i.e. what type of game that is played. It was argued that different games might respond differently to the degraded quality of service - a hypothesis that would be put to test following the realisation. Three game classes were clearly defined for this purpose. Latency was identified as the biggest obstacle in providing a high user experience. Consequently, a theoretic review was carried out in an attempt to decompose the overall system into latency-inducing components.

Based on theoretic discussions, system design and implementation was carried out in an effort to realise the core functionality of the service. Two modules were implemented. A server module was created with basis in a video game sharing utility from a related project. A client module was designed and implemented from the theoretic review with various optimisations such as bufferless streaming and a single-threaded design in order to reduce interaction latency. Client applications were created for the three major desktop platforms today, i.e. Windows, Mac OS X, and Linux.

With a functional system, evaluation of the service as a whole was carried out by user testing. In a qualitative approach, 8 test subjects carried out a set of instructions which involved using the system at different settings and with different games. During tests, their session variables such as latency and loss values were logged. After the tests the users were given a questionnaire asking them to evaluate various aspects of the system as well as the general service as a whole. Performance testing was carried out seperate from the user tests and targeted technical aspects of the service and the client implementation, such as network load and processing-induced latency.

User tests revealed that the total impression of the service varied. On average, the implementation was ranked in the upper end of mediocre. By comparing the results to server logs it was confirmed that user experience correlated with measured latency. This was also the case for media quality in that sessions with higher video bitrates scored better. Furthermore, the results from sessions involving different game classes were in harmony with the discussion on content as user experience variable. The three defined game classes performed as expected, confirming that first-person shooters is the ultimate stress test for such a system. Results from questions on the service in general confirmed that video latency was the property that most likely would prevent test subjects from using the service. Results also showed that test subjects were reluctant to paying for such a service, but very inclined to using it as a tool for game demo distribution. In terms of what features of the service that appealed to the subjects the most, all three options - i.e. ease of installation, platform independence, and lightweight client - were found attractive.

The performance tests were consistent with predictions from the theoretic review. The combined tasks of decoding and rendering were measured to be around 75 % of the processing load of the client

application, which as a whole took up 15-20 % of the system on a laptop computer. Input handling took up a mere 7 % of the client load. In terms of delay, decoding and rendering was shown to induce between 4 and 12 ms each, depending on session configuration. The network load analysis showed that the generated traffic varied with content and game situation, which is characteristic to the video codec and not the system, and also revealed an error in the server module that affects the video bitrate.

Preface

This thesis was written by Ola Mogstad during the spring of 2008 and was carried out partly in San Francisco, California and Trondheim, Norway. The thesis was assigned by the Department of Telematics as a mandatory part of the Master's Degree in Networked Services and Multimedia Systems at the Norwegian University of Science and Technology in Trondheim, Norway.

The system development presented in this thesis was primarily carried out in San Francisco in close collaboration with doctorate student and CTO of Gridmedia Technologies, Ole-Ivar Holthe, who was also the driving force of the overall project and a source of guidance and inspiration throughout the project course. Much appreciation also goes to my supervisor, Leif Arne Rønningen, who provided valuable input particularly on the thesis itself.

Finally, I would like to thank all test subjects, without whom the most important discussions in this thesis could not have taken place, and the contributors to open source projects put to use here.

Contents

List of Figures

List of Tables

Abbreviations

API	Application programming interface
AR	Arcade racing
BW2	Black & White 2
CE	Consumer Equipment
CPU	Central processing unit
DLL	Dynamically linked library
ESA	Entertainment Software Association
FC	FarCry
FPS	First-person shooter
GaL	Games@Large
GDB	GNU Debugger
GOP	Group Of Pictures
GPL	GNU General Public License
GPU	Graphical processing unit
GTK+	The GIMP Toolkit
GUI	Graphical user interface
HUD	Heads up display
HVS	Human visual system
IP	Internet Protocol
ITU	International Telecommunications Union
JPEG	Joint Photographic Expert Group
LAN	Local area network
LGPL	Lesser GPL
ME	Motion estimation
MPEG	Moving Picture Experts Group
MTU	Maximum transfer unit
NIC	Network interface card
PPM	Portable Pixmap
RDP	Remote Desktop Protocol
RFB	Remote Framebuffer
RTP	Real-time Transport Protocol
RTS	Real-time strategy
SDL	Simple DirectMedia Layer
STB	Set-top-box
TCP	Transmission Control Protocol
THiNC	Hybrid thin-client protocol
TNF	Trackmania Nations Forever
UML	Unified Modeling Language
UDP	User Datagram Protocol
VGL	VirtualGL
VOD	Video on demand
WMA	Windows Media Audio
WMV	Windows Media Video
WVK	Windows Virtual Key
X11	X Window System
XML	Extended Markup Language

Part I

Introduction

Chapter 1

Introduction

1.1 Background

Today we are in the midst of a technological revolution that has yet to show its full effects on society. High-speed broadband Internet connections are becoming more available to home users in a rapid and ever-increasing manner. Consequently, the Internet has established itself as a popular medium of choice for the delivery of on-demand multimedia content. In parallel with the development of the Internet, we have seen a dramatic increase in the production scale and quality of video games. Since their spawn in the 1980s, home video games have built up an industry with a revenue that long go surpassed the music industry and that recently caught up with the movie industry [5, 6]. The biggest game titles are flagships for technological advancement in the home market and are essentially the driving force for the development of real-time rendered computer graphics.

In the era of mobile computing, the immense sales of modern video games are counter-productive in that they require either full scale desktop computers or equipment tailored for video games only, i.e. video game consoles. In comparison, most mainstream home computer applications, e.g. office software, instant messaging, web browsing, and on-demand video streaming, have relatively light hardware requirements that allow them to be carried out on any public computer, ultramobile laptops, or even mobile phones.

Concurrently, more potential in the Internet is being realised by gradually shifting the distribution model of video games from CDs and DVDs to Internet downloads. Interestingly, this is happening at a slower pace than that of the music industry and to some extent also the movie industry. More so, while primary online distribution of music and movies today is through downloads, alternate business models based on online distribution by real-time streaming are being employed, particularly for movies. All of this provokes a concept where video games are streamed across networks in real-time.

1.2 Project Context

This thesis is written as part of the Geelix research project by Gridmedia Technologies in collaboration with UC Berkeley and the Norwegian University of Science and Technology [7]. The project originally offers "a new social gaming service for sharing high definition gameplay experiences with friends and others" and consists of a free software component that allows users to upload video clips of themselves gaming to a web portal or stream it in real-time to others. The work of this thesis is part of an extension to Geelix named LiveGames.

Overall goals for Geelix LiveGames are listed in the following (from [8] and [9]).

- Users of the service are often relatively young. It is a known fact that children from lower-income backgrounds, such as working class areas of larger cities in the US spend a lot of time playing computer games. There are great differences in the conditions for children growing up in terms of education, the environment and access to nature, clean surroundings and so on.
- The Geelix LiveGames system will exclusively offer high quality 'serious games'. I.e. games developed for the primary purpose of providing good and effective educational content and Christian values (and not merely entertainment).
- The goal is to create a non-profit service.

The goals are of high abstraction and are relatively independent of underlying solution, architecture, and technology. As such, these goals will not be explicitly revisited in the thesis but are however a source of motivation and essentially what makes up the project's context.

1.3 Problem Definition

In short, the problem definition is to conceptualise and realise a service that allows video games to be played remotely over the Internet. A solution for realising the service should then be proposed and reviewed, both from a technical and an end-user perspective. Realisation will be carried out by designing and implementing a system with the goal of promoting the identified key features of the service.

As an approach to the project assignment, research questions are created. By trying to find answers to the questions individually it is possible to focus on certain parts of the assignment at a time.

Question 1: *Is it possible to create a service for the remote playing of computer games?*
Is it technically possible to create a system that provides a quality of service high enough for meaningful use? From an end-user perspective, what are the minimum requirements?

Question 2: *What are the main challenges to creating such a service?*
What should have main focus when developing such a service? Can it be realised on public networks? What are the technological bottlenecks?

Question 3: *Are any games more suitable for such a service than others?*
How are the games affected by remote delivery? Will games with different characteristics be behave differently under these conditions?

Question 4: *What are the possible applications for such a service?*
In what ways can it be used? Does it have consumer use potential?

Implementing a system supporting the desired functionality would in a way answer question 1. However, to declare "the quality of service high enough" it will be necessary to test the system with satisfactory results. The second question will be reviewed in parallel with question 1: First in theory, where acquired knowledge of technical limitations are applied, and then in practice, by experiences gained from implementation and testing. Similarly, question 3 is answered first in theory by creating and defining various game types and then in practice by running tests on different games. Question 4 will be answered by an overall review of the first three answers as well as a user survey.

The questions define what we wish to accomplish with this thesis. Combined, these research goals will function as tools of discussion in the analysis and conclusion.

1.4 Scope

The problem domain covers a wide area with potential topics to be discussed in this thesis. It would be interesting to cover everything, however some restrictions must be made in order to keep the thesis within reasonable size and level of detail - as is highlighted by the defined research questions.

The potential business models of the service is indeed considered a pitfall and beyond the scope of this thesis. The commercialisation of telecom services is a study field on its own and purposely omitted here.

While the service is based on the Internet Protocol, no detailed analysis will be made about the workings of the Internet, e.g. in order to identify bottlenecks and ways to improve it. The Internet will be envisioned as the distribution medium of choice, but this thesis will not attempt to make any conclusions regarding the underlying network beyond deriving certain minimum requirements that the network must fulfill.

It must be stressed that the author's work, particularly the implementation, is part of a larger system and is primarily focused on the client side. That said, for the sake of readability, other parts of the system are also presented in this thesis, but in less detail. Particular focus is on the user perspective and user tests. These tests will necessarily need a full working system.

1.5 Research Methodology

From similar projects involving the creation of software, it is the experience of the author that the project often tends to become "just another software development project" and lose its perspective as research-oriented work and value. As such, it is important to review the research methods that are applied.

The research methodology used here is structured after what is commonly referred to as the *hourglass* notation [10], which on a general level can be applied to any domain of research. Its name and structure can be explained by the project starting off with a very broad area of interest, which is narrowed down, reviewed, and tested. The results are then analysed, conclusions are made and generalised back to the original broad questions. In terms of scope, this can be envisioned as an hourglass. In this project, the hourglass will be satisfied as follows:

1. Broad area of interest: Overall goals of the Geelix project
2. Narrowing down: Defining research questions
3. Review: Theoretical review of service concept, proposed solution, and technical aspects
4. Tests and analysis: Implementation, user testing, and performance testing
5. Analysis: Results from testing are presented and analysed
6. Conclusions: Try to answer research questions with basis in analysis
7. Generalisation: Revisiting the overall goals of the Geelix project with new conclusions

It should be observed that from a research-oriented perspective, the system design and implementation are only necessary tools - or a means to an end - to carry out testing and evaluation. As such, the system development is confined to its own part - as will be shown in the outline. Additionally, it should be noted that this thesis will apply three methodologies: One at the overall research level, as

presented just now, one for the development process, and one for the service evaluation. The two latter methodologies will be explained in their respective segments of the thesis.

1.6 Outline

The thesis is divided into 6 parts, each containing one or more chapters. The final part is followed by the appendix.

Part 1 - Introduction: The thesis introduction.
Chapter 1 Sets up the goals of the thesis and describes its overall context.

Part 2 - Prestudy: Work primarily derived from the state of the art.
Chapter 2 presents technologies that are deemed relevant to this thesis.
Chapter 3 presents other projects that are of similar character to the one carried out here. Much helpful information and many defining ideas from these projects will be applied in this thesis.

Part 3 - Theory: Reviews the theoretical aspects of the service and use of technologies.
Chapter 4 describes the service concepts, reviews possible applications, and proposes a solution.
Chapter 5 analyses the factors affecting end user experience for the proposed solution.
Chapter 6 reviews the technical aspects and the components that are likely to affect end user experience.
Chapter 7 proposes evaluation methods for the service based on the previous chapters. This will form the basis for the testing that takes place in later chapters.

Part 4 - Realisation: Presents the system development process.
Chapter 8 sets up the development context and process methodology.
Chapter 9 presents the requirements specification.
Chapter 10 shows the process of system design.
Chapter 11 completes the implementation, reviews what was implemented with requirements specification, and shows screenshots of the implemented system during execution.

Part 5 - Evaluation: Links what was implemented with theory.
Chapter 12 sets up test plans for user and performance testing and reviews the sources of errors.
Chapter 13 presents test results and analysis.

Part 6 - Conclusion: Revisits the original goals and hypothesis of the thesis with what was discovered.
Chapter 14 attempts to provide answers for the research questions based on results and analysis.
Chapter 15 presents future work.

Part II

Prestudy

Chapter 2

Relevant Technologies

This chapter presents technologies that are considered relevant for the discussions in this thesis and that are not considered to be common knowledge. More emphasis will be on technologies that are - in the context of this thesis - more important, and vice versa. Both hardware and software technologies are presented.

Note: For all following discussions that are based on communication over some sort of network, the underlying network is assumed to be a packet-switched network based on the Internet Protocol (IP), i.e. the Internet or some IP-enabled local area network (LAN).

2.1 Networking

Having confined networking to IP-based networks, networking will be described with the TCP/IP reference model [1] - also known as the Internet protocol suite. This model is slimmer than the more general and arguably bloated Open Systems Interconnected (OSI) model which consists of 7 layers (see Figure 2.1). The TCP/IP reference model consists of 4 layers: The *application* layer, the *transport* layer, the *Internet* layer, and the *host-to-network* layer.

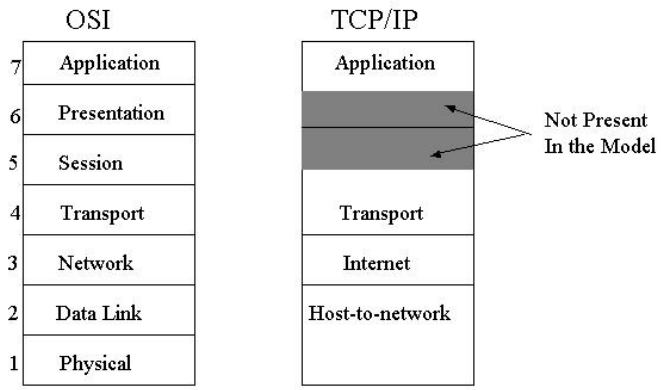

Figure 2.1: The OSI reference model compared to the TCP/IP reference model (excerpt from [1])

The host-to-network layer, also known as the link layer, is not well defined and left out of further discussions here. The Internet layer is defined as the IP protocol combined with the Internet Control Message Protocol (ICMP). The IP protocol is essentially what realises the Internet through packet-switched networking and well-defined addressing (IP addresses) but provides an unreliable and best-effort service [1]. This is approached by the transport layer, which is defined to be either the User Datagram

Protocol (UDP) or the Transmission Control Protocol (TCP), further explained in the following subsections. Finally, the application layer consists of higher level and possibly custom protocols, not pursued further here.

2.1.1 Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is one of the two end-to-end transport protocols in the Internet protocol suite. TCP is an effort to make the IP protocol reliable and connection-oriented in behaviour, allowing a connection to be established were delivery and correct order is guaranteed. It does so through an extended packet header and a retransmission scheme, i.e. where packets that are lost are resent. Correct order here refers to packets arriving in the same order as they are sent.

Due to the asymmetric and variable properties of networks, flow and congestion control is implemented. The flow control mechanism in TCP ensures that a sender will not overwhelm the receiver with data. Without this, the sender would just transmit the data as fast as possible without regard to the capacity of the receiver. Congestion control is a similar feature built into the protocol to ensure that senders do not overload the network. If there is a sign of network overload, a TCP sender will automatically lower its sending rate in order to avoid this condition.

The disadvantages of TCP are overhead due to packet headers (at least 20 bytes in size) and control packets, session initiation and teardown, and delay during retransmission since the current packet stream is paused.

2.1.2 User Datagram Protcol (UDP)

The User Datagram Protocol (UDP) is, next to TCP, the other defined transport protocol in the Internet protocol suite. UDP embraces the IP protocol's unreliable and best-effort properties, leaving any handling of events such as loss and congestion up to the application. As such, unless handling of lost packets is explicitly implemented in the application, lost packets will not be detected and simply remain lost.

The major advantage of the UDP protocol is that it does *not* have the disadvantages of the TCP protocol, unless similar features are implemented at application level. By default, the UDP header consists of 8 bytes, no connection initiation is required, and no retransmission or reordering is carried out. This makes UDP desirable for applications with strong real-time properties or for one-shot signalling where "prompt delivery is more important than accurate delivery" [1].

2.2 Media

On a general basis, the term media refers to a method or tool for communicating or delivering information. As such, media has a content form, e.g. as an image or as text. Here, when referring to different media we will be referring to media with the content form as either video or audio.

Much of the information on compression principles and fundamentals presented here is extracted from Mandal's work "Multimedia Signals and Systems" [11].

2.2.1 Video Compression

Video is essentially a series of still image frames presented at a certain frequency of frames in the time domain. Since a raw image frame of 800 x 600 pixels with 3 base colours of 8 bits each requires 800*600*3*8 = 11.520.000 bits for representation, it should be clear that raw video with 20 such frames per second requires a massive amount of bandwidth (around 220 Mbps). In effect, compression is needed in order to make digital video viable for consumer use.

In general, compression is achieved by stripping a source for redundant information. Redundancy is the duplication of information and is - for the purpose of data storage - undesirable since it is wasting bits on telling us something we already know. This can be applied in ways that make the compression *lossy* or *lossless*. In order for compression to be lossless it must be possible to reconstruct the entire original source by decompression. This way, no information and thus quality is lost but usually with a very limited compression ratio. Much stronger compression can be achieved by allowing the compression to be lossy. This implies that some information and quality will be lost and that the original source cannot be perfectly reconstructed. Lossy compression is what will be pursued further here.

Still image compression is primarily achieved by exploiting *statistical*, *spatial*, and *psychovisual* redundancies [11]. Statistical redundancy comes from the fact that the distribution of pixel values in a frame is not uniform, e.g. an image of the ocean is dominated by the color blue, thus allowing a reduction in size by spending less bits on representing pixel values of low frequency and more bits on high-frequency values. Spatial redundancy is found in that pixel values are not entirely random and independent of the values of their surrounding pixels, i.e. the pixels correlate. This allows one pixel value to be represented more efficiently by an offset to the value of its neighbouring pixels. Psychovisual redundancy is found in that the human visual system (HVS) is not perfect. Furthermore, the HVS is more sensitive to some types of distortion than others, e.g. more sensitive to distortion in smooth areas compared to areas with much variation.

Compressing each frame individually will achieve video compression, and is what is done in the Motion-JPEG (MJPEG) video compression standard. However, more efficient compression standards can be designed by also exploiting the statistical redundancies that are found in video, namely *temporal* and *knowledge* redundancies. Temporal redundancy is similar to that of spatial redundancy, only in the time domain, and refers to an expected correlation among neighbouring frames. Since any two sequential frames usually are very similar, redundancy can be stripped by representing one frame as its difference from the other. This scheme can be taken further by trying to predict the next frame based on previous ones, known as *motion estimation* (ME), increasing compression efficiency even further [11]. ME is often done in a block-based manner, which works on the premise that video contains objects that are made up by a two-dimensional group of pixels that move together. Basing ME only on past frames is known as *forward prediction*. However, imagining a video scene change, it should be apparent that this approach sometimes fails. As such, it is possible to improve the efficiency of ME by also using future frames as reference frames with the past ones, known as *bidirectional* ME. This approach improves compression at the cost of delay since it requires the encoder to hold three frames: The past one, the current one, and the future one. In effect, bidirectional ME may not be applicable for systems with strong real-time needs.

From the MPEG-1 compression standard, we have the following types of frames [11]:

- *I-frames (intra coded):* Commonly known as index frames. Encoded without reference to other frames.
- *P-frames (predictive coded):* Encoded using one-directional future prediction ME with reference to past P- or I-frames.
- *B-frames (bidirectional predictive coded):* Encoded using bidirectional prediction with reference

to past and future P- or I-frames. B-frames are never used as reference frames.

These three types of frames make up what is called a group of pictures (GOP). A GOP always begins with an I-frame and ends right before the next I-frame.

A wide range of compression standards have been developed, each employing a different subset of compression techniques. A video codec is the realisation of such a standard and many different codecs might have been realised from the same compression standard - some following the compression standard more strictly than others and implementing the techniques in different ways. As such, we say that a codec is *based* on a standard and two codecs based on the same standard might perform very differently - often depending on video content and bitrates, etc [12, 13]. Consequently, there is no absolute answer as to what standard or codec to use. For a very specific type of content it can be beneficial to customise a video codec according to that content, as is examplified with Sykora, Burianek, and Zara's work on creating a codec designed specificially for animated cartoons [14]. Today, ITU-T's H.264 (MPEG-4 Part 10) codec is generally accepted as an efficient codec for most applications and employs all the features mentioned in previous discussion in addition to many other features that are beyond the scope of this thesis. Among the many realisations of H.264, x264 is considered to be one of the best ones [12, 13] and is free under the GPL license (see Section 11.2.3).

2.2.2 Audio Compression

Audio compression is similar to video compression in that redundant information has been identified and attempted stripped from the original source. Audio is essentially sound waves and as such much temporal redundancy can be identified in that a wave generally either increases or decreases in value for some time. Consequently, sampled values can be kept lower by only recording their offset from previous values - requiring less bits to store them. Again, prediction can be employed to increase compression efficiency. Similar to the properties of the HVS, the human ear is less sensitive at some frequencies of the audio spectrum than others. Also, a stronger tone at a similar frequency of a weaker tone will dominate, or *mask out*, the weaker tone, allowing the weaker tone to be ignored by the compression scheme [11].

Similar to video compression, several standards for audio compression exists for various purposes. For instance, some standards perform very well for the lower frequency band and samplerate of voice communications (e.g. G.711) while other standards perform better for higher quality music (e.g. MPEG-3).

2.2.3 Streaming Video

Streaming media generally refers to the distribution of media over computer networks and is primarily used for the streaming of video and audio. Here, emphasis is on the streaming of video, however most principles are universal for all types of media. Streaming video has many applications and can generally be categorised in three categories [1]:

- *Video on demand (VOD):* The streaming of a video file, i.e. a file is played as it is downloaded instead of waiting for the file to finish downloading.
- *Broadcast streaming:* The streaming of either prerecorded or live content in television-like fashion.
- *Interactive streaming:* The streaming of video where a recipient in some way interacts with the content.

All video streaming over a network is susceptible to errors and a decision must be made as to how errors will be dealt with. The biggest decision is arguably if lost packets should be resent or not, i.e. whether to base transmission on UDP or TCP (see Section 2.1). Since retransmission implies a gap in playback, UDP is typically preferred as the transport layer protocol. Commonly residing on top of UDP is the Real-time Transport Protocol (RTP), which is a standardised format for delivering both audio and video, commonly used by video clients and servers. Combined with the Real-Time Control Protocol (RTCP), an RTP-enabled stream can be administered in order to adjust to varying network conditions. Additionally, the Real-Time Streaming Protocol (RTSP) can be employed to allow playback control for VOD streams, e.g. VCR-like controls such as play, pause, and fast forward.

Many of the same principles apply for the first two categories and they are as such merged into a *non-interactive* group here. Non-interactive streaming can benefit from a buffering scheme, e.g. of 10-15 seconds, in order to eliminate the effects of network jitter. Interactive streams, e.g. video conferencing, are more sensitive to delay and cannot employ the same buffering schemes as non-interactive streams.

2.3 Industry Trends

The goal of this section is to give a brief overview over current market shares and future trends for the fields of operating systems and video games. The goal is not to give a full technical presentation about the inner workings of either operating systems or video games, and as such a technical review is omitted.

2.3.1 Operating Systems

There are three major platforms for desktop computer use as of June 2008, and they differ significantly in current market shares. The biggest platform is the Windows family by Microsoft, of which the two most recent versions for desktop use are Windows XP and Windows Vista. The second biggest platform is the Mac OS X (X is the roman numeral for the number *ten*) operating system by Apple. The two most recent versions of OS X are 10.4 and 10.5, codenamed "Tiger" and "Leopard", respectively. OS X is the default operating system for Apple's line of desktop computers and laptops, and as such OS X's market share fluctuates with the popularity of those computers. OS X is based on the Berkeley Software Distribution (BSD) UNIX platform and the older NextStep operating system [15]. The third most popular platform for desktop use is Linux, which is also a UNIX derivative. Linux differs from the two others in that its core source code, i.e. the kernel, is free and open. This has spawned a community working for free software under the Free Software Foundation (FSF) [15]. Another side effect of Linux being free and open is that a wide range of versions, or *distros*, have been developed and are under continuous development. The distros are all based on the same kernel but are usually different in terms of what they are targeted for, i.e. some distros focus on stability while other distros focus on the newest of functionality and bundles applications. From a poll in 2007 the three most popular distros for desktop use are Ubuntu (30.3 %), openSUSE (19.6 %), and Debian (11.7 %) [16].

Platform	**Source 1 [17]**	**Source 2 [18]**	**Source 3 [19]**
Windows	90.89 %	91.05 %	94.5 %
Mac OS X	7.94 %	4.62 %	4.0 %
Linux	0.80 %	1.95 %	1.0 %
Others	0.37 %	2.32 %	0.8 %

Table 2.1: Operating system market shares as of April - June 2008

It has been proven difficult to find a reliable way of measuring the market share of operating systems,

likely due to the various licensing agreements in use, e.g. free distribution and vendor bundling. This is confirmed by Table 2.1 where market shares from three different sources are presented. That stated, all sources confirm that Windows is the dominating platform with over 90 % of the market share. OS X is on a distant second with a share of somewhere between 4 % and 8 %, and Linux, with all distributions combined, is on third place with a 0.5 % to 2.0 % market share [17, 18, 19].

Reviewing the current trends it is interesting to observe that the market share for both Mac OS X and Linux has over the last years increased for desktop use. This trend will be further nourished by recent events such as HP, Lenovo, and Dell offering computers optionally bundled with Linux over Windows and ASUS' line of ultramobile laptops, the Eee PC's, being based on Linux.

2.3.2 Video Games

"*On average, nine games were sold every second of every day of 2007.*"
- The Entertainment Software Association (ESA) [20]

The video game industry is in steady growth and has established itself as a leading part of the global entertainment industry, next to the movie industry and ahead of the music industry [5, 6]. Simultaneously, it is one of the fastest growing industries in the U.S. economy with an annual growth rate of 17 % from 2003 to 2006 [20].

Whereas a video game is any game where its primary content is presented as video - including games on all sorts of PCs and consoles - a computer game is a video game for a PC. Of total software sales in the gaming industry, around 10 % of the revenue was due to computer games ($910.7 million of $9.5 billion). In comparison, 69 % of global revenues were from console game sales and 21 % from portable software ($6.6 billion and $2.0 billion, respectively).

In terms of game genres, "Family Entertainment" had the strongest growth with 110 % from 2006 to 2007, accounting to 17.2 % of all games sold in 2007 [20]. Family-friendly rated titles, i.e. up to "Everyone 10+", accounted for 56.6 % of games sold, dominating the "Mature" genre which only accounted for 15 % of sales. Among video games, the best selling genres for 2007 were "Action", "Family Entertainment", "Sport Games", and "Shooter" with market shares of 22.3 %, 17.6 %, 14.1 %, 12.1 %, respectively. In contrast, among computer games the two most popular genres were "Strategy" and "Role Playing", dominating the market with shares of 33.9 % and 18.8 %, respectively [20].

2.3.3 DirectX vs OpenGL

There are two leading application programming interfaces (APIs) for 3D graphics, namely Microsoft's proprietary *DirectX* and the open standard *OpenGL* developed by Silicone Graphics Inc. Both APIs provide functionality for the rendering of 2D and 3D graphics on a hardware graphical processing unit (GPU). Their differences in terms of features and performance is an ever ongoing discussion and meaningless to pursue further here. What is of greater importance is what API the gaming industry prefers and what platforms they are compatible with. In these regards they are very different.

Due to DirectX's proprietary nature, it is supported by Microsoft's platforms only, i.e. the Windows operating system and Microsoft's Xbox series of consoles [21]. In comparison, OpenGL is cross platform, supporting Windows, Linux, Mac OS X, and Sony's PlayStation 3 console.

Today, DirectX is generally accepted as the most popular API for 3D hardware graphics. Of the top 20 computer games listed in ESA's 2007 game industry survey [20], 15 of 20 titles use DirectX. An interesting observation, however, is that the 5 OpenGL titles are all within top 7, including the top 2 spots. In effect, while DirectX has taken a dominating stance in game development, OpenGL is still

in the race. With the highly anticipated OpenGL version 3.0 (currently at version 2.1) being released soon combined with the increased popularity of non-Microsoft operating systems for desktop use, i.e. Mac OS X and Linux, the popularity of OpenGL might increase further.

2.4 Relevant Libraries

2.4.1 FFmpeg

FFmpeg is a library initiated by Fabrice Bellard, but is since developed as an open source project under the LGPL and GPL licenses [22]. It provides functionality for video and audio encoding and decoding, as well as playback functionality. It is written in the C language for the Linux platform, however ports to Mac OS X and Windows are available. Much of its content lies in its sublibraries, of which *libavcodec* is arguably the most important one as it contains a reimplementation of a wide range of codecs.

The FFmpeg library offers an API that is relatively simple to use by developers. As an example, video decoding with FFmpeg in simplified steps is as follows [23]:

1. Create, configure, and open a video codec context (*AVCodecContext*) by finding the codec by its identifier (*avcodec_find_decoder*) and setting its resolution and format
2. Allocate frames (*AVFrame*) for decompressed data
3. Decode the compressed video frames (*avcodec_decode_video*) through a series of iterations
4. Close codecs (*avcodec_close*)

Since the source code of FFmpeg is open, the library can be used in a very flexible manner and provides sufficient functionality both for file playback and streaming playback. A major drawback with FFmpeg is its poor documentation and the fact that it is developed without set releases, so versioning can become a problem. The performance of FFmpeg is difficult to assess as it fully depends on the implementation of the codecs.

FFmpeg is used in many bigger project, of which the most notable ones are *VideoLAN*, *MPlayer*, *mpeg4ip*, *xine*, and *ffdshow* [22].

2.4.2 Simple DirectMedia Layer (SDL)

Simple DirectMedia Layer (SDL) is an open source project developed under the LGPL license. It provides multimedia functionality and is an abstraction layer for video rendering through framebuffers, audio playback, and 3D graphics hardware rendering through OpenGL [24]. The SDL library is cross-platform and supports Windows, Linux, Mac OS X, and even smaller platforms such as SymbianOS. SDL is written in C.

The library API also supports additional functions handy for game development, such as event-based input, timers, and simplified networking and threading. This, combined with its portability, has made SDL a popular framework for the development of games and video playback utilities on Linux. SDL has been used to port Windows-based games to Linux, e.g. for *Doom 3* and *Unreal Tournament 2004* [24].

2.5 DirectX

DirectX is a collection of APIs created by Microsoft in order to support the development of multimedia applications on the Windows platform [21]. The subordinate APIs cover distinct properties of most multimedia application, e.g. audio (*DirectAudio*), 3D graphics (*Direct3D*), and user input (*DirectInput*).

Originally, game developers were required to explicitly support different underlying hardware - either through extra work on each project or costly third party libraries. With DirectX, developers can interact with a single set of APIs that provide access to common features assuming a generic hardware architecture.

DirectX has a layered architecture consisting of two layers above the device drivers (see Figure 2.2). The lower one is the Hardware Abstraction Layer (HAL). It interacts with the device drivers, and offers a common set of features to the API layer. The API layer lies above the HAL and exposes methods for games and other applications to use. For the developer, this means that they can work with a generic model of e.g. a graphics card or a joystick without having to consider the exact make, model and features.

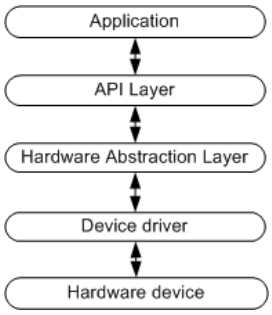

Figure 2.2: DirectX architecture

DirectX is based on the Component Object Model (COM) in which objects are not accessed directly but through interfaces that expose the available functionality. Amongst other things, this ensures that the DirectX libraries are accessible from several different programming languages.

DirectInput is the DirectX API that deals with most input devices. It can handle input from mice, keyboards, joysticks, gamepads and steering wheels. It also handles force-feedback for devices with that specific functionality. In addition to DirectInput, DirectX also includes the *XInput* API for handling input from the Xbox 360 console controller.

2.6 Thin Client Computing

Thin client computing is also referred to as server-based processing and is a type of distributed computing in the client-server domain [15]. This implies a shift in where processing takes place: Instead of having a client only request specific data from the server over a network, as would be the case with a traditional server for database storage and management, the server also contains application logic, leaving the client with the task of presentation. There are several advantages with such an architecture:

- Lightweight clients can present hardware-intensive applications to the user that would normally not run on such light hardware
- Easier administration since there are fewer potential points of failure hardware upgrades will usually apply to the server only

- It functions as a form of software distribution since applications need only to be installed at the server
- A client based on one platform can connect to a server based on another, allowing all of the server's applications to be used in cross-platform fashion

In addition to the major advantages listed above there are other ones, such as an overall lowered energy consumption and increased security. The disadvantage with thin client computing is that content presented to the user must traverse and network, thus consuming network bandwidth as well as introducing a challenge in terms of responsiveness.

A good example of thin client computing is the networked distribution of desktop environments from one computer to another, commonly referred to as *remote desktop*, popular from its integration with the Windows operating system as well as the X windowing system (X11) that is part of Linux. The concept is as follows: First a computer, i.e. the server, must share its desktop. A remote user connects and is presented with the server's desktop environment. Depending on the software used, this will involve the transfer or redirection of both graphics and audio from the remote host to the user. Concurrently, the input of the user, e.g. keyboard and mouse commands, is captured and redirected to the remote host through the network. In effect, the user gets the experience of working locally while he is in reality controlling a remote host [25]. This system property is also desirable for a remote gaming system.

The most notable remote desktop software is Microsoft Terminal Services, based on the proprietary and closed Remote Desktop Protocol (RDP) [26]. While an RDP-enabled terminal server can execute on Windows systems only, RDP client implementations exist for other operating systems, e.g. *rdesktop*, which is open source RDP client for the Linux platform. The Remote Framebuffer (RFB) protocol is an alternative where servers are Linux-based [25].

Chapter 3

Related Work

This chapter presents work and projects that are deemed relevant to the discussions shown later in this thesis. Much of what is presented here is taken from a state of the art written in the early stages of the project. The projects in the following are either in the domain of thin client computing or providing interactive content on demand.

3.1 Gaming On Demand

While the term gaming on demand is somewhat vague, it can be defined as any system where games can be played on demand, without any notion of whether the game executes locally or remotely. Consequently, gaming on demand can be realised by systems where users connect to a server which presents menu of the available games. When a user selects a game, the server uploads the game code to the user computer which then executes the game locally.

An example of a gaming on demand system is the Steam content delivery system by the Valve Corporation [27]. Through Steam, Valve is able to offer their products to users in a graphical user interface. The Steam application acts as an online game store, a manager for downloading and launching games, and as a way to interact with the game community in the fashion of a social network. From an on demand perspective, the challenge of such an approach is the time it takes to download the game code and media. Games consisting of several gigabytes are common, implying hour long waits even on high speed Internet connections.

An alternative to downloading the entire game code prior to launching the game is *progressive downloading*. This approach allows users to launch a game while the content that is not yet needed is being downloaded. The GameOD framework is an example of such an approach [28]. Initially, GameOD only downloads a small portion of the game environment, enabling the user to launch the game as soon as possible. GameOD then applies a prioritising scheme in order to progressively download content that is needed. Game content is managed in two levels: Scene level and model level. Scene level content management determines what objects that are visible to the player and prioritises objects according to how critical they are to the visual experience of the user. Similarly, model level content management does the same with models and textures. Visual data such as texture images can be compressed in order to improve performance. Additionally, GameOD utilises *prefetching*, i.e. downloading objects that are potentially visible and thus likely to need downloading soon. In tests, the GameOD framework showed startup times of less than 5 seconds for a proprietary first person shooter game [28].

Simple gaming on demand, i.e. only through remote storage of games, is essentially just a variation of digital distribution of computer games. By distributing games as digital content over the Internet, production costs are suddenly limited to server hosting whilst the brick and mortar approach must

physically produce something first. As such, games instantly reach global availability since no CDs, DVDs, or cartridges need to be manufactured and distributed to stores. The effects of this are lower prices for consumers, higher margin of profit for producers, and a more open market. It is important to mention these advantages of digital distribution, however, in terms of research and innovation, simple gaming on demand is of little interest. It has already been realised and deployed and it is technologically not very challenging.

3.2 Remote Graphics Rendering

Rendering graphics is the process of generating images from a computer model consisting of components like three-dimensional objects, textures, lighting, and shading. In a computer game, the rendering must be real-time and be fast enough to generate enough images per second in order to satisfy the user's viewing experience. This is an intensive and cumbersome task for the computer, which is why dedicated hardware is often needed in order for the game to execute well. An alternative to equipping computers with the sufficient hardware resources is to distribute the processing task over a network, relieving the client device of heavy graphics rendering. In comparison to simple gaming on demand system offering digital distribution, remote rendering shared many characteristics with that of thin client computing (see Section 2.6). The concept is as follows: Partly or fully render game graphics at server side, i.e. trusting the server with the heavy work. Then, the server transfers rendered graphics to the client, which presents it to the user on a display. Input must be captured client side and transferred to the server in a fashion similar to that of remote desktop software.

3.2.1 G-Cluster

G-Cluster is a system for providing computer games over high speed networks in Internet television (IPTV) fashion. It is primarily designed for set-top-boxes (STBs) as target devices and works by sending prerendered games to users as video and audio streams (MPEG-2 and MPEG-4) [29]. In parallel, the user interacts with the game using a USB gamepad connected to the STB. The input commands, i.e. pushing buttons and moving thumbsticks, are captured by the STB and redirected to the server. This way, the users control games that execute remotely. The user can browse what games to play in a portal-like welcome screen.

G-Cluster is a full scale remote gaming system, as defined in the beginning of this chapter. It has several drawbacks, however. The G-Cluster platform requires games to be *ported* to its platform. Porting games implies modifications to their source code, which requires access to source code as well as proprietary work for each game. Porting a game involves adding audio and video capture, tailoring the game itself to be enjoyable on an STB (here, *repurposing*), and creating game metadata such as age ratings and screenshots.

G-Cluster has been commercially deployed, primarily in hotels, in Japan since October 2004. Very little information is available in terms of performance tests and user reviews. However, looking at the current game selection of over 300 titles, it may appear as if the system is not applicable for the most resource demanding games.

3.2.2 Games@Large

Games@Large (GaL) is funded by the European Union in an attempt to "research, develop and implement a new platform aimed at providing users with a richer variety of entertainment experience in familiar environments" [30]. The GaL system is based on a centralised game server that is responsible

for the storage and execution of games, handled by the Local Storage Server (LSS) and the Local Processing Server (LPS), respectively [2]. The LPS is of most interest as it handles the resource intensive tasks such as input/ouput emulation, execution of games, and generation and transfer of game output (i.e. what is sent to the end-user). GaL investigates two approaches for game output: video streaming and prerendering.

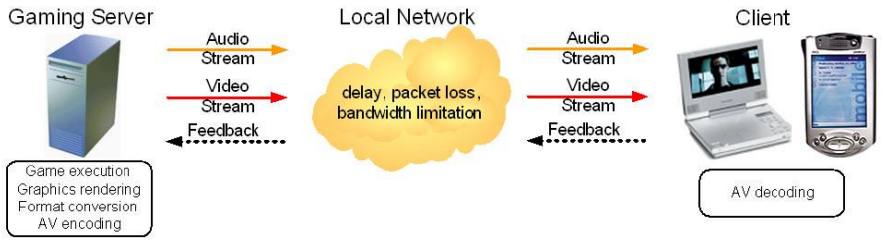

Figure 3.1: GaL network traffic in video streaming mode (excerpt from [2])

In video streaming mode (see Figure 3.1), all rendering is done server side and game output is delivered to the end-user as an image stream (e.g. MPEG-4). This approach allows lightweight devices to act as consumer equipment (CE) at the cost of very high complexity at the server side - handling game execution, graphics capturing, and video encoding. Encoding is particularly resource intensive for high resolution image streams. The high server side complexity will limit the amount of end-users that each server is capable of serving, i.e. amount of games that each server can execute and encode.

Figure 3.2: GaL network traffic in prerendering mode (excerpt from [2])

Alternatively, in prerendering mode (see Figure 3.2), game graphic commands are transferred in place of the image stream. This way, the server is only responsible for game processing while all rendering is done client side. Consequently, each server machine is capable of serving more end-users simultaneously while CE must have computing resources sufficient for graphics rendering. The latter is arguably a major disadvantage in disharmony with the concept of remote gaming. Furthermore, while the traffic load of an image stream is predictable, the traffic load of graphic commands a subject to extreme peaks of bit rate as new textures are to be retrieved and loaded. Tests of a small selection of simple OpenGL games showed maximum bit rates of 26.7 Mbit per frame [31]. A key research area of the GaL project is to find out if the prerendering approach can be implemented in a real system.

It should be observed that the two rendering modes of the GaL project have contrasting advantages and disadvantages. The GaL team implies a system where both modes are applied according to the capabilities of the CE. For instance, the video streaming mode might be the better choice for delivering games to lighter devices such as PDAs, mobile phones, and STBs, while the prerendering mode could be the better choice for delivering games to PCs.

3.2.3 VirtualGL

VirtualGL (VGL) is an open source software package which enables clients to connect to a server and play OpenGL games that are rendered using hardware acceleration [3]. VGL is based on the

Linux windowing system, X11, which in addition to a Windows like environment also provides remote access functionality similar to that of RDP and RFB (see Section 2.6). Through most remote desktop software, OpenGL games cannot be played at all or are forced to run in software mode. X11 allows it, but is essentially just transferring large amounts of raw 3D data from the executing application to the X11 server, likely congesting the network. In theory, VGL can redirect 3D data to any hardware accelerator. However, the accelerator should be located at the server - avoiding performance challenges involved in transferring 3D data on a network.

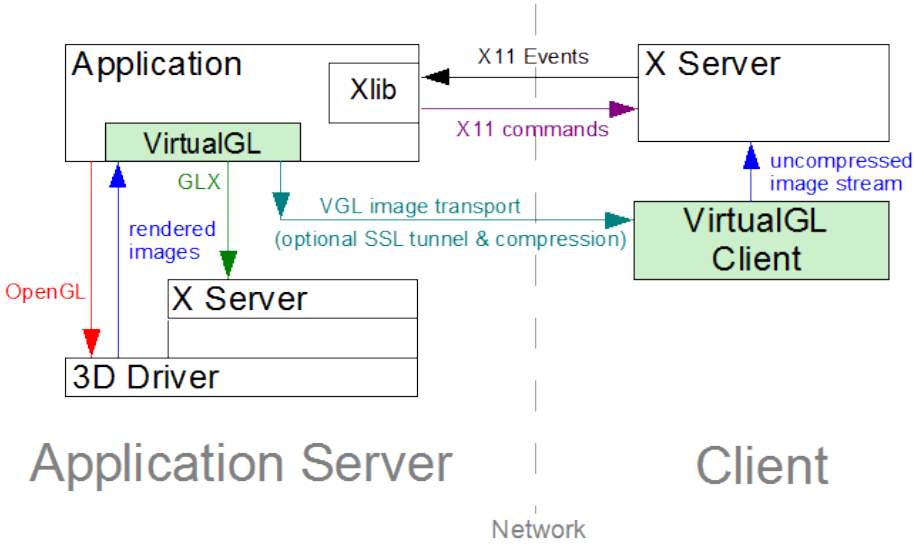

Figure 3.3: VirtualGL Image Transport mode (excerpt from [3])

VirtualGL supports 3 different modes of graphics transfer: VGL Image Transport, X11 Image Transport, and Sun Ray Image Transport. VGL Image Transport, also known as Direct Mode, is of most interest for consumer use. In VGL Image Transport mode, the application server converts rendered graphics to images and transfers them over TCP to the client in a streaming manner (see Figure 3.3). Images can be uncompressed, e.g. the PPM format, or compressed, e.g. the JPEG format. Meanwhile, input is redirected between client and server as X11 events.

VirtualGL only supports OpenGL based games running in Unix. While the client may be running a Windows system, the executing game must be running on the Unix based application server. However, few modern games are created for Unix and most games are rather Linux systems as application servers and OpenGL games.

3.2.4 Hybrid Thin-Client Protocol (THiNC)

The hybrid thin-client protocol (THiNC) developed at Ghent University by De Winter, Simoens, and Deboosere [32] is an attempt to combine full remote desktop functionality with the streaming of video games. It does so by sending graphical commands according to different layers, e.g. a higher layer would be GTK+ components while a lower layer is the X11 protocol. Since such an approach for 3D data with OpenGL and DirectX necessarily requires a GPU on the client to execute well, THiNC instead encodes all GPU-intensive data to video in real-time and streams it to the client. As such, the hybrid part of THiNC is that it combines a classic thin-client protocol with video streaming. In order to achieve this, THiNC inserts an extra driver extension abstraction layer between various graphical layers and the device driver layer. The abstraction layer inspects the complexity of calls to the GPU and decided if an application should be encoded to video or not. The server architecture of THiNC is shown in Figure 3.4.

THiNC is implemented using the x264 H.264/AVC codec over RTP for video streaming. Optimisations

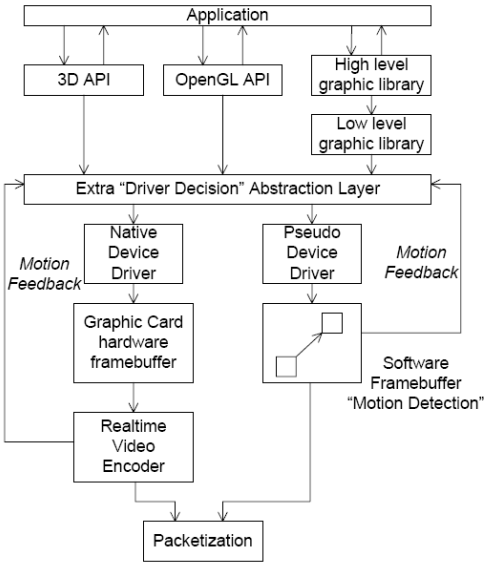

Figure 3.4: The THiNC server architecture

such as single-threaded designs and minimal buffering are made in order to minimise the induced delay. A running implementation was tested for various desktop scenarios and was for the purpose of video streaming a game set to a 640 x 480 resolution with 25 frames per second. The tests were carried out in order to measure the performance of the overall system, particularly for delay. Figure 3.5 shows the measured overall delay for the system at different video settings.

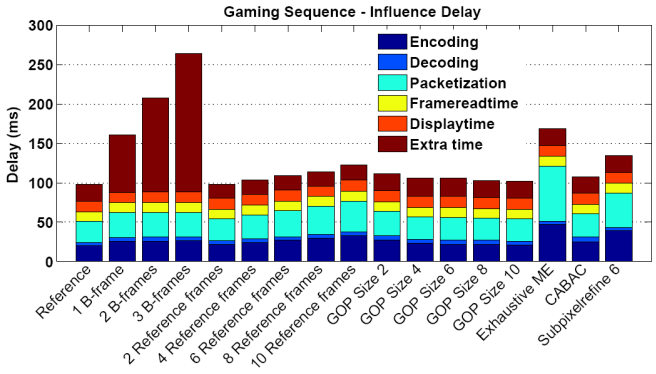

Figure 3.5: Influence on delay for the THiNC system at different video settings

It can be observed that the lowest overall delay measured is around 100 ms, which was deemed too high by the THiNC team since it was stated that the maximum delay should be 80 ms - for no particular reason. Decomposing the overall delay it can be seen that encoding and packetisation induce the most delay. No notions are made about provided video quality at various settings or perceived quality of service for users.

3.3 Summary

The work shown in the remainder of this thesis bases its ideas on several of the projects just presented. On a technical level, GaL and THiNC will be the key contributors. GaL is a well established research project and has developed not only technology but also an evaluation methodology for systems of real-time and interactive nature. This work will be of particular interest leading up to the evaluation of the system developed here. Performance-wise, THiNC is of interest since it shows the results of

extensive performance testing with focus on delay. It will be utilised here both in the technical review and during the system development.

Part III

Theory

Chapter 4

Service Description

4.1 General Proposal

We propose a service that through client and server software modules allow end users, interacting through a software *client*, to play computer games that are in fact executing and rendering on a remote computer, here *server* (see Figure 4.1).

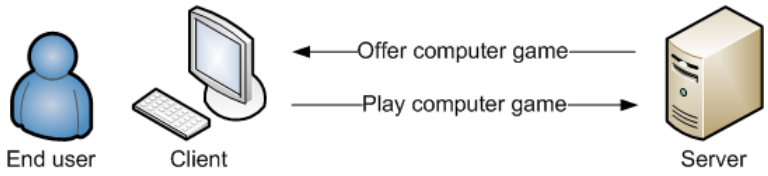

Figure 4.1: General service proposal

The concept of the service is in many ways a variation of the many solutions of thin client computing that are in full use today (see Section 2.6), and as such, the service can be thought of as thin client *gaming*. However, due to the use of that term in a related project (see Section 3.2.4) we avoid using that term here.

In a series of steps, seen from the perspective of the end user, the service will be used as follows:

1. Download and install a lightweight client application to his / her computer
2. Connect to a server over the network
3. Play a game without any further installation

The service is purposely simple from an end user perspective. Consequently, this setup offers many possible advantages over playing computer games that are executing locally:

Distribution: Play any game without installation or large downloads

Ligthtweight: Play high end computer games on lightweight computers. Consumers will never need to worry about frequent hardware updates to play the latest games again

Portability: With cross platform client software, non-portable games can be played on other operating systems

Commercial: Service spawns a whole range of different possible business models

Control: A server provider can remain in control of what to offer users. This potentially enables features such as parental guidance, games tailored to specific hardware, etc

In addition, other advantages might also be found - particularly because of the service's thin client nature. One example would be resource efficiency in that servers could be shared among multiple users, leading to increased utilisation of each hardware component. This again spawns other advantages such as a reduced overall hardware costs for the system and even an environmental gain. More on thin client computing can be found in Section 2.6.

It should be mentioned that the availability of some of the possibilities listed above depend heavily on the system architecture. Some, such as provider control, even make assumptions about the business model of the service. Since the business model of a service such as the one proposed is a study on its own, this thesis is purposely careful about making notions regarding commercialisation. Following discussions will focus on the use of the service, rather than how to make a profit from it.

4.2 Applications

As stated in the previous section, the proposed service has a wide range of possible areas of use. Here, three selected areas of use are envisioned followed by a discussion. Each application is defined with a general description, a list of the advantages in focus, and comments on possible disadvantages and restrictions of the application.

4.2.1 Commercial Distribution

Description: Commercial party sets up a farm of servers and allow registered users to play full games remotely against some sort of payment or through advertisement. For instance, game access could be acquired by the users in subscription fashion - e.g. a monthly fee for access to all sports games - or in the traditional fashion - e.g. a one time fee for full access to a single title, or even a combination of the two.

Advantages in focus: Distribution, Commercial, Control

Disadvantages: Making it profitable due to the high provider side costs. Providing a quality of service that is so strong that users are willing to pay for the service.

4.2.2 Demo Distribution

Description: A computer game distributor sets up a farm of servers and allows users to play demos of new games for free, attempting to persuade users to buy the full version.

Advantages in focus: Distribution, Commercial

Disadvantages: Providing a quality of service so strong that degraded user experience does not affect user's decision on purchasing full game or not.

4.2.3 Released Software

Description: Releasing both client and server software modules to users, either as commercial software or freeware. Users can then set up their own server and connect to it from any client.

Advantages in focus: Lightweight, Portability

Disadvantages: Much potential in the service is ignored.

4.2.4 Discussion

From the previous sections, note that commercial and demo distribution also provide the benefits of portability and lightweight. With that, it is observed that commercial distribution is the only application that take advantage of all the service's features, however the challenge of providing a quality of service worth paying for puts a lot of responsibility in the implementation as well as requirements for the available hardware and network. Whether or not - assuming the quality of service is strong enough - it is possible to profit from such a solution when including hardware and network costs, is a difficult question and is an analysis omitted in this thesis.

Using the remote gaming service to distribute game demos is of particular interest since game demos are really about *trying* a game. Modern games continue to grow in size, as do the game demos. From a quick Internet search, recent game demo download sizes of 1.77 GB, 741 MB, and 490 MB were found for the games Crysis, Unreal Tournament 3, and City Life 2008, respectively [33]. On a 2 Mbps connection downloading at 160 kB/sec, download times range from 185 to 51 minutes. Download times of several hours is long enough to demotivate the average user that is looking to quickly try the game, and is here argued to likely have a noticeable effect on total sales. Thus, the instant play functionality of the remote gaming service would likely be welcomed by both users and producers.

Releasing the service as software simplifies everything as the user is his own provider, in a sense. Possible motivations for users to use the software could be *in-home mobility*, e.g. a high end stationary computer could execute games in one room while users play the games on a lightweight computer in another, or *in-home portability*, e.g. a Mac user sets up a Windows server in order to play games on his Mac.

4.2.5 Core Application

For the sake of discussion, we set up a core application that includes only the core features of the service with no assumptions made in terms of business models or target groups.

Description: Users install client software on and play computer games that are executing on a remote computer.

Advantages in focus: Distribution, Lightweight, Portability

The core application, as described above, is very similar to the service concept - which is essentially what we wanted. The core application will be the application of choice in the following discussions and form the basis for the realisation in Part IV.

4.3 Proposed Solution

As discussed in Section 3.2.2 on the Games@Large project, there were two major solutions for realising the proposed service. One was prerendering, where the game is executing remotely but is rendered locally. This is done by sending graphic commands over the network and as a result distributes the processing load more evenly between clients and servers. The advantage of this approach is that one server likely could serve several clients. However, the idea of prerendering is complex and the balanced distribution of processing load in fact opposes the concept of thin client computing, thus failing to support the service that was envisioned in Section 4.

The alternative is to realise the service through video and audio streams. This, in contrast to prerendering, puts a heavy processing load on the server since it must execute and render a game in parallel with capturing and encoding video and audio. A much smaller load is put on the client now that it does not need to render graphical commands but video instead - in harmony with the concept of thin client computing. Based on this and the service proposal, a realisation through media streams is determined to be the best viable option.

In steps, the solution is as follows:

1. Game executes and renders on the server
2. Rendered game is captured and encoded as video and audio on the server
3. Video and audio is streamed over a network to a client
4. Client software receives, decodes, and renders the video

Concurrently, user control and interaction with the game is realised as follows:

1. Client software, in parallell with processing incoming video stream, captures input events from the user
2. Input events are sent over the network as they occur
3. Server receives input events and injects them into executing game as they arrive

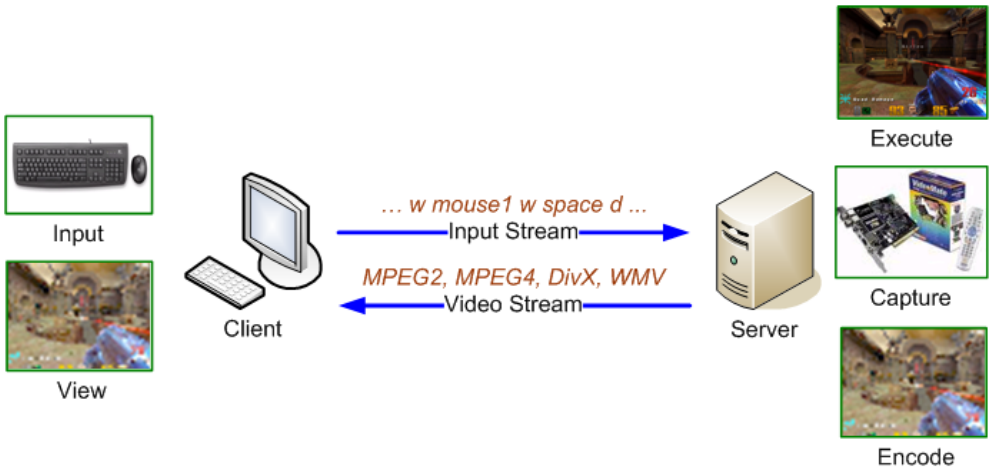

Figure 4.2: Proposed solution for realising the service

The proposed solution is illustrated in Figure 4.2. This is a general overview and in no way a complete system design. A more detailed system will gradually be derived in the chapters to come.

4.4 Summary

We have conceptualised a service and discussed its possible applications, including one core application that will be proposed for realisation. The major advantages and disadvantages of the proposed solution have been outlined. Looking at the illustration in Figure 4.2 it should again be apparent that the service will require a lot from its servers. Any frame - according to the framerate of the video - will in sequential fashion need to be rendered, captured, encoded, and sent on the network. Concurrently, the client will be receiving and decoding video.

The various resource-intensive tasks put two possible constraints on the system: It will limit the number of clients that can be served simultaneously by a server and it will induce delay from a frame is rendered on the server to it is rendered on the client. The former is primarily bad due to the costs of the system, which could affect the potential applications of the service. The latter is bad due to the strong interactive content that is processed, i.e. a computer game. Furthermore, while non-interactive content take advantage of various buffering schemes to be streamed smoothly, streaming strongly interactive content does not have that luxury. Buffering must be kept at a minimum to ensure low delay and high user satisfaction.

Finally, as with all systems based on streaming media, bandwidth will put constraints on the video and audio quality that can be supported - which directly affects the user satisfaction.

The limitations and challenges of realising the proposed service will be the essence of the next two chapters.

Chapter 5

User Experience Factors

It should be apparent that with the proposed service, the quality of the service as seen from the end users will be evaluated relative to the experience of playing a traditional game that is executing locally. In that sense, the service will at best effort - to which degree is depending on the implementation and technology available - try to mimic the local gaming experience. As such, three key factors have been identified as obstacles that will be the primary contributors to degrading the quality of service, namely *latency*, *media quality*, and *content*.

5.1 Latency

Latency was immediately recognised as a key challenge to the remote gaming service. Additionally, it was discovered that the proposed service would have to cope with a different type of latency than that of traditional online multiplayer gaming. In this section, two types of latencies are discussed, different in their origin and their effect seen from the perspective of the end user.

Note that for the purpose of detailed analysis it is an important fact that real latency, i.e. what the user experiences, necessarily will differ from pure network latency in ping-pong fashion. Consequently, overall latency should be decomposed into multiple latency enabling components, e.g. network propagation, processing times, and data transfer. This decomposition is of a more technical nature and is reserved for the technical review (see Chapter 6). For a more detailed review of the sources of latency, see Section 6.2.

5.1.1 Game Latency

In online multiplayer games, network latency creates a delay between the time of which a player performs an action on his local computer and the time of which that action is registered in the game server and distributed to other players[1]. Consequently, if a player performs an action such as a jump, there would be a varying delay before other players see the player jumping. This type of gameplay is not user friendly and much time has been spent by game developers to find ways to improve online play, generally referred to as *latency compensation techniques*.

While latency compensation techniques for online multiplayer games is a broad topic and beyond the scope of this assignment, it is worth mentioning what is arguably the most important technique in

[1]This is stated assuming that the multiplayer game has a client-server architecture, which is usually the case among popular games today. For a presentation of the client-server game architecture in comparison to other alternative architectures, see [34].

online games - namely *client side prediction*. Most online games employ a client-server architecture that revolves around managing all state and game logic within the server, treating clients as dumb [35]. Consequently, all player actions are sent to the game server which then updates its game state and sends state changes back to all the players. This might seem necessary but it also creates one major issue for online play: A player's own actions must also go through the game server before being sent back. The player will perceive this as delay between his input (e.g. key presses and mouse movement) to the game and what is presented on his screen.

Figure 5.1: Two screenshots demonstrating client side prediction in Quake 3 Arena

Figure 5.1 shows two screenshots illustrating the client side prediction in Quake 3 Arena[2]. The screenshots are taken at the during a two-player session where player 1 (left) had a latency of close to zero while player 2 (right) had a latency of around 400 ms[3]. The two screenshots are taken at the very same time while player 2 is jumping. It can be observed that while the action of jumping happens instantly at player 2's computer - even though he is playing with a 400 ms latency -while player 1 can not yet see it.

Game latency is generally accepted as the biggest technological challenge for a high quality user gaming experience [36]. It has been shown that both user experience and user performance detoriates as game latency increases, however the exact threshold between good and bad gaming experiences varies between different game categories. Research shows that for FPS games, such as Unreal Tournament 2004 and Quake 3 Arena, a game latency greater than 50-75 ms is noticeable while the upper threshold of what is acceptable lies somewhere between 100 and 200 ms [36, 37, 38]. Meanwhile, users of real time strategy games, such as Warcraft III, have been shown to tolerate a game latency of more than a second [39]. The reason for such different results likely lies in a combination of different latency compensation techniques in the game software and the different behaviour of the games - i.e. Unreal Tournament 2004 involves fast movement and high precision shooting and is of a more reaction oriented nature than that of Warcraft III.

It is worth mentioning that while online multiplayer games have strict requirements for network latency - and therefore also network jitter - the requirements for network bandwidth are more relaxed. The network traffic generated from client-server multiplayer games is typically less than 50 kbps during gameplay [40, 34].

5.1.2 Interaction Latency

First, in defining interaction latency we first define *video latency*. Having described game latency in the previous section, it might seem like the video latency of a remote game system is the very same thing. However, as will be shown, it is not.

[2]The screenshots are actually from Rocket Arena 3, a modification to the Quake 3 Arena game.

[3]For the purpose of demonstration, latency was actually simulated between the server and the player 2 computer by Shunra, a software tool for simulating network latency and loss.

We define video latency as the time it takes from an original live source is captured at the sender to it is rendered at the receiver. Due to factors such as network transmission, processing, and buffering at both the sender and the receiver, some delay will necessarily be induced. This type of delay will, as previously described in Section 2.2.3, exist in any system that utilises streaming media.

Fortunately for most applications of streaming media, since the viewers have no grasp of what state the original source is currently in, the video delay is not detected. In live televised events, for instance, video latency of several minutes is sometimes purposely added in case something unexpected happens, requiring content to be censored [41]. However, unless the viewers are present at the actual event, no delay is noticed.

Playing a modern game over a video stream differs from live television in that it is highly interactive and when something is interactive it needs to respond to user interaction. When a user interacts with something, he/she expects something to happen, i.e. a response. When a person kicks a ball, the ball is expected to move. As such, because television content is not interactive, viewers do not perceive it as delayed. The only interactive part of traditional television is switching channels, which is unaffected by the added video latency.

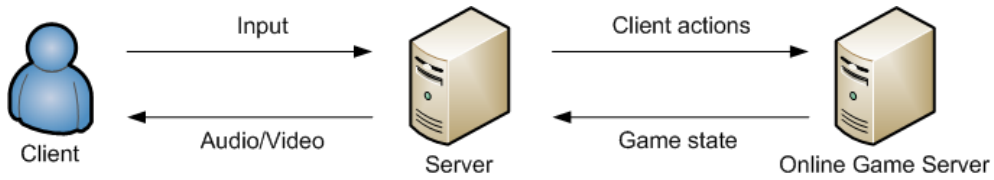

Figure 5.2: An illustration showing a combination of interaction latency (left half) and game latency (right half)

We define interaction latency as the time it takes from a user's input is given to the response is seen. In effect, interaction delay consumes the delay that affects traffic both ways, i.e. video delay and *input delay* - the time it takes from a user causes input to it is registered at the server.

Because of the thin client nature of playing a game through a video stream, client side prediction cannot be employed as it is with game latency. As an example, imagine playing Quake 3 Arena, a first person shooter type game that employs client side prediction, with 150 ms latency. As was already explained in the previous section: Because a player's *own* actions - e.g. firing a weapon or looking to the right - are simulated locally without the confirmation of the server, they appear to be happen instantly. However, reviewing the same scenario without client side prediction, the client awaits confirmation from the server before carrying out any action, i.e. a 150 ms delay from he presses a button until the game responds. It is observed that if Quake 3 Arena is offered through the proposed service, the user will get an experience that is similar to playing the game online without client side prediction.

In short, interaction latency differs from game latency in that it adds a *local* delay between user input, e.g. keyboard and mouse, and the game that is presented to the player. End users will perceive this as they would perceive game latency *without* client side prediction - which, as explained, was one of the major challenges with online play some years ago. In effect, the requirements for video latency are arguably even stricter than that of game latency, which were attempted quantified in the previous section.

5.2 Media Quality

The video presented to the user will have been compressed in size in order to make it feasible to stream it, and is necessarily of degraded quality. The information loss in the video stream may or may not be detected subjectively by users depending primarily on the compression ratio and the video

codec. Figure 5.3 shows a video stream of the game FarCry compressed with the WMV8 codec at three different video bitrates, i.e. compression ratios. The screenshots were taken from tests with video bitrates at 512 Kbps, 1 Mbps, and 3 Mbps, with 20 frames per second and an 800 times 600 resolution. The player was moving around as the screenshots were taken in order to provoke blocking errors from motion prediction. The original source had a framesize of around 11 Mb[4] which, at 20 frames per second, implies a video bitrate of 220 Mbps. In other words, from left, Figure 5.3 shows compression ratios of around 430:1, 220:1, and 73:1, respectively.

Figure 5.3: The effects of compressing a video stream of FarCry with WMV8 and three different video bitrates: 512 Kbps (left), 1 Mbps (middle), and 3 Mbps (right)

It is obvious from Figure 5.3 that a higher bitrate gives a better visual performance to the user[5], but it should be noted that for modern video codecs and higher bitrates, e.g. higher than 10 Mbps for an 800 x 600 resolution, perceived video quality is close to perfect and little is gained from increasing the bitrate. It can be observed that FarCry running at 3 Mbps is close to indistinguishable from the original game.

Another interesting observation is that quality loss appears uniform across the entire frame. This is not suitable for game content since certain areas of the frame will be of much more important value than other areas, in terms of level of detail. For FarCry and other first-person shooters the critical part of the frame is located in the middle, around the crosshair which during action is (preferably) where the enemy targets are. Imagining long-range combat across the bridge seen in Figure 5.3, it can be argued that the leftmost screenshot would not even be able to detect an enemy player.

5.3 Content

A question that should be asked when designing a service distributing content that is necessarily modified and to some extent of degraded quality, is if the service will work better on some content and worse on other content. In this section, we ask that very question with basis in the work of Claypool et. al. on the categorisation of latency effects in Unreal Tournament 2003 (UT2003) [38].

The content of the service is computer games (see Chapter 4) and since one computer game might behave very differently from another and thus respond to introduced latency and degraded media quality differently, the identity of the computer game should be considered a user experience factor. This section presents various types of games and some expectations for how games might affect the end user experience differently.

[4] 800 x 600 pixels with 3 colours each represented by 8 bits.

[5] This is stated assuming that this thesis is read from high quality printing and it is possible for the reader to distinguish between the screenshots.

5.3.1 Game Characteristics

All games are different, however some are more different than others. For instance, some games are based on the same or similar graphical engine and appear to be small variations of each other only with different textures and storyline. Games like these share many of the same characteristics and are, from a technical perspective, more or less the same game. In contrast, some games are completely different - and the interesting part when categorising content is *how*. Conclusively, we seek to determine the game characteristics that carry the primary influence on the end user experience of the proposed service.

Reviewing the two other user experience factors, i.e. latency and media quality, a game characteristic is key if it is related to how a game responds or looks, respectively. In effect, the following game characteristics are used in the remaining discussion.

Presentation

Definition: How the game is presented to the user through audio and video media. Primary variations are if the game is two- or three-dimensional and the perspective offered to the player.

Effect: It is difficult to say anything about how presentation and latency combine. In terms of degraded media quality due to compression, two-dimensional games share some visual characteristics with cartoon animations, e.g. static backgrounds with smaller objects moving around. With basis the principles of video compression regarding temporal redundancy (see Section 2.2.1), it is likely that presentation will affect the performance of compression enabled by temporal redundancy. Generally, it is expected that two-dimensional games offer better visual quality than three-dimensional games.

Detail Level

Definition: The visual level of detail the player needs to be able to enjoy the game experience.

Effect: Efficient video compression is lossy and as compression ratio increases, greater details are lost. In general, a game with high detail level will require higher video bitrates than a game with low detail levels.

User Interaction

Definition: What user interaction that is required by the game. User interaction is primarily decided by required responsiveness and type of input devices.

Effect: It should be apparent that a game requiring high responsiveness will suffer more from latency than a game with little responsiveness, as responsiveness is generally a challenge for the player to react quickly and latency prevents just that. Games that utilise mouse control will be more prone to be affected by network distortions, resulting in jitter and loss, than games that only utilise discrete controls, i.e. button presses.

5.3.2 Game Classes

The computer game industry traditionally categorises games into game genres, as the entertainment industry does with other distributable media, e.g. movies and music. As an example, Amazon.com divide computer games into the following genres [42]: Action, Adventure, Arcade, Cards & Casino,

Classic, Online, Puzzle, Racing & Flying, Role-Playing, Simulation, Sports & Outdoors, Strategy. While some genres appear to be more precise, e.g. Cards & Casino and Racing & Flying, some are more vague and likely overlapping, e.g. Action and Online. It is the opinion of the author that genres such as the ones mentioned say more about what player setting and game environment that can be expected - i.e. *subjective* content - and less about how the game actually behaves and furthermore, how well it plays with the proposed service. In effect, three game classes are selected and clearly defined according to the characteristics that were defined in the previous section.

First-Person Shooter (FPS)

Description: The FPS class games typically force the player to move around in some environment while trying to stay alive by shooting enemies. They key task here is, upon seeing an enemy, to quickly point a crosshair on the enemy and firing.

Presentation: 3D from first person perspective.

User interaction: Repeated movement with keyboard and continuous aiming with mouse.

Detail level: Varies from high (distant targets) to low (close targets).

Example titles: Quake 3 Arena, FarCry, Unreal Tournament 2003

Real-Time Strategy (RTS)

Description: The RTS class games typically put the player in a war battle like situation and in charge of units that combined must beat the enemy force. In contrast to step based strategy games, where battles are carried out with chess like movement, RTS games requires much more frequent user interaction and responsiveness. They key task here is to first click a unit and then to click where the unit should move.

Presentation: 2D/3D from bird eye perspective.

User interaction: Continuous unit selection with mouse.

Detail level: Medium and constant - being able to see the units.

Example titles: Starcraft, Warcraft III, Black & White 2

Arcade Racing (AR)

Description: The AR class games put the player in a vehicle on a track that must be completed as fast as possible. The key tasks are accelerating and turning the vehicle. In contrast to racing simulators, AR games behave in an unrealistic manner.

Presentation: 3D from first/third person perspective.

User interaction: Continuous acceleration and turning with keyboard.

Detail level: Low and constant - being able to see the road ahead.

Example titles: Trackmania Nations, Need For Speed, Out Run

The three classes defined above were chosen because of 2 reasons: First, they contain some of the most

popular game titles on the market, which makes it easy to find examples of the three as well as increasing the understanding of the average read that might have heard of some of the titles. And second, the game classes are different in all of the three game characteristics that were defined in the previous section. It should also be noted that the definition of the FPS class is based on the discussions in [38], which had particular emphasis the effects of game latency on various game situations, e.g. various types of movement and precision shooting.

Figure 5.4: Screenshot of three games representing three different game classes. Left: First-Person Shooter (FarCry), middle: Real-Time Strategy (Starcraft), right: Arcade Racing (Trackmania Nations Forever)

These three classes will be used throughout the remainder of this thesis. Particularly during the stages of testing and analysis, in Chapter 12 and 13, respectively.

5.4 Summary

Comparing the three game classes (see Section 5.3.2) with the analysis of the three game characteristics (see Section 5.3.1), it is possible to estimate what game class is better suited for remote play.

Based on the discussions on latency and media quality in this chapter, supported by notions on streaming media and the hybrid thin-client protocol - found in sections 2.2.3 and 3.2.4, respectively - it is the author's belief that latency will be the biggest challenge for the proposed remote gaming service. The hybrid thin-client protocol project, which had strong emphasis on codec delay optimisation, found a *minimum* delay of 100 ms on a local 100 Mbit network [32]. In comparison, media quality has no lower limit and is essentially a question of audio and video codec compression performance and available network bandwidth. As was shown in Section 2.2.1, media streaming is possible with quality that is close to indistinguishable from the original source with bitrates that are realistic with today's available consumer network bandwidth connections.

Consequently, user interaction is the most important game characteristic. With that, AR games will likely perform better with the proposed service. Between the two remaining classes, FPS and RTS, based on their presentation and detail level characteristics, RTS games will arguably outperform FPS games.

This holds when evaluating the *element of surprise* in each of the game classes. In AR games, the player will see the road ahead of him and to a higher extent be able to predict the actions to take several seconds ahead. Similarly, RTS games are designed with focus on strategy rather than reaction and precision, allowing the player sees his units from a bird eye perspective - attenuating the element of surprise. In contrast, in FPS games the design focus is *action*, of which unexpected and sudden events is a key tool. The player will typically walk around in some environment and repeatedly be challenged by enemies popping up from behind corners.

Conclusively, FPS games are the ultimate test of the proposed service and is expected to provide a

worse user experience than AR and RTS games. RTS is expected to give the best user experience of the three.

Chapter 6

Technical Review

In this chapter we approach the technical aspects of the service concept and the proposed solution in more detail. Particular emphasis is on reviewing critical challenges for the system and finding possible solutions, which can later be to our advantage in the realisation.

6.1 Overview

In order to form an overview of the technical aspects of the system, Figure 6.1 presents the various stages of processing that the system necessarily embraces with basis in the proposed solution, heavily influenced by a similar overview from [32]. In the illustration it can be observed that both the server and the client carry out a series of steps in order to realise the service.

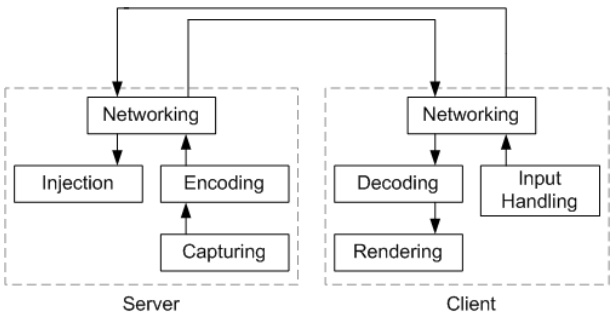

Figure 6.1: System overview

The networking is assumed to take place over a packet-switched network such as the Internet or a local IP network (LAN). Note that the vertical arrows indicate two different flows across the network with video and audio data going one direction and input data the other. The interaction latency (T_{total}) of the system is the result of the delay of the two directions (T_{input} and T_{video}) added together:

$T_{total} = T_{input} + T_{video}$

Furthermore, each of the directions (T_x) are the results of some of the same set of components - network latency ($Latency_x$), transmission time ($Trans_x$), and potentially time wasted on timing and scheduling the main process ($Sched_x$). Note that transmission time is here defined as data load divided by network throughput, i.e. the time it takes to get data on the network. Additionally, on the server we have input injection ($Inject_s$), capturing ($Capture_s$), and encoding ($Encode_s$), and on the client side we have interaction detection ($Interaction_c$), rendering ($Render_c$), video buffering ($Buffer_c$), and decoding ($Decode_c$). As such, reviewing the interaction latency on an average time basis, we have

the following:

$T_{input} = Latency_{input} + Trans_{input} + Sched_{input} + Inject_s + Interaction_c$
$T_{video} = Latency_{video} + Trans_{video} + Sched_{video} + Capture_s + Encode_s + Render_c + Decode_c$

These formulas imply that many subcomponents of the system need to be tested individually for how much time they need on each main process iteration. Consequently, it is desirable to make certain assumptions that help us simplify things:

1. Assuming identical paths, it is usually fair to assume that network latency will be identical or similar for the two directions
   $Latency_{input} = Latency_{video} = Latency$
2. It is a goal to minimise time spent on scheduling and buffering, hopefully close to being zero. However, assuming a fixed video framerate, the time gap between two video frames sent from the server will induce delay if input events are received right after a video frame is transmitted.
   $Sched_{input} = 0$
   $Min(Sched_{video}) = 0$ and $Max(Sched_{video}) = 1/Framerate$
   On average: $Sched_{video} = 1/(2 * Framerate)$
3. Assuming small input packets (e.g. less than 50 bytes) and reasonably high bandwidth (e.g. 512 Kbps), transmission time for input traffic should be negligible (here: 0.78 ms)
   $Trans_{input} = 0$
4. From Norberg's work, we have that injection time was negligible [8]
   $Inject_s = 0$

With the above simplifications, we now have the following formula for the interaction latency (i.e. both directions combined):

$T_{total} = 2 * Latency + 1/(2 * Framerate) + Trans_{video} + Capture_s + Encode_s + Render_c + Decode_c + Interaction_c$

As such, we have now identified the major contributors to interaction latency in a remote gaming system. It should be observed that latency is induced partly at all three parts that make up the system, i.e. the network, the client, and the server. In the following, we investigate the latency contributions as parts of one of the three with focus on what delay we might expect and possibly how we can affect the induced delay.

6.2 Network

If a computer network is seen as a line of trucks carrying data to some destination, the network's bandwidth is how much each truck can carry while the propagation time is how fast the trucks drive. While this in many ways is a gross simplification, it highlights the difference between network capacity and network latency - both of which will affect the overall interaction latency here.

6.2.1 Network Latency

Network latency will necessarily exist in any physical communication system. The ones and zeros in computer networking are, down at the physical level, translated to electronic impulses. These impulses travel through some medium, e.g. air or fiber, at a very - but not infinitely - high speed. Assuming a

fiber medium the impulses travel at a speed typically around 0.6 times the *speed of light* which is at a mere 300.000 kilometers/s. This will appear to be extremely fast, to the point of negligible, however networks often cover great physical distances. For instance, the aerial distance between Trondheim and San Francisco is around 8.000 kilometers. With a speed of 0.6 times the speed of light, round-trip network latency based purely on physical propagation is around 89 ms.

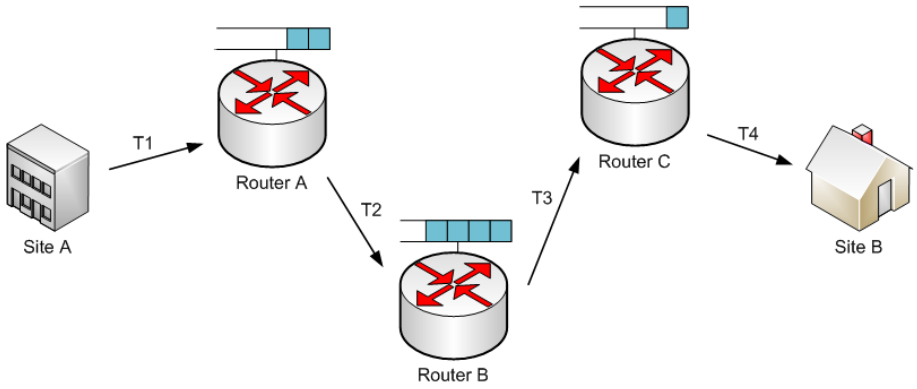

Figure 6.2: Illustration of example network architecture with router packet queues

Furthermore, a theoretical network latency based on the physical distance only holds if there is a straight single-link connection between sender and receiver, which is never the case for a public packet-switched network such as the Internet. As such, it can be expected that the network will consist of multiple segments or links connected by routers - as illustrated in Figure 6.2. A router interconnects two or more links and temporarily buffers received packets in queues before forwarding them on the correct link according to its routing table. In effect, a packet will be buffered and processed in each router on its way to the destination, further increasing network latency beyond the physical propagation delay. From the example architecture in Figure 6.2, network latency from site A to site B will be:

$$Latency_{A-B} = Prop_1 + Prop_2 + Prop_3 + Prop_4 + Queue_X + Queue_Y + Queue_Z$$

Where $Prop_x$ denotes propagation time for link x and $Queue_x$ denotes average queuing time for a packet in router x. A quick test using the Windows `ping` and `tracert` tools revealed a packet from Trondheim to San Francisco went through Amsterdam, New York City, and Washington DC with a latency of around 160 ms. For many network applications, e.g. voice communications and online multiplayer games, a 160 ms delay will be noticeable and of annoyance. That said, bits can never travel faster than the speed of light, and 160 ms is less than twice the theoretical minimum for fiber communications, which is impressive. Few technologies can claim to have a performance within a factor of 2 of its theoretical best.

While the routing of public networks is often unpredictable, physical site-to-site distance is usually a major indicator for network latency. As such, network propagation will generally not be a problem for systems that are distributed locally or relatively locally. Within the area of a city, for instance, propagation times are low enough to be ignored[1].

6.2.2 Transmission Time

For traditional video streaming it is commonly preferred to set the bitrate close to as high as the real capacity of the channel. Real capacity here refers to the maximum bitrate that the channel can carry. For minimising delay, however, the channel capacity to video bitrate ratio as well as the framerate will

[1]There is really no guarantee that two sites within the same city will have a low propagation time, since routing in public networks is unpredictable. However, assuming a reasonable routing scheme, the statement will hold. The politics surrounding the routing schemes of ISPs and other governing bodies are well beyond the scope of this thesis.

affect the transmission delay for each frame, as illustrated in Figure 6.3.

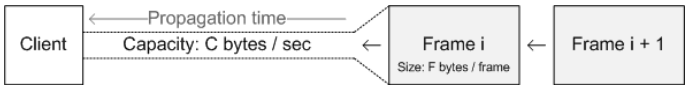

Figure 6.3: Illustration of frame transmission on a channel with specific capacity

$Trans_{video} = F/C$
$F = videobitrate/framerate$

The channel capacity (C) is determined by the available network and cannot be affected by the system. The size of each video frame (F) is - on average - the video bitrate sent by the server divided by the framerate of the video, i.e. the number of frames sent each second. So for instance, if a server is sending a 1 Mbps video stream with 20 frames per second, each frame would - if we assume the all frames are of the same size - have a size of 50 Kb. If this is sent over a 2 Mbps channel, each frame would have a 25 ms transmission delay. In comparison, if the video bitrate was reduced to 512 Kbps, the transmission delay would correspond to 12.5 ms. In reality, frame sizes for most compression schemes (see Section 2.2.1) are not constant, and transmission delay will thus vary between I-frames and P-frames, for instance. Independent of that, the point highlighted here is that while a higher video bitrate likely will improve image quality, it does so at a cost of delay - even if the channel is capable of supporting the chosen bitrate.

While we do not have the freedom to set the channel capacity high enough and the video bitrate low enough for the frame transmission delay to be negligible, it is important to be aware of these dependencies between image quality and delay. It is stressed that for remote gaming, the same principles as for choosing a bitrate for traditional video streaming do not apply.

6.2.3 Traffic Load

The network traffic load is essentially how much traffic that is generated from the two real components of the system - client and server. Calculating the generated traffic (B_{total}) based on previous discussions should be very straightforward:

$B_{total} = B_{video} + B_{audio} + B_{input}$

Total bandwidth use is thus the sum of the video bitrate (B_{video}), audio bitrate (B_{audio}), and input traffic (B_{input}). The bitrates of video and audio depends on the settings used but will - from the discussion in Section 5.2 - typically range from 512 Kbps to several Mbps and 16 Kbps to 128 Kbps, respectively.

$B_{input} = packetsize * frequency$

Generated input traffic will depend on two implementation specific parameters, packet size and frequency. The frequency will, due to the nature of input devices, be relatively high. For instance, a USB mouse has a frequency of 130 Hz, which for an optimum implementation - i.e. most similar to local execution - implies 130 packets being sent each second. The packet size depends a lot on the specifics of the protocol and on-event to periodic updates - which will be pursued further in Section 6.3.3 - and can thus vary from as little as 3-4 bytes to 100 bytes. Conclusively, traffic generated from input handling can vary from 0 to over 100 Kbps.

It should be observed from Table 6.1 that even with the most extreme of settings, video traffic will dominate the total traffic generated. For most input handling schemes, video will be followed by audio as the most significant contributor. An example could be set up with a 1 Mbps video bitrate, 64 Kbps audio bitrate, and input traffic averaging at 2 Kbps (40 events per second on average, 50 bytes per

Content	Traffic	Component
Video	512 Kbps - 10 Mbps	Server
Audio	16 - 128 Kbps	Server
Input	0 - 100 Kbps	Client

Table 6.1: Decomposition of the system traffic generation

packet). This would sum up to a total traffic generation of 1090 Kbps, of which 93.9 % is video, 5.9 % is audio, and 0.2 % is input.

6.3 Client

6.3.1 Bufferless Streaming

In video streaming, in order to keep latency at a minimum it is necessary to limit the video buffer, and - to the extreme - remove buffering entirely. Bufferless streaming has two very distinct advantages in that it a) removes all buffer-related delay, and b) can be very easy to implement from scratch. Figure 6.4 illustrates the difference between video streaming with and without a buffer. It should be observed that buffering is done on the client, thus making the decision to remove the buffer entire up to the client.

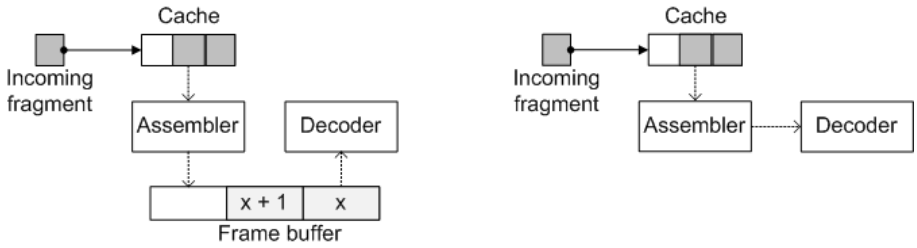

Figure 6.4: Buffered streaming (left) vs bufferless streaming (right)

While the streaming shown is attempted made bufferless, the fact that video frames likely will need fragmentation requires the bufferless scheme to in fact have a one frame buffer - here referred to as the *cache*. The cache will hold frame fragments until all the fragments have been completed and assemble them before decoding the frame in its entirety. If something stalls and the frame is not forwarded for decoding in time for new fragments to arrive, the cache is overwritten with new data and the old is dropped. It can be observed that this approach is similar to that of a one slot circular buffer. However, since fragments are being received while the frame is waiting to be completed, i.e. some fragments have been received but not all of them, the one frame buffering scheme should not add any delay.

If one or more single fragments are lost, they are - in harmony with connectionless principles - not retransmitted. It will then be an implementation-specific decision whether to try to decode incomplete frames or to drop them.

As was presented in Section 2.2.3, buffering is done for a reason. In effect, removing the buffer entirely comes at a cost. We no longer have any time to react to variations in network conditions, thus making all increased gaps between frames visible to the viewer - i.e. there is no smoothing mechanism in terms of packet delay variation or time to allow the client to catch up if bandwidth temporarily suffers.

In the work of De Winter et. al., they employ a buffering scheme that is referred to as *minimal buffering* [32]. There, two scenarios are defined: The *in regime* scenario where packets are all delivered

in sequence and in time, and the *disturbed* scenario where a lack of network capacity causes the timing of packets to be changed or packets are dropped entirely. For the regime scenario, a bufferless scheme similar to the one described above is employed. When the disturbed scenario is detected, feedback messages are sent to the server telling it to reduce video bitrate until the in regime scenario is restored.

6.3.2 Decoding and Rendering

Decoding and rendering have been highlighted as the key processing tasks for the client in that they are expected to induce the most delay of the client's tasks. Decoding a single frame involves the execution of a decompression algorithm on a set of compressed data. While this is not as computationally intensive as encoding raw data, it still takes some time, particularly since lightweight clients are part of this project's target domain. Rendering the decoded frame involves moving the data to a place in the GPU's memory and then displaying it on the monitor or *blitting* it.

These two subtasks are both deemed critical to the overall performance of the client. That stated, the objective of this project is not performance optimisation of decoding and rendering and detailed design of corresponding realisations. In that sense, the induced delay from these two subtasks will be viewed as constants to the overall systems and depend on the choice of codec, codec realisation, renderer support libraries, and finally underlying hardware. With whatever design and codec decisions that are made in the realisation, we are interested in in the actual induced delay by decoding and rendering, and conclude that this should be tested.

6.3.3 Input Handling

Input handling is a key functionality of the client and can be decomposed into three steps:

1. Event detection - listen for user input
2. Packet formatting - convert input events to packets
3. Packet transfer - send packet over network to server

Step 1 will essentially depend on the technology used during realisation. However there are two main approached that can be made. The first approach is to always wait for user input by dedicating a process to continuously listening for input events. The advantage of this approach is that - in theory - no delay is added from event detection. The disadvantage is that it requires a process on its own, which may increase the complexity of the implementation and be against the general design principles of the client software. The second approach is to periodically poll for new input events. This approach will, since polling is a non-blocking operation, not require a process on its own, but will essentially add delay to the system. For instance, if input polling is done every 5 ms and the user pushes a button right after a poll is made, a 5 ms delay will be added to that input event.

Step 2 can be approached in three different ways. The first is to format and send input events *on-event*, i.e. as they are registered - which can be done independently of the approach chosen in step 1. Advantage of this approach is packet size, which correlates with simplicity in that packets only need to read "X was pushed" or "X was released", and no added delay from event transfer aside from the necessary network induced delay. Additionally, this approach is similar to how it is done locally, and as such less of a hassle to implement. The disadvantage is arguably be resilience towards loss: If an input packet is lost, e.g. "X was released", then the server will think that X is still being pressed. This will be the case until X is pressed and released again. The second approach is to send the full state of the keyboard on every event change. The advantage is loss resilience: If a full state update

following X being released is lost, it will be corrected at the arrival of the next state update. The downside with full state updates is the increased packet size since the state of all keys, mouse buttons, and mouse position must be included in every packet. Unless some compression scheme is applied, this would necessarily include redundant information. The third approach is a hybrid of the two. The first approach would generally be applied but with periodic full state updates. This way, average packet size is kept low but with periodic updates, e.g. every 100 ms, to deal with packet loss.

The major decision with step 3 is whether packets should be sent in connectionless or connection-oriented manner, i.e. with UDP or TCP like protocols, respectively (see Section 2.1). The advantage with the latter is that all input packets are guaranteed to be received in order, which would help us ignore any considerations for packet loss. The disadvantage is that the delivery guarantee comes at the price of retransmission. Furthermore, since TCP also guarantees the correct order, a packet loss causes the data stream to stall until the lost packet is retrieved. This would be noticeable for the user, particularly while using a mouse over a non-local network connection, e.g. with 100 ms propagation time. As the mouse requires up to 130 packets every second, loss will occur every now and then even on very reliable connections at which time a delay of at least twice the propagation time can be expected. In effect, for the same reasons as with video streaming (see Section 2.2.3), it is often desirable to rather neglect lost packets instead of doing retransmission - as is the case with UDP.

The thoughts on the use of TCP for transferring input events over an imperfect connection were tested by using Synergy over a network with simulated latency and loss. Synergy is an application that allows two or more computers to share input devices over a network [43]. Since it is designed for local area use only, TCP is used for transferring input information - with no effects on user experience. However, when Synergy was tested on a network simulator with a 200 ms latency and 1 % loss, it became useless due to the frequent jumps of the mouse cursor. It was thus confirmed that TCP is not a viable solution for input transfer over non-local or wireless links. In the Synergy tests, Shunra was used for network simulation [44].

Norberg's work [8] shows that for games that require an in-game combination of keyboard and mouse, the dominating input event generator is mouse movement - e.g. 97.5 % for a user playing Far Cry. As such, assuming an on-event input handling scheme, another argument for UDP can be found since the loss of a mouse movement packet arguably has a much smaller effect, if any, on the user experience than that of a packet containing a keyboard or mouse button.

6.4 Server

Emphasis in this thesis is on the client and core server functionality was primarily extracted from an existing project (see Section 8.1), so only a somewhat superficial review of the server will be conducted. In terms of server performance and delay induction, much is extracted from the server architecture of THiNC, presented in Section 3.2.4 [32]. The server architecture of THiNC can be seen in Figure 3.4.

Independent of server design, encoding and capturing are - aside from frame transmission - the two major delay-inducing subtasks carried out on the server side. This is confirmed by THiNC's performance analysis with respect to delay, seen in Figure 3.5, where encoding (in dark blue) and capturing (in yellow) induce much more than twice as much delay as decoding (in blue) and rendering (in orange). That stated, the optimisation and and design of video codecs is not the objective of this thesis or project. In effect, and much like with encoding and rendering on the client side, decoding and capturing on the server side is treated as constants that can only be affected by underlying hardware and codec choice.

Another interesting observation from Figure 3.5 is how codec configuration affects the induced delay of other things (dark red), e.g. scheduling, buffering, etc. This is particularly the case for configurations

with bidirectional motion estimation (BME), i.e. configurations with one or more B-frames. Referring back to our presentation on video compression in Section 2.2.1 it was highlighted how BME adds delay to video encoding since three frames must be held in the buffer of the compression algorithm. As such, the THiNC analysis makes sense and while BME improves compression efficiency it adds too much delay to be applied here (from around 50 % to 160 % depending on configuration). Additionally, the THiNC analysis noted how the size of the group of pictures (GOPs) affects the stream's resilience towards loss. Smaller GOPs and more I-frames decreases compression efficiency but also resets the effect of one lost frame more often. GOP size was shown to have little effect on delay.

6.5 Summary

A technical review has been carried out by attempting to decompose the various components and tasks of the proposed system, and by analysing them individually, i.e. a *top-down* approach. Emphasis was on the latency induced by each component to the overall interaction latency, and in this regard formulas were set up. Overall latency was composed as follows:

$T_{total} = 2 * Latency + 1/(2 * \mathbf{Framerate}) + \mathbf{Trans_{video}} + Capture_s + Encode_s + \mathbf{Render_c} + \mathbf{Decode_c} + \mathbf{Interaction_c}$

Whereas the components that are affected by design decisions here are embolded. Video transmission times ($Trans_{video}$) will be affected by codec and bitrate choice for both video and audio. Similarly, the video framerate can be changed on a per-session basis. Render time ($Render_c$) and decoding time ($Decode_c$) will be affected by implementation design and third-party libraries chosen for the respective subtasks, which is also the case for interaction handling ($Interaction_c$).

Discussions on the individual components attempt to identify potential pitfalls and challenges for any isolated component, e.g.:

- Network latency cannot be improved by design choices in the system
- Transmission time for an average frame adds substantial latency depending on video bitrate and available bandwidth
- Bufferless streaming should be employed to avoid extra delay due to client-side buffers and queues
- System design cannot affect decoding and rendering delay aside from codec and library choices
- UDP should be used for the input stream
- Bidirectional motion estimation should not be enabled in the server codec configuration
- A higher framerate will increase latency induced by the time gap between frames

Chapter 7

Evaluation Methods

This chapter attempts to give a preliminary approach to evaluating the service as a whole based on the previous chapters. Following the review on such a methodology, a summary is given that will function as a starting point for the system testing that is part of the implementation.

7.1 Methodology

In evaluating the service, two important questions arise: *What* to evaluate and *how* to evaluate it. In regards of the former, evaluating the service is approached by decomposing it into smaller parts and selecting the parts that considered important for the overall completeness of the service. This seems necessary, particularly since the individual parts differ in terms of subjective and objective nature. Components of the latter can be derived with a focus on the technical challenges of the service, as outlined in Section 4.4, and the technical review of the proposed solution in Chapter 6. Subjective aspects were already discussed in Chapter 5.

The assessment methodology will to a large extent be derived from the Games@Large (GaL) project (see Section 3.2.2) and its summary on evaluation methodology, "Evaluation and testing methodology for evolving entertainment systems" by Jurgelionis et. al., based on principles from user-centered design (UCD). There, it is stressed that the evaluation process must focus on user perceptions and experience, i.e. at *user level*, essentially including the functionality of the entire system [45]. Additionally, performance tests on key parts are needed for analysis of the user level evaluation.

Primary focus will be on the subjective assessment, i.e. the user level domain. This stems from the GaL project and in previous discussions where it was stated that the service offers a user experience that is necessarily degraded from what the user might expect traditionally, i.e. playing a game that is executing locally [45]. Furthermore, it should be apparent that any subjective readings will be the result of technical properties. As such, in harmony with the conclusions of the GaL project, final evaluation will be based on user level analysis, using the performance level analysis as a tool in order to quantify subjective readings correlated to technical properties.

7.2 User Level Analysis

7.2.1 Qualitative Research

There are two contrasting scientific approaches to research and testing, namely *qualitative* and *quantitative* research. The qualitative approach requires a smaller samplesize, and will be employed here for two reasons: 1) A smaller samplesize eases the work load and helps mitigate the possible consequences of faulty client distribution, and 2) It is often the preferred approach for research involving user behaviour or perception, particularly when the output is relatively uncertain. The small samplesize, e.g. 5 to 20 subjects, makes user feedback more manageable and allows us to interact with subjects that give feedback deviating from an overall trend.

In comparison, quantitative testing requires a larger samplesize, i.e. test subjects, and it requires a population of subjects that is representative to a society or a target group. If carried out correctly, the quantitative approach is more accurate and can allow us to analyse and make conclusions on a general basis. A qualitative round of testing is often needed first in order to do quantitative testing at a later stage. As such, quantitative testing will be listed as future work in Section 15.

7.2.2 Video and Audio

Objective metrics for video and audio quality have been defined in research on codecs and streaming media, and are often preferred as they are easily quantifiable and do not require any tedious involvement of users [31]. Employing the subjective approach, however, is not as specific, but formally defined methodologies do exist. Here, much motivation is found in Sander Vorren's work, "Subjective quality evaluation of the effect of packet loss in High-Definition Video", based on ITU-R recommendation BT.500, "Subjective video quality assessment methods for multimedia applications" [46, 47].

From ITU-T P.910 there are two main categories for subjective video evaluation: The *single-stimulus method* and the *double-stimulus method*. The primary difference between the two is that with double-stimulus the video is played side by side with the reference video, from which the test subject can tell the difference between the two. In single-stimulus evaluation, only the video to be evaluated is played. As with Vorren's work, the latter will be chosen here as it is how the service will be used, i.e. without any reference video to compare with. Furthermore, since the service in a sense attempts to mimic the user experience of playing a game that is locally, which essentially sets maximum restrictions on how well the service can perform, it is decided to have the video quality graded with the adjectival category rating scale (ACR) where the highest value is defined as indistinguishable from the game executing locally [47].

To complete the user experience, the presence of audio is important. Audio quality is, however, not considered as an important a challenge as video, and is thus not given as much emphasis in the evaluation. Notions will be made about the synchronisation of audio.

7.2.3 Delay

There are no formally defined methodologies for the influence of interaction delay (see Section 5.2) for any application similar to the proposed service, which is unfortunate since service responsiveness has been established a key challenge. This is attempted solved by evaluating delay largely based on the methodology for video, defined in the previous section, particularly in that a correlation between user experience and interaction latency should be confirmed.

The difference of interaction delay influence on the three game classes that were defined - first-person

shooter, real-time strategy, and arcade racing - is of particular interest. In effect, an evaluation of delay on the three classes needs to be carried out.

7.2.4 Application

In addition to gathering user impressions of the service it would be valuable to also gather information on their opinion on the general service concept. With this follows that users should be asked to rank the features and applications of the service, as presented in sections 4 and 4.2, respectively.

7.3 Performance Level Analysis

7.3.1 Processing Load

Performance tests should be carried out, both at the server and the client. For the server, results will help determine the scalability of the server and what kind of parameters that are applicable, e.g. framerate, resolution, etc. For the client, results will help determine what hardware that is supported as the end user device, i.e. how *light* the client software is.

The results will be gathered either by the software itself, i.e. through implementation, or by third party software, e.g. system monitors or task managers. System monitors require less work but are only capable of seeing things from the system's perspective. Consequently, system monitors are suitable for finding how much CPU time that is allocated to a process but not suitable for finding how time is allocated within one process.

With basis in the technical review in Chapter 6 it is definitely of interest to see how much that is spent on average by the client doing various tasks - e.g. decoding and rendering a frame. This implies that performance testing will require implementation of some timer that is called before and after some of the major functions that are called on each iteration.

7.3.2 Network Load

Network load should be gathered in three categories, as discussed in Section 6.2: *Bandwidth*, *latency*, and *loss*.

- Bandwidth should be measured both for the media stream going from the server to the client (video and audio) and the input stream going from the client to the server
- Latency should be measured at the client
- Loss should be measured at the client and the server

As with processor load, network utilisation can be observed either by the system itself or by third party monitors.

System performance should also be tested for its resilience towards undesirable network conditions, particularly loss. Effects of loss should be recorded for different loss percentages (i.e. varying network conditions) and different values of the maximum transfer unit (MTU).

7.4 Summary

In this part we have described a service, proposed a solution for realising the service, reviewed the technical challenges of the realisation, and analysed the various factors affecting the user experience. Finally, with basis in the research from similar projects, various aspects of the proposed service and solution were reviewed for the purpose of evaluation.

In short, we have made the following decisions on how the service should be evaluated:

- Evaluation should be based on a working implementation and user testing
- Evaluation can be decomposed into various assessments in two different levels - user level and performance level
- User level consists of assessments of video and audio, delay, and application:
  - User level testing will be of qualitative orientation and need a subject population of 5-20 people
  - User testing of audio and video quality should be based on single-stimulus method with the ACR scale (see Section 7.2.2)
  - Effects of delay should be evaluated with respect to game classes (see Section 5.3.2)
  - A stronger basis for the discussion of application should be established from user impressions of the service
- Performance level consists of assessments of processing load, network load, and loss resilience:
  - Both assessments are discussed in the technical review (see Chapter 6)
  - Processing load will be reviewed by observing the technical workings of the implemented system
  - Network load will be reviewed in the same matter and in three categories: Bandwidth, latency, and loss
  - Effects of loss should be measured with different session settings

Following the realisation, these decisions set the stage for how the evaluation in Part V is carried out - particularly in terms of deriving a test plan and creating a questionnaire.

Part IV

Realisation

Chapter 8

Development Process

This chapter sets the stage for how the development process is carried out. It does so by first presenting the setting under which the development first took place and then explaining the methodology that is applied.

8.1 Starting Point

As the development process was started, much had already been created through the original Geelix project. Healthy development principles as well as common sense says that software should be reused where possible, as will be applied here.

The software component that will be reused is the Geelix Heads Up Display, or HUD. The HUD is essentially an application presented as an in-game widget, rendered over the game graphics. Its presence can be toggled. A screenshot of a game running with the HUD is shown in Figure 8.1.

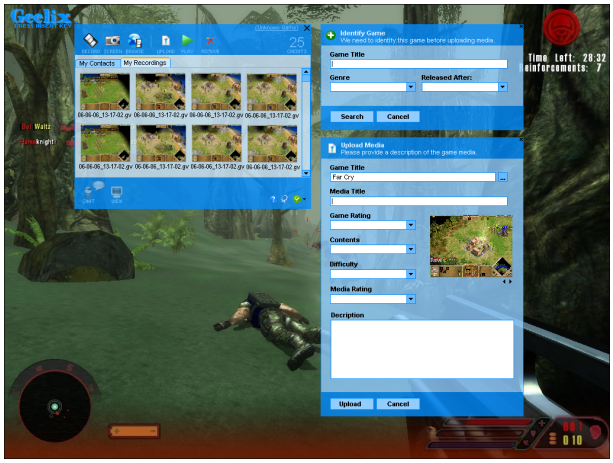

Figure 8.1: Screenshot of Geelix HUD

Without going into much detail, the in-game HUD entails the following functionality:

- Overview of friends, their online status, and what game they are playing
- Instant messaging functionality with friends
- Record gameplay as video and automatically upload it to a web portal

- Enabling shared gameplay where game graphics and audio is captured and streamed real-time to friends
- Connecting to a friend and seeing his gameplay in an in-game video window

Note that the HUD only allows users to observe the gameplay of a friend, and not interact with it. This project will reuse only parts of the HUD. More specifically: The functionality of sharing live gameplay through video and audio streams, while the rest are omitted. Conclusively, the stripped functionality of the HUD will be used to make up large parts of the server in this project, dramatically shrinking the work load for developing and implementing the server. For the client, some minor parts of the HUD were found useful for reuse, particularly the implementation of the packet cache which handles packet fragmentation and assembly.

Prior to the work shown here was carried out, a client prototype for the Windows platform had been developed. The prototype functionality was limited and none of its source code was reused in this implementation, yet it was found useful as a tool for guidance. Concurrently with the development presented in the following, the prototype was extended to a full working Windows client.

8.2 Development Methodology

The system development follows no strictly defined development process but is in a sense loosely based on the classic waterfall process model. It is said to be only loosely based because the various development steps were sporadically revisited when needed, however not in a repeating or iterative manner such as the spiral and incremental models.

For the sake of presentation, the realisation is presented in harmony with the waterfall process which from [48] is as follows:

Requirements: Deriving system requirements in order to understand exactly what to create

System design: Using the requirements to design a system architecture built up by software and hardware components and the interactions between them

Implementation: Putting the designed system to life, i.e. the actual programming

Testing: The process of testing the implementation and comparing final result to the requirements and original concept

It should be observed that this is a subset of what is defined in the waterfall model. More particularly, the steps concept analysis, integration, and maintenance are missing. Concept analysis was carried out in the Part III. The two others, however, have more or less been omitted entirely. This is due to the nature of the system to be developed and particularly that it has no explicit customer at this point.

Rolv Bræk and Øystein Haugen's work on real time system design "Engineering Real Time Systems: An Object Oriented Methodology using SDL" [49] will be used as a point of reference model for designing the system. They define a set of rules that they argue should be applied to any real time system, namely *S-rules*. Here, the rules will not be followed strictly but will be used as a basis for evaluation. Furthermore, they define step wise guidelines to implementation design. These steps will be applied during system design.

The testing will not be presented here beyond comparing the implementation with the specified requirements. More emphasis is put on performance and user testing of the realisation (see Part V),

which both includes implementation-specific tests as well as a broader evaluation of the service as the whole. Informal unit testing of smaller software components was carried out during development, but is omitted here.

8.3 Modeling

The Unified Modeling Language (UML) version 2 will be used as tool in the development process. More particularly, UML will be used for *functionality*, *deployment*, and *realisation* [50].

For modeling functionality, use case diagrams will be use to show functionality from the perspective of a user. Diagrams show how users interact with the system and the dependencies between functionality and external roles.

For modeling deployment, i.e. system architecture and design, component diagrams will show system decomposition and dependencies while sequence diagrams will show the interactions between components.

The realisation will be modeled with further use of sequence diagrams as well as collaboration diagrams.

UML was in the early stages of development chosen over Specification and Description Language (SDL), defined by ITU-T. SDL is formally complete and is supported by powerful tools that allow SDL to be used for code generation and particularly simulation [50]. SDL has emphasis on strict system design through concurrency and asynchronous communication while UML is a more flexible and loosely defined language, targeted towards visual representation. Since the envisioned system would be small in terms of components - i.e. clients and servers - simulation was not deemed necessary and UML was preferred.

Chapter 9

Requirements Specification

This chapter is concerned with translating the conceptual service that was described in Chapter 4 into a list of absolute requirements that can function as a recipe for the design and implementation. This will be approached by first reviewing overall goals with the implementation, then defining the requirements, and then putting them into the context of user scenarios.

9.1 Implementation Goals

It was demonstrated through examples that the concept service has a wide range of possible applications (see Section 4.2). However, choices regarding what application the service will be implemented for were made, and as such all requirements in this chapter are specificed in order to realise *that* system.

The reasons for implementing the service have implicitly been mentioned as the service development has progressed, but for the sake of review they are listed explicitly in the following:

- **Demonstration**
  Implementing the core functionality of the service, i.e. playing a computer game that is in fact executing on a remote computer, will serve as a proof-of-concept to others and certainly, depending on how well it performs, prove some of the potential of the service as well as the underlying technology.
- **Performance Testing**
  By implementing the service we are able to observe how the service performs and its requirements. With this we can try to derive the technical constraints on the system in order to achieve a service with acceptable user experience.
- **User Testing**
  A working implementation will allow us to test the service concept on real users and record their experiences. In the end, this is the ultimate test that decides the success of the service.

Furthermore, in order to employ the reasons listed above we create a set of overall goals for the implementation:

Completeness: System must be working according to the requirements specified in this chapter

Distribution: The client application must be distributable for all required platforms

Testability: The system must support additional functionality for testing analysis

Delivery: Being part of a project where the deadline is absolute, implementation must finish so that it leaves sufficient time for user testing

It should be noted that the various results that will be found based on the implementation will at first be valid for this specific implementation only. However, by further analysis and knowledge about how things work we hope to be able to generalise the answers to some extent and propose possible optimisations for future work (see Section 15).

9.2 User Perspective

9.2.1 Stakeholders

A stakeholder of a project is essentially any person who has a stake in the project's outcome. As such, we define three different types of roles that somehow interact with the system.

- **User**
  The user is the actor that will be using the system through the client application. The quality of service offered to the user will eventually determine the failure or success of the system.

  Primary concerns: Perceived quality of service, well designed user interface

- **Provider**
  The provider is the actor responsible for administrating the server(s).

  Primary concerns: Stability, deployment

- **Tester**
  The tester is the actor responsible for gathering test data from the server, as well as administrating it. For the evaluation part, this is us.

  Primary concerns: Retrieving test data, user feedback, deployment

The three roles can very well all be played by the first person (see Section 4.2 and "Released Software"). The tester role is necessarily included since functionality for testing was explicitly added in the functional requirements.

9.2.2 Overview

While the list of functional requirements is long (see Section 9.3), there are relatively few paths that a user can take when he / she interacts with the system[1]. This comes from two things. First of all, only the core functionality of the concept service is targeted for implementation, and secondly, end user simplicity and an easy to use interface was already deemed as critical in the previous section.

Figure 9.1 shows the overall use case diagram and should illustrate the functional simplicity of the system. Some textual use cases were also defined during development and can be found in Appendix A.

[1]Note that while use cases here are presented before the functional requirements were specified, they were in fact developed more or less concurrently.

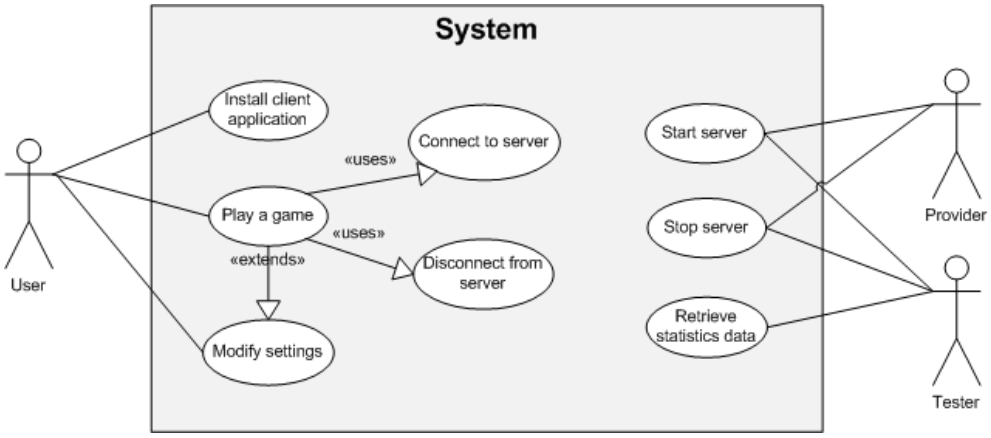

Figure 9.1: Use case diagram for the system

9.3 Functional Requirements

Functional requirements "specify services that the application must provide" [48]. These are the functional requirements (FR) for the core functionality of the service:

FR 1 User can download, install, and run a lightweight client application

FR 2 Client can connect to server over a network through a hostname or IP address

- FR 2.1 Server listens to specific port for incoming connections
- FR 2.2 Server allows only one connection at a time
- FR 2.3 Server responds with a "busy" message if a second client attempts to connect

FR 3 Client can request certain session settings upon connection

- FR 3.1 Server take requests into consideration but may override them as desired
- FR 3.2 Client will leave final decision up to server and no further negotiation is carried out

FR 4 Client can disconnect from a server

- FR 4.1 Client sends a "bye" message and tears down connection
- FR 4.2 Server receives "bye" and tears down connection

FR 5 Server streams a computer game to a client upon connection

- FR 5.1 Server sends game graphics as compressed video
- FR 5.2 Server sends game sound as compressed audio
- FR 5.3 Client sends periodic heartbeat message to keep connection alive

FR 6 Client receives and renders video and audio streams

- FR 6.1 Video stream is presented to the user on the desktop
- FR 6.2 Video can be shown in windowed or full screen mode
- FR 6.3 Audio volume can be changed through the operative system settings

FR 7 User is able to interact with the game through the client application

- FR 7.1 Client supports keyboard, mouse, and joystick input

FR 7.2 Client only captures input events when the video window is in focus or in full screen

FR 7.3 Video window can grab the input so that the mouse cursor disappears into the window and key presses are only registered by the application

FR 7.4 Input grabbing can be enabled/disabled by a certain combination of buttons (similar to ctrl + alt + delete)

FR 7.5 Client forwards input events to the server

FR 7.6 Server receives the input stream and inject the input events into the game

FR 8 User receives continuous updates on the state of the network connection through the application

FR 8.1 Heartbeat messages is used to determine latency between client and server

FR 8.2 Packet sequence numbers is used to calculate loss

FR 9 Server should detect any client crashes in order to avoid freezing

Additionally, the following requirements are added for testing and debugging purposes - referred to as *testing requirements* (TR):

TR 1 Client can take screenshots of the video stream

TR 1.1 User takes screenshot by pressing a predetermined key combination

TR 1.2 Screenshot is saved on hard drive as a raw (i.e. non-compressed) image file with indexed name

TR 2 Client measures the processing times of certain events

TR 3 Both server and client logs important events

TR 3.1 Log files should be stored on hard drive of respective host

TR 4 Client sends messages to server with latency times and loss values

TR 4.1 Server will log the messages received from client

TR 5 Client should gather host computer information (OS version, CPU, GPU, memory, etc) at application spawn and send to server

9.4 Non-Functional Requirements

Non-functional requirements "qualify a service or services (specifies something *about* them)" [48]. We categorise the non-functional requirements (NR) as follows:

Performance: Timing and quality constraints that are of particular interest for real-time systems

NR-P 1 Perceived interaction latency should not exceed 200 ms and be should be perceived as constant

NR-P 2 Video quality should be acceptable from 512 kbps and up and should support a resolution of at least 640 x 480

NR-P 3 Audio playback should be continuous, i.e. glitch free, and by synchronised with video

NR-P 4 Session setup time should not exceed 2 seconds

Reliability and availability: Deals with system and module crashes and is of less importance for experimental purpose systems

NR-R 1 Server should on average handle at least 10 game sessions between each crash

NR-R 2 Server should automatically reboot if it crashes

NR-R 3 Client should on average do at least 5 game sessions between each crash

Note: These values are purposely set low due to the experimental nature of the system and will be altered at post development stages.

Interface requirements: Requirements for how a user interacts with the system / application

NR-I 1 Client application should offer a simple to use graphical user interface (GUI) for all video settings, audio settings, and session controls

NR-I 2 Video and audio settings should be set to default values at application spawn and should in most cases not need to be changed

NR-I 3 GUI should include the following input fields:

- NR-I 3.1 Session: Server address, port, packet size
- NR-I 3.2 Video: Bitrate, scaling, framerate
- NR-I 3.3 Audio: Bitrate

NR-I 4 Server field should accept both IP addresses and host names

NR-I 5 GUI should include the following action buttons:

- NR-I 5.1 Native ”exit” button (in application title header)
- NR-I 5.2 ”Connect” & ”disconnect” buttons - with according results

NR-I 6 Upon connection, GUI will spawn a video window that shows the game

- NR-I 6.1 Video window titlebar should present latency and loss information
- NR-I 6.2 Native ”exit” button on video window should do the same as the ”disconnect” button of the application window

Constraints: Limits or conditions on things like accuracy, design, and platforms

NR-C 1 Application should run on Windows, Linux, and Mac OS X

NR-C 2 Application installation package should be smaller than 5 MB and install easily for all versions

NR-C 3 Application should require little more processing power than a normal streaming video application

9.5 Summary

The functional specification was created out by first reviewing the overall goals with the implementation, then deriving a functional overview for the system, and finally both functional and non-functional requirements were specified. Due to its experimental nature, the system will have particular focus on non-functional requirements.

Basis for the testing requirements can be found in chapters 5 and 7. More specifically, requirement TR 2 is necessary for the purpose of performance testing induced delay of individual components. TR 1 will be used to compare the visual quality of various settings and TR 4 is necessary to analyse potential correlations between latency and user experience. Similarly, implementation goals can be found represented by non-functional requirements, such as NR-C 1 and NR-C 3 that represent cross platform and thin client features, respectively.

Chapter 10

Design

10.1 Approach

From Bræk and Haugen we have the following guidelines for implementation design [49]:

1 Perform the trade-off between hardware and software

2 Define the hardware architecture

3 Define the software architecture

4 Restructure and refine the functional design

These steps provide an organised and purposeful order in which to present architecture related decisions and are all employed in the following sections.

10.2 Hardware and Software Tradeoffs

Before defining any type of architecture it is crucial to decide what components or modules to integrate as software and what to integrate as hardware. We will focus on the following major considerations in this step:

- **Physical distribution and physical interfaces**
  The system will, at a minimum, need to be physically distributed at two sites - the server and the client. Furthermore, without assuming full control over all network resources, no elaborate distribution schemes can be applied (e.g. with regional media server proxies). In effect, all network traffic must traverse the entire path between the client site and the server site.

  It is assumed that any participating computer has dedicated hardware for networking, i.e. a network interface card (NIC). The NIC will always be colocated with its host computer and not change the physical distribution of the system.

  User input devices and client output will necessarily be realised as dedicated hardware such as keyboard / mouse and computer monitor, respectively.

- **Time constraints versus processing capacity**
  It should be obvious from the non-functional requirements that time constraints, i.e. delay, and

processing capacity is critical for this system. Delay and processing of management signals, e.g. connect and disconnect, is however not as critical.

- Rendering should be processed with hardware, i.e. with the hardware support of the GPU if available
- Encoding will not be processed with dedicated hardware due to cost limitations and will be processed by the CPU
- Decoding will be processed by the CPU
- Game execution will be processed by both the CPU and the GPU

It was mentioned that any computer is assumed to have a NIC. This is good in that software does not need to worry about the very basics of networking, however the software must still read data off the NIC and as such the NIC does not solve any time constraints for the system.

- **Costs**
  As a rule, hardware is more expensive than software. This was a key reason for using a software encoder instead of dedicated hardware.

Furthermore, by using software decoders and encoders, the system is more easily deployed since we make no unusual assumptions about the availability dedicated hardware for encoding (server side) or decoding (client side). We do, however, assume that a GPU is present server side, for game execution. This is not necessarily true since most current day server hardware is not meant for graphical operations. A GPU is more likely to be present at the client but is an optional requirement for the system, enabling lighter end user devices without GPUs to participate.

10.3 Defining Hardware Architecture

Based on the discussion in the section above, we define a hardware architecture that is illustrated in Figure 10.1.

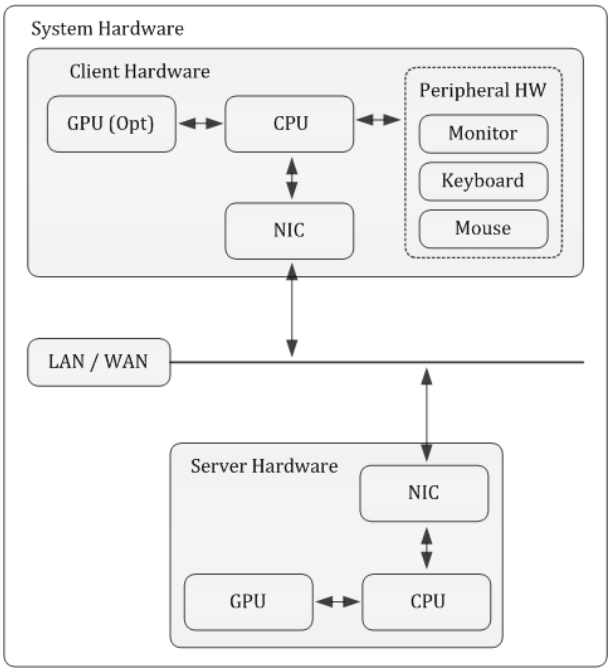

Figure 10.1: Informal SOON diagram of hardware architecture

Bræk and Haugen states "The emphasis at this stage is the overall hardware structure in terms of computers, peripheral equipment and communication channels" [49] and is what is done in the SOON diagram. The diagram follows SOON aesthetics but not strictly and is thus classified as an *informal* SOON diagram.

Things to note on the design:

- Server requires a hardware GPU while it is only optional (but preferred) at the client
- Peripheral equipment at the client is modeled somewhat loosely and is shown to communicate only with the CPU. This is not necessarily the case as the presence of a GPU would require communication with the monitor and the GPU
- Some hardware components that are part of the typical computer were not modeled, e.g. hard drive, memory, sound card, etc

10.4 Defining Software Architecture

10.4.1 Server Design

The design and implementation of the server software module was not carried out by the author but by other participants in the project and was to a large extent based on reuse of software from a different project. In short, the architecture is as illustrated in Figure 10.2.

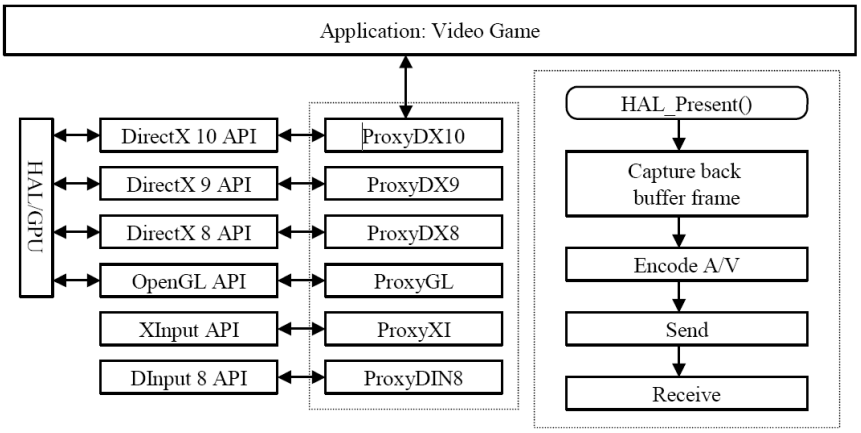

Figure 10.2: Server software architecture (excerpt from [9])

The server software is designed to be implemented as dynamically linked library (DLL) files that acts as proxies that load as the game spawns and override the graphics and input APIs called by the game. Note that these APIs are property of DirectX, further explained in Section 2.5. In effect, by putting code in the DLL proxies that carry out various tasks and then call the original APIs, the DLL proxies allow us to add certain functionality without modifying game code or device drivers.

More specifically, the two DLL proxies carry out the following tasks:

- **Graphics Capture (DirectX / OpenGL)**
  - Capture snapshots of game graphics by accessing the memory of the GPU backbuffer

- Encode and compress snapshot by snapshot as video frames
- Encode and compress game audio as audio samples
- Send data

- **Input Injection (XInput / DirectInput)**

  - Receive and parse input data
  - Recreate input events and inject them into the executing game

Additionally, the graphics proxy also contain all other server functionality such as listening for incoming connections, negotiating video settings, and so on. As a result, whenever the game is running the server is also running.

10.4.2 Client Design

The design of the client was a gradual process and is as a result divided into components with very specific tasks, as illustrated in the informal SOON software structure diagram in Figure 10.3.

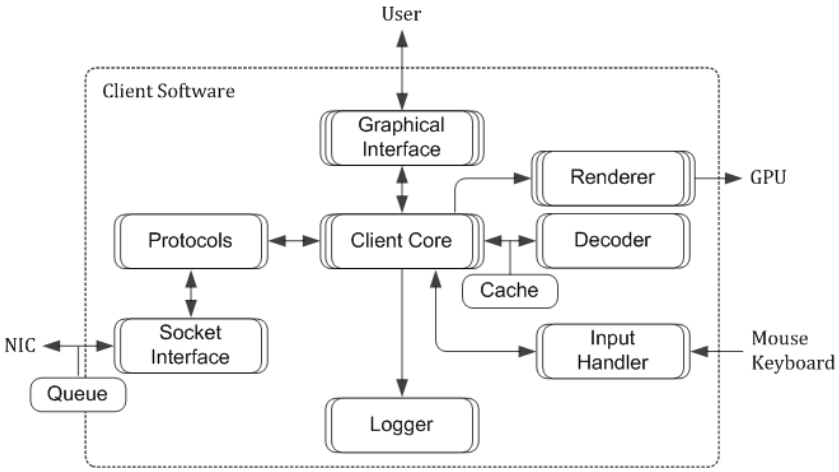

Figure 10.3: Client software architecture

The software structure shows that central behaviour is placed in the *client core*, which makes all major decisions and call the other components functions when needed. It receives instruction from the user via the *graphical interface*, i.e. the main application window. Networking is then carried out by doing function calls on the *protocols* API, which functions as a wrapper class for sending things with the right format. The protocols API interacts with the NIC through the *socket interface*, i.e. sending data to and reading off the sockets. The queue that is modeled is not meant to be implemented but illustrates the queuing functionality that exists in the socket and on the NIC [1].

All packets that are received go through the client core, which then decides what to do with them. If they are session control packets, i.e. a connection response, it is processed directly. Media stream packets, however, are immediately forwarded to the *decoder* through a packet *cache*. The packet cache is needed in order to assemble fragmented packets. The decoder processes packets as they are completed in the cache.

When a packet has been decoded it is again forwarded to the *renderer*. The renderer processes both audio and video and plays either for the end user through the GPU or a sound device - which is then represented on the monitor or through speakers. Concurrently, the *input handler* works as a listener

for input events as they occur when the user does something with either the mouse or keyboard. The client core then processes input events and forwards them to the protocols API which converts them to a packet format that is readible by the server and sends the packet. Finally, the *logger* is called by the client core when something needs to be logged. The logger then writes it to a file on the hard drive (not modeled).

10.5 Component Interaction

While the overall software and hardware architecture give an idea of how components are linked, it is purposeful to discuss the communication - particularly the network traffic between client and server - in a more detailed way.

10.5.1 Behaviour

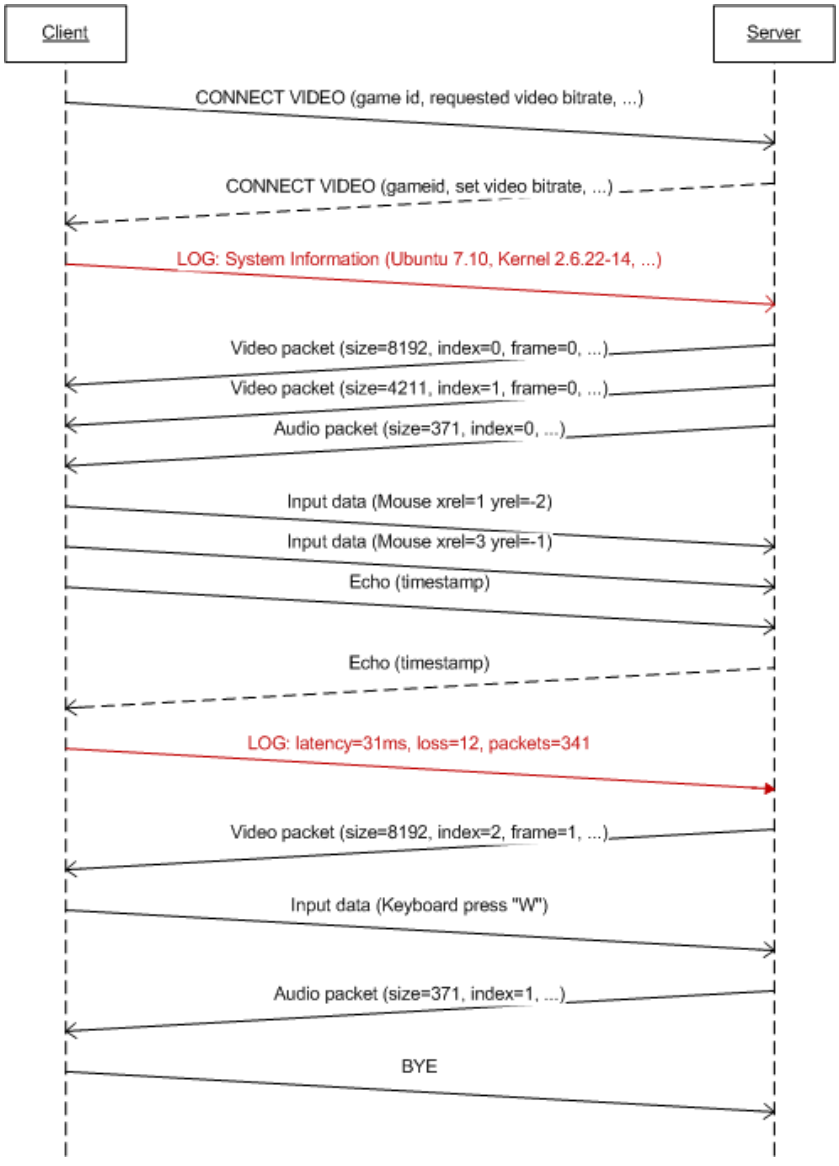

Figure 10.4: Sequence chart of network traffic between client and server. Messages for testing are in red

Figure 10.4 shows a sequence diagram of the signals that are exchanged between the client and the

server in relatively normal operations. Sequentially, it shows the following:

Connection: Client sends a `CONNECT` message with requested session settings. The server then evaluates the settings, makes a decision, and (in this case) accepts the connection with a response including the final settings. For logging, the client sends a message with host computer information.

Receiving media: Following a successful session initiation, the server starts streaming game content in audio and video packets. They may be fragmented and are sequentially numbered in order to detect packet loss.

Sending input: Concurrently with receiving media, the client sends a stream of input packets containing input events - one packet per event.

Heartbeat: Concurrently with media and input streams, the client periodically sends an `ECHO` message containing a clock which, upon the server receiving it, is immediately returned with the same clock. The client can then calculate latency which is sent to the server as a message together with packet loss information, for logging.

Disconnection: Finally, the client disconnects by sending a `BYE` message. No further signalling is done. If the message is lost, the server will simply time out due to the lack of `ECHO` messages.

Additional sequence charts of the inner workings of the client can be found in Appendix B.

10.5.2 Protocols

A protocol was worked out in order to achieve communication between server and client. We originally define three categories of packets: *Control* packets, *media*, and *input*. Table 10.1 shows an overview of the different packet types. The control packets go both directions are used for session initiation and teardown. Media packets go only from the server to the client and contain either audio or video and may be fragmented.

Type	**Direction**	**Tasks**
Control	Both	Session initiation, teardown
Media	Server → Client	Audio and video
Input	Client → Server	Keyboard, mouse, echo
Testing	Client → Server	Logging

Table 10.1: Protocol packet types

The input packets go from the client to the server and are either keyboard events, mouse events, or echo packets. Keyboard events are button presses or releases and mouse events are either button presses, button releases, or mouse motion. Echo packets are sent periodically and let the server know that the client is still there. If the server does not receive an echo packet for a set number of seconds, it deems the client as timed out and closes the session.

Input packets with keyboard presses or releases use the Windows Virtual Key (WVK) format. This was done in order to ease server implementation. For the client developed for the Windows client, this is trivial since the WVK format is available natively in the operating system. For the Linux and OS X versions developed here, however, translation is needed (see Section 11.2.2).

Additionally, a packet type was creating for testing. This packet type (shown in red in Figure 10.4) is sent from the client to the server and essentially instructs the server to log something. This something will usually be latency and loss information or client host information.

More details on the protocol input format can be found in Norberg's work [8].

10.6 Design Evaluation

10.6.1 Single-Threaded Design

Reviewing Figure 10.3 we can, according to SOON notation, spot 3 software processes (i.e. squares with three vertical borders) and 5 software procedures (two borders). This suggests that much of the application will be implemented as *poll-based*, i.e. a single threaded design where events are polled for and handled in the iterations of a main loop - presented in pseudo code in the following:

```
while ( true )
{
    if ( PollSocket() == true )
        ReceivePacket()
    if ( PollPacketCache() == true )
        DecodeData()
    if ( PollDecoder() == true )
        RenderData()
    if ( PollInputHandler() == true )
        SendInput()
    LogEvents()
}
```

By polling each module before calling a function that takes action, one makes sure that the iteration does not stall at any point. Also, as is, the loop would iterate as fast as possible. Avoiding this would require the use of a wait or sleep function.

A major downside with the single threaded design is that functions cannot be interrupted or interleaved with other functions. This is not a primary concern in this project due to its very sequential nature. As long as video packets on average are fully processed, i.e. decoded and rendered, before a new one arrives, it would appear that no scheduling-induced delay occurs. However, the following scenario demonstrates that the single threaded design may indeed induce delay: A frame fragment is received, completing the frame, thus pushing the completed frame for decoding and rendering. At the very same time, the user performs some input event, e.g. shooting. In the single threaded domain, the decoding and render will complete in X ms before the input event is detected and sent. In effect, the scheduling has induced an X ms delay where X is the summarised time for decoding and rendering. Referring back to the technical review in Chapter 6, this implies the following:

$Sched_{input} = Render_c + Decode_c$

Note that this is only the case for certain input events, i.e. a worst case scenario. Nonetheless, the scheduling-induced delay of the single threaded design is not optimal and should be revised.

10.6.2 Overall Design

In order to discuss the overall design, we include a set of selected S-rules from Bræk's work.

S-Rule: Physical distribution (Bræk, p.258)
Distribute processes in a way that minimises the bandwidth needed over physical channels

This was discussed in Section 10.2 on hardware and software tradeoffs. It was concluded that little could be done in terms of process distribution that would affect the bandwidth needed. As such, this S-rule is *satisfied*.

S-Rule: Input-output-layering (Bræk, p.282)
Try to hide the physical details of input-output in separate software modules that handle the physical layer (...)

All network input and output is carried out in the socket interface module which acts as an abstraction layer for calls on the system sockets. This S-rule is *satisfied*.

S-Rule: Input-ouput processes (Bræk, p.282)
When input-output modules are time-critical, (...), they should be implemented i software processes that may be scheduled independently from the application software

As discussed in Section 10.6.1, the design implies a single-threaded implementation of the main behaviour. Having stated that modules such as networking and input handling are time critical, we find that we have not designed them as independent processes, i.e. threads. In effect, this rule is *not satisfied*.

S-Rule: Max waiting time (Bræk, p.282)
Always specify a max waiting time when waiting for on external events in order to avoid infinite waiting in error situations. (...)

As can be seen in the functional requirements, timers should be implemented whenever request-response communication is needed. This S-rule is *satisfied*.

S-Rule: Hardware similarity (Bræk, p.265)
Look for similarity between hardware modules. Increase cost/benefit by using similarity to minimise the number of different component types needed and maximise the reuse of each

We use no custom hardware components and all hardware modules used (see Figure 10.1) are arguably mainstream. This is S-rule is *satisfied*.

S-Rule: Generality (Bræk, p.304)
Preferably use the most general and flexible implementation techniques wherever they satisfy the design constraints (...)

For the purpose of reuse, the protocols API module is separated from the client core. This is purposeful since it allows the protocol to be implemented only once and shared between any implementation that interacts with components in the system, here client and server. Additionally, the separation of the socket interface, which will likely need to be implemented separately on different operating systems, allows the protocol API to be used on all different operating systems, as well. This is particularly useful for implementing protocols since any change on one end will need changes on the other end. This S-rule is *satisfied*.

Chapter 11

Implementation

This chapter describes what was actually implemented and compares the final result with the work presented in the previous chapters. First, the environment under which the implementation took place is presented, then comments on differences between implementation and the design are made. Finally, screenshots of desktops with the software in use are shown.

11.1 Environment and Tools

System development was carried out primarily in the offices of Gridmedia Technologies in San Francisco, California. Several computers were dedicated for development in addition to one computer dedicated as a game station, which allowed us to test various games and run them with the server module. What would turn out to be a major problem was a faulty local area network in the office building where development took place. Even local sessions experienced high jitter and loss which was confusing at first since it implied that something was wrong with the implementation.

Since the client application was to be supported on three different types of operating systems, i.e. Windows XP / Vista, Mac OS X, and Linux, all mentioned operating systems were used during development. The author's work was targeted towards the OS X version and Linux versions for the different distributions that were supported, i.e. Ubuntu, Debian, and OpenSUSE (see Section 2.3.1). The application was first implemented for Linux, and then ported to OS X. The only major difference between the Linux versions and the OS X version is the implementation of different GUIs on different frameworks.

Development tools used include Eclipse C/C++ Development Tools (CDT) as the integrated development environment (IDE), Glade Interface Designer for GNOME interface design (Linux GUI), the GNU Compiler Collection (GCC) for compilation, the GNU Debugger (GDB) for debugging, and the XCode Interface Designer for interface design on the Carbon framework (OS X GUI). The GNU tools Automake and Autoconf were used for configuration and compilation, in addition to the features of Eclipse CDT.

Additionally, some third-party libraries were used during implementation:

SDL: Simple DirectMedia Layer, for rendering audio and video and listening for input events (see more in Section 2.4.2) [24]

FFmpeg: For decoding, encoding, and converting audio and video (see more in Section 2.4.1) [22]

GTK+: The GIMP Toolkit, a widget toolkit for creating GUIs in the X window system [51]

libglade: A library for reading the XML files generated by the Glade Interface Designer

Both GTK+ and libglade are written in C, however it was found useful to use C++ API wrappers named gtkmm and libglademm, respectively. A major source of inspiration for the use of FFmpeg and SDL was the video player tutorial based on those two libraries, by Stephen Dranger [23].

All code was written in C++.

11.2 Challenges

Many challenges were encountered during the implementation. Some were general and not specific to this project, e.g. linking errors due to the extended use of third-party libraries, while others were more particular for this system and of more interest to be mentioned here.

11.2.1 Debugging

Debugging the implementation was difficult for two reasons. First, the real-time and streaming nature of the system means that some main loop will be iterating frequently so that functions are being called repeatedly several times a second. Furthermore, since the client will continuously need to communicate with the server it is not straightforward to pause the application on certain events. Second, due to the of third-party libraries, errors induced by our code would often spawn faults in third-party code, e.g. FFmpeg. These types of errors were difficult to trace back to their source.

Debugging was handled through development by writing events and variable states to the log file. Upon unexpected errors such as sudden segmentation faults 2 minutes into a running session, GDB was applied to pinpoint the source of error. The FFmpeg library was at times very difficult to work with and its use will be reconsidered for future projects.

11.2.2 Input Translation

The protocol specification (see Section 10.5.2) showed that keyboard button events were to be sent in Windows Virtual Key (WFK) code and is thus what is expected server side. As such, translation was required on the Linux and OS X version implemented here. Since SDL is used on both platforms, it was found useful to use the SDL names, i.e. *SDLKey*, for translations. This way, the same translation maps could be used on all platforms using the SDL API. This would not have been the case had we used platform specific scancodes instead.

11.2.3 Licensing

Licensing is important when working with third-party libraries. In the Linux domain much software is open source but still regulated by software licenses. There are two dominating software licenses that are widely used in the open source paradigm. The GNU General Public License (GPL) [52] is a strong *copyleft* license, thus requiring that any derivative project of GPL must also be available under GPL. An alternative to GPL is Lesser GPL (LGPL) [53] which is similar to GPL but that does not require the copyleft restrictions to apply to software that only links to LGPL projects.

This project is not open source and the implementation could thus not benefit from GPL-licensed libraries. We could however link to LGPL licensed libraries. These conditions affected the project in

the following ways:

- Could not use libx264 (GPL) for encoding and decoding (see Section 11.2.4)
- Could not use libswscaler (GPL) for pixel formatting
- SDL (LGPL) could be used for rendering together with SDL_image and SDL_ttf
- Glade and libglade (LGPL) could be used for GUI development
- libcurl (LGPL) could be used for HTTP requests

It should also be mentioned that both GPL and LGPL has strict rules for modifying libraries enabled by them. This was however not a problem for this project as the third-party libraries were only used through their respective APIs and not modified.

11.2.4 Codec Choice

We have from Section 2.2.1 that the choice of video codec can greatly affect the video quality delivered to users for various compression ratios. Better subjective quality means a better user experience and, as such, the very best video and audio codecs should be used for the compression of media in our system. Having based the implementation on FFmpeg, a wide range of encoders and decoders are supported by default or by adding auxiliary libraries. As was mentioned in the prestudy, one auxiliary codec library that performed well in tests was the libx264 library - an implementation of the x264 codec based on the H.264 standard [54, 12]. The library is open source but unfortunately under the GPL license - as was discussed in the previous section - not an option under the constraints that this project is being carried out.

To completely avoid issues revolving licenses and patenting, it was deemed necessary to go back in time and use a codec that had been abandoned in terms of commercial use and that is thus free without restrictions. The WMV8 codec was chosen - first on this merit and also because of good test scores in the 512 Kbps to 1 Mbps bitrate domain [13]. Furthermore, WMV8 was already in use in the existing server module that had been developed prior to this project (see Section 8.1) and is supported by FFmpeg by default. The same reasons apply for the choice of using WMA8 as the audio codec.

11.3 Source Code

The client source code consists of around 4600 lines of code with comments, without the code of third party libraries such as SDL and FFmpeg. Additionally, a library of shared code between the Linux project and the Windows project consists of around 1300 lines of code. The GUI XML file consists of 1100 lines of code autogenerated from the Glade designer.

The various components of the client (see Figure 10.3) are implemented as separate classes:

- `GXLinuxClient:` The client core. Main intelligence and decision making, as well as the main function to be iterated.
- `GXLinuxDecoder:` The decoder. Manages frame and audio decoding by calling the FFmpeg API.
- `GXLinuxInputHandler:` The input handler. Listens on SDL window for user interaction and reacts accordingly.

- `GXLinuxRenderer:` The renderer. Creates and closes the SDL video window. Blits decoded frames to the SDL window and plays back audio samples.
- `LiveGamesApp:` Graphical interface and entry point. Loads GUI XML file and binds events to actions. Manages the main timer loop, calling the main function of the client core on each iteration.

in addition to these classes, other smaller classes were implemented for administrating and supporting functions, such as logging, system info retrieval, performance testing, etc.

As an example, an extract of the source code of the client core main function, i.e. `GXLinuxClient::DoWork()`, is shown in Listing 11.1.

Listing 11.1: Extract from `GXLinuxClient::DoWork()`

```
/**
 * @function    DoWork()
 * @brief       One iteration of the main loop. Calls all DoWorks
 */
void GXLinuxClient::DoWork()
{

[ ... ]

#ifdef PERFORMANCETEST
    SW_Active.LapStart();
#endif

    // Let input handler do work
    if (gxl_input.DoWork() == -1)
        CloseClient("Requested by user");

    // Get packet from cache
    CGSPktCache::GXPKT* pPktCC = m_cache.Get();

    // If there was a packet pending, decode and queue it in renderer
    if (pPktCC != NULL)
        DoDecQueue(pPktCC);

    // Then, render items from queues
    gxl_renderer.DoWork();

    // Check for more packets and put potential packets in packet cache
    if (DoListen() > 0)
        OnVidPkt();

#ifdef PERFORMANCETEST
    SW_Active.LapStop();
#endif
}
```

The similarities of the implemented source code with the single-threaded design pseudo code in Section 10.6.1 should be clearly revealed. First, the input handler is called and allowed to poll itself for input events not yet handled. Then, a complete packet is retrieved from the packet cache and, if one was found, the packet is decoded and then rendered - independent of whether it is audio samples or a video frame. Finally, networking is allocated time to read data off the socket and to the packet cache.

It should also be noted from Listing 11.1 that in the beginning and end of the presented call, a `StopWatch` instance is started and stopped, toggled by the global `PERFORMANCETEST` define statement.

This is done for the purpose of performance testing and will be revisited in more detail in the evaluation (see Section 12.3).

Additional extracts of central code from the decoder, renderer, and input handler can be found in Appendix F.

11.4 Review

The implementation satisfies the requirements specification to a large extent in terms of functionality. This likely comes from the fact that the specification is not meant for public release but for some specific purposes (see Section 9.1) combined with a development process were steps were taken back and forth in order to add, remove, or change requirements as the implementation progressed.

Requirement FR 7 is not fully satisfied since no support for joystick is implemented.

Determining if the non-functional requirements are satisfied is not a trivial task since non-functional aspects of a system tend to be difficult to specify in quantifiable terms. Furthermore, many of the requirements make unrealistic assumptions about the system conditions.

Requirement NR-P 1 is likely not satisfied. We have from [32] that even in perfect network conditions, time spent in the various steps of processing, e.g. encoding and decoding, will be more than 100 ms for non-index frames (see Section 3.2.4). As such, perceived latency values higher than 200 ms should be expected. This is pursued further in Section 12.3.

Requirement NR-P 3 is likely not satisfied. Audio and video data is decoded and rendered as it is received by the client. However, due to the larger data size of a video frame compared to the data size of audio samples, some audio buffering was deemed necessary in order to have smooth, i.e. glitch free, sound. In effect, even with perfect network conditions, audio and video rendering is not perfectly synchronised. A solution would be to delay the video somewhat in order to achieve synchronisation. This, however, increases interaction latency. A more elaborate scheme should be derived in order to ensure synchronised audio and video without increasing interaction latency.

The requirements for reliability and availability - i.e. NR-R 1, 2, and 3 - are from the author's experience with using the system likely not satisfied. Proper methods for fixing faulty situations such as memory leaks should be applied.

11.5 Screenshots

11.5.1 Main Application Window

A screenshot of the Linux version of the main application window is shown in Figure 11.1.

The non-functional requirements stated that the graphical user interface (GUI) should be easy to use and at spawn be set with default values that rarely should need to be altered (NR-I 1 and 2). This is approached in the GUI by indeed having default values as well as hiding the more technical settings inside an *expander*, i.e. a button that can be clicked that essentially increases the size of the application window containing the technical settings. The expanding button can be seen as "Advanced settings" in Figure 11.1.

Furthermore, the checkboxes "Fullscreen" and "Grab input" decides if fullscreen and / or grabbed input should be enabled on connection. While connected, these settings can be toggled by pressing

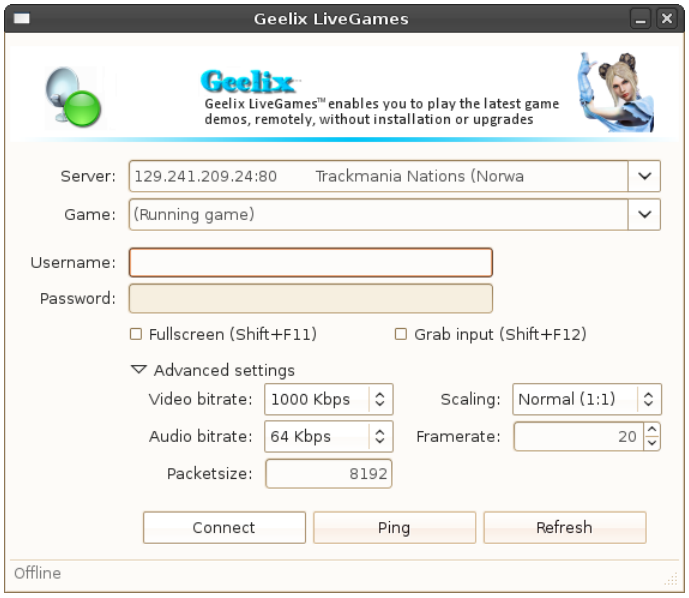

Figure 11.1: A screenshot of the Linux client main application window

shortcut keys as stated in the GUI.

Things to note:

Game: Here, the user can choose which game on the server that he or she wants to play. Since the system yet only supports one game per server, this combobox is not in use.

Username and Password: These entry fields were added for the purpose of supporting some sort of authentication from the client if needed. This is not in use and the password entry is thus disabled. The username entry is in use for testing purposes, i.e. the test subject can type in his or her email address for identification.

Ping and Refresh: These two buttons are not in use. The ping button was intended to run a quick network test towards the server, giving some indication of the status of the network connection, while the refresh button was intended to update the server and game lists from some web server.

Offline: The statusbar in the very bottom of the main window shows the session status as well as error messages. For instance, during a session it would say "Connected" while if a connection attempt fails due to the server being busy it would say "Error: Busy".

Connect: While connected, this button will switch and read "Disconnect" with respective functionality.

11.5.2 Game Window

A screenshot of the Linux version of the game window is shown in Figure 11.2. Here, the client is connected to a server running the game Trackmania Nations.

The game window should look as if the game was played in windowed mode. What can be noticed is that the titlebar, i.e. the top of the window, shows latency and loss statistics for the current session. This is in harmony with requirements FR 8 and NR-I 6. Also, no local mouse cursor can be seen which is because the game window has grabbed the input. If that is not the case, the local mouse cursor can be seen and moved around like with any other desktop application.

A gallery of screenshots of the application running the three different game classes is shown in Figure

Figure 11.2: A screenshot of the Linux client game window

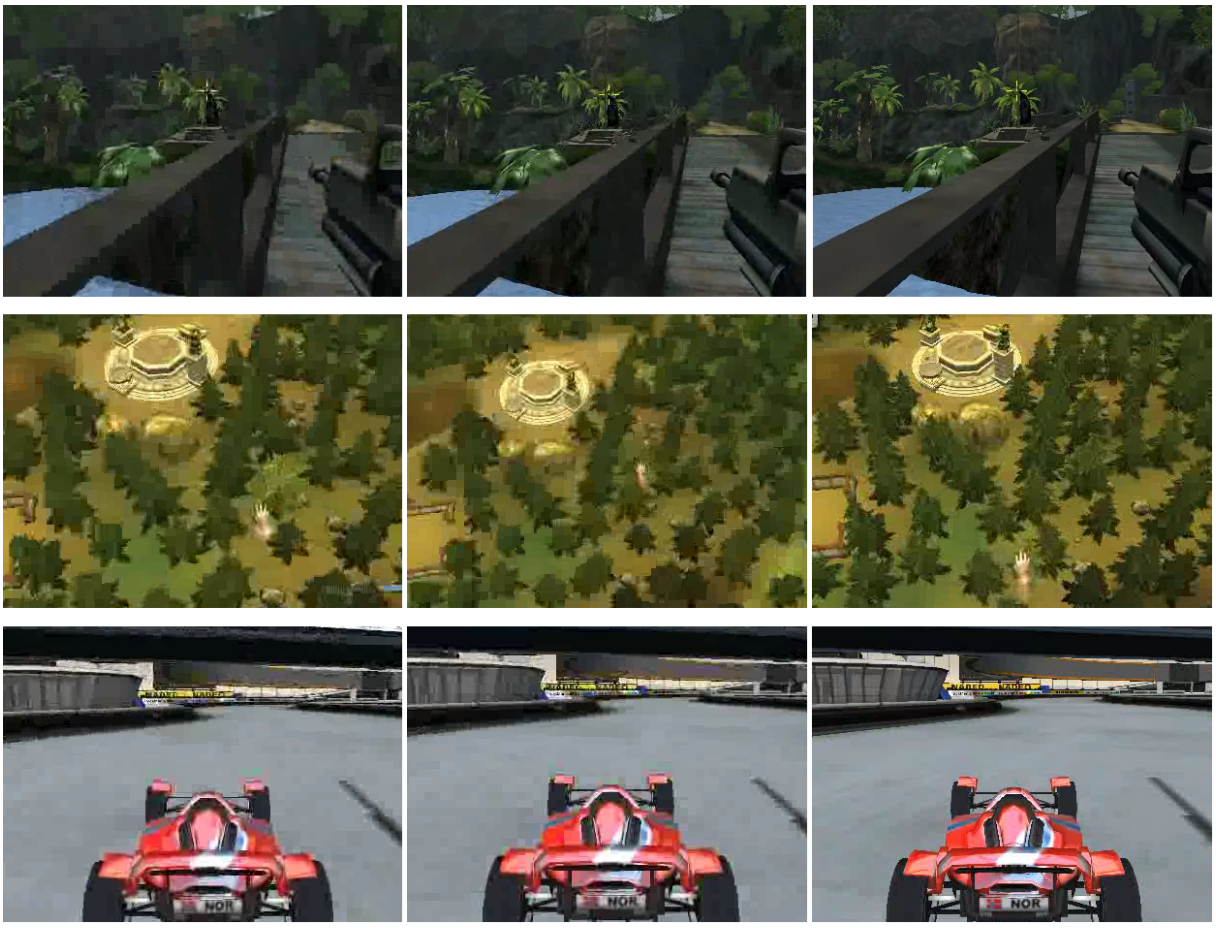

Figure 11.3: Screenshot gallery of the three game classes as content provided by the service. From top: FarCry, Black & White 2, and Trackmania Nations. Each class is shown with three different video bitrates: 512 Kbps (left), 1 Mbps (middle), and 3 Mbps (right).

11.3. Each class is shown as content of the service with three different bitrate settings. Player view was moving while all screenshots were taken in order to show the worst of blocking errors. In a sense, this is not fair to BW2 since the view is fixed most of the time - as will be discussed in the test analysis in Section 13.4.1.

11.6 Client Versions

As has been mentioned, the work shown here is the development of the Linux version which was then ported to OS X. Additionally, the Linux version was compiled and distributed for 3 different Linux distros, namely the most recent versions of Ubuntu, OpenSUSE, and Debian. Two screenshots showing the client application running in Mac OS X and Ubuntu 7.10 are shown in 11.4. More information on deployment and distribution is found in Appendix C.

Figure 11.4: Two screenshots showing the client application running in Mac OS X (left) and Ubuntu 7.10 (right)

In terms of functionality, all versions are identical and use the very same source code. It should be observed that the GUI in the OS X version looks somewhat different from the Linux versions since it was developed in Carbon and not GTK+, however the input fields and functionality should all be the same.

Part V

Evaluation

Chapter 12

Test Plan

The purpose of a test plan is to describe the process of testing a system. It may include terms such as approach, scope, criteria, schedule, etc. This chapter is to a large extent based on the discussions of Chapter 7. In effect, deriving a test plan is here presented in two categories of scope, i.e. user level and performance level. Additionally, the former is divided into two sequential phases, as illustrated in Figure 12.1.

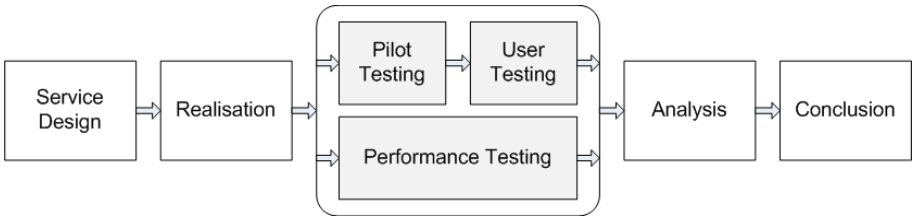

Figure 12.1: Overview with emphasis on testing

12.1 Pilot Tests

The general approach for phase A is as follows:

1 Complete implementation

2 Setup at least one game server

3 Invite a few selected users over e-mail with instructions and questionnaire

4 Distribute client software on a web page that is linked to in the instructions (see Figure 12.2)

5 Have test subjects carry out test cases found in the instructions and then fill out the questionnaire

6 Review returned questionnaires and server logs

7 Derive and optimise testing process based on that analysis

As should be observed, the tests of Phase A are brief and are carried out in order to improve the testing process of phase B - i.e. "testing the test" - commonly referred to as a *pilot test*.

It was early on decided to distribute the software client on an available web page. This instead of having users show up at some test site where computers with the client software are already set up.

The major advantage of our approach is that it will test both the system and the client software as it is intended to be used in post-development. This is deemed more realistic and puts more aspects of the system to test, e.g. in-home network conditions and software distribution. This is discussed further in the following section.

The data gathered from such a test will not be used in the analysis directly but instead as a tool used to determine the following:

1 The willingness of users to participate in the testing, i.e. how difficult it is to acquire test subjects

2 If the implementation provides a quality of service that is strong enough and a good representation of the intended service

3 If setting up the system and distributing the implemented software is reliable and suitable for user testing

4 If functionality for logging is implemented and working according to expectations

5 If the test instructions given to users are understood without ambiguity and followed correctly

6 If the questionnaire given to users following the test is asking the right questions and phrasing them correctly, i.e. without ambiguity

Pilot testing is to be carried out on 1-3 test subjects. After the testing, the subjects should be available for follow-up questions that may help us optimise and specify the test plan further.

12.2 User Tests

12.2.1 Setup

User testing is strongly based on the discussions revolving user experience factors identified in Chapter 5, i.e. latency, video and audio quality, and content. Consequently, this implies three conditions for the user testing:

1 Tests should be carried out with subjects in different network conditions

2 Test subjects should try sessions with different video bitrate settings

3 Each test session should include several test cases involving different game classes

The first condition is realised by itself since test subjects will be at different locations and on different networks. Ideally, the test subjects should represent the whole spectre of network conditions, i.e. from high latency to low latency, from high loss to low loss, and from high bandwidth to low bandwidth. The extent to which this is the case will be determined by server logs that measure these different values.

The second condition is partly realised by requirement NR-I 3 (see Section 9.3), i.e. that the main application window should have a setting for requested video bitrate. To fully satisfy the condition, the test subjects must be instructed to play the same game at different video bitrates and then be asked about the video quality at different bitrates in the questionnaire.

The third condition will be realised by setting up at least three game server running at least one instance of each of the three game classes defined in Section 5.3.2: First-person shooter (FPS), real-time strategy (RTS), and arcade racing (AR). In choosing what instances of each of the game classes to use it was necessary to consider one of the major constraints of the server software module - it is only compatible with games running on DirectX 8 or 9. As such, it was opted to use *FarCry* as FPS, *Black & White 2* as RTS, and *Trackmania Nations Forever* as AR.

After pilot tests were carried out with 3 test subjects as well as in-house testing, the following was observed:

1 **Participant willingness**

- Not as high as hoped. Invitations were sent out to 7 people in order to acquire the 3 test subjects.

2 **Implementation quality of service**

- Sufficient for testing if network loss is low or non-existent
- Not testable for lossy connections such as over wireless

3 **Distribution**

- Windows installation OK
- Mac OS X not yet ready at this point in development
- Problems with Linux packaging dependencies for different versions of third-party libraries

4 **Functionality for logging**

- Worked as planned and according to needs
- The need for a log file parser was identified

5 **Test instructions**

- Was generally deemed as too brief and vague
- Pparticularly lacked instructions on what to do within a game, i.e. how to play the game

6 **Questionnaire**

- Some questions were found missing, particularly how the service affected the user experience of different games and the various tasks carried out in games
- Grading scale and how the questions were phrased were vague and ambiguous
- Not enough user information gathered, particularly where the test was carried out and on what network connection
- Questions about the service concept were lacking and should be revised according to discussions around the applications of the service as presented in this thesis

Consequently, the discoveries made were reviewed and at best effort attempted solved and integrated in phase B. Little could be done with the strict requirements for a stable and relatively fast network connection in order to use the system. Concurrently with preparing phase B, however, efforts were made to improve Linux distribution files as well as to complete the version for Mac OS X, which indeed it was.

Furthermore, much time was spent revising the test case instructions and the questionnaire. The instructions were extended and organised in steps to be strictly followed by the test subject in a

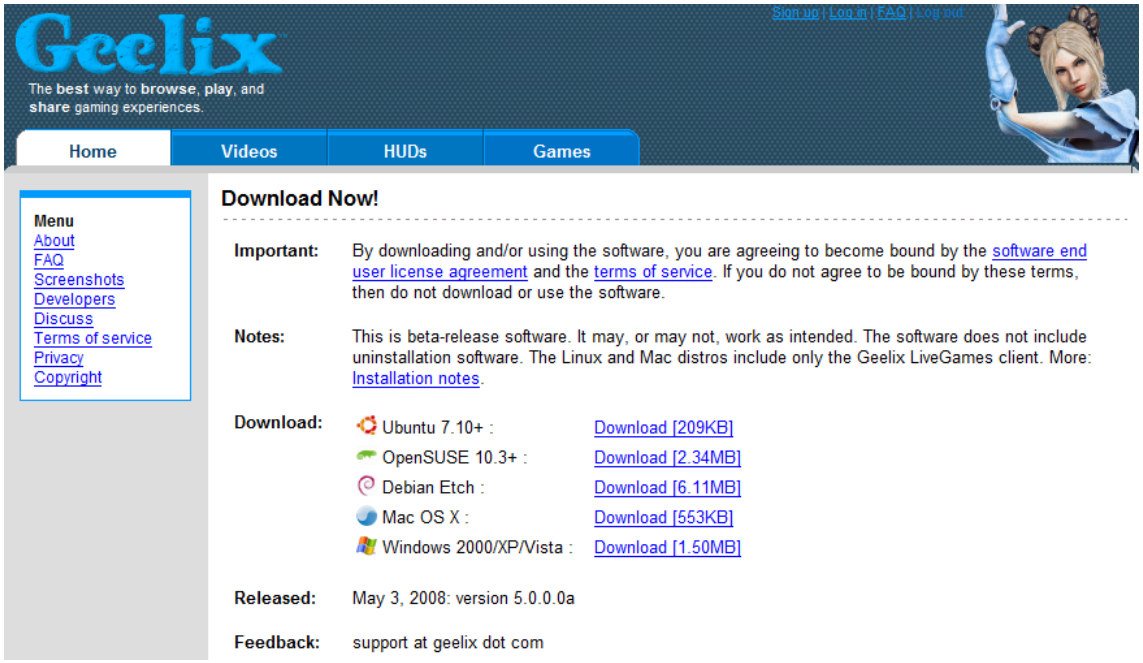

Figure 12.2: A screenshot of the web page for client software distribution during user testing [4]

”click this button” fashion. It was explicitly stated what the subject should do within each game, which relates to the revisions of the questionnaire. It should also be mentioned that the subjects were instructed to type in their e-mail address in the username field of the application. This was done in order to be able to compare server logs to the subjects questionnaires at a later stage.

The questionnaire, i.e. the list of questions to be answered by the test subject after the test instructions have been carried out, was separated in three parts: Test subject info, implementation questions, and service statements. In part 1 the subject was asked to provide information about the context of the test session. This included date and time of test, platform, location, network connection information, hardware information, etc. In part 2 the subject was asked questions on his impressions of the implemented system. The subjects were asked to evaluate the tested system in regards to overall impression, difference between game classes, difference between certain parts of some games, difference between video bitrates, if audio and video was out of synch, and if loss of input data or video and audio data was noticeable. In accordance with the discussion in Section 7.2.2, all questions in part 2 are to be answered with the adjectival category rating scale ranging from 0-10 where 10 is defined as ”no different than playing computer game that executes locally” and 0 is defined as ”extremely different from executing locally, unplayable”.

Due to the subjective nature of the tests, it is challenging to define a criteria for what is deemed satisfactory for the scores of part 2. While the rating scale is presented to subjects in a manner that we hope will receive results that can be considered linear, there is no guarantee that this is the case. That stated, it is a goal that the implementation on average is rated within what is defined as ”good” in the questionnaire, i.e. 7-9. Consequently, the satisfactory criteria is set to 7.

Part 3 asks the subjects to evaluate statements about the service concept in general and are less dependent on the quality of the implementation. More specifically, the statements try to reveal the primary reasons for why a user would *not* use the service as well as the applications for the service that a user finds most useful. Here, the subjects will use an absolute scale ranging from 0-5 where 5 is defined as ”completely agree” and 0 is defined as ”completely disagree”.

The questionnaire used during phase B can be found presented in Appendix D.

12.2.2 User Test Cases

While more detailed and step-by-step instructions were given to the test subjects, the user test cases are summarised in Table 12.1.

Test Case	Game	Bitrate 1	Bitrate 2	Bitrate 3
Case 1	FarCry	512 Kbps	1 Mbps	3 Mbps
Case 2	Black & White 2	512 Kbps	1 Mbps	3 Mbps
Case 3	Trackmania Nations	512 Kbps	1 Mbps	3 Mbps

Table 12.1: User test cases

It can be observed from the table that the test cases differ only in terms of game title and video bitrate. All other settings were set to default values and remained identical through all the test cases. The different game titles are server-specific settings and can only be controlled by the test subject through connecting to different servers. The video bitrate is a client-specific setting and must be set by the subject.

The test subjects were for each test case and bitrate instructed to "play actively" for 3 minutes. For FC this involved walking around and shooting enemies. For BW 2 it involved picking things up and solving simple tasks. For TNF it involved completing tracks as fast as possible. More specifically, for TNF, subjects were instructed to drive different tracks for the three different bitrates. This was done for the purpose of testing the effects of latency, so that subjects would not memorise parts of the track and be able to predict it more easily for one bitrate than another.

12.3 Performance Tests

Performance tests will be done separately from the user tests and for this purpose separate test cases will be used. From this follows that performance testing will be carried out by the author and not with the help of test subjects.

12.3.1 Traffic Generation

Measuring the network traffic that was generated by the system is done with a third-party network protocol analyser, Wireshark[1] [55]. Wireshark is available for free under the GNU General Public License (GPL) on multiple platforms and is capable of monitoring and logging network data from the physical layer and up. Wireshark will be running on the client computer set to filter out all packets not exchanged between the known client and server.

12.3.2 Processing Loads

In order to carry out tests on the various client-side processing loads it is deemed necessary to make changes to the source code (see Section 7.3). We implement the `StopWatch` class, which indeed works as a stopwatch that can be started and stopped through function calls `LapStart()` and `LapStop()`, respectively. In addition to starting and stopping time, the stopwatch also registers the number of stops, i.e. the number of laps. Three `StopWatch` instances are injected into the source code:

[1] Wireshark was prior to June 2006 named Ethereal.

- `SW_Decoder` - time spent on decoding video
- `SW_Renderer` - time spent on rendering video
- `SW_Input` - time spent on handling user input

It should be observed that these three stopwatches correspond with the conclusions of the performance level analysis (see Section 7.3) and are employed in order to find the average delay per frame for the various tasks. Additionally, a stopwatch measuring the total execution time of the application process is injected, `SW_Total`, in order to find processing shares for the other three tasks.

12.3.3 Performance Test Cases

While the test cases for performance testing are similar to that of user testing, they are tailored more for the technical aspects of the service and less for content. The test cases are listed in Table 12.2.

Test Case	**Game**	**Bitrate**	**Resolution**	**Framerate**
Case 1	FarCry	512 Kbps	800 x 600	20
Case 2	FarCry	1 Mbps	800 x 600	20
Case 3	FarCry	512 Kbps	1024 x 768	20
Case 4	FarCry	512 Kbps	800 x 600	5
Case 5	Black & White 2	512 Kbps	800 x 600	20
Case 6	Trackmania Nations	512 Kbps	800 x 600	20

Table 12.2: Performance test cases

As can be observed from the table, 6 test cases are set up that differ in game, video bitrate, resolution, or framerate. All test cases will run for 5 minutes of active playing with what is deemed as typical in-game behaviour. For FarCry, for instance, typical in-game behaviour means that the player will move around and not lay still. Furthermore, test cases should not involve time in the menus. Audio will be set to 64 Kbps for all tests.

Performance tests will be carried out with certain goals in terms of what questions we hope to answer, with case 1 being the reference model for all other cases. Comparing results from cases 1, 5, and 6 will possibly reveal if different game content - i.e. the three game classes defined in Section 5.3.2 - performs differently on the client. Furthermore, cases 1 and 2 will compare loads from different video bitrates, cases 1 and 3 will compare two different resolutions, and cases 1 and 4 will compare two different framerates.

In addition to the 6 performance test cases listed above, a case 7 is defined. The settings of case 7 are identical to case 2, i.e. running FarCry at 1 Mbps. However, for case 7 various types of behaviour are strictly scheduled:

1 60 seconds: Enter game menus and change some settings and idle some

2 60 seconds: Walk around and look around in-game

3 60 seconds: Walk around but do not look around

4 60 seconds: Stand still and do not look around

The objective with the scheduled behaviour is to monitor network traffic generation and packet frequency for variations according to the different types of behaviour.

12.4 Sources of Error

Before presenting the results gathered by employing the test plans it is deemed necessary to identify any possible sources of error in the testing process. These errors can potentially degrade the quality of the test results and can in a sense be thought of as the test process *risks* - as such, following their identification, steps should be taken in order to ensure the quality of the test results by mitigating these risks. We identify the following sources of error:

Test subjects: The population of test subjects may not be representative to the population of the targeted users, e.g. in terms of size, age, gender, gaming experience, etc

Test conditions: Test scores will depend not only on network conditions but also on other things such as monitor size, light setting, hardware specifications, etc

Game representation: It is assumed that the chosen game titles will represent the three defined game classes, which may not be the case

Game preference: Test subjects may prefer one game over another and be affected by this during testing

Questionnaire: Ambiguous questions or statements will affect scores since test subjects might interpret a question or statement differently

Implementation: Implemented functions for performance testing, especially the `StopWatch` class, might be we wrong or inaccurate

Dealing with the problem of test subjects is arguably straightforward. We should make sure we have enough test subjects and that they represent the population that most likely will be using the service. Population size is essentially a question of the scale and duration of the testing process while representation is problematic since testing is carried out without a specific application or target group in mind.

The varying test conditions between sessions may or may not be a source of error. As was discussed in Section 12.2, a more clinical lab-like environment could have been used but would at the same time have increased the work load of the user tests and, more importantly, have been contradictory to how the service is envisioned being used. That said, the test conditions are random and can potentially affect different test sessions to different extents. As such, test conditions should be considered as a possible reasons for test scores deviating from the overall trend.

Correct game representation is attempted ensured by reviewing the defined game classes (see Section 5.3.2) and finding game titles that match the class definitions on all points.

In any subjective testing, preference is a potential source of error. Here, it could be attempted mitigated by selecting test participants that have a widespread history of gaming and who prefer different games. Additionally, it could be helpful to use game reviews in order to select game titles that are generally accepted as being on the same level, quality wise. Furthermore, test subjects are instructed to focus on the delivery of the games and not the games themselves.

The questionnaire has - through the process of pilot testing - been reviewed and modified according to the findings in those tests.

The implementation of the `StopWatch` class has been tested with other timing devices and was deemed adequately accurate. It is, however, still possible that inaccurate values are registered for quick starts and stops. Furthermore, it is also possible that the function calls are at the wrong places in the client source code. In this regard it is also important to note that operating system scheduling likely will

affect stopwatch precision, i.e. by allocating CPU time to other processes while a stopwatch for a subtask is running. This is attempted mitigated by running a minimum of other processes on the test computer during testing.

After the test results have been presented and analysed, these risks will be reviewed again to help determine the validity of the results.

Chapter 13

Results and Analysis

This chapter presents the results that were gathered from employing the test plans derived in the previous chapter. Test data will be based on questionnaire responses as well as statistics found in server logs. Concurrently, the results are analysed. Emphasis will be on establishing a basis to answer the research questions that were created in Section 1.3 as well as to highlight deviations from expectations or overall trend.

13.1 Testbed

13.1.1 Server Specifications

In order to support three different game titles, three game servers were set up at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. These servers were used for pilot tests, user tests, and performance tests.

Model: Dell Optiplex GX 620
Processor: Intel Pentium D 3.0 GHz
Memory: 2 GB
Graphics card: Intel 82945G Integrated
Operating system: Microsoft Windows XP SP2

Server 1: FarCry
Server 2: Black & White 2 demo version
Server 3: Trackmania Nations Forever

The servers were connected to NTNU's 100 Mbps campus network.

13.1.2 Client Specifications

Performance tests were carried out on a laptop computer connected to NTNU's campus network, i.e. communicating with the servers locally.

Model: HP NX 7000 laptop
Processor: Intel Pentium M 1.6 GHz
Memory: 512 MB
Graphics card: ATI Radeon 9200M

Operating system: Ubuntu 7.10, Linux kernel 2.6.22-14

During pilot and user testing, client hardware was whatever the test subjects had access to and is as such not specified here.

13.2 Implementation

In the following, various figures of test results are presented. All scores in this section are on a scale from 0 to 10, where 10 is better.

13.2.1 Video Quality

The experienced video quality divided by video bitrate is shown in Figure 13.1. There should be an obvious trend here that higher bitrates provide a higher video quality. However, it was hoped that at 3 Mbps, video quality would score higher than 6.5 of 10. This can be explained with variations in test scores, likely due to loss which affects higher video bitrates more than lower bitrates (see Section 6.2).

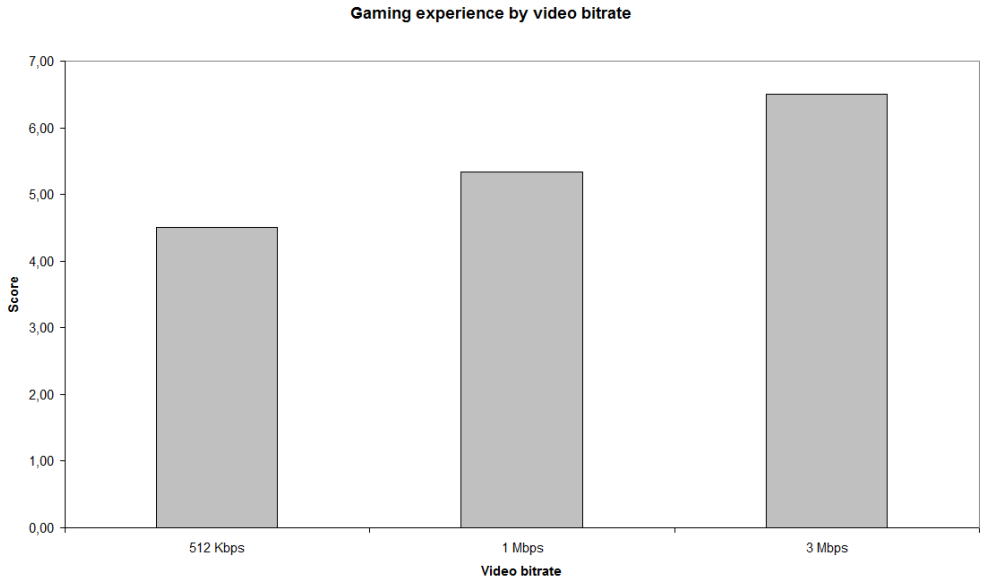

Figure 13.1: Gaming experience for different video bitrates

By reviewing test scores it was observed that all test subjects scored video quality either in segment 0-3 or segment 8-10. One test subject even scored video quality opposite of the overall trend, i.e. putting 512 Kbps highest. This could have been to misunderstanding the rating scale, however user comments confirmed that the scale was used correctly. In the 8-10 segment, it was observed that 3 Mbps scored 3-4 points higher than 512 Kbps with 1 Mbps somewhere in the middle.

13.2.2 Audio Quality

Audio quality was only trialed at 64 Kbps, which should be of adequate quality. Degraded sound quality was thus primarily due to implementation errors and audio being out of synch with video - which was not handled (see Section 11.4). This was confirmed by user responses. Overall, sound quality received a score of 5.8 of 10 (see Figure 13.2). It was observed that the audio quality scores to a large extent correlates to the video quality scores, and as such it is likely that audio quality is affected by the same factors.

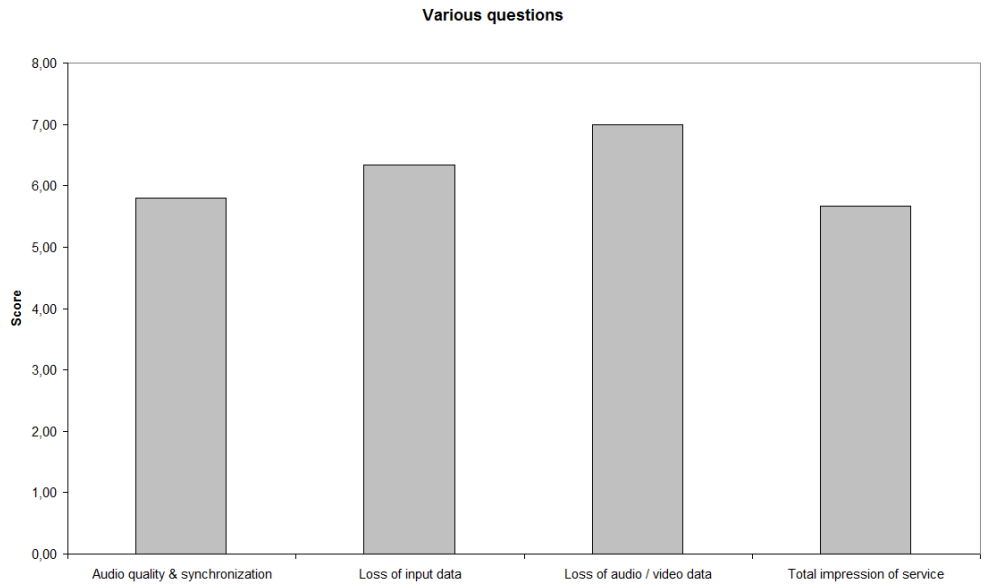

Figure 13.2: Questions 3, 4, and 5

13.2.3 Loss

Loss of input data and audio / video data received scores of 6.3 and 7.0, respectively, with values from the entire scale. From reading user comments, it seems that 1 or 2 test subjects did not enable the "grab input" functionality of the application, and thus getting a different user experience in terms of interacting with the game. This might have affected the score for loss of input data. Loss of audio or video data was likely determined by network conditions.

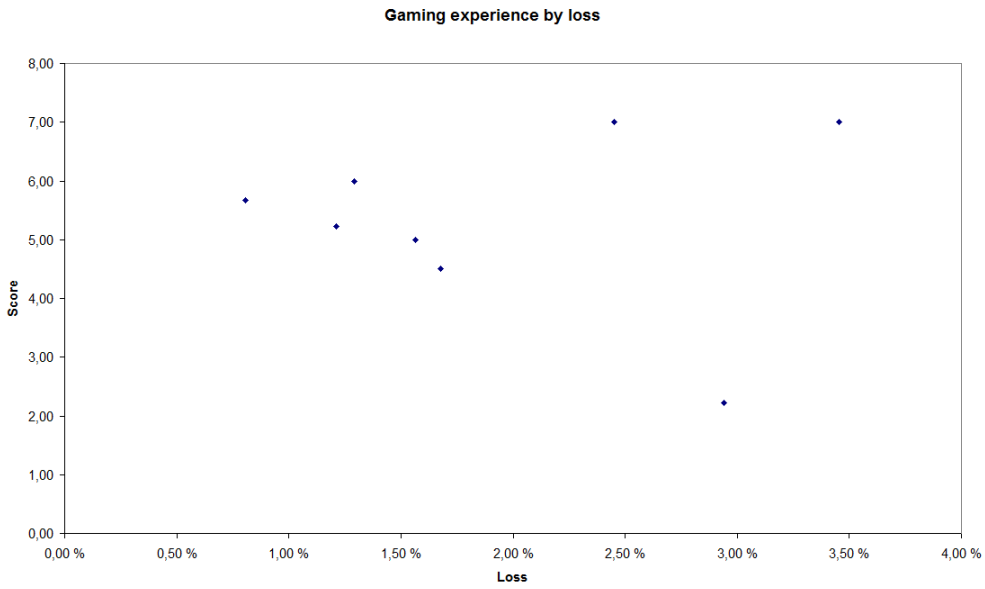

Figure 13.3: The average score by loss

Furthermore, average score is showed compared to measured network loss in a scatter diagram in Figure 13.3. It can be seen that there is an overall trend in that higher loss results in lower test scores, however there are two exceptions at test scores of around 7.0. By examining the two subjects closer, it can be seen that they both have low average latencies and by looking at their individual server logs it is revealed that most of the loss that is detected arrived as bigger sets in occasional fashion, commonly referred to as spiked. This type of loss is arguably experienced as very different from more constant

loss, or in other words that losing every 50th frame is probably worse than losing 10 frames every 500th frame. With this, it is difficult to confirm the validity of the loss measurement.

13.2.4 Game Classes

Figure 13.4 shows that different game titles indeed are affected differently by quality of service that is provided with the service. FarCry is by far the worst performer of the three with an average score of 3.7 while Black & White 2 and Trackmania Nations Forever get 6.0 and 6.6, respectively. Furthermore, all test subjects ranked FarCry as the title giving the worst gaming experience compared to the others.

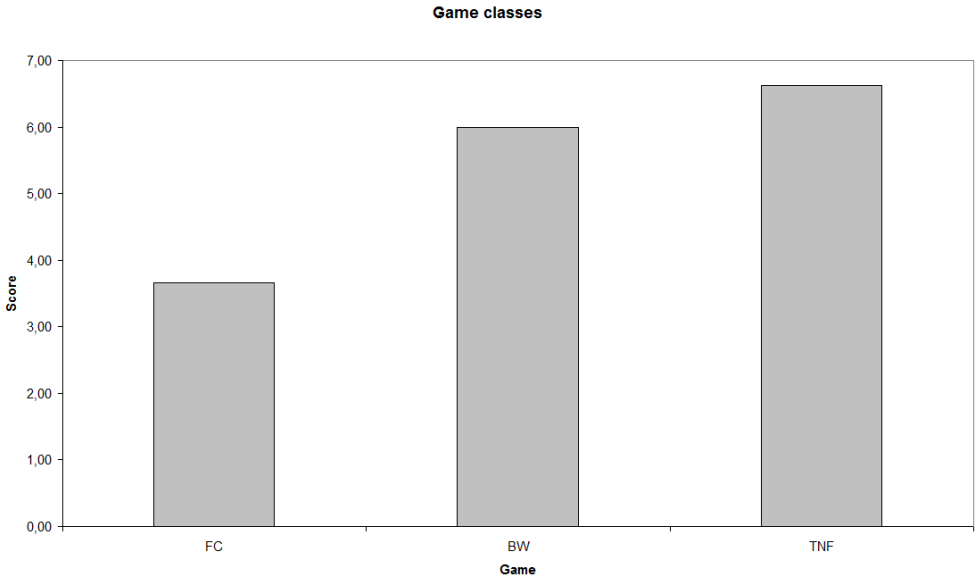

Figure 13.4: The average score by game class

13.2.5 Latency

Figure 13.5 shows a scatter diagram of average scores, i.e. the average score of the three game titles for each subject, placed according to measured latency. With exceptions, it can be observed that there is a general trend that lower latency gives better test scores than a higher latency.

13.3 Service

Are scores in this section are on a scale from 0 to 5 where 5 is higher, i.e. agrees with the statement the most.

13.3.1 Statement 1

The answers to statement 1 in Figure 13.6 show that video is a bigger concern for users than audio. All test subjects agreed on this. Furthermore, between video quality and video latency, users are more concerned about the latter. Only one test subject stated that video quality was more of a concern than video delay.

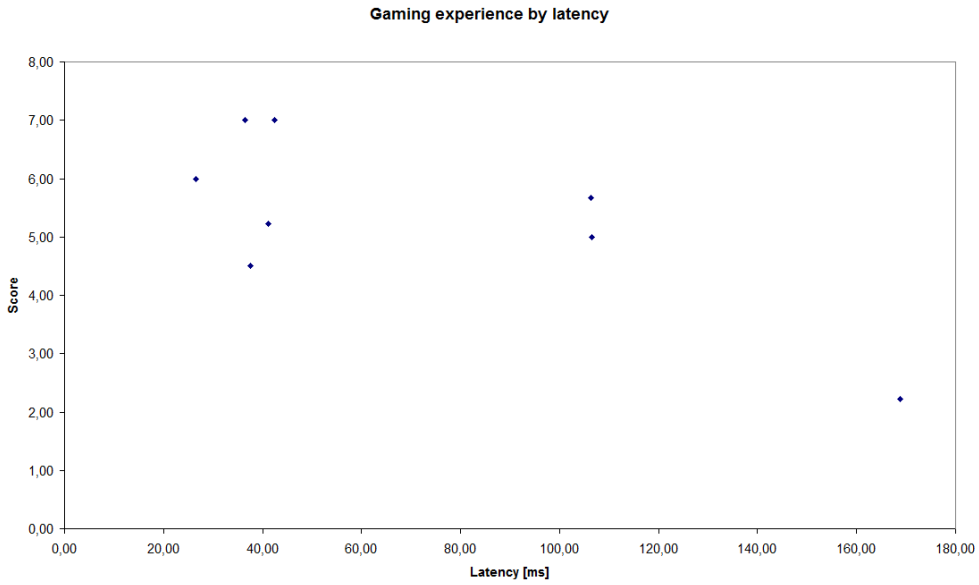

Figure 13.5: The average score by latency

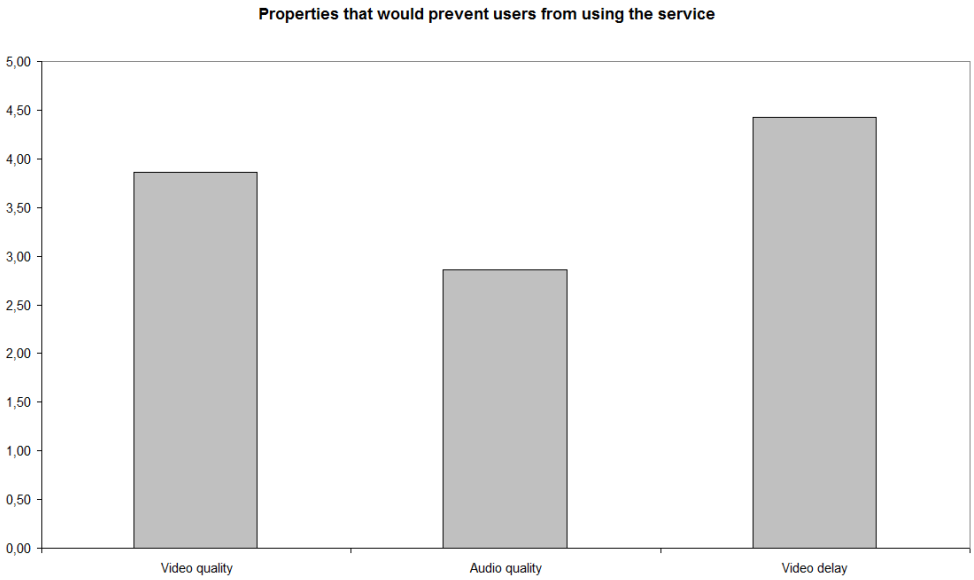

Figure 13.6: Statement 1: Properties that would prevent users from using the service

13.3.2 Statement 2

The results from statement 2, depicted in Figure 13.7, showed that the primary reason was the ease of installation, but with all 3 options scoring 4.0 or higher on average. Platform independence came in second, however two test subjects ranked lowered hardware requirements higher.

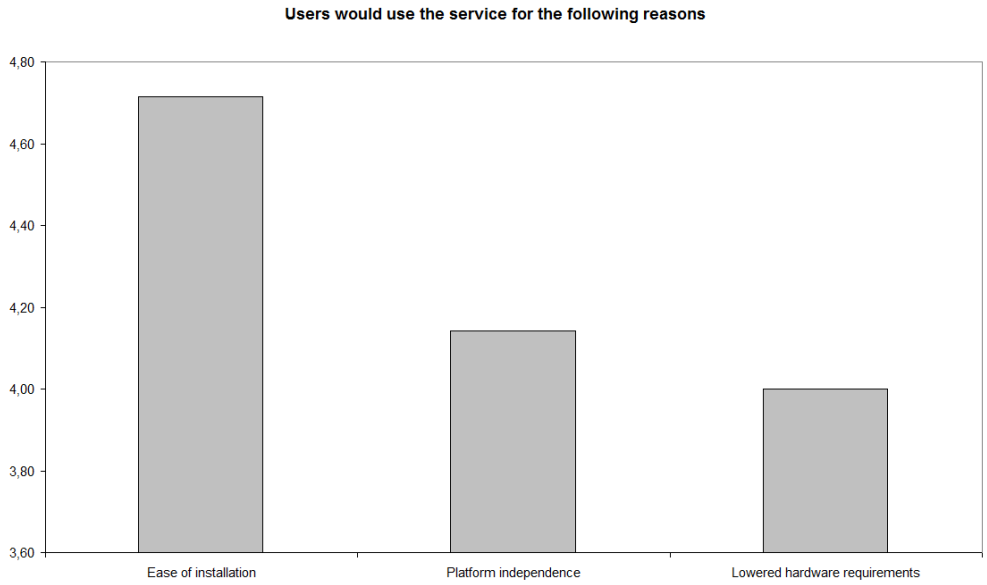

Figure 13.7: Statement 2: Users would use the service for the following reasons

In general, all options seemed to be good reasons for test subjects to use the service.

13.3.3 Statement 3

The results from statement 3 can be seen in Figure 13.8. It should be observed that they are in harmony with the results of statement 2, particularly by observing the three rightmost bars in the figure.

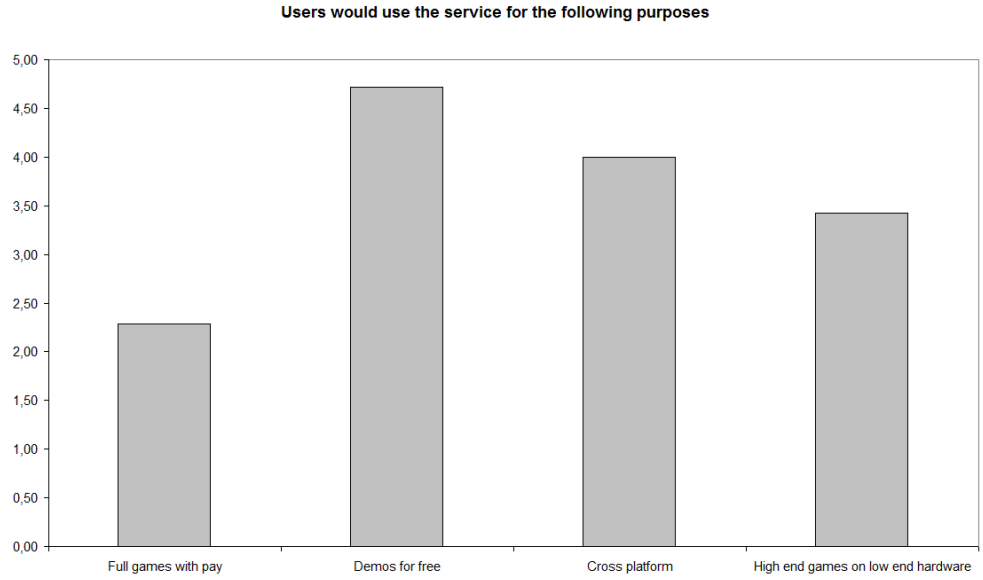

Figure 13.8: Statement 3: Users would use the service for the following purposes

What is more interesting is comparing the two leftmost bars in Figure 13.8. They imply that while test subjects see much potential in using the service, they are more reluctant to the option of paying for it, i.e. commercialisation.

13.4 Performance Tests

The test cases for the performance tests were presented in Section 12.3.3 and are listed in Table 12.2.

13.4.1 Decoding and Rendering

Figure 13.9 shows the induced delay for decoding and rendering organised by the different test cases. Induced delay here refers to the number of milliseconds that is spent doing both tasks on each frame.

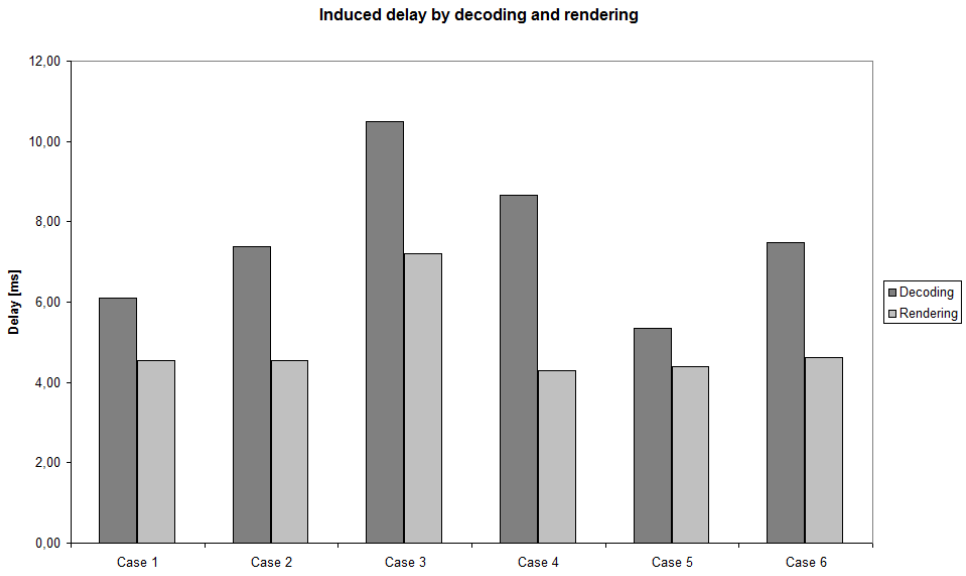

Figure 13.9: The induced delay for decoding and rendering according to performance test case

It should be observed that the average time spent rendering a frame is very similar for all test cases except case 3, which differed from the others in that it had a higher resolution - 1024 x 768 instead of 800 x 600. In fact, the proportional delay is almost identical with the ratio of the two resolutions: Case 3 induced 58 % more delay than the average of the other cases and the 1024 x 768 resolution contains 63 % more pixels than 800 x 600. This was similar for the decoder, as well.

The delay induced by decoding was for all cases higher than that of rendering, however with greater variations in the results. Case 2 - with a higher video bitrate - took 21 % more time than case 1. This is additional to the extra delay that is induced to higher bitrates from transmission delay. Second to case 3, case 4 - with 5 frames per second - took the longest time of all being 42 % higher than case 1. With this, we have that decoding-induced delay is not only dependent on the amount of pixels but also on the data size that needs to be decoding, which essentially is 4 times higher per frame for case 4 than case 1.

It was interesting to observe that decoding-induced delay differed for the three different game titles. Trackmania Nations induced most delay, followed by FarCry on second and Black % White 2 on third. Case 6 (TNF) was 22 % higher than case 1 (FC) while case 5 (BW2) was 13 % lower. At first, these results actually created doubts whether or not the implemented test functions were accurate, and as

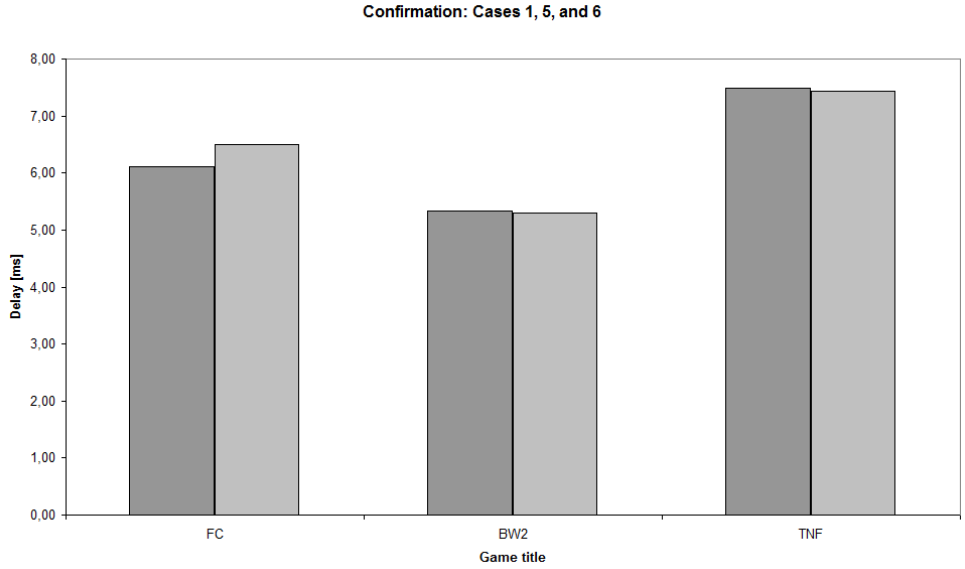

Figure 13.10: Repeated tests for cases 1, 5, and 6

such the tests for cases 1, 5, and 6 were repeated. As can be seen in Figure 13.10, the repeated tests confirm the original results. One possible reason for these results lies in the visual characteristics of the games - an explanation that has basis in Section 2.2.1 that explained how some compression is achieved by using delta frames that represent the difference between other frames. For BW2, the main view of the game does not move but is static while the mouse cursor and game units move around in a more 2D-like manner. In FC the player moves around more, and whenever the player moves his mouse the full view will move and not just the cursor. The colours, however, are relatively monotonous in accordance with the environment that the player is in. The colours brown, green, and blue dominate the game and represent sand and gravel, bushes and trees, and water and sky, respectively. TNF is not only fast paced but also uses a somewhat unrealistic palette of colours in high contrast. As such, one frame in TNF differs more from the previous in comparison to FC and BW2.

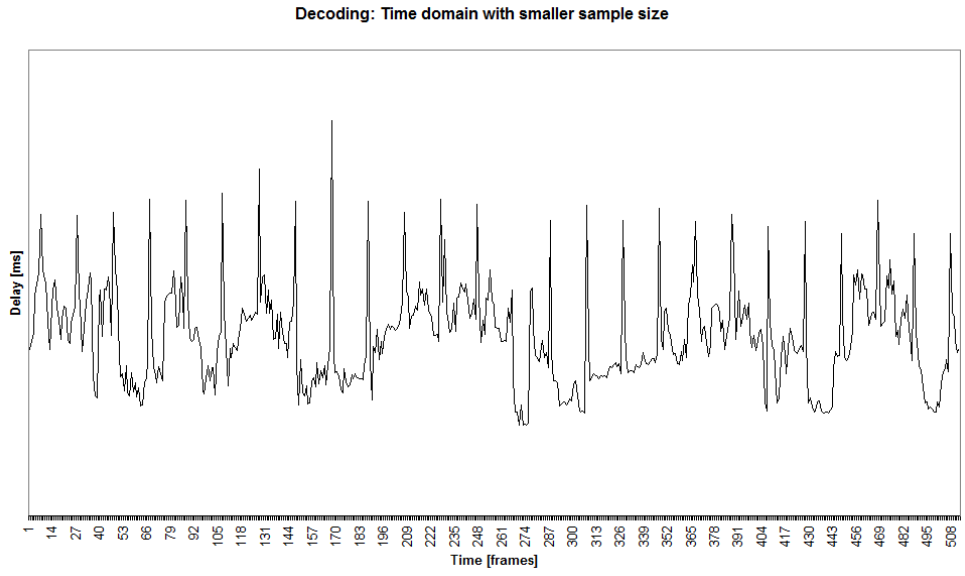

Figure 13.11: Decoding-induced delay shown in the time domain with small sample size

Data from decoding and rendering in the time domain was also attempted analysed and is partly shown in Figure 13.11. However, for such small time periods and with the implemented testing functions based

on the system clock, the frame-by-frame values are deemed too inaccurate for detailed analysis. The periodic spikes, however, are likely due to the index frames.

13.4.2 Input Handling

Both input frequency and input-induced delay was measured for cases 1,5, and 6. Input event frequency was in harmony with expectations in that it was much lower for TNF than the two other games. FC and BW2 registered around 84 and 114 events per second, respectively, and TNF registered 3. This is because FC and BW2 use the mouse extensively while TNF only uses the keyboard as a source of input.

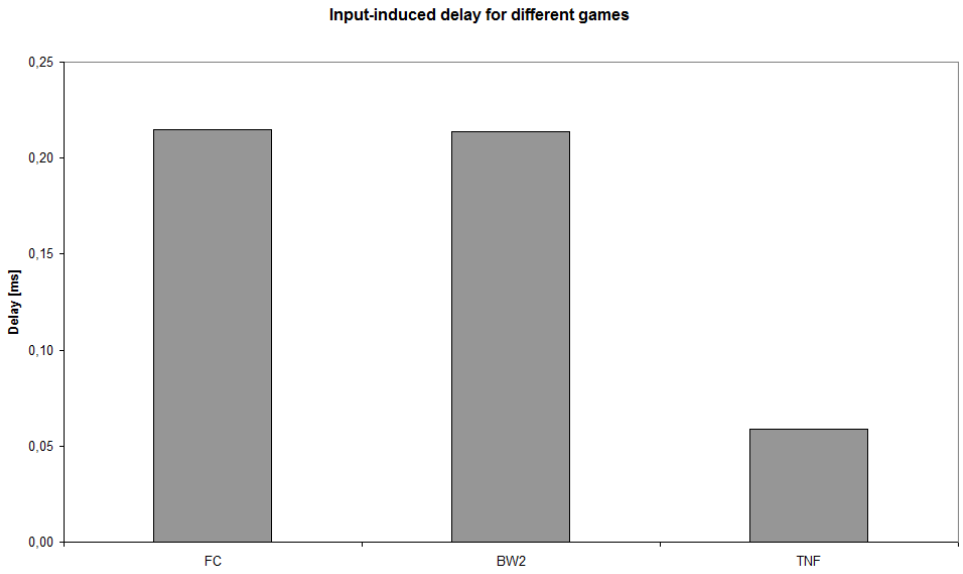

Figure 13.12: Input-induced delay for different games

Furthermore, the induced delay for each input event on average is shown in Figure 13.12 categorised by game. Here, it is interesting to observe that delay is lower for TNF than the others. This is due to the different handling of mouse motion events and key press events. By reviewing the source code it was found that mouse motion events need to do multiple calls to the SDL library in order to relative mouse movement and to set the mouse cursor directly if the input is grabbed.

13.4.3 Processing Share

Processing share times were measured for cases 1, 5, and 6. Combined, the three measured subtasks of the client dominated its total execution time, as was expected. As can be seen in Figure 13.13, decoding and rendering on average take up 43 % and 31 % of the CPU time, respectively, in harmony with expectations and the previous results (as they should). As was found in the previous section, the CPU time of the input handler depended on the game class, varying from 13 % for the RTS game to 0.08 % for the AR game.

On average, the three subtasks took 81.50 % of the total execution time, meaning that 18.50 % of the time was spent on tasks not explicitly measured during performance tests. This time will primarily consist of client iterations when no input or media needs processing - controlled by a timer - and smaller subtasks such as packet assembly and logging.

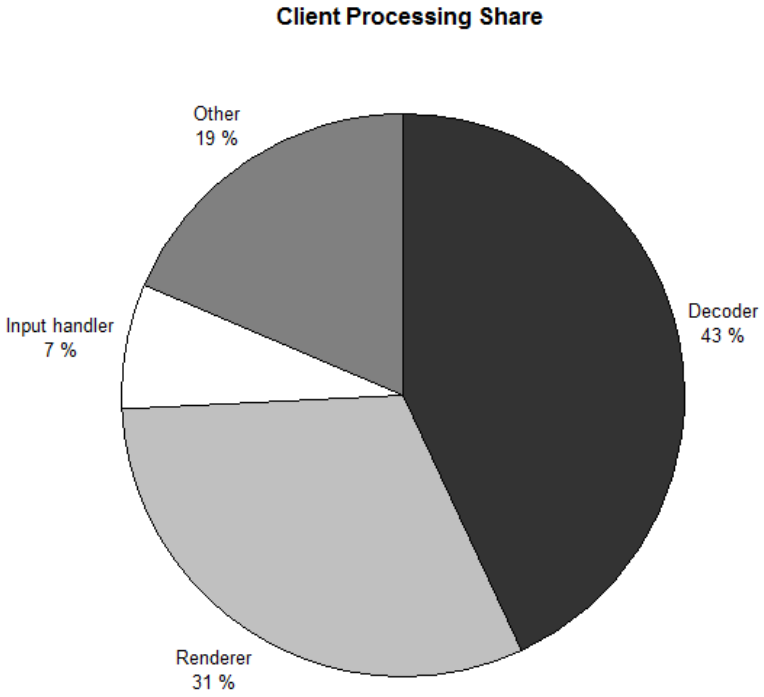

Figure 13.13: Distributed execution time for the client over various subtasks

13.4.4 Network Traffic Analysis

The results from the cases 1 to 6 are listed in Table 13.1 with measured average bandwidth (in Mbps), packet frequency (in packets / sec), and average packet size (in bytes).

Test Case	Bitrate	Frequency	Packet size
Case 1	0.494	103.7	595.9
Case 2	0.663	117.2	796.4
Case 3	0.518	93.6	692.4
Case 4	0.632	110.0	718.0
Case 5	0.394	94.8	518.7
Case 6	0.902	104.4	1079.8

Table 13.1: Performance test cases with network traffic statistics

The most interesting thing to observe from the table is that measured bitrates do not match the bitrates that were set respective to each case. This particularly goes for cases 2 and 6. Case 2 was set to have a bitrate of 1 Mbps, however the measurements show a real bitrate of only 0.663 Mbps. All other cases were set to have a bitrate of 512 Kbps, but Case 6 - running TNF - was in fact sending video at 0.902 Mbps during tests. These results deviate from the specifications and were investigated further.

The client requested and was confirmed the correct video bitrates, so the flaw must reside in the server. It was found that the error lies in the server's interpretation of framerate, not strictly enforcing the session framerate but rather using it to set the audio samplesize and capturing video interleaved with the sampling of audio. In effect, the output framerate is not accurate. This is confirmed by the measurements also by comparing cases 1 and 4, running the same session only with framerates of 20 and 5 frames per second, respectively. Case 4 is in fact sending more packets per second than Case 1,

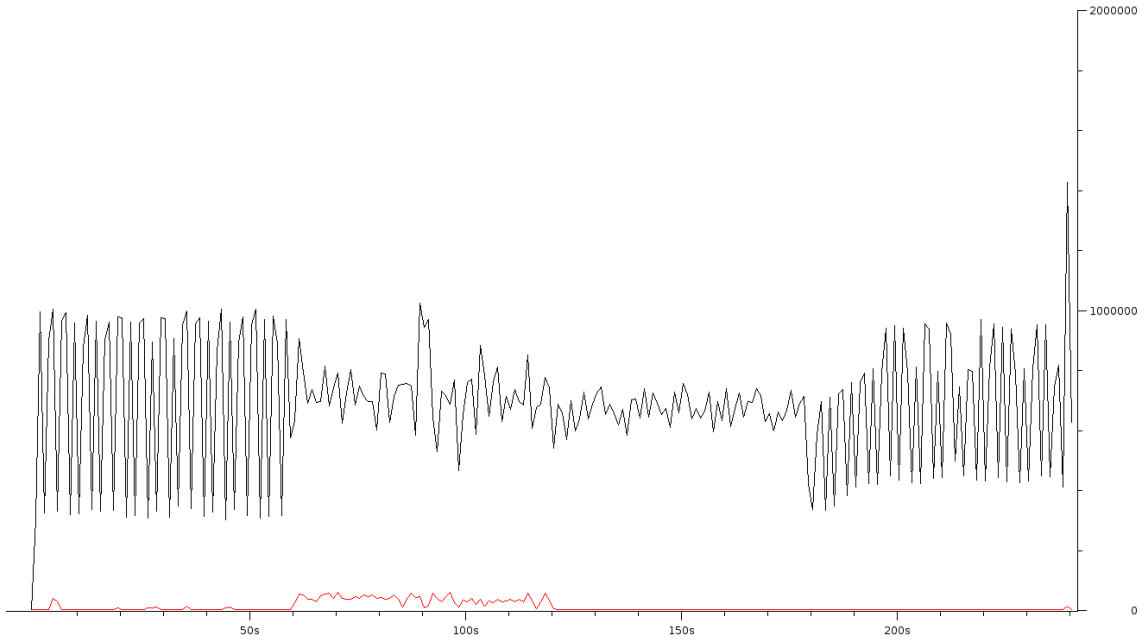

Figure 13.14: Case 7: Time domain by measured bitrate with ingoing data (black) and outgoing data (red)

which is not in harmony with projections.

The results from case 7, i.e. the scheduled behaviour test, are represented according by the measured bitrate in Figure 13.14 and by measured packets per second in Figure 13.15.

It can be observed how both figures vary in 60 second segments, according to the duration of the test steps. During step 1, while navigating through the menus, it is observed that input packets are sent sporadically, i.e. when the mouse is moved. The video stream seem to jump from about 200 Kbps to 1000 Kbps, likely due to delta frames of small size (little change) and periodic index frames. During step 2, while in-game moving and looking around, input data packets jumps as the mouse is moved continuously with a bitrate peaking at around 50 Kbps. Video stream is now much more constant at around 600 - 1000 Kbps since that delta frames are of larger size (more change) than before. During step 3, while in-game with no mouse movement, input data is almost nonexistent due to the lower frequency of keyboard button presses and releases. The video stream looks unchanged. During step 4, while in-game but standing still, no input data is seen aside from echo packets and the video stream appears similar to that of step 1.

13.5 Revisiting Sources of Error

Having identified possible sources of error as part of developing the test plan (see Section 12.4), it is now - following the presentation of the results and analysis - purposeful to revisit them.

Test subjects: It is deemed that the test results presented are lacking due to the low number of test subjects that participated. In total, only 8 subjects participated. Furthermore, it is observed that the subjects are of:

- same age and gender: All males 23-25 years old
- same background: All graduate students within computing science related fields
- same gaming experience: All subjects have extensive experience with playing computer games

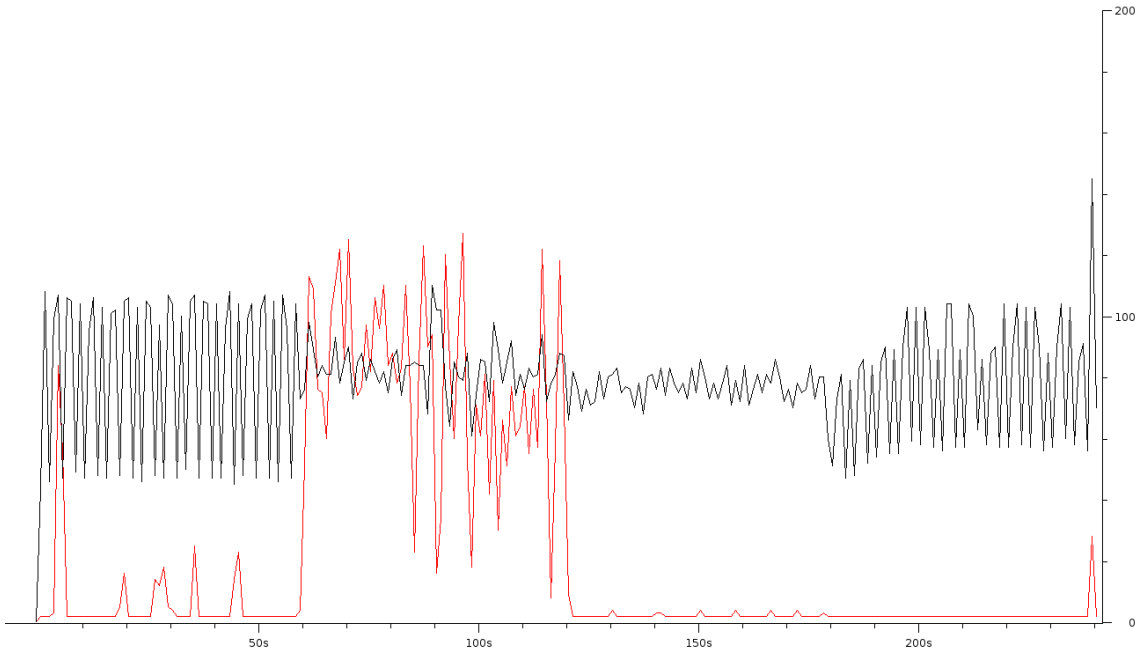

Figure 13.15: Case 7: Time domain by packets per second with ingoing data (black) and outgoing data (red)

As such, the subject population cannot be deemed representative and the lack of a sufficiently sized population is particularly unfavourable for subjective testing.

Test conditions: The randomness of the test conditions of subjects taking the test in their homes is desirable due to the realism of it but is a problem when our subject population is so low. For instance, we only have one test subject with an average latency over 160 ms, who - in harmony with projections - gave low test scores. There is a chance that his test session was degraded by other factors, e.g. ambient lighting or faulty hardware, which affected the gaming experience more than the latency. If all tests were carried out on the same computer in the same room, this could have been ruled out.

However, we find that with the basis for our projections combined with matching results, there are grounds for deeming it unlikely that other factors came into play in an extent high enough to invalidate the scores.

Game preference: Figure 13.4 shows Trackmania Nations and Black & White 2 outperforming FarCry. Coincidentally, this is in contrast to reviews of the three titles found on the most popular online gaming magazines [56] [57]. While this in no way confirms it, it implies that test subject game preferences did not have a big impact on test scores.

Questionnaire: Reading the subject comments on question when they were in doubt was usually enough to confirm that the subject understood the question correctly. Furthermore, ambiguity was to a large extent already mitigated from the pilot testing.

Implementation: The `StopWatch` class was tested and compared with other clocks and was found to be accurate for longer intervals. For shorter intervals down to a few milliseconds, however, its accuracy has not been confirmed. Further testing should be carried out.

It was also discovered from the network traffic analysis that the server had a design issue that affected its ability to send video at the bitrate agreed upon in session initiation. This might invalidate the representation of the test results, since bitrates different than the what was defined in the cases were in fact delivered. What is worse is that bitrates differed between game classes, which potentially invalidates the confirmation that different game classes perform differently. It does explain the results shown

in Figure 13.10, i.e. that the decoding-induced delay differed between game classes, thus invalidating the former analysis of those results. We do, however, argue that this discovery does not invalidate the confirming results that game classes perform differently - for two reasons: 1) For the results shown in Figure 13.4 the subjects were explicitly asked to evaluate the perceived effects of delay on each class and not the overall impression (including video quality) and delay increases with the video bitrate, and 2) Measured bitrates do not correlate with original game class scores.

Part VI

Conclusion

Chapter 14

Conclusion

We have conceptualised and realised a system that enables video games to be played remotely on lightweight computers. Clients supporting the three most popular desktop platforms today have been created. The system has been used to carry out both performance and user testing in a realistic and cross-platform setting, and the results have been analysed and discussed.

In the following we review the 4 research question that were set up in the introduction. We will attempt to answer them by using information and results derived from the work presented in this thesis and the experiences from carrying out the work.

Question 1: *Is it possible to create a service for the remote playing of computer games?*

We have designed and implemented a system that realises the functionality of the proposed service. By itself, this should confirm that it indeed is possible to create such a service. However, whether the work presented here confirms that such a service can be realised with satisfactory user experience is a different matter. Without any notion of application it was from testing found that users on average ranked the service in the upper end of "mediocre" (5.7 / 10) on our scale. While this is below our criteria for what is deemed satisfactory, a strong correlation was shown between test scores and network conditions. As such, we believe that satisfactory test scores can be achieved if the network conditions are right.

Processing requirements were also analysed and were found to be realiseable on relatively light hardware for both the servers and the clients. The client software utilised only 15-20 % of the CPU of a somewhat outdated laptop and will likely run well on even lighter hardware without inducing too much delay. It would have been interesting to run extensive performance tests on even light hardware in order to establish strict minimum requirements for the client software. The server software also ran surprisingly well even on hardware without dedicated graphic adapters. Conclusively, the answer to question 1 is yes, but with strong constraints for the network conditions.

Question 2: *What are the main challenges to creating such a service?*

The primary user experience factors that have been discussed in this project are latency, media quality, and content. The extent to which they are noticeable is determined by processing resources and network conditions, and as was already pointed out from performance tests - processing resources is not a major problem for most hardware setups. Consequently, the bottleneck is in the network. From both a developer's and a user's standpoint this is unfortunate. Neither can do anything about it: Having the client buy a more expensive hardware setup will not help and there is little that can be

done implementation-wise in order to mitigate the effects of loss without adding delay. Furthermore, the underlying network infrastructure is not property of the system and cannot be upgraded easily.

Performance tests also revealed that little delay was added by the client with the tasks of decoding and rendering using on average 10 ms and 5 ms each, respectively. Similarly, input event processing took less than 0.25 ms on average. However, latency was still noticeable even for local connections, implying that much delay is induced by the network and the server. It was highlighted how delay was caused by both network propagation and frame transmission, together contributing to significant delay. The server deals with capturing and encoding raw data concurrently with executing the game. While the performance of the server could not be analysed in detail, the technical discussion and comparison with a similar system expected server-induced delay to be somewhere between 50 and 80 ms - depending on settings and hardware - which is in harmony with the client results and perceived interaction latency by users.

From test results it was observed that users considered delay to be the primary reason for not using the service, as we predicted. Video quality was a close second, however we argue that for a relatively loss free connection with reasonable bandwidth, e.g. higher than 1 Mbps, video quality will be acceptable. For 3 Mbps and up, most users perceived the video quality as in the upper end of "good" (8-9 / 10) and close to indistinguishable from the original source. A higher bandwidth also mitigates transmission latency and thus overall interaction latency.

Conclusively, findings were as expected in that latency is the biggest challenge in creating a remote gaming service with satisfactory user experience. Primary causes for latency are the network and server processing. We argue that the minimum specifications for the enabling network should be at least 3 Mbps of bandwidth with zero or close to zero loss and a one-way latency of less than 25 ms. Server processing times should be lowered by further optimising the implemented server module and upgrade hardware.

Question 3: *Are any games more suitable for such a service than others?*

We attempted to clearly define three different game classes, all with properties that were very characteristic to their class: First-person shooter (FPS), real-time strategy (RTS), and arcade racing (AR). It was found from user testing that these game classes indeed performed differently under conditions that necessarily degraded their perceived quality. The example titles for RTS and AR gave much better test scores than that of FPS, in accordance with our discussion and expectations. AR was on average given close to a twice as high score as FPS. Between AR and RTS, AR performed somewhat better. Furthermore, this helps to confirm that FPS games are the ultimate stress test for services of thin-client nature. What was interesting is that this is in contrast to the performance tests, where the AR game title required more processing time for both decoding and rendering than the others. The various reasons for these results were outlined in the theoretical and analytical discussions.

Question 4: *What are the possible applications for such a service?*

User surveys showed that all three candidate reasons for using the service, i.e. ease of installation, platform independence, and lowered hardware requirements, were attractive to users. However, when asked what purposes the users thought the service could serve, it became clear that they were much more reluctant to paying for such a service to play commercial games (2.3 / 5) than to using the service for free to try game demos (4.7 / 5). In effect, we deem it unrealistic to deploy the service for full commercial use unless interaction latency is much improved.

We conclude that if the core application that was developed was to be extended to a consumer use application, it would most likely be as a distribution technology for game demos. Quality of service

requirements are not as strict for services that are free to use and we further argue that users will have a much more complete impression of a game title compared to game screenshots and video clips. Various advantages over releasing game demos have been highlighted, i.e. instant gameplay with no downloads or installation.

Overview: *How does the realised service correspond with the overall goals of the Geelix project?*

Finally, we revisit the overall goals of the Geelix project, defined in Section 1.2. The research in this thesis was not targeted towards type of content, e.g. violent versus non-violent games, but rather on the perceived characteristics of different games from a technical standpoint. We did however note that content control was one of the possible advantages in this service, defined in Section 4.1. This lies in that a provider could offer not only games of a particular genre, e.g. a subscription for sports games only, but also games of specific content in terms of violence and maturity. Parents could then trust the provider to offer only non-violent games to their household account, thus not requiring them to monitor their children's video game habits.

Making such a service non-profit is possible, however it is difficult to envision a way that the service could be fully free due to the provider-side costs of deployment and maintenance. The user survey showed that as is, subjects were sceptical towards paying for such a service. On this premise, improvements in terms of delay optimisation and stability are required for commercial deployment - profit or non-profit - to be viable.

Chapter 15

Further Work

15.1 Implementation

While the realised system is fully functional as a core application (see Section 4.2.5), some work remains for optimisation and completion.

Audio synchronisation: As was discussed in reviewing the implementation, no real efforts were made to synchronise the streamed audio with the video. In effect, the audio stream is lagging behind the video stream on the client side. Solving this necessarily involves delaying the video rendering on the client, however the delay time can also mitigated by shortening the audio samplesize.

Video codec: Due to licensing and patenting restrictions, a relatively old and outdated video codec, WMV8, was used. Newer codecs employing more efficient compression algorithms have since been created and would likely improve the perceived quality of service if employed here. One that has been mentioned already is the x264 H.264/AVC codec that was also used in the THiNC project (see Section 3.2.4).

Protocol feedback: In the current version, video and audio settings requested and confirmed in during session initiation are kept constant throughout the entire session. Network conditions vary and so should the session parameters. This could be enabled through protocol feedback from the client to the server, e.g. reporting increased loss likely as a result of congestion, enabling the server to react accordingly, e.g. decrease video bitrate. Furthermore, it would be of interest to the user to have some sort of functionality auto-detecting the optimal settings at session initiation, e.g. by running quick tests upon connection. This way, the user would always get the best possible experience from any given conditions.

OpenGL support: As of now, the server is only capable of grabbing graphics and injecting input from / to games running on DirectX 9.0. This leaves many games out in the cold, and it would be desirable to also support games on other platforms - primarily OpenGL. From the author's experience with the SDL library, it would be trivial to also create a similar server module for SDL-enabled games.

Stability: At this stage the software still contains many bugs to the annoyance of users, providers, and testers. They should be fixed.

15.2 Research

Possible sources of error were discussed in the test analysis and should be used as input in a new round of testing in order to strengthen the research-related value of this project. More specifically, larger scale testing should be done in quantitative fashion. This will generate much more precise numbers and allow the results and analysis to be applied on a general basis. Based on the outcome of this, a study in the possible business models of the general service should be carried out.

Performance testing of the client's loss resilience was not carried out due to time limitations. This testing is of interest, particularly in terms of potential correlation with session packet size settings (MTUs), and should take place later.

During the project, additional areas of research became of interest to the author, but that were deemed beyond the scope of this thesis. Examples of such areas are codec optimisation for game videos, in-depth analysis of the temporal redundancy in different video content, and more research on how human perception reacts to visual delay.

15.3 Commercialisation

In the final stages of this project the author discovered that a similar service had been released[1], named *StreamMyGame* (SMG) [58]. While the technical details of SMG are not available, it seems to realise the application here defined as "Released Software" in Section 4.2.3 where the users are also their own providers. Windows, Linux, and PlayStation 3 (PS3) are supported - however the PS3 version is essentially just a Linux version since it assumes that the PS3 is running a version of Linux called Yellow Dog Linux (YDL) [59]. Our client does not currently support YDL, however we argue that it would be trivial to do so - in which case we could add PS3 to the supported platforms. The SMG service focuses on the portability and lightweight advantages of remote gaming, however Mac OS X does not appear to be supported. The business model of SMG seems to be based on releasing a limited version of the software for free, i.e. limited to a 640 x 480 resolution, and offering two higher scale versions, namely "HDTV" and "Unlimited", against payment. The free version includes adverts.

SMG does not realise all of the potential in remote gaming but confirms that there is commercial and consumer use interest in these types of services. It has received positive reviews and is praised both for allowing Linux users to play Windows games in their homes and for allowing the new generation of ultraportable computers, such as the Linux-enabled ASUS Eee PC, to play high-end games.

[1] The author was made aware of StreamMyGame on June 3rd 2007.

Bibliography

[1] Andrew S. Tanenbaum. *Computer Networks*. Pearson Education Inc., 2003.

[2] Yoav Tzruya, Alex Shani, Francesco Bellotti, and Audrius Jurgelionis. Games@Large - a new platform for pervasive entertainment. 2006.

[3] The VirtualGL Project. VirtualGL - about. `http://www.virtualgl.org/About/Background`. Accessed 7 February 2008.

[4] Gridmedia Technologies. Welcome to Geelix - LiveGames download. `http://www.geelix.com/downloads2a.html`. Accessed: 21 June 2008.

[5] MSN Money. Video-game sales overtaking music. `http://articles.moneycentral.msn.com/Investing/Extra/VideoGameSalesOvertakingMusic.aspx`. Accessed: 28 June 2008.

[6] GameDaily. Video games explode: Global revenues now on par with box office. `http://www.gamedaily.com/articles/news/video-games-explode-global-revenues_now-on-par-with-box-office/?biz=1`. Accessed: 28 June 2008.

[7] Gridmedia Technologies. Welcome to Geelix. `http://www.geelix.com/`. Accessed: 21 June 2008.

[8] Frederik Norberg. Remote game play for broadband IP networks. Technical report, Norwegian University of Technology and Science, June 2008.

[9] Ole-Ivar Holthe, Ola Mogstad, and Leif Arne Rønningen. Geelix LiveGames: Remote playing of video games. Pending ACM approval, June 2008.

[10] Research Methods Knowledge Base. Structure of research. `http://www.socialresearchmethods.net/kb/strucres.php`. Accessed: 28 June 2008.

[11] Mrinal Kr. Mandal. *Multimedia Signals And systems*. Kluwer Academic Publishers, 2003.

[12] Dmitriy Vatolin et. al. MPEG-4 AVC/H.264 video codecs comparison. Graphics Media Lab Video Group, November 2006.

[13] Tilemachos Doukoglou, Stelios Androulidakis, and Dimitrios Kagklis. Subjective video codec evaluation for streaming services up to 1 Mbps. In Z. Mammeri and P. Lorenz, editors, *HSNMC 2004*. Springer-Verlag Berlin Heidelberg, 2004.

[14] Daniel Sykora, Jan Burianek, and Jiri Zara. Video codec for classical cartoon animations with hardware accelerated playback. In *Advances in Visual Computing*, pages 43–50, 2005.

[15] William Stallings. *Operating Systems: Internals and design principles*. Pearson Education International, 2005.

[16] DesktopLinux.com. 2007 desktop Linux market survey. `http://www.desktoplinux.com/cgi-bin/survey/survey.cgi?view=archive&id=0813200712407`. Accessed 8 July 2008.

[17] Net Applications. Operating systems market share for June 2008. `http://marketshare.hitslink.com/report.aspx?qprid=8#`. Accessed 8 July 2008.

[18] W3Counter. Operating systems market shares for June 2008. `http://www.w3counter.com/globalstats.php`. Accessed 8 July 2008.

[19] XiTi Monitor. Operating systems market shares for April 2008. `http://www.xitimonitor.com/en-us/internet-users-equipment/operating-systems-april-2008/index-1-2-7-129.html`. Accessed 8 July 2008.

[20] Entertainment Software Association. 2008: Essential facts about the computer and video game industry. Annual report.

[21] Wendy Jones. *Beginning DirectX 10 Game Programming*. Thomson Course Technology, 2008.

[22] Fabrice Bellard. FFmpeg. `http://ffmpeg.mplayerhq.hu/`. Accessed: 12 April 2008.

[23] Stephen Dranger. How to write a video player in less than 1000 lines. `http://www.dranger.com/ffmpeg/`. Accessed: 12 April 2008.

[24] SDL Team. Simple DirectMedia Layer. `http://www.libsdl.org/index.php`. Accessed: 12 April 2008.

[25] Tristan Richardson. The RFB protocol version 3.8. Technical report, RealVNC Ltd, 2007.

[26] Jason Nieh, S. Jae Yang, and Naomi Novik. A comparison of thin-client computing architectures. Technical report, Columbia University, 2000.

[27] Valve Corporation. Welcome to steam. `http://www.steampowered.com/v/index.php`. Accessed 7 February 2008.

[28] Frederick W.B. Li, Rynson W.H. Lau, and Danny Kilis. GameOD: An internet based game-on-demand framework. In *Proceedings of the ACM symposium on virtual reality software and technology*, 2004.

[29] G-Cluster. Games on demand streaming games for IPTV, PC and mobile. 2007.

[30] Games@Large. Games@Large home. `http://www.gamesatlarge.eu/`. Accessed 7 February 2008.

[31] Audrius Jurgelionis, Francesco Bellotti, Peter Eisert, and Jukka-Pekka Laulajainen. Testing and evaluation of new platforms for delivery of distributed gaming and multimedia. Games@Large, 2004.

[32] D. De Winter, P. Simoens, and L. Deboosere. A hybrid thin-client protocol for multimedia streaming and interactive gaming applications. In *ACM NOSSDAV 06 Newport, Rhode Island, USA*, 2006.

[33] Gamer.no. Demoer - Gamer.no (Norwegian). `http://www.gamer.no/demoer/?s=50`. Accessed: 6 June 2008.

[34] Joseph D. Pellegrino and Constantinos Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *Proceedings of 4th ACM SIGCOMM workshop on network and system support for games*, 2003.

[35] Valve Corporation and Yahn W. Bernier. Latency compensating methods in client/server in-game protocol design and optimization. `http://developer.valvesoftware.com/wiki/Lag_Compensation`. Accessed 7 February 2008.

[36] Matthias Dick, Oliver Wellnitz, and Lars Wolf. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM workshop on network and system support for games*, 2005.

[37] Grenville J. Armitage. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, pages 137–141, 2003.

[38] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In *ACM SIGCOMM04 Workshops*, 2004.

[39] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in Warcraft III. In *Proceedings of the 2nd workshop on network and system support for games*, pages 3–14, 2003.

[40] Michael Claypool, David LaPoint, and Josh Winslow. Network analysis of Counter-Strike and Starcraft. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pages 261–268, 2003.

[41] Gary Levin. CBS to air Grammys with up to 5-minute delay. `http://www.usatoday.com/life/television/news/2004-02-03-grammy-precautions_x.htm`. Accessed: 20 March 2008.

[42] Amazon. Amazon.com: Online shopping for electronics, apparel, computers, books, dvds more. `http://www.amazon.com/gp/homepage.html`. Accessed: 2 June 2008.

[43] Chris Schoeneman et. al. Synergy. `http://synergy2.sourceforge.net/`. Accessed: 16 February 2008.

[44] Shunra. Network simulation, network emulation. `http://www.shunra.com/`. Accessed: 16 February 2008.

[45] Audrius Jurgelionis, Francesco Bellotti, Wijnand Ijsselsteijn, and Yvonne de Kort. Evaluation and testing methodology for evolving entertainment systems. In *ACM Conference 04*, 2004.

[46] Sander Vorren. Subjective quality evaluation of the effect of packet loss in high-definition video. Master's thesis, Norwegian University of Science and Technology, 2006.

[47] ITU-T. ITU-T recommendation P.910: Subjective video quality assessment methods for multimedia applications. ITU telecommunication standardization sector, 1999.

[48] Eric J. Braude. *Software Engineering: An Object-Oriented Perspective*. John Wiley & Sons, Inc, 2001.

[49] Rolv Bræk and Øystein Haugen. *Engineering Real Time Systems: An Object Oriented Methodology using SDL*. Prentice Hall, 1993.

[50] Rolv Bræk. On methodology using the ITU-T languages and UML. *Telektronikk 4*, 2000.

[51] GTK+ Team. GTK+ - about. `http://www.gtk.org/`. Accessed: 23 April 2008.

[52] Free Software Foundation. GNU General Public License. `http://www.gnu.org/licenses/gpl-3.0.txt`. Accessed: 04 July 2008.

[53] Free Software Foundation. GNU Lesser General Public License. `http://www.gnu.org/licenses/lgpl-3.0.txt`. Accessed: 04 July 2008.

[54] VideoLAN. VideoLAN - x264. `http://www.videolan.org/developers/x264.html`. Accessed: 5 March 2008.

[55] Gerald Combs. Wireshark: Go deep. `http://www.wireshark.org/`. Accessed: 28 June 2008.

[56] IGN. IGN: FarCry. `http://pc.ign.com/objects/482/482383.html`. Accessed: 21 June 2008.

[57] Gamespot.com. FarCry for PC review. `http://www.gamespot.com/pc/action/farcry/review.html`. Accessed: 21 June 2008.

[58] StreamMyGame. Streammygame.com. `http://www.streammygame.com`. Accessed: 03 June 2008.

[59] Terra Soft Solutions. Yellow Dog Linux. `http://www.terrasoftsolutions.com/products/ydl/`. Accessed: 03 June 2008.

Appendix A

User Scenarios

Use cases are essentially user scenarios. A subset of the use cases from Figure 9.1 are described in text - commonly referred to as *textual* use cases - in the following:

Use case name	Play a game
Actor	User
Event path	1: User connects to the server 2: User interacts with the game 3: User disconnects from server
Alternative path	1: User modifies settings (UC 2) 2: User connects ...
Exceptional path	1: User connects to the server 2: Server is not available (busy, fault)
Trigger	The user wants to play a game.
Pre condition	Client application is installed and running. Server is running.
Post condition	Server logged session data.

Table A.1: User scenario

Use case name	Retrieve statistics data
Actor	Tester
Event path	1: Start server 2: Carry out testing, i.e. user sessions 3: Collect server logs 4: Stop server
Alternative path	
Exceptional path	
Trigger	The tester needs test data.
Pre condition	Available client and server hardware.
Post condition	Test data has been retrieved.

Table A.2: Tester scenario

Appendix B

Message Sequence Charts

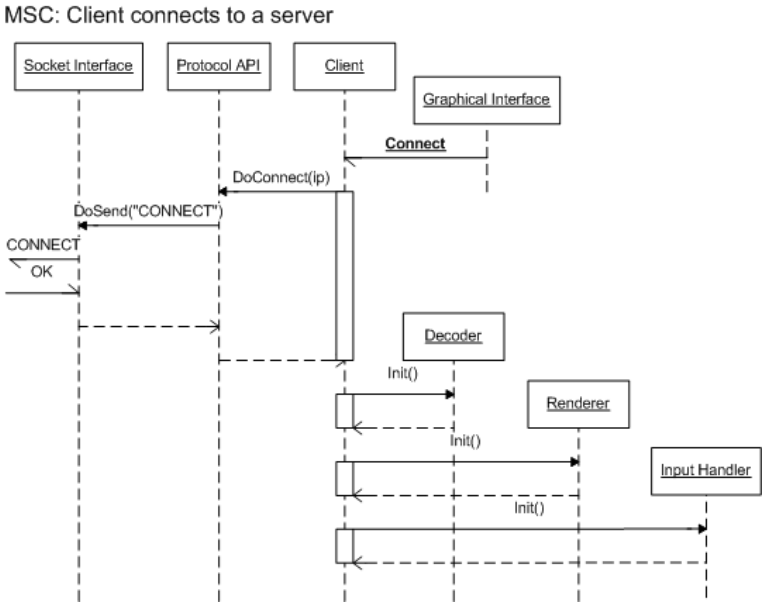

Figure B.1: MSC: Client connecting to a server

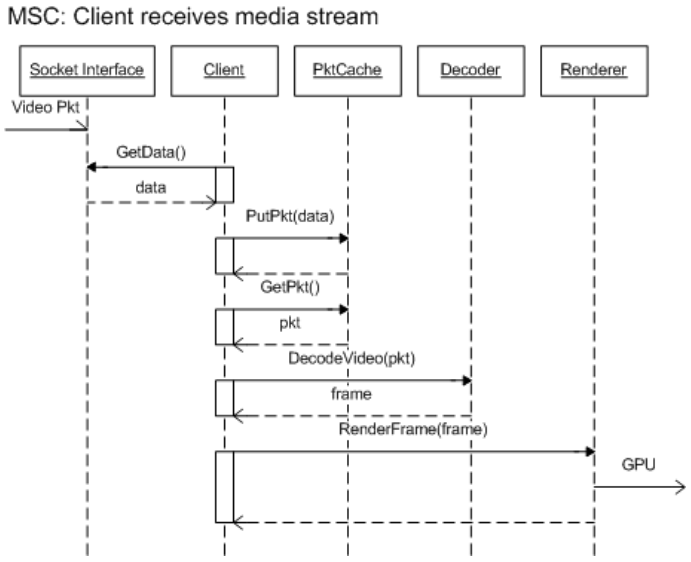

Figure B.2: MSC: Client receiving a video packet

Appendix C

Deployment and Distribution

In summing up previous notions on how the system is deployed, Figure C.1 shows an overview of the various files that make up a working system. Additional files as part of third party libraries are not included, e.g. FFmpeg and SDL.

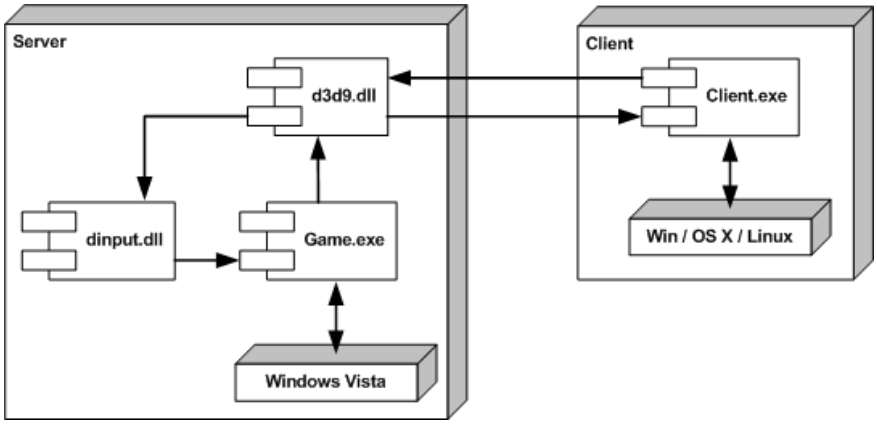

Figure C.1: System deployment

Distributing cross-platform software is a non-trivial task since different approaches apply for different operating systems (see Section 2.3.1). Furthermore, while the Windows and Mac OS X operating systems are available in versions that are developed by the same organisation and thus are compatible with other versions of the same operating systems - at least in terms of file formats - Linux is available in a whole range of versions developed by different people - referred to as distributions or *distros*.

Before deciding how to distribute software, one must decide if it is better to distribute pre-compiled binary files, i.e. as executable files often bundled with various resource files, or as source code that needs to be compiled, which is more common with Linux software than the other operating systems. The former is usually preferred for two reasons: 1) Unless it is an open-source project the developers will usually be hesitant to release the source code, and 2) It is more user friendly since it requires no compiling environment to be set up. Consequently, we wish to release the client software as binary, pre-compiled files.

Windows: Software is typically distributed either as executable self-extracing (.exe) files or as compressed archives containing the file tree - usually zip files (.zip).

Mac OS X: Applications are distributed primarily as compressed disk image (.dmg) files that essentially are very similar to compressed archives.

Linux: Different Linux distros prefer different software package formats. Debian and its derivatives,

Platform	**Version**	**File type**
Windows	XP, Vista	.exe
Mac OS X	10.4, 10.5	.dmg
Linux	Ubuntu 7.04, 7.10, 8.04	.deb
Linux	Debian 4.0	.deb
Linux	OpenSUSE 10.3	.rpm

Table C.1: The file types used for distributing the client software for the various operating systems

e.g. Ubuntu, all use .deb files that consist of a compressed archive with the files to install as well as control files. The control files allow us to list dependencies that must exist on the system before the the distributed software will work. Red Hat and its derivatives, e.g. OpenSUSE, all use .rpm files. These files are structured somewhat differently from the .deb files but behave in the very same way.

Table C.1 gives an overview over the different operating systems and the packaging file type that was used for the client software. It should be noted that source code is essentially the same for all Linux distros and the only difference is linking and small variations in the versions of external libraries - depending on what version that is in the repository of that distro. As such, it will in most cases be a trivial task to release the software for currently non-supported distros. Furthermore, it is very possible that the listed distributions of the client software are compatible with additional Linux distros.

Appendix D

Test Questionnaire

Following is the questionnaire that was answered by subjects during user testing. Note that introductory parts explaining the project have been omitted and that formatting is different.

PART 1: Test Subject Info
E-mail: ... (e.g. ola.mogstad@geelix.com)
Platform: ... (e.g. Ubuntu 8.04
Date and time of test: ... (e.g. May 15th 2008 11:42 am)
Location of test: ... (e.g. San Francisco, USA)
Connection information: ... (e.g. ComCast 768/3000)
Hardware information: ... (e.g. Pentium M 1.6 GHz laptop)

PART 2: Implementation Questions
For each question, we ask that you provide a score and a full answer (where possible). For scoring, use a scale from 0 to 10, where 10 is better. 10 is better no matter how the question is phrased - so for instance, if video delay had no effect on the gaming experience whatsoever, the score would be 10.

10: Perfect: No different than playing computer game that executes locally
7-9: Good: Somewhat noticeable different from executing locally, but enjoyable
4-6: Mediocre: Noticeable different from executing locally, but tolerable
1-3: Poor: Very different from executing locally, not acceptable
0: Worst: Extremely different from executing locally, unplayable

Question 1: For an individual game, how did video delay affect the following game aspects?
1.1: FarCry: Movement / walking - Score: _ Answer: ...
1.2: FarCry: Shooting at enemies - Score: _ Answer: ...
1.3: FarCry: Navigating menus - Score: _ Answer: ...
1.4: Black & White 2: General gameplay - Score: _ Answer: ...
1.5: Trackmania Nations: General gameplay- Score: _ Answer: ...

Question 2: How was the gaming experience affected by decreased video quality at different video bitrates?
2.1: At 512 Kbps - Score: _ Answer: ..
2.2: At 1 Mbps - Score: _ Answer: ..
2.3: At 3 Mbps - Score: _ Answer: ..

Question 3: How was the gaming experience affected by audio being out of synch or low quality?
Score: _ Answer: ..

Question 4: How was the gaming experience affected by loss?

4.1: Loss of input data? (keyboard presses, mouse movement, etc) - Score: _ Answer: ..
4.2: Loss of video /audio data? (glitches in audio, distorted video frames, etc) - Score: _ Answer: ..

Question 5: What was your total impression of the remote gaming service?
Score: _ Answer: ..

PART 3: Service Statements
Following is a series of statement about the remote gaming service. Use the scale below to describe to what extent you agree or disagree with each statement.

5: Completely agree
4: Mostly agree
3: Somewhat agree
2: Somewhat disagree
1: Mostly disagree
0: Completely disagree

Statement 1: The following property would prevent me from using the service:
1.1: Video quality - Score: _
1.2: Audio quality - Score: _
1.3: Video delay - Score: _

Statement 2: I would use the service for the following reason:
2.1: Ease of installation - Score: _ (Comment: No installation required after client is installed)
2.2: Platform independence - Score: _ (Comment: Client runs on Windows, Linux, and Mac OS X)
2.3: Lowered hardware requirements - Score: _ (Comment: Light computers can play any high-end game)

Statement 3: I would use the service for the following purposes:
3.1: Playing full games for payment - Score: _
3.2: Trying game demos for free - Score: _
3.3: Playing games on unsupported platforms - Score: _
3.4: Playing high-end games on laptop - Score: _

Appendix E

Detailed Results

User	Latency	FC	BW2	TNF	AVG
User 1	37.49	2.00		7.00	4.50
User 2	42.48	6.00	6.00	9.00	7.00
User 3	106.56	5.00		5.00	5.00
User 4	26.58	4.00	7.00	7.00	6.00
User 5	36.49	5.00	7.00	9.00	7.00
User 6	41.18	3.67	5.00	7.00	5.22
User 7	106.25	2.00	8.00	7.00	5.67
User 8	168.85	1.67	3.00	2.00	2.22

Table E.1: Latency and game scores of all users

Question							AVG
Q2 - 512 Kbps	4	1	5	4	5	8	4.50
Q2 - 1 Mbps	6	2	6	6	7	5	5.33
Q2 - 3 Mbps	8	3	8	8	9	3	6.50
Q3	5	3		8	6	7	5.80
Q4 - Input	6	2	6	10	7	7	6.33
Q4 - Video	5	3	8	10	10	6	7.00
Q5	6	2	7	7	6	6	5.67

Table E.2: User answers to questions 2 - 5

Statement								**AVG**
S1 - Video quality	5	5	3	3	4	4	3	3.86
S1 - Audio quality	4	5	2	3	1	2	3	2.86
S1 - Delay	5	5	4	5	3	5	4	4.43
S2 - Installation	5	4	5	5	5	5	4	4.71
S2 - Platform	3	4	4	5	4	5	4	4.14
S2 - Hardware	4	5	4	5	4	3	3	4.00
S3 - Full games	2	3	2	5	1	1	2	2.29
S3 - Demos	4	4	5	5	5	5	5	4.71
S3 - Cross platf.	4	4	4	5	4	4	3	4.00
S3 - High end	4	4	2	3	4	3	4	3.43

Table E.3: User answers to statements

	Case 1-b	**Case 2**	**Case 3**	**Case 4**	**Case 5-b**	**Case 6-b**
Decoder - Total	24632	34887	35712	13906	17327	33751
Decoder - Frames	3794	4729	3403	1604	3273	4539
Decoder - Avg	6.49	7.38	10.49	8.67	5.29	7.44
Renderer - Total	17706	21455	24502	6894	14526	21415
Renderer - Frames	3794	4729	3403	1604	3273	4539
Renderer - Avg	4.67	4.54	7.20	4.30	4.44	4.72
Input - Total	5343				6559	50
Input - Events	24865				30710	861
Input - Avg	0.21				0.21	0.06
Total time	312061	386582	370238	321124	283504	286926
Active time	58319				50545	63667
CPU load	18.69 %				17.83 %	22.19 %

Table E.4: Performance tests for all cases

	Case 1-b	**Case 5-b**	**Case 6-b**	**AVG**
Decoder	42.24 %	34.28 %	53.01 %	43.18 %
Renderer	30.36 %	28.74 %	33.64 %	30.91 %
Input handler	9.16 %	12.98 %	0.08 %	7.41 %
Other	18,24 %	24,00 %	13,27 %	18,50 %

Table E.5: Client subtask CPU load share

Appendix F

Source Code

Listing F.1: Extract from `GXLinuxDecoder::DecodeVideoPacket()`

```
/**
 * @function    DecodeVideoPacket()
 * @brief       Uses the ffmpeg decoder to decode a video packet.
 * @returns     -1 if error, 0 if ok
 */
int GXLinuxDecoder::DecodeVideoPacket(const uint8_t* inCache, int inCacheSize,
    AVFrame* outFrame, int* outFrameSize)
{

[ ... ]

    // Create AVPacket instance
    AVPacket packet;
    if ( av_new_packet(&packet,inCacheSize) != 0 )
    {
       LOG.MLog("Failed to create AVPacket");
       return -1;
    }

[ ... ]

    // Copy function parameter data to the AVPacket instance
     memcpy(packet.data, inCache, inCacheSize);
     packet.size = inCacheSize;

     int frameFinished;
     int ret = 0;
     BOOL doConvert = FALSE;

     // Check if the codec's pixel format matches desired output format
     if (GetVideoOutFormat() == pCodecCtx->pix_fmt)
     {
       // Decoding the frame from AVPacket instance to parameter AVFrame instance
           (outFrame)
       ret = avcodec_decode_video(pCodecCtx, outFrame,
          &frameFinished, packet.data, packet.size);
       doConvert = FALSE;
     }
     else
     {
       ret = avcodec_decode_video(pCodecCtx, pFrame,
```

```
            &frameFinished, packet.data, packet.size);
        doConvert = TRUE;
    }

[ ... ]

    // Do pixel conversion if needed
    if (doConvert)
    {
        LOG.DLog("Decoder pixel formats were different - converting:
            GetVideoOutFormat() = %i pCodecCtx->pix_fmt %i",
                GetVideoOutFormat(),pCodecCtx->pix_fmt);
        #ifdef USESWSCALE // Note: Disabled by default
            sws_scale(swsCtx, pFrame->data,
                    pFrame->linesize, 0, pCodecCtx->height,
                    pFrameRGB->data, pFrameRGB->linesize);
        #else
            img_convert((AVPicture *)pFrameRGB, avVideoOutFormat,
                (AVPicture*)pFrame, pCodecCtx->pix_fmt,
                pCodecCtx->width, pCodecCtx->height);
        #endif
            *outFrame = *pFrameRGB;
    }
    else
        LOG.DLog("Decoder pixel formats were the same - not converting!");

    av_free_packet(&packet);

    return 0;
}
```

Listing F.2: Extract from `GXLinuxRenderer::RenderFrame()`

```
/**
 * @function    RenderFrame()
 * @brief       Reads an RGB AVFrame and renders it on SDL window
 */
int GXLinuxRenderer::RenderFrame(AVFrame* avFrame)
{

[ ... ]

    // Checks if a screenshot was requested by user
    if (takeScreenshot)
    {
        SaveFramePPM((AVPicture*) avFrame, PIX_FMT_YUV420P, frameIndex++, width,
            height);
        takeScreenshot = 0;
    }

    // Locking YUV overlay before moving data
    SDL_LockYUVOverlay(this->overlay);

    // Setup new location for frame data
    AVPicture pict;
    pict.data[0] = overlay->pixels[0];
    pict.data[1] = overlay->pixels[2];
    pict.data[2] = overlay->pixels[1];

    pict.linesize[0] = overlay->pitches[0];
```

```
    pict.linesize[1] = overlay->pitches[2];
    pict.linesize[2] = overlay->pitches[1];

    // Convert and transfer data to new memory space
    #ifdef USESWSCALE    // Note: Disabled since it is GPL licensed
        sws_scale(swsCtx, avFrame->data,
            avFrame->linesize, 0, this->height,
            pict.data, pict.linesize);
    #else
        img_convert(&pict, PIX_FMT_YUV420P, (AVPicture *)avFrame,
            inFormat, width, height);
    #endif

    // Unlock YUV overlay
    SDL_UnlockYUVOverlay(overlay);

    // Setup display rectangle
    SDL_Rect rect;
    rect.x = 0;
    rect.y = 0;
    rect.w = this->width;
    rect.h = this->height;

    // Blit the YUV overlay
    SDL_DisplayYUVOverlay(overlay, &rect);

    return 0;
}
```

Listing F.3: Extract from `GXLinuxInputHandler::DoWork()`

```
/**
 * @function    DoWork()
 * @brief       Calls DoInput() on an amount of input events according to
     LIMITINPUTPUSH
 * @returns     -1 if SDL_QUIT event, 0 otherwise (normal)
 */
int GXLinuxInputHandler::DoWork()
{
    SDL_Event event;

[ ... ]

    int extractedEvents = 0;

    // Extract a limited amount of input events from queue
    while(extractedEvents <= LIMITINPUTPUSH)
    {
        if (SDL_PollEvent(&event))
        {
            // If a quit message from SDL is received, client should close
            if (event.type == SDL_QUIT)
                return -1;
#ifdef PERFORMANCETEST
            SW_InputHandler.LapStart();
#endif
            // Does actual input handling
            DoInput(&event);
#ifdef PERFORMANCETEST
            SW_InputHandler.LapStop();
```

```
#endif
            // Needed in order to limit amount of events handlet sequentially
            extractedEvents++;
        }
        else
            return 0;
    }

    return 0;
}
```

Listing F.4: Extract from `GXLinuxInputHandler::DoInput()`

```
/**
 * @function    DoInput()
 * @brief       Checks if there is an input event, and handles it if there is
 * @returns     -1 if error or exit, 0 if ok
 */
int GXLinuxInputHandler::DoInput(SDL_Event* event)
{

[ ... ]

    switch(event->type)
        {
            //////////////////////
            // KEYBOARD EVENTS
            //////////////////////

            case SDL_KEYDOWN:

                // Some special functions if SHIFT key
                if (!TESTMODE)
                if (event->key.keysym.mod & (KMOD_LSHIFT | KMOD_RSHIFT))
                {
                    // Have shift+f11 toggle fullscreen
                    if (event->key.keysym.sym == SDLK_F11)
                        gxl_renderer->ToggleFullscreen();
                    // Have shift+f12 toggle grab input
                    else if (event->key.keysym.sym == SDLK_F12)
                        gxl_renderer->ToggleGrabInput();
                    // Have shift+f10 take a screenshot in PPM format
                    else if (event->key.keysym.sym == SDLK_F10)
                        gxl_renderer->TakeScreenshot();
                }

            case SDL_KEYUP: // Note: No break in previous case

                // Set nDevice according to protocol
                nDevice = 1;

                // Set nData1 (press or release) according to protocol
                if (event->type == SDL_KEYDOWN)
                    nData1 = 0x80;
                else if (event->type == SDL_KEYUP)
                    nData1 = 0x00;

                // Retrieve Windows Virtual Key
                nVirtKey = MapSdlKeyboardToWindowsVK(&event->key);
```

```
        // Send the actual input message through protocol interface
        clientLGV->DoSendInput(nDevice, nVirtKey, nData1, nData2);
        break;

    ////////////////////
    // MOUSE EVENTS
    ////////////////////

    case SDL_MOUSEBUTTONDOWN:
    case SDL_MOUSEBUTTONUP:

        // A mouse button has either been pressed or released
        nDevice = 2;

        // Check if it was a mousewheel event (X handles mwheel as button 4
            and 5)
        if (event->button.button == 4 )  // Event: mwheelup
        {
            if (event->type == SDL_MOUSEBUTTONUP)  // Needed or mwheel will be
                sent twice
                break;

            nVirtKey = 0x01;
            nData1 = 120;
        }
        else if (event->button.button == 5) // Event: mwheeldown
        {
            if (event->type == SDL_MOUSEBUTTONUP)  // Needed or mwheel will be
                sent twice
                break;

            nVirtKey = 0x01;
            nData1 = -120;
        }
        else                            // Any other mouse button
        {
            if (event->type == SDL_MOUSEBUTTONDOWN)
                nData1 = 0x80;
            else if (event->type = SDL_MOUSEBUTTONUP)
                nData1 = 0x00;

            nVirtKey = MapSdlMouseToWindowsVK(&event->button);
        }

        clientLGV->DoSendInput(nDevice, nVirtKey, nData1, nData2);
        break;

    case SDL_MOUSEMOTION:

        // Check if mouse and keyboard are grabbed
        if (!TESTMODE)
        if (gxl_renderer->GetGrabInput() == 1)
        {
            // Check if this was the event that moved the mouse to the center
                (see below) -> IGNORE
            if ( (event->motion.x == gxl_renderer->GetWidth()/2) && (event->
                motion.y == gxl_renderer->GetHeight()/2) )
                return 0;

            // Set mouse to center of screen, or grabbed control will reach
```

```
                    boundries of video window
                SDL_WarpMouse(gxl_renderer->GetWidth()/2, gxl_renderer->GetHeight
                    ()/2);
            }

            nDevice = 2;
            nVirtKey = 0x00;

            // Reading relative x and y movement, according to protocol
            nData1 = event->motion.xrel;
            nData2 = event->motion.yrel;

            clientLGV->DoSendInput(nDevice, nVirtKey, nData1, nData2);
            break;

        default:
            break;
    }

  return 0;
}
```