An Investigation of Measurement-Based Load and QoS Management in a Distributed Audio Environment

Mathew W. Kinghorn

Thesis submitted for the degree of

Masters of Science

in

Applied Mathematics

at

The University of Adelaide

(Faculty of Mathematical and Computer Sciences)

School of Mathematical Sciences

November 24, 2005

Contents

List of Tables

List of Figures

Signed Statement

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

SIGNED: ..... DATE: .....

Acknowledgements

Foremost I would like to thank my supervisors, Professor Nigel Bean, Dr David Green and Professor Michael Rumsewicz, for their strong encouragement and high level of support throughout the entire period of my studies.

I greatly appreciate the assistance offered by Alison Jobling, who gave me much advice as to proper programming design and structure, for which I would be lost without. I would like to thank Justin Viiret for his programming assistance and googling skill.

I would like to acknowledge the financial support provided by the Smart Internet Technologies Cooperative Research Centre (SITCRC). In addition, I wish to thank the staff of TRC Mathematical Modelling for making me feel welcome for the duration of my studies.

Finally, I would like to thank my family and any friends not already acknowledged for their support and understanding throughout the entire period of my studies.

Dedication

This thesis is dedicated to Karen, who always knows how to make me smile.

Abstract

Recently, Massively Multiplayer Online Games (MMOGs) have received significant popularity. Made up of populations of physically distributed players, MMOGs provide user immersion into the playing environment through realistic graphics and sound. However, many lack real-time communication between users, relying upon text-based and simple VoIP systems. The Immersive Audio Communications for Massively Multiplayer Games (IACMMG) and Dense Interactive Communications Environment (DICE) projects, currently being developed by the Smart Internet Technologies CRC, aim to change this by providing realistic two dimensional voice communications. As MMOGs cater for a global audience, the projects are designed to utilise a network of audio processing servers to which users are connected for the purposes of communication.

These projects face a number of challenges, two of which we address in this thesis. The first is concerned with the finite resources of each server and is thus a question of how to effectively balance user load over the available audio servers. The second problem is concerned with ensuring that users receive acceptable audio Quality of Service (QoS).

In order to address the first problem, a load balancing architecture is proposed. Four load balancing algorithms are designed and implemented and their respective strengths and weaknesses explored.

To address the second problem, we explore how QoS should be defined in an MMOG context. The IACMMG and DICE projects are VoIP based, thus QoS is primarily dependent upon packet delay and loss. We explore techniques for information gathering from packet traffic and use it for equating perceived QoS. This leads to the development of a QoS algorithm designed to measure the QoS of conversations within the virtual environment of the MMOG, reacting to the perceived level of QoS, and moving users to servers capable of providing higher levels of QoS should this be necessary.

The load balancing and QoS aspects of the IACMMG and DICE projects are dependent upon one another, and thus we are left with a problem of two potentially conflicting objectives. To address this we integrate the two architectures, modifying each to make it compatible with the other. The culmination is a complete architecture capable of both load and QoS management.

Chapter 1

Introduction

The rapid advancement of the Internet has seen the introduction of a number of exciting new technologies. Of these technologies, one that has received significant attention in recent years is that of Massively Multiplayer Online Games (MMOGs). As the name suggests, MMOGs cover a class of computer games that involve a large number of users and are played over a computer network, in particular the Internet. MMOGs can be likened to traditional single-player adventure computer games in that they share similar genres and provide the end user with realistic graphics and sound. However, the major departure from traditional single-player computer games that MMOGs provide is that the population of the game environment is made up of many players. This proves to be the greatest appeal of MMOGs, as users are able to compete and interact with other users from around the world.

As technology advanced, MMOGs have become increasingly more realistic, with immersive graphics and sound. However, the inherent appeal of MMOGs is still the interaction between users, and unfortunately the majority of MMOGs lack advanced means of communication between users. For the majority of MMOGs, interaction between users is restricted to either text-based or simplistic Voice over IP (VoIP) systems. In particular, current VoIP systems do not allow for accurate, 2-dimensional scene creation. That is, current MMOGs lack real-time voice communications be-

tween users that are directionally and spatially correct.

The concept of incorporating real-time, spatially correct voice communications with MMOGs has inspired the development of a number of projects through the Smart Internet Technologies Cooperative Research Centre (SITCRC). The culmination of these projects has led to the Immersive Audio Communications for Massively Multiplayer Games (IACMMG) and Dense Interactive Communications Environment (DICE) projects, the aim of which is to develop a real-time, 2-dimensional audio architecture that can be incorporated seamlessly with current MMOGs.

Whilst Chapter 2 will explore the IACMMG and DICE projects in greater detail, a generalised background is given here. The IACMMG and DICE projects are designed to be separate entities to the game for which they are providing voice communications. This is to ensure that a generic system capable of providing voice for a number of MMOGs is implemented. The system works by overlaying the virtual environment of the MMOG with an invisible grid, designating areas of conversation. Within each grid square a conversation may take place, with the audio scene creation for each user within the grid square being spatially accurate. In addition, any users within the surrounding grid squares of the current one, will hear the conversation take place with accurate distance and direction attributes. However, the background voice, or noise, is a secondary objective, and as such may not be in real-time and may be of lower audio quality.

As the population of MMOGs reside worldwide, the IACMMG and DICE projects must be able to also cater for geographically distant users. To do this, the IACMMG and DICE projects will incorporate a number of servers to which users are connected and have their audio scene creation managed. Each server is responsible for the audio scene creation of the user population of a number of grid squares, with all users in each grid square being hosted by a single server. However, the use of a network of

servers to provide real-time audio scene creation to a geographically distant population of users leads to two of the major problems faced by the IACMMG and DICE projects, which are addressed by this thesis.

The first problem that arises is how to manage a large user population over a network of servers, each with a finite user capacity. Whilst the second problem that arises, is how to maximise user perceived Quality of Service (QoS), given the physical location of both the users contributing to a conversation, and the servers. The difficulty of the two problems is that whilst they seem quite separate problems, we shall see that they are in fact inextricably linked. A detailed examination of both problems will be presented in future chapters, but we again give a brief introduction here.

Load Management:

Within the IACMMG and DICE projects, each user can be thought of as consuming a quantity of resources. This consists of both the processing and bandwidth resources required for the audio streams, both to and from each user. As each user is connected to a server within the IACMMG or DICE networks, the current capacity of the server is reduced by the amount of resources required by each user. If the resources required by the users of a server is greater than the maximum capacity of the server, then it becomes overloaded and the ability of the server to process user requests is diminished. The impact of this is that user requests take longer to process, potentially reducing the user QoS for the server.

As briefly discussed, there are multiple servers within the IACMMG and DICE networks upon which users may be hosted. However, users reside within grid squares, within the virtual world, which are in turn solely hosted by a single server. Thus a user must be hosted by a particular server, if their grid square is on that server.

This can easily lead to a server becoming overloaded if it is hosting a large number, or some particularly popular, grid squares.

To overcome this, grid squares can be moved from one server to another, using a technique known as load balancing. Load Balancing is the process of moving tasks, in our case grid squares, between a number of processors (servers), in order to balance the load experienced by each processor. The end result of this process is that it facilitates the quicker completion of tasks, or in our case, it allows for the concurrent processing of a greater number of tasks. The field of load balancing is well researched and will be explored in greater detail in Chapter 3. However, the general principles of load balancing are that the current capacity of each of the servers be known so that should a server be found to be overloaded, grid squares can be moved from the overloaded server to one with spare capacity. Movement of load is performed using a load balancing algorithm, for which we shall examine four different schemes.

QoS Management:

The second problem encountered by the IACMMG and DICE projects is that users have varied geographical distances between them and the servers of the network. A user must connect to the server which is hosting the grid square they reside in, which may result in a large variation of delay values for each user, even for those users within the same grid square. The amount of delay encountered by a user directly affects the perceived audio QoS the user experiences. We develop an empirical measurement of the perceived audio quality a user receives, through the use of the ITU-T E-Model [24]. Using audio packet delay values, the E-Model is capable of producing a numerical figure for the perceived audio QoS a user is currently receiving. Using the E-Model, we develop an algorithm with the following goals. The first is to measure the perceived QoS of the users of a grid square. The

algorithm must then determine if the QoS received is either adequate or not. Finally, given that the QoS is found to be inadequate, the algorithm will then attempt to find a server that is able to provide an adequate level of audio QoS.

Integration

In order to completely address the problems faced by the IACMMG and DICE projects, each problem discussed must be aware of the other and take this into consideration. For instance, a traditional load balancing algorithm may be indiscriminate as to the reallocation of grid squares with respect to audio QoS. On the other hand, our unmodified QoS algorithm will find the optimal server for a grid square to move, with respect to perceived QoS, but will not take into consideration the current capacity of the server in question. Thus, in order to construct a viable load and QoS management architecture for the IACMMG and DICE projects, both components must be aware of the other and be willing to make decisions based upon information collected by the other.

1.1 A Brief Introduction to Voice over IP

Voice over IP (VoIP) [11] is a technology that allows for transmission of voice to be made over an Internet Protocol (IP) data network, such as the Internet. VoIP has a number of benefits which have made it a booming technology. For example, by implementing VoIP, big businesses are able to merge their existing telephone and data networks into a single network. However, as VoIP uses the data network for the transmission of voice, it must send voice in a data friendly form. To do this, the voice must first be digitised through the use of an audio *codec*. There are a number of different audio codecs, each with their own strengths and weaknesses, and we shall explore a number of them in the following chapters.

Having digitised the audio, it is packetised and a number of protocols from

the TCP/IP protocol suite are used to transmit the audio packets from the source address to the destination address. Typically VoIP is transmitted over the data network using the User Datagram Protocol (UDP) [41] which provides addressing information to route the packetised audio through the network. However, UDP provides little else apart from source and destination addressing for each packet. Real-Time Transport Protocol (RTP) [46] is a further protocol that provides information that UDP does not. RTP provides packet numbering and time stamping. Packet numbering allows the receiver to determine if packets are missing or if packets have arrived out of order, whilst time-stamping allows the receiver to control the playback of the encoded audio [9]. The receiver can determine the variability in packet arrival and can thus determine how much delay at the receiver, will be required to ensure smooth playback.

1.2 Road Map

This thesis is organised as follows. In Chapter 2, we explore the motivation behind the IACMMG and DICE projects, as well as an introduction to its operation. In Chapter 3, we examine current methods for balancing load in computer networks. In Chapter 4, we introduce our load balancing architecture, and present results detailing its performance. In Chapter 5, we look at QoS management and in particular methods for gathering information from the network relevant to QoS. In Chapter 6, we examine our QoS management algorithm, and present results detailing its performance. In Chapter 7, we investigate the integration of both load and QoS management into a single architecture, and present results examining its effectiveness as a possible solution to the IACMMG and DICE projects' problem. Chapter 8 concludes the thesis.

1.3 Contributions

The contribution to the field that is made by this thesis is that we combine two disparate systems, a load balancing architecture, and a QoS management architecture, to form a complete system. Through analysis of the two architectures, we develop an integrated load and QoS management architecture that is currently absent from the literature.

Chapter 2

Motivation

Recently Voice over IP (VoIP) [11] has gained increased popularity as it is proving to be a relatively low bandwidth and good quality means for conducting voice communication over the Internet. However, at present, its main use is for a conversation between two parties. Extensions to this model in the form of allowing multiple users to take part in a single conversation is attractive, especially for use in online chat-rooms and Massively Multiplayer Online Games (MMOG) for which the traditional means of communication relies on typed text.

The attractiveness of an amalgam of VoIP and current technologies, such as Internet chat-rooms and MMOG, in part motivated the introduction and development of a number of projects through the Smart Internet Technologies Cooperative Research Centre (SITCRC). These projects are known as the Virtual Café and its successors, the Immersive Audio Communications for Massively Multiplayer Games (IACMMG) and Dense Interactive Communications Environment (DICE) projects. The aim of the Virtual Café was to merge current VoIP technology with the Internet chat-room, to create as the name suggests a 'virtual café' in which users can interact in both small and large scale conversations via VoIP. The concepts behind the Virtual Café have further application, especially in online computer games, allowing for the ability to hold real-time conversations with other players without the need

to type, adding another level of realism and interactivity. This extension in which real-time 2-dimensional voice is incorporated into online computer games is what is known as the Immersive Audio Communication for Massively Multiplayer Games (IACMMG) and Dense Interactive Communications Environment (DICE) projects.

However, unlike the traditional circuit-switched telephone network which is dedicated to the transmission of voice, the Internet is inherently a best effort network. There are no guarantees that an audio transmission will arrive at its destination on time, or even at all. Thus, in order to provide users with an acceptable level of audio quality, or Quality of Service (QoS), we strive to minimise factors that contribute to reducing levels of QoS. In the following chapter we present a detailed discussion of QoS and some of the many factors that contribute to the reduction of QoS in online audio applications. The rest of this chapter is dedicated to the motivation behind this work. In particular we begin by looking at current applications that may benefit from the addition of real-time voice communication. Following this, we discuss the Virtual Café, IACMMG and DICE projects, giving some insight as to how they operate, as well as providing a footing from where we can begin to examine the major topic of this thesis.

2.1 Current Technology

For some time the Internet has increasingly been used as a means of communication between friends and colleagues. Email is a widely used medium through which users send messages. However, electronic mail is a poor method for communication between a group of users. To that end, online chat rooms were developed to allow groups of users to communicate. Through the use of text, users are able to post messages to the bulletin board of the chat-room, to which other users may respond. The attractiveness of this technology is that users are able to post messages quickly, ensuring the momentum of a conversation is maintained. Unfortunately, this attrac-

tiveness can also be viewed as a significant flaw of online chat-rooms in that users who are faster typists can easily dominate a conversation. In addition, confusion can occur when a user responds late to a comment as the two messages may not be contiguous. Hence the integration of Voice over IP with the Internet chat-room appears to be a natural progression, reducing the awkwardness of the traditional text-based system.

Even more recently, Massively Multiplayer Online Games (MMOG) have had near exponential growth with nearly every computer game released now including some form of online multiplayer component. Massively Multiplayer Online Games differ from other games that are released in that they are made with the sole intention of being played whilst connected to the Internet. MMOG allow for players from around the world to pit their skills against each other. As with their counterparts, MMOG have evolved to the point where they sport realistic graphics and sound, adding to the realism of the rendered environment. The problem is that they have limited voice capabilities. Thus by adding voice, it is envisioned that it will culminate in complete immersion within the game world. First we need to investigate the three major genres of MMOG and discuss what they currently offer for communications between users and if the IACMMG and DICE projects might be applicable.

2.1.1 Real Time Strategies

The basis of the Real Time Strategy (RTS) game is that each user within the virtual environment proceeds to construct an army with the sole purpose of destroying the armies of the other players. Complexity is added to this basic formula by requiring that each user gather resources that are used to buy the individual units of a player's army. In addition, many of these games require the building of structures that facilitate the collection of resources and the construction of fighting units. Balance must be maintained between the construction of fighting units and the development

of new units through the construction of new unit building structures and upgrades.

The majority of RTS games are played from an isometric or "God's eye view" perspective and each individual game usually only hosts a small number of players. Whilst each game may only host a small number of players, a game server may host a large number of individual games, with the users of one game being ignorant of the users in another game. Both text and VoIP based systems are available (discussed in the next section) for use with RTS games and provide the ability to communicate with the other users of the game. Whilst RTS games can be played in teams, they are more often played in a solitary manner.

2.1.2 First Person Shooters

First Person Shooters (FPS) are similar to Real Time Strategy games in that each game server hosts a number of individual games, which in turn host a number of players. The game's basic premise is that each player wants to eliminate the other players within the game whilst trying not to be eliminated in the process. Apart from these similarities, FPS differ considerably from RTS. As the name suggests, FPS take a first person perspective. In addition, instead of building an army with which to defeat the other players, FPS have the player control an 'avatar', representing themselves in the virtual world. Many FPS are very simplistic in nature where each player runs around the virtual environment, collecting weapons and ammunition, and then eliminating the other players. However, some have evolved, adding greater levels of strategy to the game. A major evolutionary step introduced into FPS, was to make them team-based. Making FPS team-based allows for team objectives to be completed along with the elimination of the opposition.

Traditionally, FPS relied upon text-based and hot-key triggered systems for communicating between users. These have proven to be a suitable means of communica-

tion since there is not often a reason to communicate between players. The problem with these systems is that vital moments are lost whilst the user switches from playing to typing. The fast paced nature of FPS means that a moment wasted typing could be the moment the player is eliminated. The introduction of team-based FPS necessitates that users work together: to communicate with one another to achieve mission objectives. The text-based system is too slow to allow for effective communication between team members. For this reason VoIP based applications have been developed to facilitate voice communications. This has been seen as a significant step forward in adding realism, something that improving 3-D graphics and sound could not deliver.

There are some problems with the current VoIP chat programs in terms of their level of realism. Messages are broadcast throughout the group of users. The end effect of the use of these chat applications can be thought of as each team member being in contact with each other through a walkie-talkie or radio headset. Whilst this will be realistic in a game simulating a counter-terrorism situation, the same cannot be said for a game set in a fantasy environment. However, the crux of the problem is that whilst the audio systems within these games have evolved to include 3-D sound effects, the same cannot be said for the VoIP chat programs that are available. For example, whilst a player can hear the footsteps of an approaching enemy and work out from where they will attack, the player cannot be privy to the conversation the enemy may be taking part in.

2.1.3 Role Playing Games

Role Playing Games (RPG) differ greatly from both RTS and FPS games and yet still retain some similar aspects, namely the concepts of survival and elimination. Whilst both RTS and FPS game servers host a number of games to which players are connected, RPG servers can quite often only host a single game to which a large

number of users connect. Depending on the popularity of the game, the number of users connected can range from hundreds up to thousands, all interacting within the same virtual environment. This is possible since the virtual worlds created are immense in size. Unlike RTS and FPS games where virtual environments are made small to ensure fast paced action, RPG are made large, allowing players to explore and to be thoroughly captivated by the simulated world.

The idea behind RPG is that the player guides their avatar through the game world gathering both experience and currency through the completion of missions and the elimination of creatures located in the environment. Experience is used to 'level-up', increasing the vital statistics of the player's avatar such as strength or dexterity, or is used to develop skills such as spells. As the avatar gains experience, they become more difficult to eliminate and are able to tackle the difficult monsters found in the game world. Currency is used to purchase equipment, such as armour or weaponry, to make it easier to gather more experience and currency. RPG utilise either an isometric or first-person perspective to the virtual world, with the first-person view point becoming essentially standard as users, computers and Internet connections have improved.

Unlike RTS and FPS, RPG are actively maintained and enhanced by the game developer. This is because even though a player pays for the game they install on their computer, they must still pay a monthly fee in order to have access to the game servers and thus the game itself. Without the monthly fee, the installed game is useless. Accordingly, game developers have to ensure that the game never becomes stale, or else they will lose players to the competition. To this end, game developers aim to make their online RPG as immersive as possible. This includes graphics and sound effects, but also includes making the keyboard layout easy to remember and to make the interface clear and simple.

An interesting phenomenon that has been revealed with the introduction of online RPG is that many players spend a considerable amount of time talking to each other inside the game world. This has proven a boon for developers of well designed online RPGs since players with friends within the game are more likely to remain loyal. The problem is that communication between users is via text. The result of a large assembly of players can be text messages covering the screen. In some games, conversation is held via a chat window, increasing playability but the problems with Internet chat-rooms remain, and so voice communication is desirable. The problem is that VoIP applications used in FPS are not suitable in this situation. They cannot scale to allow for the large population as all users are connected to a single server. In addition, the environments are large to the point that it is not realistic to be able to speak clearly with players on the other side of the virtual world. Realism proves to be an important factor for many RPG.

2.1.4 Summary

Whilst mechanisms are available, be they text or voice based, for communication between users within online computer games, they prove problematic. Text-based systems add little realism and can be difficult to use. VoIP based systems can provide much greater levels of realism to both RTS and FPS games, but are not suitable for RPG. The main problem we find with VoIP systems is that they have not been designed to cope with the sheer scale of many of the RPG. VoIP programs run either in a peer to peer fashion, where each user is connected to each other user, or one user must act as a VoIP server to which the other users connect. As one can imagine, as the number of users connected to the system increases, either the connections between individual users becomes excessive or one user (the server) becomes overloaded. For these reasons, independent VoIP systems will never be able to scale to accommodate the large numbers of users we see in many MMOGs.

The idea behind the IACMMG and DICE projects is that we design a system that is independent of the individual game architectures. In this way, it can be made compatible with a series of different games. It is then run in conjunction with the game such that support in the form of servers run by the game developers is available.

2.2 The IACMMG and DICE Projects

Of the three genres of online computer games outlined, role playing games prove to have the greatest need for an intelligent voice communications system. First Person Shooters have simple VoIP applications which are all that are necessary for communications between a small community of users. RPGs require a completely new system. The Virtual Café and its successors, the Interactive Audio Communications for Massively Multiplayer Games and Dense Interactive Communications Environment projects aim to provide such a communication system [43]. The idea is to provide a completely immersive 2-D audio scene creation. What this means is that voice heard by a user has realistic distance and direction components. Naturally, if one was conversing with a fellow user within short range, the conversation will be clear. On the other hand, if one happens to be eavesdropping on another conversation from a distance, then the volume will be reduced.

2.2.1 How It Works

The Virtual Café, the IACMMG and DICE projects work in a similar way. They aim to provide realistic, 2D audio for users within a virtual environment. The difference between the projects is in the way the 2D scene is created. The Virtual Café consists of a static 2D audio scene, in which a user is represented by an avatar. Through the avatar, the user may navigate through the Virtual Café. Tables are located within the café and it is at these tables that conversations are held. The tables of the Virtual Café are analogous to the single rooms of the chat-room application. At

each table of the Virtual café a user will hear other users at that table with equal volume, albeit from differing directions depending upon user positions around the table thus creating the 2D audio scene. Conversations at tables surrounding the user's table are heard at a reduced volume, with audio direction being maintained. The purpose of this is to give users a more immersive environment. An additional feature is that users can walk between tables listening in on conversations before sitting down at a table where they believe the conversation will interest them the most.

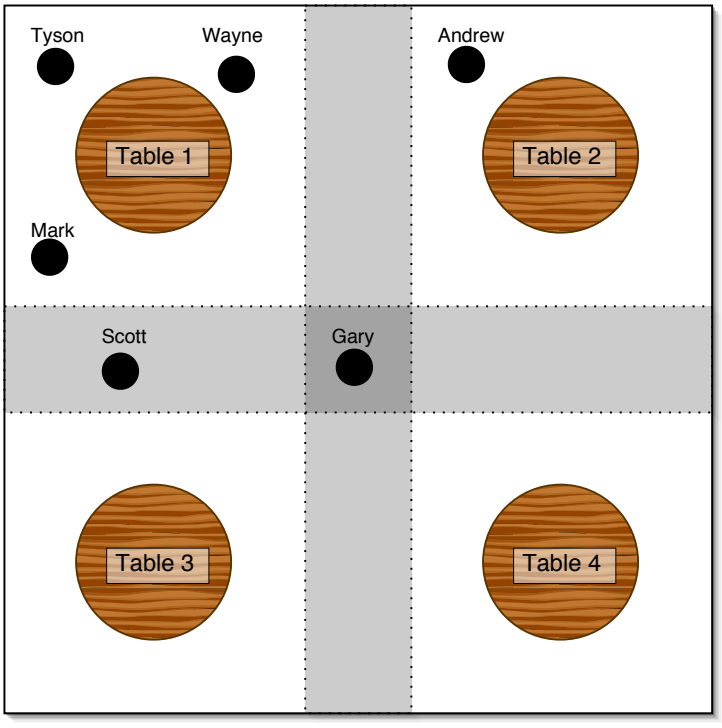


Figure 2.2.1: Depiction of a four table Virtual Café.

For example, if we look at Figure 2.2.1, we see that the environment is divided into four zones, with a table in each. Located at Table 1 are Tyson, Wayne and Mark. They are involved in the conversation at Table 1 and benefit from realistic 2D audio scene creation. Scott is between Tables 1 and 3 and thus may take part in any conversation held on these tables. In this scenario, Scott may converse with

Tyson, Wayne and Mark. Gary is between all four tables and so may take part in all conversations being held. Andrew is by himself on Table 2 and so is restricted to conversing with Gary only.

The IACMMG and DICE projects are an evolutionary step up from the Virtual Café, in which real-time, 2D audio scene creation is incorporated into Massively Multiplayer Online Games which scale to allow for large numbers of users. The IACMMG and DICE projects run as an overlay to current and future online games. For the purposes of prototyping and research, the IACMMG and DICE projects overlay a grid over the virtual world. Each grid square within the virtual world represents a possible conversation location. Grid squares adjacent to another particular grid square are declared its neighbours and it is from these squares that background voice is collected. Figure 2.2.2 is a simple example where the grid overlay for the virtual environment consists of 25 grid squares. If we consider only the concerns of the central square, we see that the three users: Mark, Wayne and Tyson are in the same conversation. The sound that they hear is both clear and spatially correct. The eight squares adjacent to the central square provide the directional background voice data. For example, background voice only from the square containing Andrew and Scott, will be sent to Mark, Wayne and Tyson. In this way, Mark, Wayne and Tyson are able to listen in on Andrew and Scott's conversation and vice versa.

To create these virtual audio scenes, we need to send audio through the digital data network. A codec digitises and then packetises an analogue signal such that it is ready to be sent over the data network. On the receiving end of a conversation, the codec is responsible for decoding the packetised audio and reconstructing the original analogue signal. A variety of audio codecs are available for use in data networks, with each using a different bit rate and delivering a different quality. In order to add robustness to the design of the Virtual Café and IACMMGs projects, both should be compatible with a number of codec choices. In order to get a good cross

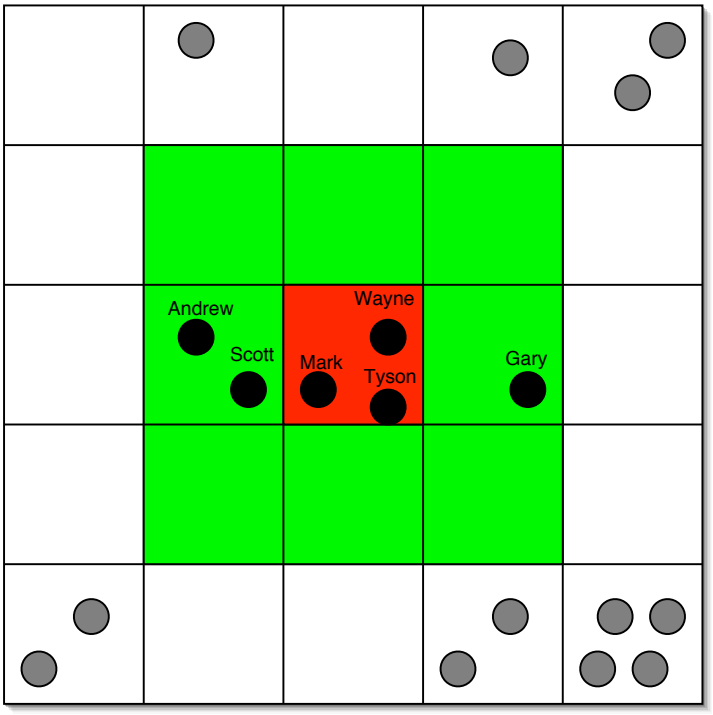


Figure 2.2.2: Example of a grid overlay.

section of the many audio codecs available, we have chosen the following three for further investigation: G.711 [21], G.729 [18] and G.723.1 [17]. G.711 is also known as Pulse Code Modulation (PCM) and is used in current telephone networks. It is a 64 Kbps codec and is regarded as the best codec to use with respect to audio quality. The drawback of G.711 is that it has a relatively large bandwidth requirement. The other two codecs, G.729 and G.723.1 have bit rates of 8 Kbps and 6.3 Kbps respectively. They sacrifice some quality for reduced bandwidth. The bit rates stated above for the three codecs give the total bit rate of the audio component of the packetised audio stream.

In order to give both distance and direction to the voice transmitted through the game a combination of 5.1 audio and the in-game coordinate system is used. The 5.1 audio system provides six separate audio channels representing front left

and right, centre, back left and right, and bass. This system allows for the delivery of 2D audio to the user. The in-game coordinate system provides the location of each user with respect to one another. The IACMMG and DICE projects use this to determine for each user the direction audio is coming from and thus which audio channels to use.

A focus of the three projects is that users interact with other users from all over the world. This can result in great physical distance separating users in a particular conversation. As the distance between users increases so too does the delay and the result is that audio QoS decreases. In order to maximise audio quality, a number of servers should be located throughout the world. From the point of view of the users, the multiple servers are invisible, there is simply a single virtual server to which they connect (Figure 2.2.3). The use of grid squares simplifies the allocation of users to servers, in that all users in a given grid square must be hosted by the one server, though a single server may host many grid squares. Figure 2.2.4 gives a physical interpretation of the allocation of users to servers. Using the grid square arrangement of Figure 2.2.2, Wayne, Mark and Tyson reside within the same grid square and thus must be connected to the same server. The same can be said for both Andrew and Scott.

MMOGs are installed upon a user's computer. From then on, only small updates, signaling the player's actions, are required to be sent between the user and server. Voice on the other hand cannot rely on this. Thus it is imperative that neither the audio servers nor the network become overloaded with audio streams. To combat this, grid squares are assigned and then moved between servers in an attempt to provide the best audio quality for their users. As servers will be located worldwide, it is probable that a server will be able to provide higher audio QoS for a grid square due to its closer physical distance to the users of the grid square. Thus the grid square can be moved to this server, allowing for greater QoS to be had

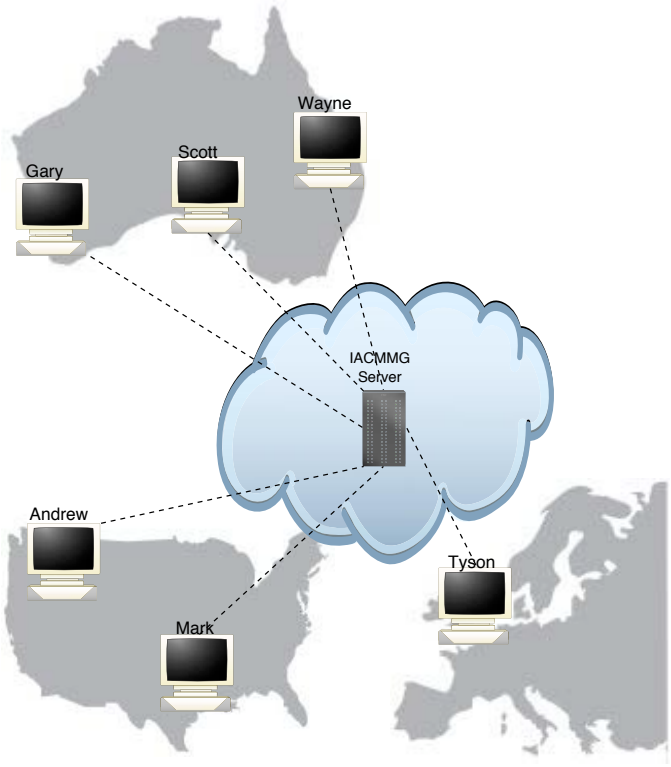


Figure 2.2.3: Users see a single server to which they connect

by the resident users. Unfortunately participants in local conversations within the virtual world would rarely remain static for long periods of time. For both projects, conversations may last for long periods of time and it is within this time period that the condition of the server may change. More users may join the current server, reducing its ability to provide an adequate service to all users. Links of the path between a user and the server may become congested, increasing the delay between audio packets and hence reducing the overall quality of the audio stream. The IACMMG and DICE projects are even more dynamic in that whilst one aim of a RPG is to converse with other players, another is to complete quests and through doing this the user's avatar will come into earshot of other users located in different regions of the virtual environment and on separate servers. In a similar way, users in the Virtual Café can move from one table to another, changing both the conversation

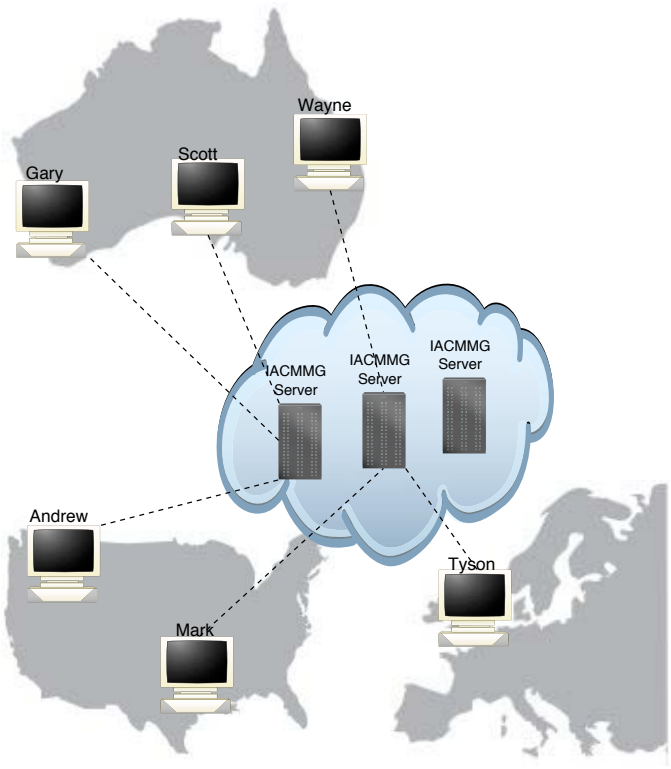


Figure 2.2.4: A network of servers allow for load sharing and maximising audio quality.

they were involved with, and the one they join. If a server becomes overloaded, or the audio quality deteriorates for other reasons, the grid square can be moved from one server to another with the aim of providing greater audio quality for the users involved within the conversation. This, however, incurs a cost which must be weighed against any improvement in audio quality experienced by the users.

2.3 Issues

There are two main research issues on which we shall focus. The first issue is trying to determine when a conversation and hence a grid square is suffering from poor audio quality. Chapter 5 introduces the ITU-T E-Model, which is the industry stan-

dard for measuring audio quality between a pair of users and will be used as the basis of the work presented here.

The second issue is that given a grid square is found to be suffering from poor audio quality, how do we best allocate grid squares to servers so as to maximise the overall service quality? This involves balancing server processing capacity with Internet delay and loss characteristics. We introduce server load balancing in Chapters 3 and 4 and discuss in detail how to design an architecture to address these issues and best achieve this tradeoff.

Chapter 3

Discussion of Load Management

Load Management is a large and widely discussed topic within the literature. Load management comes under a number of guises. There are a large number of factors which distinguish various load management techniques. To this end, we use this chapter to examine many of the different methods for load management. We begin by giving a brief discussion of how load management is performed and in what circumstances it is used. After this we give some explanation as to many of the factors that separate different load management techniques and methodologies. Following this, we explore a selection of load management schemes. Finally, we relate what we have discussed in this chapter with the load management needs of the IACMMG and DICE projects discussed in the previous chapter.

3.1 Load Balancing Literature

Load Balancing, also known as load sharing, load scheduling, or load distribution, is the process of distributing a workload over a number of processors, or nodes, in a network in an attempt to efficiently utilise the combined processing power of the network [48, 32, 45]. A simple example of a load balancing architecture would be to look at a scenario in which a single server handles client requests. As the volume of requests increases, the single server is bound by its processing capacity and requests

may be rejected. Through the interconnection of a number of servers, if a particular server becomes inundated with requests, then requests that would otherwise be denied may be distributed to other servers for processing. Apart from faster task completion, there a number of other benefits of load balancing. For example, having multiple connected nodes increases the reliability of the system, in that it allows for redundancy [32]. If one node should fail for any reason, computation can be moved to one of the other nodes in the network.

3.2 Definitions

We use the following section to characterise the load balancing architectures present in the literature.

3.2.1 Static vs Dynamic

Load management schemes can be divided into two categories based upon the amount of prior knowledge or assumptions made about the nature of incoming tasks the scheme has. These categories are static and dynamic [30, 48, 32, 3, 27]. A static system makes load management decisions based upon trends that develop within the system. To work well, a static load management scheme requires that the load balancing algorithm have *a priori* knowledge of the tasks that will enter the system, using average behaviour information of the network, whilst ignoring the current state of the individual nodes [28].

A dynamic system, on the other hand, does not necessarily require prior knowledge of incoming tasks but may react to changes in the system on the fly. Consequently, dynamic systems require greater complexity in that they require information on the state of the system before balancing decisions can be made.

3.2.2 Homogeneous vs Heterogeneous

Load management schemes can be further categorised, depending upon whether the nodes of the network are either homogeneous or heterogeneous. In the case of having homogeneous nodes, each and every node within the network is of equivalent processing power, with equal resources. An example of a homogeneous system would be a network of identical computers.

A heterogeneous network consists of nodes of varying processing capabilities. A simple example of which would be a network of local computers of various ages and models and thus various speeds.

The significance of whether the network is either homogeneous or heterogeneous is that load balancing schemes for heterogeneous systems need to be of greater complexity: since nodes are not equivalent, the load balancing algorithm must take into account the varying processing capacity of each machine.

3.2.3 Sender-Initiated vs Receiver-Intiated

Load balancing algorithms can be split into two categories; sender-initiated [28, 13], or active [30], algorithms and receiver-initiated [28, 13], or passive [30], algorithms. Sender-initiated load balancing algorithms involve the currently overloaded node initiating the load balancing process. Once a node becomes overloaded, it proceeds to poll the other nodes of the network to determine their load. If there is a node that is either idle or has spare capacity (depending upon the specific algorithm), excess tasks on the overloaded node are transferred to the node with spare capacity. If however, there are no servers with spare capacity, the overloaded server remains overloaded until either: processes have finished execution, reducing the number of queued processes, or the overloaded node is given another chance to load balance.

Receiver-initiated algorithms, on the other hand, rely on nodes which find themselves idle, or in a state where their load is below a set threshold, to actively look for excess tasks to process. Once a node becomes idle (it has no tasks to process), or its load is below a set threshold, the node polls other nodes in the network in an attempt to find a node which is overloaded. If such a node exists, tasks are moved from the overloaded node to the idle node. Otherwise, the idle node either waits until tasks are generated locally, or it is given another chance to load balance.

The benefit of using sender-initiated strategy is that overloaded servers can attempt to move excess load as soon as it develops, instead of having to wait for an idle or low loaded node to ask. However, the main disadvantage is that overloaded servers must be further burdened by having to initiate load balancing.

The advantage of using a receiver-initiated algorithm is that there is very little overhead when all of the nodes have enough work to keep them busy [30]. In addition, the majority of effort required for load balancing is given to the least loaded nodes, reducing the strain on the overloaded nodes. On the other hand, receiver-initiated algorithms have problems when facing lightly loaded environments as idle nodes take up bandwidth with requests for tasks.

One final category, a hybrid, is to use both sender-initiated and receiver-initiated methods in the same algorithm [30]. The idea here is to combine the advantages of the two strategies. Overloaded nodes may ask the rest of the network for any processors capable of taking excess tasks whilst any idle processors may also poll the network looking for load.

3.2.4 Centralised vs Distributed

The responsibility of how load balancing is performed is determined by whether the network and corresponding algorithm is either centralised or distributed [13]. A centralised load balancing scheme involves assigning all responsibility for both scheduling new tasks to each of the nodes, as well as moving tasks from overloaded nodes to nodes with low load, to a central controller. This controller could be one of the nodes of the network, or as is most often the case, a separate machine.

On the other hand, a distributed scheme, consists of independent nodes which are linked together and know of each other, but may not necessarily know of each other's current state. Each node must share its processor power and resources between completing tasks and keeping in contact with the other nodes to allow for load balancing.

Both algorithm styles have their advantages and disadvantages and each is best suited for a particular circumstance. For instance, a centralised architecture allows for complete knowledge of the network to be maintained, ensuring that any load balancing performed is optimal. However, it also has its disadvantages, most notable of which is that the central controller becomes a performance bottleneck as the number of nodes in the network increases [14, 13]. It is also a single point of failure for the network [29]. Thus, a centralised architecture is possibly better suited to a smaller network. A distributed architecture is not as restricted by the size of the network and so proves easier to scale [28]. However, as we shall see, maintaining accurate and up-to-date state information can be problematic.

3.2.5 Maintaining State Information

Within a centralised system, a central node is dedicated to the control of the network as well as to the execution of the load balancing algorithm. The result of which

is that the central controller is responsible for the scheduling and movement of all tasks within the network. Thus, it is very easy to maintain complete information of the state of the network.

On the other hand, distributed networks have the problem that due to the individual nodes being responsible for scheduling local tasks and their own load management, they may have an incomplete picture of the rest of the network. Even with polling, problems exist as there is always a period of time between when information is sent by one server and it is used by the others in the network. This can be a problem for a number of reasons, most importantly a node may be reported as being lightly loaded when it is in fact overloaded, causing neighbouring nodes to transfer tasks to it. The other important problem is the opposite case resulting in missed opportunities for neighbouring nodes. To remedy this, state information is distributed throughout the network in a regular fashion. However, the method in which this is performed and the frequency of information distribution can be problematic.

The simplest method is to have periodic distribution of state information. Each node, after some fixed period of time, sends its load information to the other nodes in the network (if it has changed from last time) creating a complete picture of the network from the point of view of each node. The problem is that the frequency of information distribution plays a significant role in the effectiveness of this method. If the frequency is high, then state information is kept accurate and up-to-date, but at the expense of nodes having to spend resources processing load update messages. However, some argue [27] that the resources spent on processing update messages is negligible when compared to the resources required to move tasks.

If the frequency of updates is low, then the problems of stale information that would plague a system without the updating of state information exist, but nodes

are free to process tasks. However, even with stale information, the system can perform no worse, and often better, than if random selection was to be performed [35].

An alternative technique is to send update messages once something significant occurs at the local node. This can range from sending updates whenever a task is processed or a new task is scheduled and queued, to setting thresholds for which load must either get above or below before an update message is sent.

Whatever way state information is maintained, it is critical that it is kept accurate and up-to-date to ensure that load balancing mistakes are minimised.

3.2.6 Costs

Sharing load over multiple servers comes at a cost. In both centralised and distributed load balancing environments, there are a number of costs that may limit the effectiveness of the architecture. For example, there is the time wasted in physically moving tasks between nodes. In addition, a task may preferably be processed on the current node, perhaps due to previous tasks in the series that have been already processed, or there may be resources that have been already allocated for the task. To move the task would require consideration of having to move the associated resources. Ultimately, it may prove more costly to move a task to a lightly loaded node than it is to make it wait at the currently overloaded node [48].

There is also the cost of maintaining up-to-date state information about the nodes of the network, which comes in the form of server polling. However this is a cost that is mostly applicable to distributed architectures. As we have discussed previously, state information must be distributed throughout the network on a regular basis. If updates are made too often then nodes spend a large proportion of their

time maintaining this information. There is also the cost in terms of bandwidth for all of the update messages sent. However, if updates are made infrequently, then nodes may make misinformed decisions due to stale state information.

Ultimately for a load balancing architecture to be successful, it must be able to effectively weigh up the cost imposed by task migration against the potential benefits the movement may bring [27].

3.3 Example Methods

We now explore a number of load balancing algorithms. In [4], a dynamic, homogeneous architecture utilising a simple distributed receiver-initiated load balancing scheme is described in which idle processors may take queued tasks from their immediate neighbours, the choice of which is based upon the current load of each of the neighbours. In order to ensure that tasks are eventually processed, once a task is moved it may not be transferred to another idle node. The problem with [4] is that the authors do not consider any of the costs associated with the movement of tasks from one node to another.

The *gradient model*, a distributed load balancing algorithm utilised on a dynamic, heterogeneous system, is presented in [31], where each node is given a label based upon its state using two thresholds. Each node can either have a light, moderate or heavy load. In addition, nodes are only connected to their immediate neighbours. Because processors are connected to their immediate neighbours, a concept of "proximity" is developed for the purposes of load balancing. The proximity of a node is the distance in node hops between the current processor and the nearest lightly loaded node. If a node has a light load, it is a given a proximity of zero. It then informs its neighbours, which modify their proximity and so forth. When a node is found to be heavily loaded, tasks migrate to the nearest lightly loaded node.

Lin *et al.* [31] give a thorough analysis of their model, taking into account a number of critical factors. However, as load balancing is performed between neighbouring nodes, this model is better suited to nodes of close proximity or smaller processes. As the distance between nodes, or the size of processes, increases the cost to move processes between heavily and lightly loaded nodes, through intermediary nodes, becomes significant.

In [39], a dynamic, homogeneous network with a distributed, sender-initiated, threshold scheduling policy to forward calls through a network is examined. Whilst calls prefer to be forwarded to their original destination node, they may be scheduled elsewhere should the current load of the node be found to be larger than a pre-set threshold. In this case it is assumed that it is less costly to send the process to the remote node than it is to allow it to wait for processing at the original node. The only problem with this method is that the scheduling policy may be difficult to scale due to the algorithm's complexity.

Another dynamic, homogeneous network with a distributed, sender-initiated approach is given in [14]. Called *diffusion scheduling*, it involves nodes deciding either to keep or send off any newly created processes to their immediate neighbours. This decision is based upon the load of both the node creating the process and its neighbours. As processes arrive at a node, they can be further passed on. However, the further away a process is moved from its creator node, the higher the cost, as local data must also be moved.

The work in [15], [37] and [7] looks at a hierarchical load balancing architecture in which a collection of local processors are connected together to form a LAN or domain. Independent LANs or domains are in turn connected together to form a WAN. The individual LANs can use either a distributed or centralised, sender-initiated load balancing scheme. However, due to the smaller number and close

locality of nodes within LANs, a centralised algorithm will provide better results. Load is balanced between the individual domains through the use of a distributed sender-initiated algorithm. Thus load balancing is first attempted locally before attempts are made globally. However, it is really only [7] which goes into detail as to the effects that lag times between the individual domains can have upon a load management architecture.

Hwang *et al.* [16], look at a sender-initiated method in which the offloading processor selects the most idle node through the use of a load average. The load average is defined as the approximate increase in the amount of time it would take to run on the current machine versus the time needed for the same process to run on a completely idle machine, and is calculated from a number of factors including the number of currently running processes and the amount of effort required to move the task.

In [34], a simple sender-initiated, receiver-initiated and a combined algorithm in a distributed systems environment is considered. Emphasis is placed upon the problems associated with out of date state information and significant task transfer delays, finding that with high delays, load balancing is no longer effective. However, at high loading of each node, load balancing becomes important no matter the transfer delay.

Schaar *et al.* [45] look at three algorithms, a receiver-initiated, a sender-initiated and a threshold algorithm, examining the communication traffic of each over a heterogeneous network. The receiver and sender-initiated are only able to send or receive excess load if the lightly load node is in fact idle. The threshold algorithm can migrate load as long as the load of the lightly loaded node is below a threshold. They found that the threshold algorithm has the largest amount of communication traffic, which can be expected since the likelihood of either of the sender or receiver-

initiated algorithms to meet their task movement conditions is far less than the threshold algorithm. Thus there are less tasks being migrated and hence less traffic.

Finally, [42] explores load balancing in peer-to-peer networks, which for our purposes can be thought of as large scale, heterogeneous, distributed systems. In particular they look at balancing the demand for content in these systems through the use of "replica" nodes, which are used to cache popular content. Request for content arrives at a system node which then directs the request to one of the replica nodes which is currently hosting the required content. They found problems with traditional distributed load balancing schemes in that they react slowly to changes in the network, due to the large geographical distance between nodes. For example, a node may become overloaded and report this to the other nodes in the network. Suddenly, the overload node is dropped from further forwarding, reducing its load significantly, but potentially overloading the remaining nodes which host the same content.

3.4 Load Management Scheme

We now propose a load management architecture for the IACMMG and DICE projects. Due to the unique nature of the IACMMG and DICE projects, we shall spend some time investigating the similarities and differences between our load management architecture and the literature presented. We propose a dynamic and heterogeneous load management architecture utilising both centralised and distributed sender-intiated load balancing algorithms. A dynamic approach is taken as it is difficult to ascertain user movements through MMOGs. Whilst much of our analysis will examine a homogeneous system, our architecture is designed to allow for a heterogeneous network. Of the four algorithms we shall examine, one is centralised and the others are decentralised, whilst all are sender-initated.

The pure load balancing aspects of our proposed algorithms have a number of similarities with schemes proposed elsewhere. For example, instead of looking at individual processes, we acknowledge that the processes that make up the audio streams of a single user are equivalent to those of another user, and thus we abstract our processes to the user level. In a similar respect, we do not look at a node, or server, as a processor with an event queue, but instead see it as a single processor that is capable of the concurrent computation of a set number of user processes. Thus, if we allow each user to be represented as a single unit of load, and a server is capable of processing 50 units of load, then it is capable of hosting 50 users concurrently. We find that it is necessary to represent the processing abilities of our servers in this way because the user processes are long-lived (ranging from minutes to potentially hours), since, although the constituent processes of a user may be processed, the user will remain within the system from the moment they enter until they leave.

Our scheme has some significant differences from the literature. The biggest difference between our system and the literature is that we have two constraints to fulfill simultaneously. The first constraint is the normal load balancing constraint of maximising overall throughput whilst minimising overload. The second constraint comes about because we wish to maximise user perceived QoS. Thus, as we shall discover in greater detail in Chapter 6 each user and server pair will have an associated delay, representing both the physical distance between them and the processing delays of audio traffic. None of the literature reviewed or that we know of, combines these two constraints simultaneously.

In addition, as will be discussed in greater detail in Chapter 4, our load management architecture relies upon a series of grid squares which dictate whether a user is in close enough proximity to another user to talk. We reason that it is more difficult to let a conversation span over multiple servers, and thus we require that

users within the same grid square be located on the same server. Thus, as users move from grid square to grid square, they may also move from server to server. We assume that there are high speed connections between servers such that any hand over procedure can be conducted with minimal disruption to the end user. In addition, for the purposes of load balancing, the grid squares themselves become the load balancing entities, each with differing figures of load depending upon the number of users they are currently hosting.

3.4.1 Maintaining State Information

Within our distributed load management architecture, we maintain periodic distribution of state information. The reason for this is that we identify load of the server as the average utilization. Since it is an average, we require some time to obtain an accurate measure. There are a number of other reasons for choosing to update periodically. For instance, with a virtual environment population, users are free to move from server to server as they move from grid square to neighbouring grid square, thus an event triggered update policy has the potential to be very costly.

3.4.2 Costs

We find that there are two main costs associated with our load management architecture.

The first is the cost associated with the network. This includes the bandwidth cost to transfer tasks and to send update messages to maintain state information. In addition there is the cost to the individual nodes to hand over tasks, as well as to process update information. Whilst we do not quantify these costs, we do recognise that it is highly desirable to minimise them. This will be explored further in Chapter 4.

The second cost, is with regards to the QoS received by the end user. As the design of our load balancing architecture is to eventually encompass both load management and QoS management, load balancing decisions will be affected by changes in QoS. The main cost will be the result of an overload condition in the network. If a server becomes overloaded, we can assume that it will no longer be able to process users as effectively, with the end result being that each user on the server will have reduced QoS.

Chapter 4

Load Management Schemes

As discussed in Chapter 3 current methods of load balancing are not suitable for a distributed, real-time audio conferencing application, since we are dealing with long-lived, real-time processes. The purpose of this chapter is to investigate load balancing architectures meeting the requirements of this project. The load balancing system described here does not attempt to directly address QoS management issues as this is introduced in Chapter 7.

4.1 The Architecture

MMOGs generally have large user populations. Audio processing tends to be a processor intensive activity and hence it is likely that server farms will be required to meet load requirements. In addition, due to the geographical distance separating users, the servers may need to be located around the world, reducing the distance of the client/server connection.

As discussed in Chapter 2, the audio system utilises a grid overlay of the MMOG virtual world in order to divide the user population into a number of conversations. The users currently residing in a single grid square are assumed to be in the same conversation. A conversation held within a particular grid square is in real-time

and is perceived clearly and spatially correctly by the participants. Conversations held within grid squares adjacent to a particular grid square are deemed background noise and may not necessarily be heard with the same quality or even in real-time by the users of the particular grid square. With regards to allocation of grid squares to servers, each server can host only whole grid squares, meaning that multiple servers cannot share the task of hosting a single grid square. In addition, each server has a limited user capacity, resulting in a server potentially running out of capacity should a large number of users migrate to a single grid square.

The system may be either centralised or decentralised in nature. In a centralised system there is a central controller whose job it is to collect server information and initiate load balancing, whilst a decentralised architecture relies upon the servers themselves to manage their load management. For the purposes of analysis, the biggest influence a centralised or decentralised system has is upon the load balancing algorithm we may choose. For this investigation, we examine both centralised and decentralised algorithms.

4.2 Load Balancing Algorithms

The heart of the load balancing architecture is the load balancing algorithm. Four algorithms were chosen for analysis. Each has the characteristics of low computational complexity and relatively small number of grid square rearrangements required when load balancing is required. The four algorithms include both centralised and decentralised algorithms. For the IACMMG and DICE projects, a centralised algorithm requires a central controller that would watch over the set of audio servers, keeping up to date measurements of load for each. When a server is found to be overloaded, the central controller initiates any rearrangement. A decentralised algorithm leaves the individual servers to initiate load balancing. With the knowledge of how to contact other servers in the network, an overloaded server will make contact

with a server to see if it has spare capacity. If it does then the transfer is made, otherwise the overloaded server moves on to the next available server. We examine three decentralised algorithms: Minimum Offload, Maximum Offload and Minimum Redistribution; and one centralised algorithm: Simple Packing [38]. These are described in detail in the following section.

The servers are denoted *Server 1, Server 2, Server 3, ...* which also denotes the order in which the load balancing algorithm progresses. For example, once Server 1 becomes overloaded, it attempts to offload grid squares to Server 2 before it tries to offload to Server 3, and so on. Additionally, each user constitutes a single unit of load. Hence, the load of each grid square is the total number of users within that square. We now discuss the four load balancing algorithms in detail.

4.2.1 Minimum Offload

The grid squares residing upon the overloaded server are sorted from smallest to largest in terms of the amount of load they contribute (number of users per grid square). If there is a tie, then grid squares are sorted by grid square identification number.

The overloaded server establishes contact with one of the alternate servers within the network. The overloaded server asks for a measurement of spare capacity on the alternate server. If the amount of spare capacity is greater than or equal to the size of the overloaded server's smallest grid square, then the smallest grid square is reassigned to the alternate server. If the overloaded server is still in an overloaded state, it continues to offload grid squares to the alternate server until either: the overloaded server is no longer overloaded, in which case the algorithm is successful; or, the alternate server runs out of spare capacity. If the latter occurs, the algorithm moves on to the next alternate server and so on, until the overloaded server is no

longer overloaded, or we run out of servers to offload to. Should there be no more servers to offload grid squares to, then the overloaded server remains overloaded, until the above procedure is repeated.

There are a number of advantages and disadvantages of the Minimum Offload algorithm. For example, if we consider the scenario of a heavily loaded system, the Minimum Offload algorithm is able to more readily utilise the small pockets of spare capacity found on alternate servers. However, the downside of this algorithm is that in order to remove any significant amount of overload, we must offload a potentially large number of grid squares. Whilst the effect of grid square movement on the end users is not entered into in any detail in this document, we believe it to be something we wish to minimise, as it would be either annoying to the end user due to possible loss of audio during the move, or expensive for the network if done invisibly to the user.

4.2.2 Maximum Offload

This algorithm is identical to the Minimum Offload algorithm, except that grid squares are sorted in *descending* order of size. In this respect, any grid square moved from the overloaded server is the largest the server is hosting. If the largest grid square cannot be offloaded to the first available server, then the second and subsequent servers are questioned. As with Minimum Offload, this algorithm continues until either the overloaded server is no longer overloaded, or there are no alternate servers with spare capacity to offload to.

The algorithm attempts to offload the largest grid square from the overloaded server and hence results in the redistribution of fewer grid squares than the Minimum Offload algorithm. However, in a heavily loaded system the Maximum Offload algorithm will have difficulty finding a server with enough spare capacity to accept

a large grid square.

4.2.3 Minimum Redistribution

The Minimum Redistribution algorithm is similar to the Maximum Offload algorithm, with the difference being that instead of offloading the first and largest grid square, we attempt to offload the smallest grid square which would sufficiently decrease the load on the overloaded server to take it out of overload. The overloaded server calculates the amount of load that is required to be offloaded in order to relieve the overloaded state. If this amount is larger than the first and largest grid square, then this square is offloaded, otherwise the grid square whose load exceeds the excess overload by the smallest margin is offloaded. As with Maximum Offload, this algorithm continues until either the excess overload is eliminated, or it fails to find any alternate servers to offload to.

4.2.4 Simple Packing

Simple packing is a centralised algorithm and as such the servers of the system communicate directly with a central controller. The central controller keeps track of the level of load of each server, as well as the location of each grid square and their respective load. Once a server is found to be overloaded, the grid squares of the entire system are sorted in descending order of load. The algorithm then selects a server and proceeds to allocate grid squares to it, trying to pack it with as much load as possible. The algorithm continues on with the next server and so on until either the sorted list of grid squares is exhausted (we have a feasible packing) or there are no longer any servers with spare capacity. If the latter occurs, then the remaining grid squares are allocated to the server they were originally assigned to.

Grid squares are packed in the following manner. If the grid square at the head of the list has a load that does not exceed the capacity of the server, then it is accepted.

Otherwise, the list is iterated through to find a grid square that will fit. This process is repeated until either the server no longer has any spare capacity, or there are no grid squares within the list with a load not exceeding the spare capacity of the server.

The benefit of having greater control over the system is that the Simple Packing algorithm can tightly pack each server even when the system is heavily loaded, ensuring that the event of servers becoming overload is minimised. Being a centralised algorithm, it requires a central controller collecting and collating information from the rest of the network, and this requires greater overhead than the distributed algorithms. In addition, the nature of the algorithm requires that a complete reassignment of the grid squares of the system be performed each time the algorithm is used. This results in a large number of grid square movements which may prove expensive.

4.3 Simulation Analysis

4.3.1 Simulation Architecture Assumptions and Parameters

The following are a number of assumptions made for the purposes of simulation:

- As a user moves within the game world, the grid square and server they are residing on changes. User movement is recreated within the simulation, with each user moving independently of all others. Every 10 seconds, each user has a fixed probability of moving from one grid square to a surrounding grid square or staying in their current grid square.

- Each server within the system periodically measures its load, which can then be used to determine if the server is overloaded. To ensure an orderly process of coordinated load balancing, server load measurement is performed in specified cycles. The reason for performing cyclic measurement is to ensure

that each server gets an equal chance to measure and perform load balancing, if necessary. It also stops the *herd effect* [35], where overloaded servers simultaneously offload to a lightly loaded server, swamping it. To highlight the bias of using a cyclic ordering, three forms of measurement cycle are used: *forward*, *backward* and *random*. These dictate the order in which servers are measured. *Forward* cycles through the servers from the first one in the list to the last; *backward* the opposite; and *random* cycles through the list of servers in a random order each time.

- Initially Server 1 is assigned all of the grid squares of the system, and thus performs all audio processing. As more users are introduced into the system, the spare capacity on Server 1 reduces, until it becomes overloaded. Once in an overloaded state, the algorithm attempts to move grid squares from the overloaded server to any servers with spare capacity to remedy the overloaded state.

We now examine a series of scenarios to test the effectiveness of each algorithm.

Scenario 1:

- The simulation run length is set at 10000 seconds.
- The network consists of four servers, each with a capacity of 50 users.
- The load on each server is measured every 30 seconds, initially using a forward cycle. Reverse and random measurement cycle methods are used in further simulation runs.
- The virtual world is made up of 40 grid squares, arranged in a 5 by 8 grid.
- Users may not leave the system. Once placed within the virtual world, a user remains until the simulation ends.

- The population of the virtual world is fixed at 200 users (the maximum capacity). Users enter the system at random locations, every 0.5 seconds, until the number of users reaches 200. Once this occurs, users are no longer admitted to the system. Whilst a system such as this running at maximum capacity is unrealistic, it does provide us with good insight as to the ability of the load balancing algorithm under heavy load conditions.

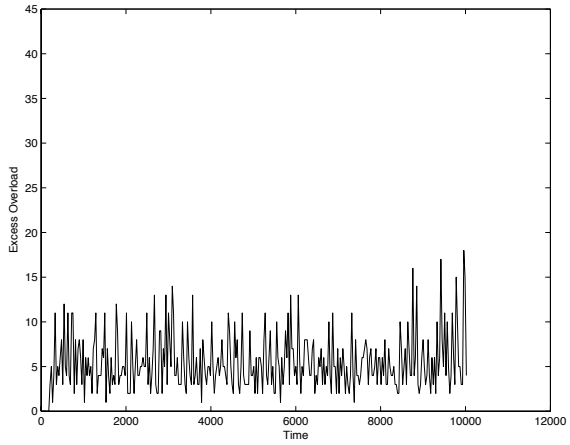
- Every 10 seconds from the moment they enter the system, a user has a chance to move from their current grid square to a neighbouring grid square. This process is random, with a probability of 0.2 of moving to a neighbouring grid square. Otherwise the user remains in their current grid square and must wait a further 10 seconds for another chance to move. A neighbouring grid square is defined as a grid square that shares a common edge or vertex with the current grid square. As the grid of the simulated system is arranged in a rectangular pattern, each grid square can have either three, five or eight neighbouring grid squares depending upon their location on the grid. When determining which grid square a user moves to, the grid square is chosen uniformly from the list of neighbouring grid squares.

The graphs of Figure 4.3.1 plot the *Excess Overload* over the running time of the simulation for the four load balancing algorithms. We define *Excess Overload* as:

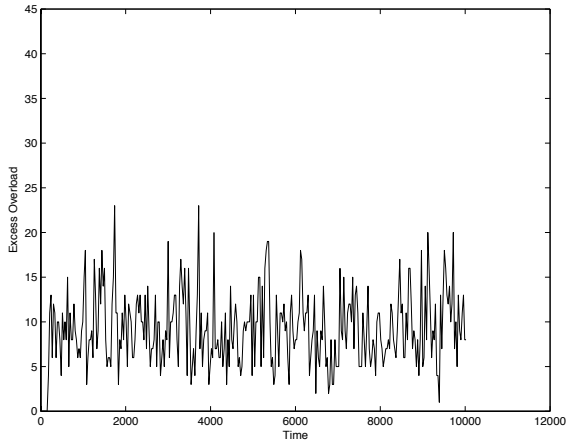
$$Excess\ Overload = \sum_{j=1}^{4} (L_j - 50)^+, \tag{4.3.1}$$

$$where\ x^+ = \begin{cases} 0 & \text{if } x \leq 0 \\ \text{x} & \text{if } x \geq 0, \end{cases} \tag{4.3.2}$$

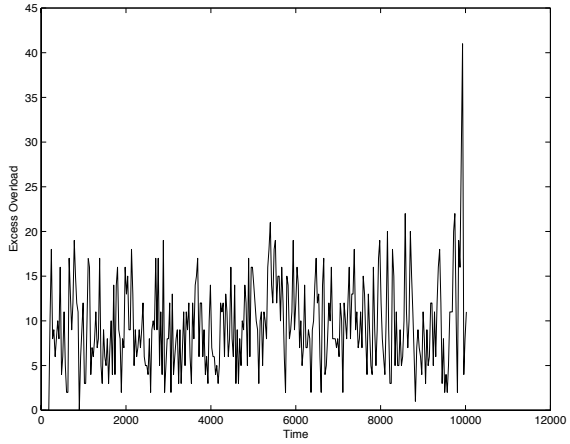
and L_j is the current load of server j. This is the total amount of load above the maximum capacity figure for each server after they have had a chance to balance their load. Thus, if a server has a load of 51 after load balancing, then the excess overload is one unit. A load of 50 or less will result in a value of excess overload of



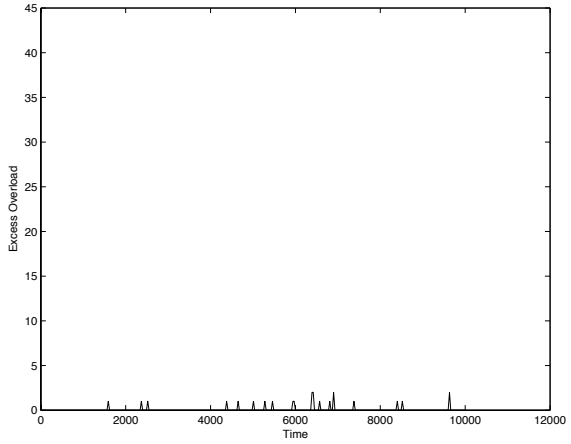
(a) Minimum Offload.



(b) Maximum Offload.



(c) Minimum Redistribution.



(d) Simple Packing.

Figure 4.3.1: Excess Overload for the four load balancing algorithms.

0. Figure 4.3.1 shows the total excess overload for the four servers within the network. As we can see in Figure 4.3.1(d), the centralised algorithm, Simple Packing, can withstand the heavily loaded system with ease, rarely leaving servers overloaded. On the other hand, all three of the decentralised algorithms have trouble keeping the servers out of overload. However, the algorithm Minimum Offload (Figure 4.3.1(a)) appears to perform the best, keeping excess overload lower than either Maximum

Offload or Minimum Redistribution. Calculating the mean and standard deviation of the results of the four algorithms, yields the following results.

Table 4.3.1: Mean and Standard Deviation Pairs

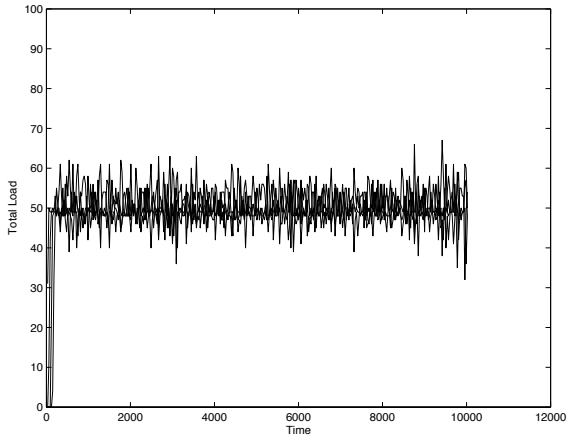
Algorithm	Mean	Standard Deviation
Minimum Offload	5.4401	3.1700
Maximum Offload	9.2246	4.1227
Minimum Redistribution	9.4401	5.1577
Simple Packing	0.2972	0.2972

Performing two-sample t significance tests comparing the results of each of the four algorithms with each of the other result sets, enables us to conclude that: Simple Packing is able to significantly outperform the other algorithms with a confidence of greater than 99.9%, and Minimum Offload is able to outperform both Maximum Offload and Minimum Redistribution with a confidence of greater than 99.9%.

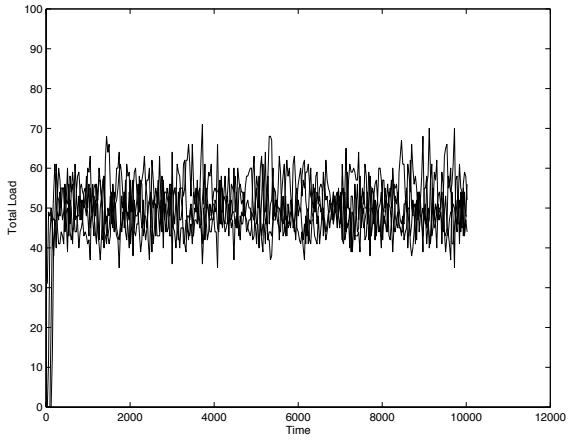
Figure 4.3.2 gives plots of load for each server at the end of each cycle (after they have all had a chance to balance load), for the length of the simulation run. Unsurprisingly, Simple Packing (Figure 4.3.2(d)) gives the best performance of the four algorithms. The algorithm, Minimum Offload (Figure 4.3.2(a)), is the best performer of the decentralised algorithms, providing a tighter range of load around the maximum server capacity of 50 units.

4.3.2 Implementation Issue

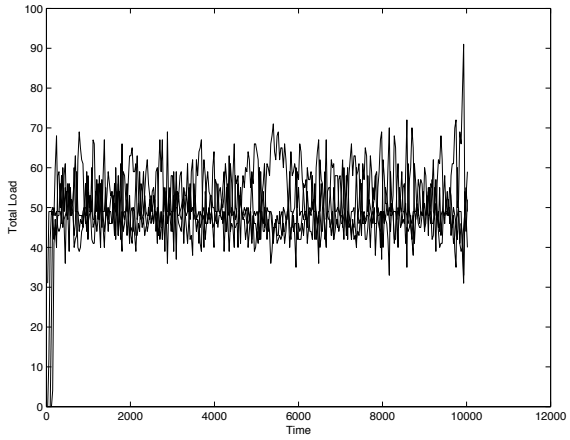
The Simple Packing algorithm was clearly the best performer, however the cost involved in the use of a centralised algorithm that would reorganise the grid squares significantly is believed to make it prohibitive. Thus we proceed with further investigation of the best decentralised algorithm: Minimum Offload.



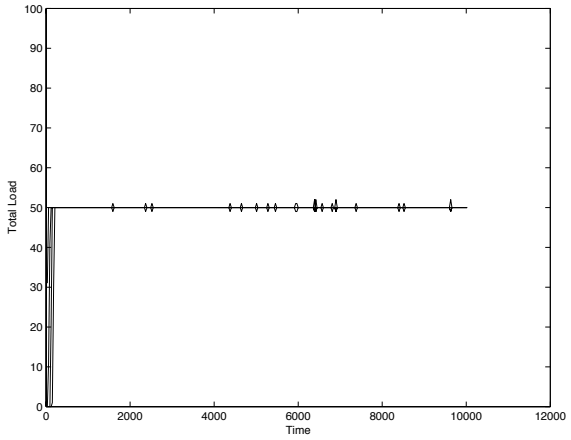
(a) Minimum Offload.



(b) Maximum Offload.



(c) Minimum Redistribution.



(d) Simple Packing.

Figure 4.3.2: Total Load for the four load balancing algorithms.

Having investigated the capability of the Minimum Offload algorithm with a fixed inter-measurement time of servers at 30 seconds, we tested the algorithm with a number of different inter-measurement times. The system was parameterised the same except that the time between server load measurement cycles was varied, and the simulation run time was increased to 1,000,000 seconds. The inter-measurement

times selected for analysis were: 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 seconds. In addition, the simulation was run utilising the three types of measurement cycle: *forward*, *backward* and *random* and each simulation run was executed ten times. This allows for a 95% confidence interval to be established for each parameter set.

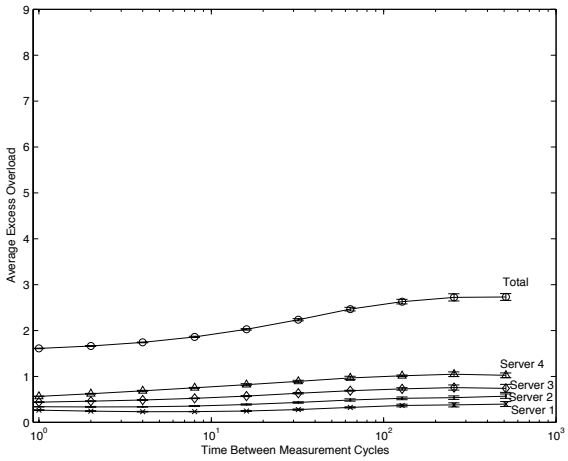
Instead of plotting the excess overload after each measurement cycle, the average excess overload for the length of each simulation was plotted. Figure 4.3.3(a)-(c) are logarithmic plots of the average excess overload over our range of inter-measurement times. We present the results with three levels of information completeness. Figure 4.3.3(a) depicts a system in which we have complete information. When a server wishes to load balance, each of the other servers measures its load to ensure that they are reporting accurate, timely, information. Figure 4.3.3(b) looks at the same system but in which measurement of a server is only performed once per measurement cycle, immediately before their attempt to load balance. Figure 4.3.3 (c) explores a hybrid of the two measurement systems. Immediately before the measurement cycle is to begin, all servers measure their load, and this information is distributed to every other server within the network. Having gathered up-to-date information, the algorithm proceeds in the same way as the algorithm with no further information. Each server has its chance to measure its load and load balance if necessary. It is envisaged that by measuring all of the servers immediately before the measurement cycle, will reduce the problems associated with perceived knowledge, as opposed to complete knowledge, of the network. The plot consists of five lines, four of which plot the average excess overload for each server, and one to plot the total average excess overload. In addition, the variability between simulation runs is displayed in the form of the error bars for each point. Each individual plot will receive further explanation in due course. However, in general if we look at the line of total average excess overload for each plot, we see an expected result. As the time between measurement cycles increases, the average excess overload also increases to a limit. Due to the random nature of user movement within the virtual world, as the

time between measurement cycles increases, the correlation between the state of the system from measurement cycle to measurement cycle decreases and the average excess overload tends to that of a system without load balancing. Note that the static number of players means that the average excess overload remains bounded.

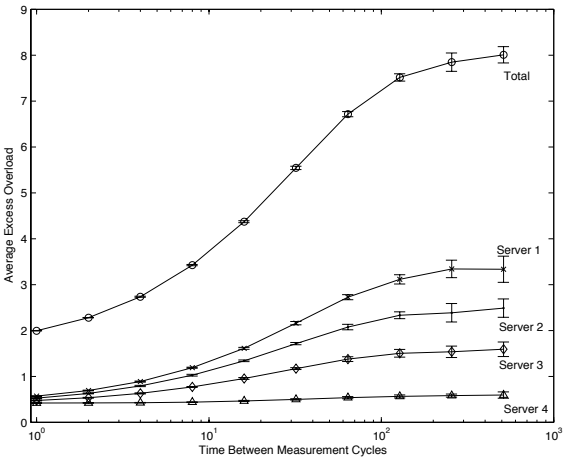
When gathering the results we theorised that each server would perform differently depending upon the order in which they are measured. Our reasoning was that due to the decentralised nature of the Minimum Offload algorithm, each server is greedy in its load balancing, with no concern for the other servers within the network. Thus if we use a forward measurement cycle, Server 1 has the best opportunity to offload any excess load. By the time Server 4 gets its chance to load balance, the other servers will presumably have little free capacity since they have already been used in load balancing and thus Server 4 has a high possibility of remaining overloaded. Figure 4.3.3(a) shows what we expected. Server 1 has the best performance, followed by Server 2, 3 and finally 4.

Figure 4.3.3(b) is the plot of the results we originally observed. Apart from the sharp increase in average excess overload, the most noticeable difference is that the order of performance of the servers is reversed. The reason for this difference is that servers are only able to measure their load once per measurement cycle, which is a reasonable restriction in a production environment. Between measurement cycles, users are able to move from grid square to grid square and thus from server to server. The result of which is that when a server load balances, it has a stale picture of the rest of the network. Table 4.3.2 steps through a single measurement cycle for our exemplar system.

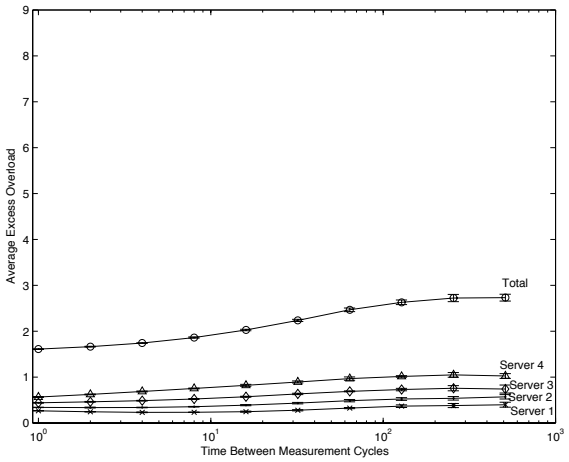
Initially we assume that from the previous measurement cycle, each server ended with load at maximum capacity (50 units). Reading across row 1, Server 1 measures its load and finds that it is overloaded, with a load of 55 units. The other servers



(a) Maintaining Complete Load Information.



(b) Maintaining No Additional Load Information.



(c) Maintaining Partial Load Information.

Figure 4.3.3: Average Excess for Minimum Offload using Forward Measurement Cycle.

have not yet had a chance to measure their load and so their load as perceived by Server 1 is 50 units each. Server 1 attempts load balancing but upon finding all other servers at maximum capacity fails.

Table 4.3.2: Limited Information

	Server 1	Server 2	Server 3	Server 4	Total
Load Perceived By Server 1 After Measurement	55	50	50	50	205
Load Perceived By Server 2 After Measurement	55	52	50	50	207
Load Perceived By Server 3 After Measurement	55	52	47	50	204
Load Perceived By Server 4 After Measurement	55	52	47	46	200

Reading across row 2, Server 2 now proceeds to measure its load and finds it to be at 52 units. Like Server 1, Server 2 attempts load balancing only to find that the other servers are either at or above maximum capacity. Note that Server 1 is reporting an accurate account of its current state, and Servers 3 and 4 a stale account, Server 2 has no ability to separate the two. By this time, the perceived total load of the network is 207 units, greater than the actual load of 200 units.

Reading across row 3, Server 3 now proceeds to measure its load and finds it to be at 47 units. Thus its actual load is 3 units less than what was perceived by Servers 1 and 2 when they were measured. Similarly, when Server 4 measures its load it finds that it is at 46 units, ensuring that our figure of total load is still 200 units.

In the same way that servers can report that they have more load then they actually do, they may also report less, resulting in grid squares being offloaded to already overloaded servers.

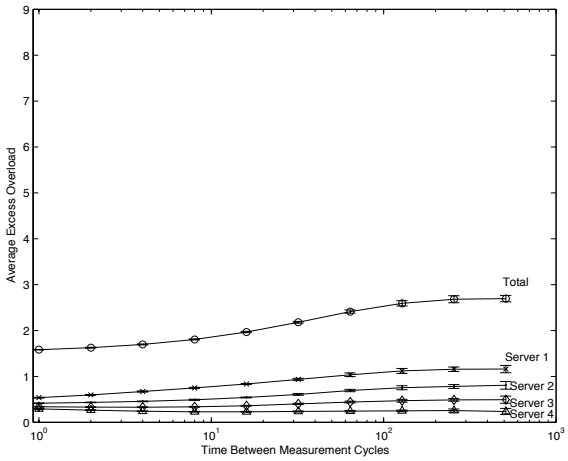
Such a situation is clearly untenable and hence we enhance the measurement framework to ensure accurate information is used during load balancing and thus to achieve the results of Figure 4.3.3(a). We do this by having each server measure and report its load at the beginning of the measurement cycle, and by further updating load balances as each server performs balancing. In Figure 4.3.3(c) we find that the use of the partial measurement scheme as previously described, produces near identical results to a system with complete information. Part of the reason as to why the results gained with partial measurement are so good is that the time between server measurement is small, the result of which is that little change in server load can occur between servers measuring.

What is immediately obvious from the results of Figure 4.3.3 is that the completeness of information plays a significant role in the overall quality of the system.

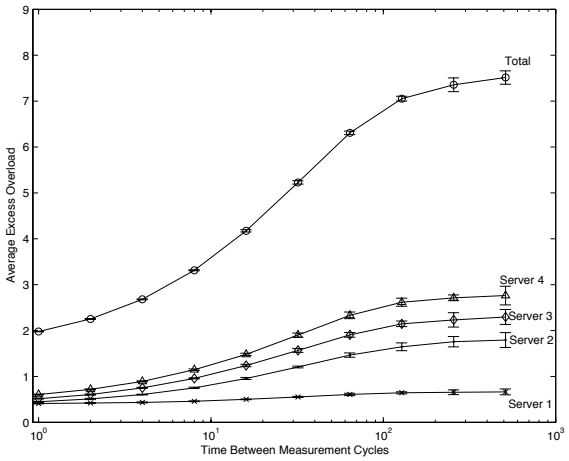
The plots of Figure 4.3.4 explore the same system as before except that we are using a backward measurement cycle. The results are similar and show the same problem that stale information has upon the network.

The plots of Figure 4.3.5 explore the use of a random measurement cycle. As with both the forward and backward measurement cycle cases, stale information has a marked effect upon the system. The difference is that due to the random nature of the measurement cycles, stale information has an equal effect upon all servers within the network.

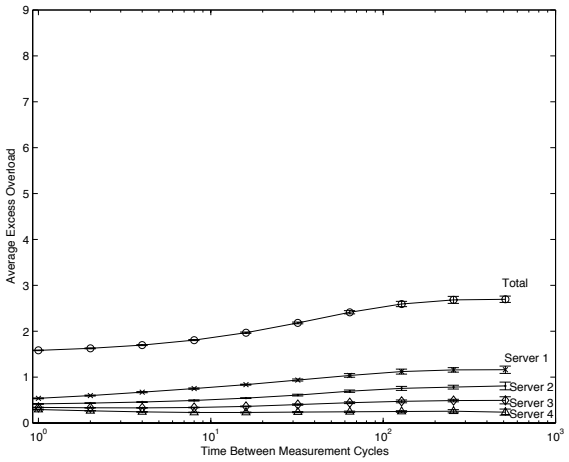
The results presented in Figure 4.3.3, Figure 4.3.4 and Figure 4.3.5 indicate that a significant requirement is for information to be kept current. The consequence of using stale information is that the average excess overload increases substantially as we increase the time between measurements.



(a) Maintaining Complete Load Information.



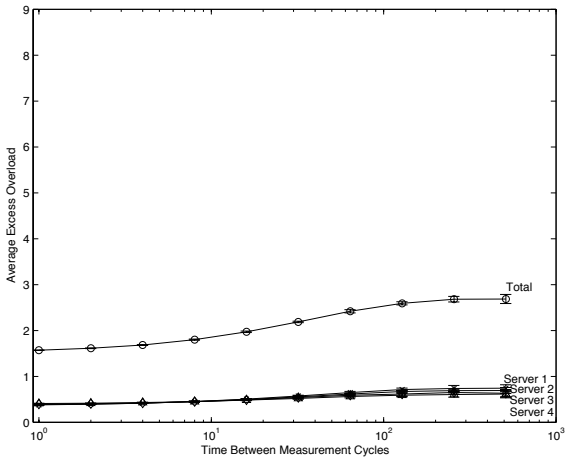
(b) Maintaining No Additional Load Information.



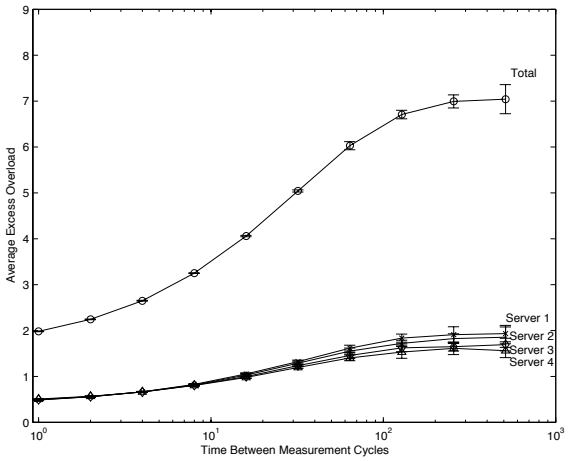
(c) Maintaining Partial Load Information.

Figure 4.3.4: Average Excess for Minimum Offload using Backward Measurement Cycle.

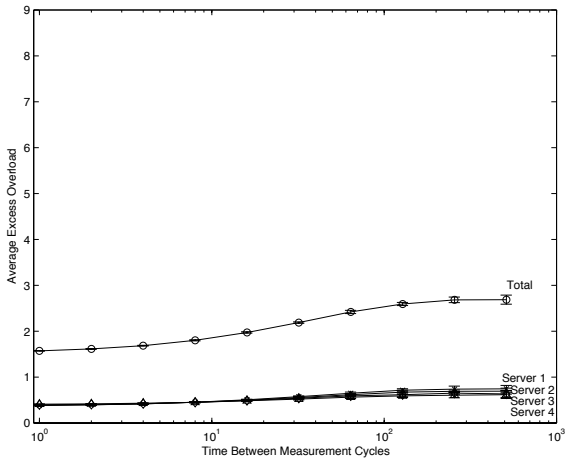
In conclusion, we find that there is a marked improvement in performance as the period between load measurement, and hence load balancing, is reduced. More



(a) Maintaining Complete Load Information.



(b) Maintaining No Additional Load Information.



(c) Maintaining Partial Load Information.

Figure 4.3.5: Average Excess for Minimum Offload using Random Measurement Cycle.

importantly, we discovered the importance that current information has upon the performance of the network. It is at this point that we must decide as to the level of importance complete information has on the system. On one hand, by restrict-

ing server measurement to a server's preordained turn means that we keep network costs associated with server measurement to a minimum. However, we run the risk of unnecessary grid square movement due to the use of stale information. The impact of this is that user quality may be reduced as grid squares are moved from one overloaded server to another, possibly even more overloaded, server.

On the other hand, to maintain complete information of the system requires that when a server needs to load balance, all other servers within the network potentially have to remeasure their load. This increases the network costs and the time required in gathering server measurements increases the overall time of load balancing. However, at the same time complete system knowledge reduces the number of unnecessary grid square movements performed and increases the overall performance of the network.

Maintaining partial information, in that each server is allowed to measure and distribute their load statistics at the beginning of a measurement round, reduces average excess overload significantly. With this minor modification in measurement methodology, maintaining partial information, the performance is only slightly worse than if complete information is maintained. In addition, maintaining partial information requires significantly less overhead, in terms of measuring and distribution of load statistics, than the complete information scheme.

Thus, we recommend that all servers within the network measure and distribute their load information before the load balancing cycle begins.

4.3.3 Further Analysis

Having examined the case in which the population of the virtual world was fixed, we now examine the more realistic situation in which simulated users may also leave

the system. We modify our movement rules such that when a user is permitted to move, they have a probability of 0.05 of leaving the system, reducing the probability of staying in their current grid square to 0.75 (the probability of moving remains at 0.2). In addition, our maximum population cap of 200 users is removed. On average, the population of the system remains at 200 users but the instantaneous population can vary significantly.

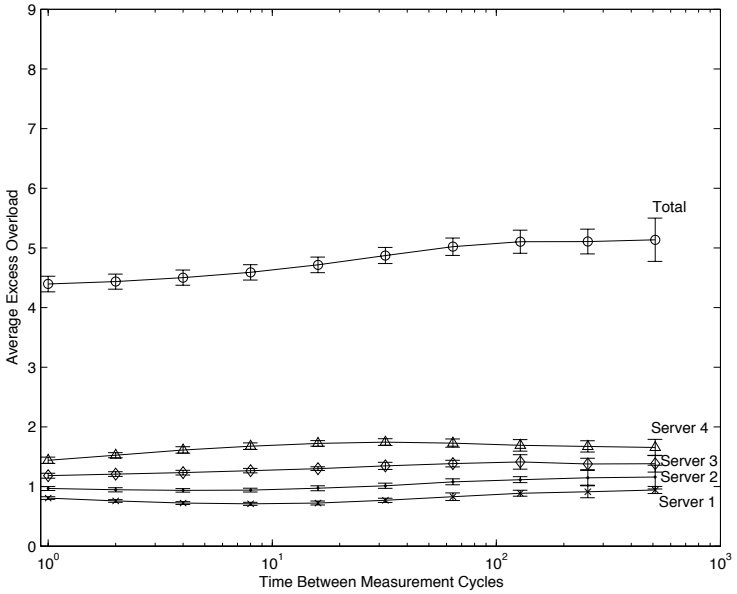


Figure 4.3.6: Average Excess for Minimum Offload using Forward Measurement Cycle.

Figures 4.3.6 - 4.3.8 show the results of the variable population simulation for each type of measurement cycle (forward, backward and random respectively). As it was established in the previous section, by maintaining partial information we are still able to maintain small levels of excess overload, with only a small increase in bandwidth usage. Thus our results reflect this by only considering the case where partial information is maintained.

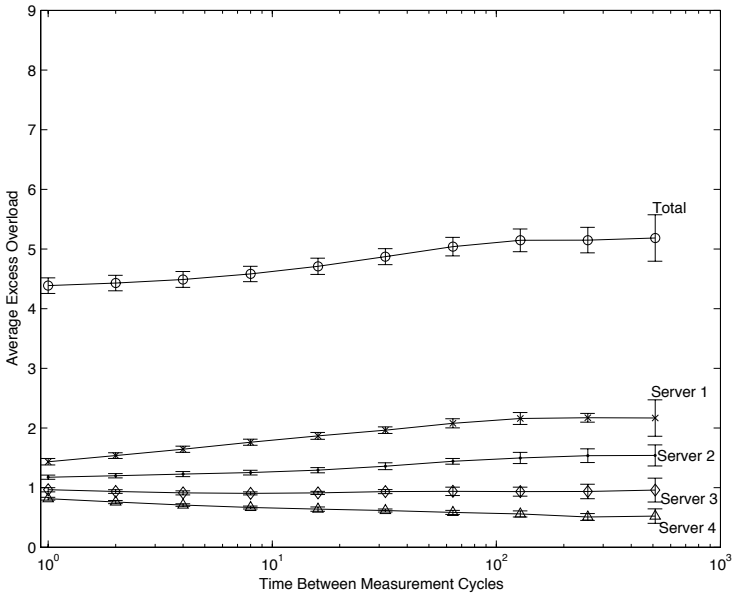


Figure 4.3.7: Average Excess for Minimum Offload using Backward Measurement Cycle.

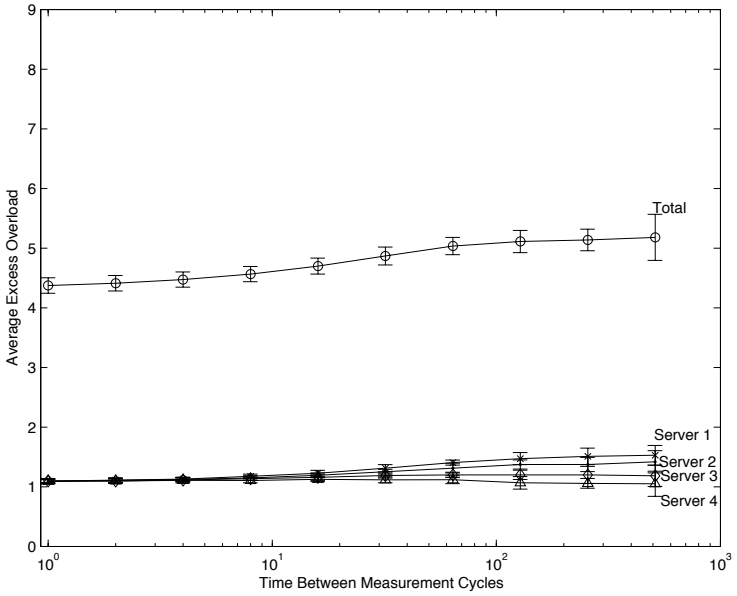


Figure 4.3.8: Average Excess for Minimum Offload using Random Measurement Cycle.

The most noticeable differences between the results gained from the the variable and fixed population systems is that the figures of average excess overload are much higher when the system has a variable population and that the variance between simulation runs is significantly higher.

The first difference is due to our result metric. Since a result is only observed when a server is found to be overloaded, a network that is overloaded past maximum capacity will result in a greater excess overload. With a variable population, there are periods of time when the instantaneous population of the system is far greater than the maximum capacity, resulting in a large excess overload. Conversely, there are also periods of time when the instantaneous population is far lower than maximum capacity. However, the metric does not consider this, resulting in only the times when the network is under strain being recorded.

A curious result is found when examining the differences between the results of forward and backwards measurement (Figures 4.3.6 and 4.3.7). Whilst both give a similar total result, the plots for the servers of Figure 4.3.6 appear to level off. It would be expected to give a similar result to that given in the backward case. The reason for this difference is due to the load balancing algorithm's preference of offloading grid squares to servers in numerical order. Thus the algorithm will first attempt to offload to server 1, then server 2 and so on. The consequence of this is that any grid squares of zero size, and thus no users, will be offloaded to server 1. When users move through the system, these empty grid squares will fill and the server may become overloaded. This is more obvious when server inter-measurement time is large and empty grid squares have a greater chance of filling. It is also for this reason that there is a greater variability of results in the backwards case of Figure 4.3.7.

Having investigated the effect that changing server inter-measurement time has on systems with both fixed and variable populations, we can conclude that a greater

frequency of measurement will result in lower excess overload. This is to be expected, as the system is given more opportunities to balance server load. Unfortunately, load balancing more often may result in an increased cost, in terms of bandwidth and user QoS, to the system. Thus a compromise between excess overload and potential network cost is required. By maintaining partial information, we find that even for a variable population, the excess overload resulting from an increase in server inter-measurement time only increases slightly. Thus larger server inter-measurement times can be chosen which will result in only a small negative effect to the network. However, larger variability of results is introduced as server inter-measurement time increases. From a user perspective, larger server inter-measurement times may result in longer periods of time for which a user must tolerate potentially poor QoS caused by excess server load. Whilst a user may tolerate poor QoS for a short period of time, say a minute, it is unlikely that larger delays will be tolerated. For these reasons, we recommend that the server inter-measurement time be somewhere in the range of less than a minute.

In addition, we recommend that a random measurement cycle be chosen, as it produces a result that is equal to either forward or backward cycles, and yet allows for greater more consistent performance by all servers.

4.4 Summary

A review of the literature in Chapter 3 has enabled us to design and implement a load balancing architecture for the IACMMG and DICE projects. Through simulation we have been able to gauge the effectiveness of a number of load balancing algorithms. The algorithms in question are arranged into two categories: centralised and decentralised, where the centralised algorithm requires further infrastructure in the form of a central controller. Through simulation analysis, we found that the centralised algorithm, Simple Packing, has the best performance out of the four

algorithms investigated. However, this performance is at the cost of system-wide reassignment of grid squares each time the algorithm is initiated. As the load upon the system increases, the need to load balance also increases. Thus having to regularly rearrange all of the system's grid squares would impose a significant cost which is assumed to be prohibitive. For this reason, the best decentralised algorithm, Minimum Offload, was examined closely for its suitability. Whilst not being able to perform nearly as well as the Simple Packing algorithm, it is superior in performance when compared to the other two decentralised algorithms, that we proposed.

Following our examination of the performance of the four load balancing algorithms, we investigated the problem that stale information has within a real-time system. This brings to light the need to ensure that state information is accurate, either through shorter periods of time between measurement updates, or by waiting until all measurements are gathered before grid squares are moved.

Throughout this chapter, we have investigated the problem of maintaining the IACMMG and DICE projects over a number of servers, with the emphasis of ensuring that the servers do not become significantly overloaded. However, we are dealing with a real-time, audio based technology, where end users require a certain level of QoS. The problem is that each server can give a different level of QoS for each user due to geographical distance. Thus, our current load balancing architecture is incapable of directly providing adequate levels of QoS. To this end, the following chapters introduce the concept of QoS, and present an algorithm for ensuring acceptable levels of QoS that is then integrated with the load balancing architecture presented here.

Chapter 5

Discussion of QoS Management

5.1 Goals and Definitions

The primary goal of Quality of Service Management with respect to a distributed audio conferencing application is to ensure that each user receives an adequate level of audio quality. If we look at our exemplar system, the IACMMG project, the idea of providing an adequate level of audio quality to each user translates to providing an adequate level of audio quality to each grid square within the system and in turn to each user within a grid square. However, what do we mean by an adequate level of audio quality?

QoS with respect to audio comes down to a user's perceived audio quality. However as each user is different, how they perceive a sample of audio will also differ. Difficulty arises in that we cannot poll each user to determine whether or not the audio stream that they are receiving is what they deem to be adequate. A standard technique in human factors research is to use Mean Opinion Scores (MOS) [19] in order to provide a standardised numerical value of perceived audio quality. Questionnaires are used to gather user's opinions of a set of audio samples, which are in turn collated to produce a set of Mean Opinion Scores. The MOS ranges from

a value of 1, representing unusable quality audio, up to a value of 5, representing excellent quality audio. A score of 3 is deemed as fair quality, but it is not until we reach a value of approximately 3.6 that the audio sample is deemed to be acceptable [24]. Whilst it is useful to have some form of numerical scale with which to evaluate an audio stream's quality, the problem remains as to how to measure the quality of an audio stream. In order to do this we must identify the factors that contribute to the quality of an audio stream when distributed over a data network.

QoS in terms of voice can be attributed to a number of factors. The quality of the equipment used by the client and the physical environment of the client both affect the quality of service received [20]. From the point of view of trying to provide an audio conferencing application, one can recommend specific equipment for clients to use, but this doesn't necessarily mean that they will use it. Thus there is little control over many of the factors that contribute to QoS. However, the factors that we have some ability to control are delay and loss of audio packets. By reducing the delay experienced by packets through the network from source to receiver, and by minimising loss of audio packets, we are able to maximise the QoS for the user. The problem is that we do not have any physical ability to directly affect the delay and loss between two points within the network as the network itself is not under our control. What we are able to do however, is detect when delay and loss become so large as to reduce the quality of the connection. As we shall see in the next chapter, once we have found that a particular connection is providing insufficient QoS, we are able to intelligently reassign the affected grid square to another server in order to provide adequate service.

It has been established that delay and loss have great influence over the overall quality of an audio transmission [24], and from the point of view of a service provider, they are the only two factors that can be both measured and used to give a measure of service quality. Later in this chapter we discuss techniques for measuring both the

delay and loss affecting a connection. However, we are still left with the problem of translating delay and loss statistics into a figure for the quality of the audio connection and hence an estimation of the level of QoS a user will receive. The ITU-T E-Model [24], which is discussed in detail in the next section provides a means by which delay and loss statistics gathered from the network can be transformed into a usable metric of audio quality.

5.2 Measuring Audio Quality

In order to ensure that users receive adequate levels of QoS, we need to reduce delay and loss to acceptable levels along the path between the client and the server. However, before we can begin to try to reduce these factors, we must first be able to identify what values of delay and loss produce inadequate levels of QoS. A mechanism is required which can translate these figures of delay and loss into some qualitative measure of QoS. The industry standard mechanism is known as the ITU-T E-Model.

5.2.1 The ITU-T E-Model

The ITU-T provides a model [24] known as the 'E-Model' for calculating the audio quality of a two party connection. The E-Model uses a number of impairment factors to determine an overall transmission rating factor 'R' of audio quality. R-values range from 0 to 100, with 0 representing unacceptable quality and 100 being very good quality. For the purposes of judging audio quality, an R-value of 50 or less is considered so poor that all users will be dissatisfied [22]. An R-value of 70 or more is considered satisfactory, and it is at this level that the service is recommended for use [20]. In addition, the R-value produced by the E-Model can be further translated easily into a MOS value if that is required. For the sake of comparison, an R-value of 70 translates to a MOS of 3.6.

Unfortunately, the E-Model is not designed for use with multi-party Voice over

IP conversations, and thus may be inadequate. Work has begun, both by independent researchers [47] as well as by the ITU-T, to improve the E-Model for use with Voice over IP. We are unaware of any research into determining QoS in multi-party environments. Since the IACMMG and DICE projects rely upon users connecting to a server to converse, the greatest delay end-to-end path for a particular user becomes the path from that user to the server, and then from the server to the user within the conversation with the largest delay. As there are no industry standard mechanisms for estimating audio in a multi-party environment, the E-model will be used here as a first approximation for providing insight into the audio quality derived.

The transmission rating factor R is given by [24]:

$$R = R_o - I_s - I_d - I_{e,eff} + A, \tag{5.2.1}$$

where R_o represents the signal-to-noise ratio, including circuit noise and room noise; I_s is a collection of impairments which occur simultaneously with the voice signal; I_d represents impairments caused by delay; $I_{e,eff}$ represents impairments caused by low bit-rate codecs and impairments due to packet loss; A is the advantage factor and compensates for the impairment factors when there are advantages that the network provides. For example, users are more willing to put up with poorer audio quality with regards to mobile telephones for the convenience of being able to give and receive telephone calls in places where fixed lines cannot reach [36].

The factors in formula (5.2.1) can be broken down into sub-factors, which are described in detail in [24]. Of these, the majority are of little interest here and will be given default values in our analysis. The remaining factors of interest are:

- Absolute Delay in echo-free Connections (T_a): This is the one-way delay experienced from the server to the user, or vice versa. This factor is measured in ms and has a range of 0 up to 500 with a default value of 0.

- Round Trip Delay in a 4-wire Loop (T_r): Due to the fact that there is no 4-wire loop with regards to packet based audio, this factor is set to twice the absolute delay T_a. This factor is measured in ms and has a range of 0 up to 1000 and has a default value of 0.

- Mean one-way Delay of the Echo Path (T): Since echo can only occur from the length of the audio path, T becomes equal to the absolute delay T_a as long as echo is present in the system.

- Equipment Impairment Factor (I_e): This factor takes into consideration all of the impairments that may occur due to the use of low bit-rate codecs. This factor has a permitted range of 0 up to 40 and a default value of 0.

- Packet-loss Robustness Factor (Bpl): This factor is concerned with a particular codec's ability to cope with packet-loss. It has a range of 1 up to 40 and a default value of 1.

- Random Packet-loss Probability (Ppl): This is the probability that packets are randomly dropped from the system. This factor has a permitted range of 0% up to 20% and a default value of 0%.

The factors T, T_r and T_a are incorporated within the factors I_s and I_d and are directly related to changes in the R-value due to delay and possibly echo. The factors I_e, Bpl and Ppl are all incorporated within the factor $I_{e,eff}$, and are directly related to changes in the R-value due to loss in the network. Due to the additive nature of (5.2.1), the effects of loss and delay are independent.

There is also an associated potential problem introduced by echo, which is briefly discussed here and in greater detail in [8]. Echo occurs when the sender receives a sample of their own speech a considerable time after it was initially made and sent to the receiver. With regards to the quality of the audio, the larger the delay between the original sample and when the echo is received, the worse is the effect of

the echo. The same is true for the loudness of the echo, with a louder echo causing a lower quality transmission. The E-Model is designed to incorporate echo into its quality estimations and so echo can be easily incorporated via the E-Model later on and does not directly affect our investigations. Further examination of echo is also beyond the scope of our analysis.

We now consider the delay and loss experienced by packets and how these factors affect the overall quality of the transmitted audio because we have some ability to control them.

The rest of the examination of the ITU-T E-Model will focus upon estimated audio quality results obtained using the E-Model. As each audio codec is different, they have different impairment and packet-loss robustness factors. The codecs discussed in Chapter 2 have the following impairment factors [23]: G.711 has a value of 0, G.729 has a value of 11, and G.723.1 has a value of 15. Using these values, along with the packet-loss robustness factors for each codec (25.1 for G.711, 19.0 for G.729, and 16.1 for G.723.1 [25]), and measured values for delay and loss, an estimate of the expected quality of the transmission can be made.

Figure 5.2.1 is a graph of E-Model rating R versus delay for the three codecs discussed, with a 0% loss rate. From this figure an understanding of the difference in audio quality between the three codecs without loss can be gleaned. As one can imagine, the higher bit rate of G.711 results in a significant improvement in the resultant audio quality. However, for an approximate 10% reduction in the quality of the audio, the bandwidth required by the audio stream can be reduced to an eighth of its original size. To allow access to audio conferencing applications over low bandwidth Internet connections, it may be necessary to use a lower bandwidth and thus lower quality codec. What is also important about this figure, is that even for the low bit rate codec G.723.1, acceptable quality audio can be achieved with

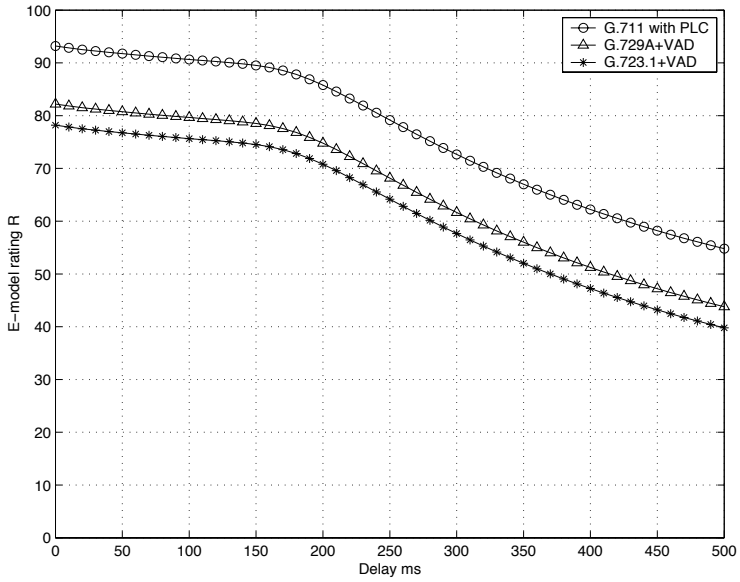


Figure 5.2.1: E-model rating R with G.711, G.729 and G.723.1 with 0% loss and all other parameters set at their default values (T=Ta=Tr/2).

delays of up to approximately 200 ms.

Figures 5.2.2 - 5.2.5 are similar to Figure 5.2.1 in that they all show the effect that delay has upon the three nominated codecs. The difference is, that they all show different rates of loss. Figure 5.2.2 shows the effect a loss rate of 0.5% has upon the three codecs, whilst Figures 5.2.3, 5.2.4 and 5.2.5 examine loss rates of 1%, 2% and 5% respectively. The obvious conclusion from these figures is that loss is very influential with respect to the quality of conversation audio. Of note is that for rates of loss greater than 1%, the codec G.723.1 fails to be able to generate any rating that is of acceptable quality. The other two codecs are still capable of producing acceptable quality audio at a 1% and 2% loss rate, with G.729 unable to produce acceptable quality audio for a loss rate of 5% and above. The codec G.711 proves to be the most robust, able to withstand significant levels of loss whilst still producing acceptable audio.

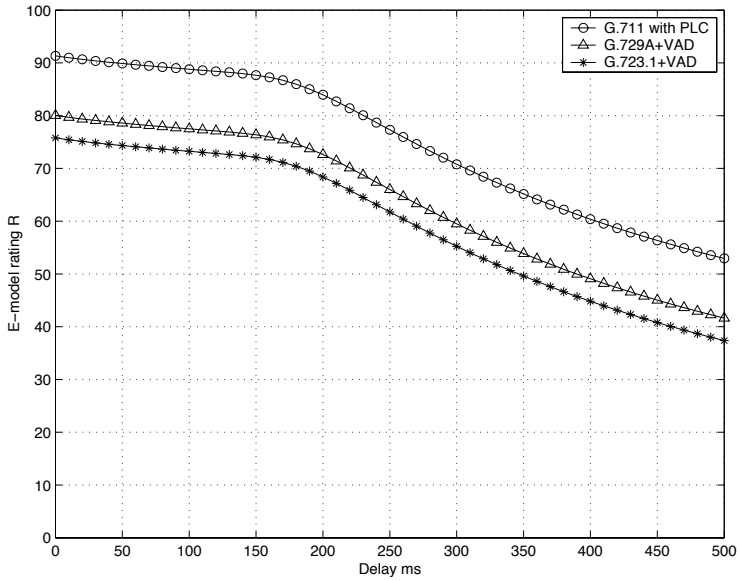


Figure 5.2.2: E-model rating R with G.711, G.729 and G.723.1 with 0.5% loss and all other parameters set at their default values (T=Ta=Tr/2).

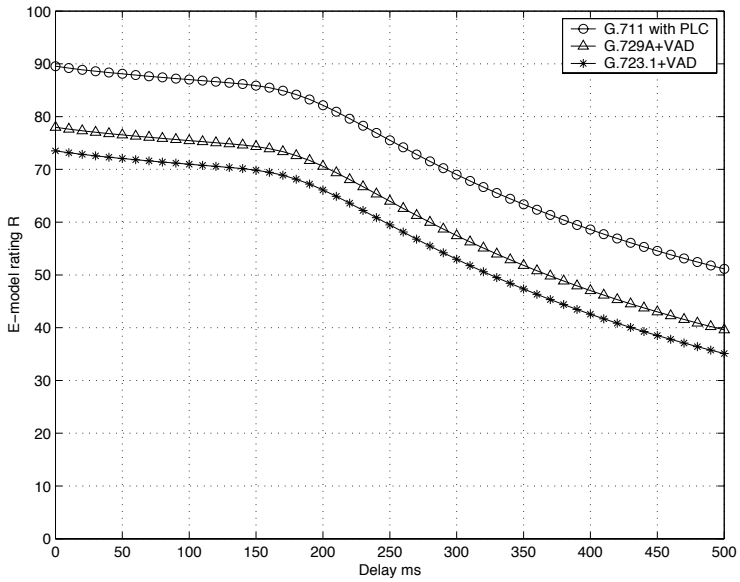


Figure 5.2.3: E-model rating R with G.711, G.729 and G.723.1 with 1% loss and all other parameters set at their default values (T=Ta=Tr/2).

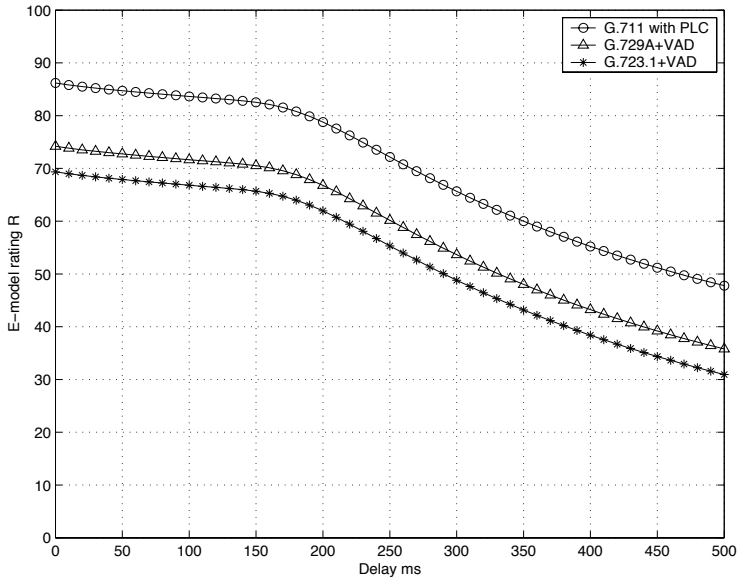


Figure 5.2.4: E-model rating R with G.711, G.729 and G.723.1 with 2% loss and all other parameters set at their default values (T=Ta=Tr/2).

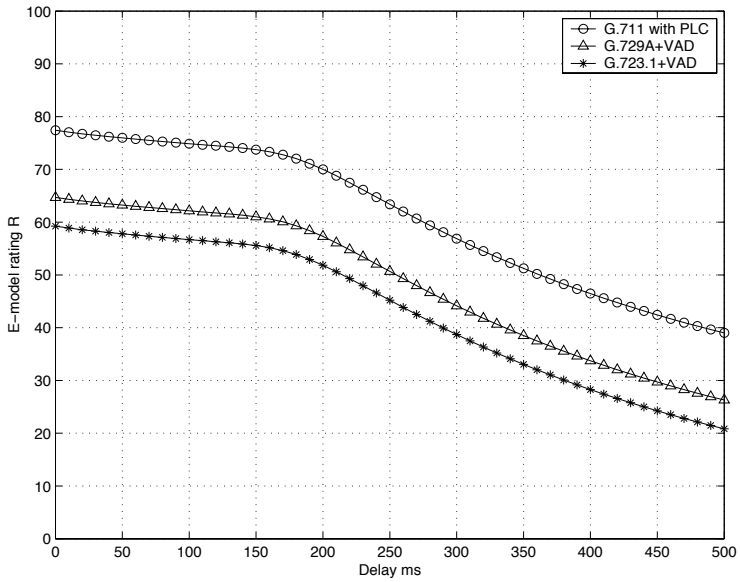


Figure 5.2.5: E-model rating R with G.711, G.729 and G.723.1 with 5% loss and all other parameters set at their default values (T=Ta=Tr/2).

5.2.2 Discussion of Measurement Approaches

A distributed, audio conferencing application requires some form of mechanism for determining the level of audio quality that a particular server can provide. The E-model is a standardised means for estimating the level of audio quality from objective measurements. However, before one can use this model, measurements of delay and loss must be gathered. In current networks there are two distinct approaches for gathering this information. These are known as passive and active measurement, and are discussed in the following sections.

Passive Measurement

Passive measurement is the process of monitoring network traffic passing measurement point(s) without impacting the traffic flow. It is usually used for the measurement of throughput and packet size statistics. A common method for performing passive measurement involves the use of Simple Network Management Protocol (SNMP) [6] measurement tools [12]. Through the use of SNMP one can keep track of the number of packets that have passed through a particular router, the number that have been dropped and the number of transmission errors that have occurred on a link. Unfortunately SNMP cannot be used for collecting end-to-end statistics such as delay. However, by using the time-stamp data within the RTP header [46] of each packet we are able to measure some delay statistics.

We shall examine two methods for using passive measurement for the purposes of a distributed, audio conferencing application. The first is to utilise two measurement points, one at the server and the other at the client. By doing this we are able to gather time-stamps from both locations which can then be used to calculate a delay measurement. The problem with this method is that it requires accurate clock synchronisation between the server and the client, which may not be possible. Another problem with this method is that it requires the time-stamp data to be

moved to a central location to be processed (the server). This results in additional data being pushed into the network which is undesirable. The second method consists of the server measuring the time required for it to receive audio packets from the client. By using the time-stamp data within the RTP header [46] of each audio packet we gain an approximation for the time taken to travel through the network.

Some positive aspects of passive measurement are that it is non-invasive and thus does not affect the packets that are being measured [26], nor does it affect the flow of the data network. It is also able to collect statistics on all data that flows through a measurement point. This enables more accurate measurement collection. In a real-time scenario, passive measurement would allow for the prompt determination of problems in the network as measurements are continuously being made.

On the other hand, passive measurement has some limitations. Passive measurement cannot be used to measure a currently unused path. It relies on data to be already flowing through the measurement point in order for statistics to be collected. Thus passive measurement would prove unsuitable for a number of measurements required in the audio conferencing application. These measurements include: initially determining the best server for a given grid square and determining a better server to move to if the need arises.

Another problem of passive measurement is that it potentially measures all data that passes through an interface. This potentially results in a considerable amount of collected data, in some instances a number of Terabytes of data could be collected per day [12], depending on the decisions as to what constitutes necessary data. However, if we are selective as to the data collected then this is not such a problem. Overall, passive measurement appears to be a useful mechanism for obtaining delay statistics. For the requirements of an audio conferencing application its usefulness is limited to measuring a currently used path.

Active Measurement

Active measurement is the process of introducing artificial probe packets into a network and measuring the effects the network has on these packets. It has been traditionally used for the measurement of end-to-end statistics such as delay and loss.

The attractiveness of active measurement over passive measurement is that it can be used to measure currently unused paths. With complete control over the probe packet stream, both currently used paths and unused paths can be measured for delay and loss. In addition, probe packets can be of any size and have any inter-departure time, which allows for the closer fitting of probe packets to the data under investigation. Appropriate choice of the size and inter-departure time of probe packet traffic is examined below.

Whilst active measurement allows for greater flexibility with regards to the probe packet stream, it does have some disadvantages. The main disadvantage of active measurement is that it is invasive. Hence the probe packet stream generated may affect the network upon which it traverses. The problem with this is that we are dealing with a real-time audio application and thus if there is any additional traffic being injected into the network, reductions in available bandwidth and application effectiveness may result. However, this invasive nature of active measurement, does not necessarily have to be a disadvantage. For example, if we are investigating the best server upon which to hold a conversation, then by using active measurement (with packet size and inter-departure time equal to that of the audio codec stream) we effectively load the path between the user and the server with the equivalent of an audio stream. Thus a greater understanding of the ability of the path to handle an audio stream is achieved. We now consider two methods for performing active measurement.

One method is the one-way end-to-end measurement method, which consists of a sender and a receiver on either side of the network. The sender generates a stream of probe packets that are timestamped and uniquely numbered. The packets travel through the network to the receiver, where they are given another time-stamp and calculations can then be made using the timestamps and the packet number. Delay and delay variation can be calculated from the time-stamp values and lost packets can be identified from the packet numbers.

An alternative method is the round-trip, end-to-end measurement method, which consists of a sender, a receiver and an intermediate node. As with the first method, the sender generates a stream of probe packets that are time-stamped and uniquely numbered. However, in this method, the packets traverse the network to the intermediate node, where they are echoed back through the network to the receiver. In this model, the sender and receiver usually reside in the same computer and so this model measures the round-trip time from the sender to the intermediate node and back again.

The main problem with the one-way end-to-end method is that it is difficult to ensure synchronisation of the sender and receiver clocks for the purposes of timestamping. If the clocks are not synchronised then the timestamps and hence delay measurements become inaccurate. This is because the sender and receiver are usually separate entities. This results in the need to use a method for synchronisation. The most common method for distributing time clocks in the Internet is Network Time Protocol (NTP), which is not sufficiently accurate for synchronisation purposes [40]. On the other hand, whilst the round-trip method described does not suffer from this problem as the sender and receiver usually share the same computer, it does have problems of its own. Most notably, the delay metric is that of the round-trip delay which cannot necessarily be halved to give the one-way delay, as delay paths can change.

5.3 Implementation Recommendations

Whilst we have identified that both passive and active measurement are valid methods with which to gather statistical information from the network, we must consider how we utilise these measurement approaches to gather enough information to be useful. Passive measurement has less variables to consider as we use packets already flowing through the network to gather statistics. The only factor we must consider is "how long must we measure?". With active measurement, likewise, we must consider how long we must measure, but in addition we must also think about the size and inter-departure time of the probe packets we send across the network. In the following, we discuss these factors as well as the problem of measuring levels of loss in order to make recommendations for use with our proposed measurement architecture.

5.3.1 Time

Since we are dealing with a real-time application, minimising the time of measurement is also of great importance. Whilst some implementations of active measurement in the literature, require considerable time to gain results, they do this purely to obtain greater accuracy. By reducing the level of accuracy, we can significantly speed up the acquisition of measurement data. An adaptive measurement architecture will be presented that minimises measurement time. Grossly inadequate servers can be ruled out quickly leaving only a small subset of available servers that require further measurement. Although this adaptive process reduces the effort required to measure a number of paths through the network, it still does not, *a priori*, determine the number of probe packets required to measure delay and loss over a path. If a quick result is needed, how many packets are required to give a result with some accuracy? Two probe packets are all that are required to obtain a confidence interval of the delay on a path, but that is obviously not enough to give a reasonable first approximation. One option that does present itself is to use the length of samples

that are used for audio quality determination [19]. When deriving audio quality, listeners are subjected to a series of 2-5 second audio samples. Opinions upon these samples are then used to develop scales of audio quality. As we wish to measure the audio quality between users, measuring delay in 2-5 second bursts will result in enough packets to provide some accuracy. A 2-5 second burst of probe packets with an inter-departure rate of say 20 ms results in 100 to 250 probe packets being sent. However, since there will be multiple users measuring at the same time, the total number of probe packets may be too great for the network to cope with. Thus, it may prove necessary to increase the inter-departure time between probe packets to reduce strain. Using a maximum server measurement time of 5 seconds will result in a maximum measurement time of 10 seconds (time to measure current server + time to measure alternates). This should provide enough delay measurements to give a reasonable level of accuracy, whilst maintaining a small measurement time.

5.3.2 Packet Size and Inter-departure Time

As discussed, both active and passive measurement are suitable approaches to measurement. However before they can be used, parameters for packet size and inter-departure time need to be set. As passive measurement uses existing traffic, the packet size and inter-departure time depends on the properties of that stream. Active measurement on the other hand can use any size packet and inter-departure time.

The two papers, [2] and [44] investigate the use of the round-trip method for measuring delay and loss in a network. Both parties chose packet sizes of 32 bytes in order to minimise impact on the network. In [44] packets were sent out every 39.06 ms, reflecting a multiple of the clock tick time (3.906 ms) of the DECstations used in the experiment. On the other hand, [2] used a range of inter-departure times

to identify the effect inter-departure time can have on the network and the probe stream. The times used were: 8, 20, 100, 200 and 500 ms. Finally, the length of time for which the experiments lasted were 10 minutes for [2] and approximately 65 minutes for [44]. Both experiments were performed with emphasis upon accuracy, not the quick acquisition of results and hence their use of small packets and variable inter-departure times.

With regards to probe packet size and inter-departure time, measurement projects generally try to maintain an average bandwidth of less than 10 kbps to minimise their affect on the network [10]. However with regards to the IACMMG and DICE projects, whether this is the most appropriate choice needs to be investigated. Obviously there is a lower bound on the size of probe packets as we need to transmit some information. On the other hand, if we make our packets too large then we shall have a negative impact on the network. For our measurement architecture, the purpose of active measurement is to either determine if a grid square is suffering poor audio quality or to measure paths connecting users to other servers. Obviously for the first case, we wish for the probe packets to have as little impact upon the network as possible. Whilst for the second case we are trying to determine if a selected path is suitable to have 5.1 stereo audio data pass over it. Since we are measuring the effect the network has on the probe packets it seems reasonable to mimic the audio stream closely in order to simulate the effect of audio data. It would not be recommended to create a probe stream resembling 5.1 stereo audio data, as this may adversely effect other network users, but a probe stream resembling a single audio stream would give some indication of the path's suitability. However, the same result can be obtained by using smaller probe packets while maintaining the original inter-departure time.

5.3.3 Measurement Approaches

The basic requirements of a measurement architecture were discussed in Chapter 2 with the conclusion that two types of measurement are required. The first is to measure the current quality of the paths between the users of a grid square and the server they are connected to. The second is to measure the paths between the users and possible alternative servers. As explored earlier in this chapter, active and passive measurement approaches are possible candidates for these measurements. However, passive measurement cannot be used to measure a currently unused path, which rules it out for the second measurement required.

One possible approach is to consider a hybrid measurement architecture. The first component of the hybrid architecture consists of using passive measurement for determining whether the current server is providing adequate quality audio. The second component is to use active measurement to determine the best server to move to, should the current server prove to be unsuitable. This architecture has a number of advantages and disadvantages. For example, by using passive measurement we are able to determine relatively easily when a user's audio quality becomes poor. On the other hand, an outcome of using multiple measurement approaches is that it leads to added complexity of the architecture.

Another approach is a measurement architecture which utilises active measurement for both required measurements. Again, this architecture has its own strengths and weaknesses. The main strength of this architecture is that it is a simpler architecture, since there is only one measurement approach. On the other hand, the architecture needs to periodically probe current paths to find ones that are poor. A decision has to be made as to how often current paths should be probed to ensure customer satisfaction. If probed too often, then the probes are merely adding unnecessary traffic to the network; too seldom and users may be forced to contend

with poor service for long periods of time before the poor service is detected.

With the two architectures mentioned, we have up to now assumed that the results gathered from the measurement approaches can be directly compared. This includes comparison between the passive and active approaches, as well as the case when we are using active to measure two different scenarios. However, is this a valid assumption to make? For example, in the hybrid architecture, passive measurement monitors the effect the network has upon the currently flowing audio data giving an accurate measure of the delay experienced. In order to make a comparable measurement for the unused path, active measurement must use a probe packet stream with the same characteristics as that of the audio data. In this way active measurement is able to measure the potential effect the network will have upon the audio stream should it be moved to this path. However, as we are dealing with 5.1 stereo audio, the use of active measurement would require a probe packet stream six times larger than a single audio stream in order to remain comparable with passive measurement. This would place an unnecessary load on the network.

On the other hand, if we consider the totally active approach, we find that no matter the size of the probe packet stream, the measured metrics produced are not ideally comparable. The reason for this is that for the first measurement, we are measuring the effect the network has upon the combination of the audio stream and our probe packet stream. Whilst for the unused path, active measurement measures the effect that the network has upon the probe packet stream. The result is that the metrics produced cannot be accurately compared, we are potentially claiming that grid squares are suffering from unacceptable QoS when in fact this may not be the case. If we then size the probe packet stream to resemble 5.1 stereo audio the potential for inducing unacceptable QoS increases.

From this argument, it would seem that the hybrid model is the ideal mea-

surement architecture. However, we have neglected to consider the possibility of multiple grid squares measuring unused paths concurrently. Concurrent measurement will see probe packet streams from multiple grid squares which may, or may not, give a correct indication of the state of the path because as little as one, or as many as all, of the grid squares could be measuring the same server concurrently. Thus, in some instances the hybrid measurement model could produce comparable results but in general it will not, and therefore we are unsure of the accuracy of these measurements.

We have already examined a totally active architecture for a single grid square with the conclusion that results produced are not ideally comparable. With the concurrent measurement of multiple grid squares the problems of comparability of results is further exacerbated. In addition to the problems described with the concurrent measurement of unused paths, there is also the possible concurrent measurement of the currently used path should two grid squares on the same server measure at the same time.

Thus, we essentially have the situation where both measurement architectures yield results which are not ideally comparable and therefore we choose the simplest system for measurement, that is, a totally active architecture. As this is the case, the use of probe packets to simulate 5.1 stereo audio for active measurement, will have no real positive effect for our measured results, but may have a detrimental effect upon the overall quality of the network. As such, we recommend that probe packet size should be minimised, so as to have a minimal effect on the network.

5.3.4 Loss

As was observed from the figures in Section 5.2.1, loss has a significant effect upon the quality of a sample of audio. It is also a common problem in data networks

with loss rates often ranging from 0.1% to 31% [1]. For these reasons loss must be taken into consideration with respect to the design and construction of the measurement architecture. However, whilst loss is important, it is unfortunately difficult to measure. This is because in order to measure it, packets first have to be lost. Therefore, in order to calculate a loss rate to a suitable level of accuracy, a large number of packet transmissions needs to be considered. If passive measurement is used for the measurement of the current server, then it is possible that rates of loss could be measured, as measurements are taken over a long period of time with little impact on the underlying network or the user. For active measurement, loss can not be effectively measured, as it would take too long to ensure an adequate level of accuracy of results. For example, if we observed 1 packet being lost in 100 packet transmissions, we could conclude a 1% packet loss. However, this is only a point estimate with no information about how accurate it is. Because of this, it is necessary to establish a fixed loss allowance into the measurement architecture. 5% loss was shown to produce audio of poor quality for both G.729 and G.723.1, with G.711 not too far behind. This gives the impression that no codec can withstand loss rates of this magnitude or higher and still enable acceptable audio quality. In addition it is unrealistic that a real network could survive with sustained rates of loss in excess of 2%. A fixed loss rate of 2% is therefore assumed as it allows for both G.711 and G.729 to produce adequate results over a large range of delay values. Whilst having a fixed loss rate does not reflect the true nature of the network, it does give some level of robustness to the system. An unfortunate consequence is that G.723.1 is unable to produce good quality audio with a loss rate of this magnitude, which may indicate the need to re-evaluate this choice of codec.

5.3.5 Codec Choice

The choice of codec is important as different codecs can withstand varying degrees of delay and loss and yet still provide acceptable levels of audio quality. Whilst our

choice of three codecs for examination is only a small subset of available codecs, it does give insight into the tradeoff between robustness and audio stream size that codecs possess. It is this tradeoff that inevitably decides which codec to use. Since we are dealing with a 6 channel audio system, serious consideration must be made as to the impact the IACMMG and DICE projects will have on the network and the impact the network will have on the projects. If we consider our sample codecs and take into consideration a fixed loss rate, then we can immediately rule out G.723.1. This leaves G.711 and G.729 which are both capable of producing acceptable quality audio with a 2% loss rate. As discussed in Chapter 2 the bit rate for G.711 is 64 Kbps, whilst for G.729 it is 8 Kbps. Since we require 6 channels of audio we have 384 Kbps and 48 Kbps for G.711 and G.729 respectively, before any header information, compressed or not. Whilst G.711 is capable of producing good quality audio for a larger range of delay statistics when compared to G.729, due to the size of the stream it will cause more delay than a smaller stream. In addition, the user's connection to the Internet will greatly affect the codec choice. Whilst it must be assumed that a user has a broadband connection, it cannot be assumed that a user's connection can cope with both the download and the upload of 6 G.711 audio streams in addition to game data. Thus, it must be recommended that a tradeoff be made, sacrificing some audio quality to reduce the size of the packet stream. Since we have only discussed three codecs, G.729 is recommended for use.

Chapter 6

QoS Measurement Schemes

The purpose of this chapter is to take the techniques explored in Chapter 5 and to combine them to create a measurement architecture that could be used as an effective means of QoS management for an audio conferencing application. To simplify the overall complexity of the architecture, throughout this chapter we take the point of view of managing the QoS of a single grid square within the virtual environment. In addition, we shall initially look at the case where the environment remains static. This translates to neither new users entering our grid square, nor current users leaving. These assumptions are dropped when we integrate this architecture with the load balancing architecture in Chapter 7.

6.1 Proposal

In Chapter 5, emphasis was placed upon providing adequate QoS for users within a distributed audio environment. In order to do this, we need to gain some insight as to the quality of the connection between the users of the grid square and the server. We want to be able to identify when the users of the grid square are suffering poor QoS and then proceed to alleviate this by moving the grid square to a server that will provide adequate QoS if such a server exists.

We consider a proposed architecture to measure delay and loss for an audio conferencing application. As has been discussed, in order to determine whether a server should be replaced by another, the current server and all available servers need to be measured to determine the best server to use. However, there is no need to measure all available servers if the current server is providing a reasonable service. This leads to two basic measurement requirements. The first measurement is to determine if the current server is undesirable and needs to be replaced. If this is found to be true, then the second set of measurements should determine the best server (out of the possible remaining servers) to move to. From the previous chapter, we see that both passive and active techniques can be used to identify whether the current server is undesirable or not, whereas active measurement is the only method suitable for determining the best server to move to.

We focus on delay as the key indicator of poor QoS based on the discussion of Chapter 5. We divide the delay range into two regions of quality based upon ITU-T E-Model ratings. Difference in codec choice will result in a different range of E-Model ratings and these two regions will change. As we have discussed we shall be using the codec G.729 in our discussion. Figure 6.1.1 displays E-Model rating R versus delay for the codec G.729 for a number of loss rates. As discussed in Chapter 5, long measurement times are required to calculate loss rates and it is for this reason that we recommend that loss not be measured initially. Instead, a selected loss rate is incorporated into the measurement architecture to accommodate any reasonable level of loss that may be tolerated. As we can see from Figure 6.1.1, if a maximum loss rate of 2% is used, then in order to obtain acceptable quality audio for G.729, that is an R-value of 70 or better, the maximum delay an audio stream can have is approximately 160 ms. Thus if an audio stream is suffering from 2% loss it can maintain good quality over the delay range 0 ms to 160 ms. Higher delays may result in an audio stream of poor quality. Hence an R value in the range of 70 to 0 will indicate an undesirable connection and thus the need to find a better

alternative server. As the two regions have different bounds depending upon the codec used, we simply label these regions as green and red indicating the regions of acceptable and poor quality, respectively.

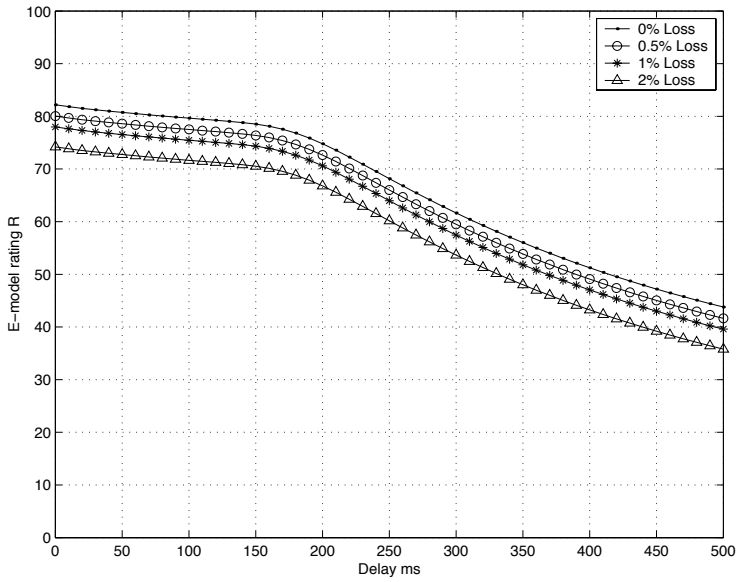


Figure 6.1.1: E-model rating R using G.729 & 0%-2% loss.

The composition of the two measurement architectures was explored in Chapter 5 with the conclusion that neither architecture is capable of producing comparable results. However, due to the reduced complexity of the totally active measurement architecture, it was recommended. Now we can investigate how the measurement architecture would perform. Since the data network is dynamic, there is a need for an adaptive means of measurement. In addition, since active measurement puts added load onto the network, we need to reduce the amount of active measurement performed as much as possible. In order to facilitate these requirements the following algorithm has been developed. It is split into two stages. The first deals with measurement of the current server, and the second deals with the measurement of available servers should the current server be found to be undesirable.

Whilst we recommend that active measurement be used for the measurement architecture, the algorithm used for the measurement architecture should not be limited by this. Thus, we ignore the method of measurement acquisition and simply assume that delay measurements are received by the system. In this way we are able to focus primarily upon the performance of the architecture algorithm.

The measurement architecture is divided into two stages reflecting the required measurements.

6.1.1 Stage 1

A collection of delay measurements of the current server holding our exemplar conversation are taken at regular intervals. A mean, variance and a confidence interval are created from the data and are converted into E-Model R-values. As delay measurements are gathered, the mean, variance and confidence interval are updated.

Colour Coding

Within the algorithm, the server hosting the grid square for which we are measuring, is given a label deeming it either 'green' or 'red'. This label expresses the ability of the server to provide adequate QoS for the grid square. Labels are assigned in correspondence to the location of the grid square's calculated confidence interval over the E-Model's R-value range. The R-value range is divided into two regions (defined in the previous section). If the confidence interval is completely contained by the green region, then we assume that the server is capable of providing adequate QoS for the grid square and is given a 'green' label. If, however, the confidence interval is either contained by the red region or spans both regions then the server is deemed incapable of maintaining adequate levels of QoS for the grid square and is given a 'red' label. If this occurs, then Stage 2 measurement is initiated.

Delay Measurements

The algorithm must deal with the delay measurements of a group of users. Thus, the delay measurement we consider is not an actual delay measurement as such, but a combined delay measurement taking into account the delay measurements of all the users of the grid square. The simplest way to do this is to take the mean of the user's delay measurements. However, for our current system, we use the following. To begin with, the delay measurements received from all users are sorted in descending order. The first two measurements (the worst two) are taken and added together. This gives a worst case figure of the 1-way trip time for the users of the grid square (from one user to the server and then to another user).

Parameters

A number of parameters are used in each stage of the algorithm. Those used in Stage 1 are defined as follows:

- Measurement Time := Minimum amount of time for measurement during which delay measurements are collected (default value of 2 seconds).
- GREEN := Defines the range of R-values of a grid square to be acceptable. It has an upper bound of 100 R-value units and a lower bound of 70 R-value units.
- RED := Defines the range of R-values of a grid square to be undesirable. The region is bounded by 0 and the lower bound of the GREEN region.
- Inter-measurement period := Time between measurement cycles (default 60 seconds).

Algorithm

The following is the detailed algorithm for Stage 1 measurement:

```
While current Measurement Time loop
   Get the delay measurement and convert to E-Model R-value
   Keep running lists of delays and number of measurements
End while

Calculate the E-Model Confidence Interval based upon mean and variance of E-Model
  R-Values

If the confidence interval is within the green region  then
   The current server is deemed 'GREEN'
   Wait a pre-set inter-measurement period before measuring again
Else
   The current server is deemed  'RED'
   Initiate stage 2 measurement (which includes further measurement of the
     current server)
End if
```

6.1.2 Stage 2

Algorithm

The following is a detailed outline of the Stage 2 algorithm:

```
Loop over the list of available servers (including current server)
   While current Measurement Time loop
      Get the delay measurement and convert to E-Model R-value
      Keep running lists of converted delays and number of measurements
   End while
   Calculate the E-Model Confidence Interval based upon mean and variance of E-Model
     R-values
End loop

Loop over the list of available servers (including current server)
   Check confidence interval of each server (see Section 6.1.4)

   If server is GREEN then
      Construct t-test using current and available server (see Section 6.1.5)
      If result is greater than or equal to Minimum Green Change then
```

```
         Shortlist server to Green list
      End if
   Else (The server is RED)
      Construct t-test using current and available server (see Section 6.1.5)
      If result is greater than or equal to Minimum Red Change then
         Shortlist server to Red list
      End if
   End if
End loop

If Green shortlist is not empty then
   Choose best server to replace current server and quit (see Section 6.1.6)
Else if Red shortlist is not empty then
   Choose best server to replace current server and quit (see Section 6.1.6)
Else
   Keep current server and quit
End if
```

As discussed in Chapter 2 there is an inherent cost associated with the movement of a grid square from one server to another. We endeavor to minimise this cost as much as possible by limiting movement only to that which will significantly improve the quality of the audio the grid square receives. The need for there to be a significant improvement in audio quality from a server change results in two minimum movement parameters depending upon the audio quality of the current and prospective servers. In order to accomplish this, a two-sample t confidence interval is established between the current and prospective servers and we compare this with our minimum movement parameter. If the confidence interval is greater than the movement parameter then the difference between the current and prospective servers is deemed significant. The two minimum movement parameters depend upon the region within which the comparison is made. The first, known as the 'Minimum Green Change' parameter is when the comparison is made between a green prospective server and a red current server and has a value of 1 E-model units, whilst the second, the 'Minimum Red Change' parameter, is for a comparison between a red prospective server and a red current server, and has a value of 5 E-model units. The reason for this is that there is less benefit to both the user and the system to move from one undesirable server to another, whereas by moving from

an undesirable server to an acceptable server will hopefully result in less further measurement for the system.

6.1.3 Colour Coding

When we consider Stage 2 measurement we have a similar definition for colour coding as we did for Stage 1. For the current server, we have hitherto assumed it is red as defined in Stage 1. Thus it may only change from red to green, and this may only occur if further measurement of the current server reduces the span of the confidence interval such that it resides solely in the green region. For the list of available servers, the green and red labels have a similar meaning in that a red label will indicate that that particular server is unable to provide an adequate level of service, whilst a green label will indicate that it can. The difference in Stage 2 is that in order for an available server to become suitable for grid square movement, its confidence interval must be significantly better than that of the current server. Thus, whilst we place emphasis upon only green labeled servers as being capable of providing adequate QoS, it may prove necessary to move a grid square to another red server that is significantly better should there be no other option. In this way we recognise that whilst all red servers will provide inadequate QoS, the QoS provided by a red server with a high R-value will be better than a red server with a low R-value.

6.1.4 Check Confidence Interval

The process of comparing confidence intervals is identical to the process demonstrated in Stage 1. The only difference is that in Stage 2, we compare the confidence intervals of all available servers including the current server. If the confidence interval of the current server is found to be solely within the green region, then the server is deemed adequate to continue hosting the grid square, and we cease further measurement.

6.1.5 Comparison of Two Servers

Within Stage 2 of the algorithm, a 2-sample t confidence interval is established between the current server and an available alternate server. The result of which is compared with either the *Minimum Green Change* or *Minimum Red Change* figures. The purpose of calculating a t confidence interval is to determine whether the alternate server will provide better QoS to a set confidence. An example of a 2-sample t confidence interval is as follows. Delay measurements are collected for both the current and alternate servers. As they are collected, they are converted into E-Model R-values. Once a number of delays are collected and converted, mean and variance values are equated. For our example, five delay values are collected for the current server (which is red) giving an E-Model rating mean of 60.51 units and standard deviation of 1.38 units. Six delay values are collected for the alternate server (which is green) giving an E-Model rating mean of 71.17 units and standard deviation of 0.77 units. The following formula is used to calculate the 2-sample t confidence interval:

$$(\overline{x}_A - \overline{x}_C) \pm t^* \sqrt{\frac{s_A^2}{n_A} + \frac{s_C^2}{n_C}}. \tag{6.1.1}$$

Here $\overline{x}_A$ and $\overline{x}_C$ are the means for the alternate and current servers respectively, s_A and n_A, and s_C and n_C are the standard deviations and number of measurements, of the alternate and current servers respectively. The critical value t^*, requires the degrees of freedom to be calculated using the following formula:

$$df = \frac{\left(\frac{s_A^2}{n_A} + \frac{s_C^2}{n_C}\right)^2}{\frac{1}{n_A - 1}\left(\frac{s_A^2}{n_A}\right)^2 + \frac{1}{n_C - 1}\left(\frac{s_C^2}{n_C}\right)^2}. \tag{6.1.2}$$

This gives a value of 6.0157 and a resulting t^* of 2.447. Using formula 6.1.1, the 2-sample t confidence interval is (8.97, 12.35). The left-hand value of the confidence interval gives the minimum QoS improvement the alternate server will provide over

the current server with a confidence of 95%. This value is then compared with our minimum change parameters. As the alternate server is green, the change parameter is 1 R-value unit, and the server is accepted as suitable to move to.

6.1.6 Server Ranking

In our efforts to find a suitable server for a conversation to move to, we may find multiple capable servers. In this instance, we must decide which one of the group of capable servers we wish to move the grid square to. We have a number of options as to how this decision is made. If we are particularly pessimistic of the network, we may choose the server with a confidence interval with the greatest lower bound. In this way, we are more confident as to the server's ability to maintain a base level of quality. Alternatively, if we choose the optimistic approach, we could look at the upper bound of each server's confidence interval. A higher upper bound may indicate that the server is capable of producing better QoS. There are a number of other methods to choose from, however, since we are dealing with a list of servers which have proven themselves capable of providing adequate levels of QoS, all methods should produce acceptable results. The simplest method to use is to compare mean values of the capable servers, accepting the one with the best mean score. For the purposes of simulation, this simple method of server selection is used.

6.1.7 Parameters

The parameters used for Stage 2 are given as follows:

- Minimum Green Change := Minimum distance (in R values) between the current server and a prospective server to allow for a server change. Note that the prospective server must be classified as green (default value of 1 R-value points).

- Minimum Red Change := Minimum distance (in R Values) between the current

server and a prospective server to allow for a server change. Note that the prospective server must be classified as red (default value of 5 R-value points).

6.2 Simulation Analysis

6.2.1 User Satisfaction

Unlike the excess overload metric established in Chapter 4, the emphasis of the algorithm described in this Chapter is upon maximising user QoS. Thus, we introduce a new metric, *User Satisfaction*, which is designed to give the proportion of time the user remains in a 'happy' state, over the full length of the simulation run. A user is deemed 'satisfied' if their instantaneous delay, when converted into an E-Model R-value, is greater than 70 R-value points. Over the course of a simulation run, a user is polled periodically. If the user is found to be satisfied, then a counter is incremented by one, otherwise it is not. At the end of the simulation run, the counter for each user is divided by the number of times the user is polled giving an average satisfaction rating for each individual user. These ratings are then averaged to give an overall average satisfaction of the system.

6.2.2 R-value Improvement

A second metric used is the E-Model R-value improvement in grid square QoS. This is an instantaneous measure, taken at the moment a grid square is moved from one server to another during Stage two of the QoS Algorithm. The QoS, in R-Value units, is measured for the worst two users residing within the moving grid square, before and after movement occurs. The total R-value improvement of the collection of grid squares is then collated.

6.2.3 Simulation Architecture Assumptions and Parameters

The following are a number of assumptions made for the purposes of simulation:

- The Algorithm is simulated using a simulation environment similar to that described in Chapter 4. The only difference being that a number of the load balancing features of the simulation have been disabled in order to test the effectiveness of the QoS algorithm. Thus, load balancing is not performed within the simulation environment.

- Since we are only testing the effectiveness of the QoS algorithm, each of the servers upon which users reside are given infinite capacity.

- Each user is assigned a delay distribution for generating delay measurements of the delay path between them and each server within the simulated network. Since a complete delay path consists of the path from one user to the server and then to another user, the distribution assigned to each user is half of what would normally constitute a delay path. We consider eight different distributions that can be assigned to a user. Whilst this does not cover the full spectrum of delay distributions that could be experienced, it is inherently difficult to encapsulate a realistic range of delay distributions. Delay values can vary significantly, dependent upon a number of factors, including path choice, congestion and even time of day. In [33], they found significant variability in the fixed delay, or propagation delay, of measured paths. Once delays caused by congestion and buffering are incorporated, delay values can easily range from less than 50 ms up to around 300 ms. From this we have selected a small range of delay distributions that will produce realistic delay values. The delay distributions are all assumed to be Normally distributed with mean and variance given in Table 6.2.1. Each distribution has equal probability 0.1 of being chosen, except for the 50/25 and 50/225 pairs which have probability 0.2 of being chosen.

We now examine scenarios to test the effectiveness of the QoS algorithm.

Table 6.2.1: Mean and Variance of the 8 delay distributions

Mean	Variance	Mean	Variance
50	25	100	25
50	225	100	225
75	25	125	25
75	225	125	225

Scenario 1:

- The simulation run length is inconsequential as we are after an instantaneous result.
- Each simulation is run 10 times with different random numbers each time.
- The network consists of four servers, each with infinite user capacity.
- Grid squares are randomly arranged, such that each grid square has equal probability of being initially assigned to any one of the four servers.
- The virtual world is made up of 40 grid squares, arranged in a 5 by 8 grid.
- Users enter the system at random locations every 0.5 seconds, until the number of users reaches 200.
- Once allocated to a grid square, each user must remain in that grid square until the end of the simulation run.
- Users may not leave the system. Once placed within the virtual world, a user remains until the simulation ends.
- A single grid square measurement is performed for each grid square, allowing for grid square movement should it be necessary.

The minimum green and red thresholds are set to 1 and 5 R-value units respectively, resulting in an accumulated system R-value improvement of 112.29 units, with a standard deviation of 18.51 units when the environment is simulated. A curious outcome is the large standard deviation of our results. Two reasons for such a large standard deviation are that: very little happens over the run time, and the simulated environment changes dramatically from one run to the next. When the simulation is run again, the environment may be significantly different resulting in a considerably varied result.

We next tried changing one of the delay distributions, as listed in Table 6.2.1, to see what kind of effect a change in delay value distribution would have upon our results. The mean was changed from 125 to 200 while the variance remained at 225. We tested thresholds of 1 and 5 units which yielded a total system improvement of 328.38 units, and a standard deviation of 67.65 units. It is interesting to note the effect a greater distribution of delay measurements has upon the improvement that can be obtained through the use of the QoS algorithm. Giving each grid square a small probability of having a large delay metric for a particular server, means that there will be greater variation in grid square QoS and thus greater improvement when a grid square moves.

Scenario 2:

The network of Scenario 2 is an extension to that of Scenario 1 with the following inclusions:

- The simulation run length is 10000 seconds.
- Users may move between neighbouring grid squares as described in Chapter 4.
- The inter-measurement time of user QoS for each grid square varies with each

simulation run. The inter-measurement times selected for analysis are: 8, 16, 32, 64, 128, 256, and 512 seconds.

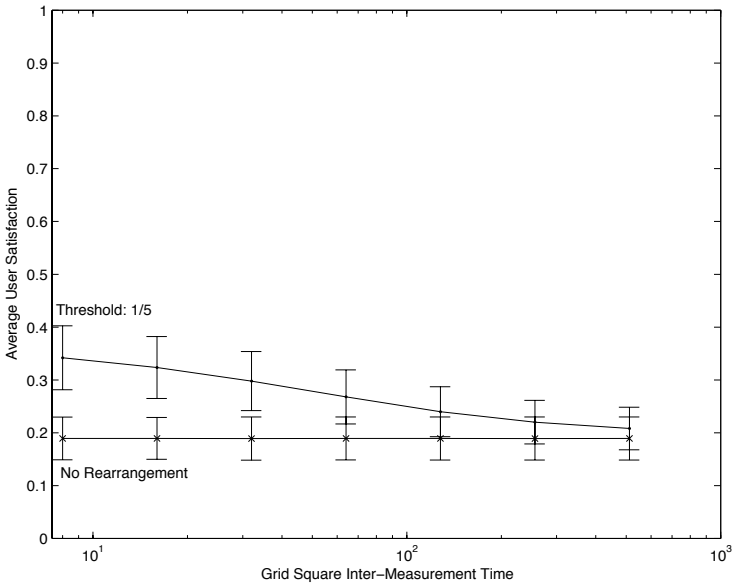


Figure 6.2.1: Average User Satisfaction

Figure 6.2.1 shows the average user satisfaction of the network, for the threshold set of 1 and 5 R-value units, over each of the grid square inter-measurement times given, against a system in which no rearrangement is performed. What we notice is that the algorithm has a positive overall effect upon the QoS experienced within the system. However, user satisfaction is significantly reduced as the grid square inter-measurement time increases.

Similarly, Figure 6.2.2 explores the situation in which one of the delay distributions has its mean increased from 125 to 200 while keeping the variance constant at 225. Again we examine the average user satisfaction when the thresholds are set at 1 and 5 R-value units, against the system in which no rearrangement is allowed. Again we see that an increase in grid square inter-measurement time has a significant effect upon the improvement that can be achieved by the QoS algorithm. Thus

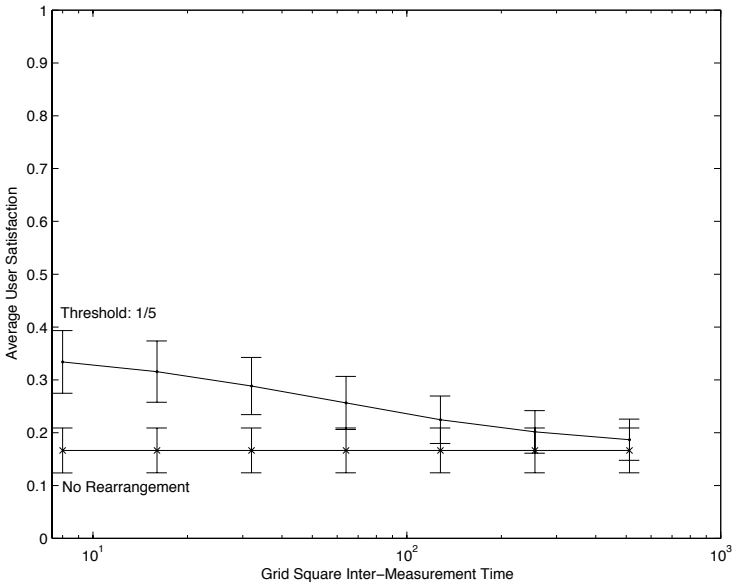


Figure 6.2.2: Average User Satisfaction

it seems desirable to keep the grid square inter-measurement time quite low.

It must be noted that for the cases examined in Scenario 2, users within the system may move between adjoining grid squares, as described in Chapter 4. This movement will ultimately affect the overall user satisfaction, since slower users will result in grid squares remaining static for longer and allow for a longer grid square inter-measurement time to be used. Thus, whilst we have shown that user satisfaction drops significantly as the grid square inter-measurement time increases, the appropriate time scale must be chosen carefully in a real system, where user movement will differ.

6.3 Summary

This chapter has developed and investigated the performance of a measurement architecture that can gauge the level of QoS received by users of a grid square and attempt to alleviate any poor levels of QoS by moving the grid square to a different

server, that is able to provide the required level of QoS or a significantly better QoS. In addition we have weighed up the added benefits against the additional costs of its implementation and have found it to be a useful addition to the distributed audio environment.

Upon examining the performance of the QoS algorithm we find that it produces a definite improvement in the QoS of users. However the full benefit of the measurement architecture cannot be fully realised until we integrate it with the load balancing architecture to develop a complete system. Thus we leave further discussion until Chapter 7 when we can make a more accurate assessment of the architecture as a whole.

Chapter 7

Integrated QoS Management Schemes

Having defined a load balancing architecture in Chapter 4 and a QoS algorithm in Chapter 6, we now integrate the two into a complete QoS management architecture.

7.1 The Architecture

Recall that the QoS algorithm takes measurements from each user within a grid square and combines them to form a grid square metric before a decision about the grid square's quality of service is made. If the QoS of the current grid square on the current server is poor, then delay measurements for each user in that grid square to every server within the network are collected. These measurements are used to give quality metrics for that grid square should it be located on each of the servers. Having estimated the quality of the grid square on each potential server, the grid square is moved to the potential server with the highest quality for that grid square, so long as the improvement exceeded a given threshold.

However, development of the QoS algorithm in Chapter 6 assumed servers of infinite capacity. This assumption can no longer be used in the integrated system

where server capacity is a constraint. Thus, in our integrated architecture, the algorithm is altered to reflect finite server capacity.

When a grid square is found to have poor QoS, a list of alternate servers is constructed for the grid square, in descending order of measured audio quality. In accordance to the QoS algorithm, the constructed list contains only those servers with a quality metric sufficiently greater than that of the current server to warrant the grid square movement. Assuming that the list constructed for the grid square is non-empty, the first server on the list is chosen and is queried to determine if there is sufficient available capacity to accommodate the grid square. If there is enough spare capacity, then the grid square is moved. Otherwise, the algorithm progresses through the list, until either it finds a server with adequate spare capacity, or the list is exhausted.

A consequence of integrating the QoS algorithm and the load balancing architecture is that there are now two kinds of measurement that are performed within the system, which have the potential to conflict with one another. Since the measurement of a grid square takes a few seconds, server measurement and hence load balancing may be performed at the same time. If a grid square that is currently being measured is moved to a different server due to load balancing, then the current measuring of the grid square is rendered useless and must be restarted. Thus, it is important to ensure that a grid square cannot be involved in load balancing while QoS measurement is being undertaken. The simplest method to achieve this, is to stop grid squares that reside on the currently overloaded server, from being offloaded during QoS measurement. The grid squares being measured for QoS remain locked to the overloaded server, requiring that other grid squares be offloaded instead. The problem with this scheme occurs when all resident grid squares of an overloaded server are being measured for QoS. An improvement can be made by ensuring that measurement of grid squares and servers is performed at different times that will not

conflict. This is achieved by delaying the QoS measurement of a grid square should it coincide with the load measurement of a server. When a grid square's QoS is to be measured, an approximation to the finish time of measurement is calculated. If this time conflicts with server measurement, then measurement to be performed on the grid square is delayed until after the server load has been measured. By doing this, a server will always have the ability to offload grid squares should it be necessary.

In addition to the changes made to the QoS algorithm to take into consideration server capacity, the load balancing architecture must also undergo some modifications in order for it to acknowledge aspects of QoS. As will be discussed in the following, each of the load balancing algorithms is modified to allow for grid squares to choose the server they would prefer to move to should they be required to move.

7.1.1 Minimum Offload

As with the original Minimum Offload algorithm, when a server attempts to offload a grid square, the grid squares located on the server are sorted from the one of smallest load to the largest. However, when the grid square is preparing to move, the alternate servers are polled to determine if any have sufficient capacity and those that do are shortlisted. Of the shortlisted servers, the preferred server, based upon measurement, is chosen for the grid square to move to. The process continues until either, a server with sufficient spare capacity is found, or there are no longer any servers in the network to interrogate for available capacity.

7.1.2 Maximum Offload, Minimum Redistribution & Simple Packing

The modifications made upon the Maximum Offload, Minimum Redistribution and Simple Packing algorithms are similar to those made upon the Minimum Offload algorithm. When load balancing is initiated, for each grid square, alternate servers

are shortlisted to determine which have adequate capacity. The grid square is then moved to the server in the shortlist with the highest preferred QoS.

7.2 Simulation Analysis

7.2.1 Simulation Architecture Assumptions and Parameters

The following are a number of assumptions made for the purposes of simulation:

- As with the simulation from Chapter 4, each server within the system is periodically measured, to determine if the server is overloaded.
- As a user moves within the virtual world, the grid square and server they are residing on changes. User movement is recreated within the simulation, with each user moving independently of all others. Every 10 seconds, each user has a fixed probability of moving from one grid square to an adjoining grid square or staying in their current grid square.
- Initially Server 1 is assigned all of the grid squares of the system, and thus performs all audio processing. As more users are introduced into the system, the spare capacity on Server 1 reduces, until it becomes overloaded. Once in an overloaded state, the algorithm attempts to move grid squares from the overloaded server to preferred servers with spare capacity to remedy the overloaded state.
- Each grid square is periodically measured for QoS, which can then be used to determine if the grid square is suffering from poor QoS, triggering the QoS algorithm.
- As discussed in Chapter 6, the QoS algorithm requires two seconds to gather enough measurements to effectively measure a particular grid square. If the grid square in question is found to be suffering poor QoS, then a further

two seconds of measurement are required to look for a better server. In the situation where grid square measurement is scheduled to occur at the same time as server load measurement, the grid square measurement is deferred until load measurement and possible load balancing is completed.

- In Chapter 6, we introduced a concept of user satisfaction. Here we look at the reciprocal, we call *user dissatisfaction*. The reason for this is that with the integration of the two architectures, dissatisfaction for a user can be caused by either poor audio QoS, an overloaded server, or both. Thus we wish to categorise our results. For results in which user dissatisfaction is examined, we define three result sets. The first is the user dissatisfaction due to load, and consists of the case where a user has adequate audio QoS, but the server upon which they currently reside is overloaded. The second result set is the dissatisfaction due to poor audio QoS, and consists of the case in which a user has inadequate QoS, but the server upon which they reside is not overloaded. The third case is the dissatisfaction due to QoS and Load, and is the case in which a user is suffering from both poor audio QoS as well as an overloaded server.

We now examine the effectiveness of the integrated architecture. For the scenario presented, we shall analyse the average excess overload for the integrated architecture. The reason for this is that with an integrated architecture we are now dealing with a problem with two constraints, which may conflict. Thus we wish to examine the compromise made to the existing load balancing architecture by incorporating the QoS algorithm.

In addition, we shall also examine the average dissatisfaction of users within the system. This is broken into three factors: dissatisfaction due to server overload, dissatisfaction due to poor QoS; and, dissatisfaction due to poor QoS and server overload.

Scenario

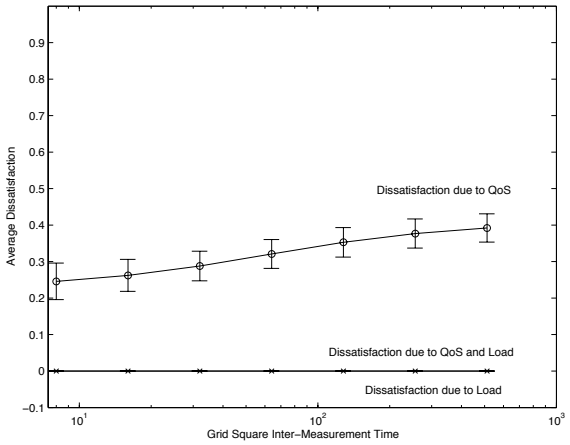
Our scenario explores the integration of the QoS algorithm with the load balancing architecture.

- The simulation run length is set at 100,000 seconds.
- The network consists of four servers, each with a capacity of 50 users.
- The inter-measurement time of load on each server varies with each simulation run. The inter-measurement times selected for analysis are: 16, 32 and 64 seconds.
- Measurement of the servers is carried out using a Random cycle.
- The inter-measurement time of user QoS for each grid square varies with each simulation run. The inter-measurement times selected for analysis are: 8, 16, 32, 64, 128, 256 and 512 seconds.
- The virtual world is made up of 40 grid squares, arranged in a 5 by 8 grid.
- Users enter the system at random locations, every 0.5 seconds, until the number of users reaches 200. Once this occurs, users are no longer admitted to the system.
- Users may not leave the system. Once placed within the virtual world, a user remains until the simulation ends.
- As users enter the virtual world they are assigned delay distributions for each of the servers within the network. Delay values are randomly generated from the distribution when required. There are eight delay distributions as described in Chapter 6.
- Minimum change thresholds of 1 and 5 R-value units shall be used.

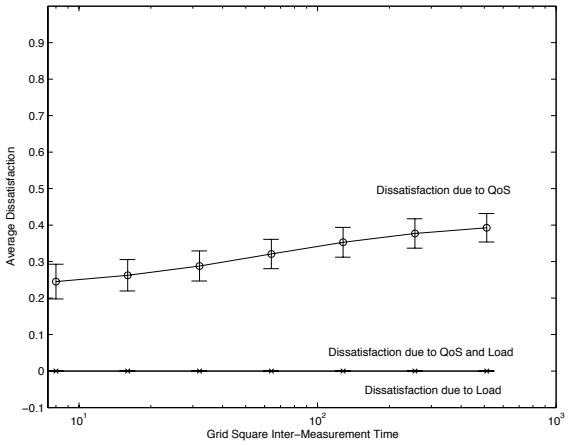
- The E-Model is used for QoS comparison purposes, where an R-value of 70 or greater is deemed acceptable QoS.

- Every 10 seconds, a user has a chance to move from their current grid square to a neighbouring grid square. This process is random, with a probability of 0.2 of moving to a neighbouring grid square. Otherwise the user remains in their current grid square and must wait a further 10 seconds for another chance to move. A neighbouring grid square is defined as a grid square that shares a common edge or vertex with the current grid square. As the grid of simulated system is arranged in a rectangular pattern, each grid square can have either three, five or eight neighbouring grid squares depending upon their location on the grid. When determining which grid square a user moves to, the grid square is chosen uniformly from the list of neighbouring grid squares.

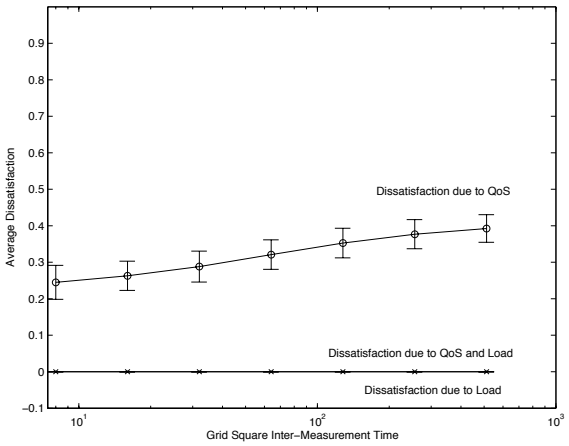
The plots of Figure 7.2.1 explore the average dissatisfaction of the network with a loading of 25% (or 50 users), for our range of server inter-measurement times (16, 32 and 64 seconds). What is immediately noticeable is the high dissatisfaction due to QoS experienced within the system. However, this can be attributed to a number of factors including user movement within the environment and our choice of delay distributions. Thus, this is not necessarily an intrinsic issue of our architecture design. Over the three plots of Figure 7.2.1, we find that change in server inter-measurement time has either no or a very minor effect upon the average dissatisfaction recorded. In fact, this trend continues for the remainder of our result sets. Thus, we shall only discuss the results with a server inter-measurement time of 16 seconds. In addition, we find that both average dissatisfaction due to load and average dissatisfaction due to QoS and Load give a result of zero. This is to be expected as the user population within the system is small enough to be hosted entirely upon one server, rendering load balancing unnecessary.



(a) Server Inter-Measurement Time = 16s.



(b) Server Inter-Measurement Time = 32s.



(c) Server Inter-Measurement Time = 64s.

Figure 7.2.1: Average Dissatisfaction with Network at 25% loading.

For a load of 50%, the user average dissatisfaction is given in Figure 7.2.2, where we find the most noticeable difference from that of Figure 7.2.1 is that the average dissatisfaction due to poor QoS has increased significantly. This is due to a number of factors. One factor is that unlike the system modelled for Figure 7.2.1, in which there were 40 grid squares with only 50 users to occupy them, there are now twice as many users. Thus there is a higher probability that a user will be in a conver-

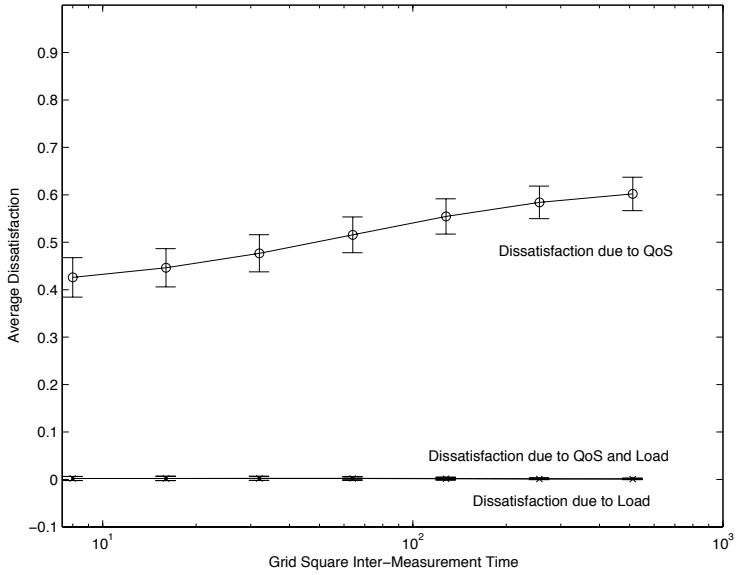


Figure 7.2.2: Average Dissatisfaction with Network at 50% loading and Server Inter-Measurement Time = 16s

sation with another user, who may not be 'geographically' close, and thus QoS will be lower. Another factor is that whilst there will always be large amounts of spare capacity for servers to load balance, there may not be enough for the QoS Algorithm to move a grid square to a preferred server.

Additionally, we find that like Figure 7.2.1, change in average dissatisfaction due to change in server inter-measurement time is negligible, with only minor change being noticed for the smaller grid square inter-measurement times. Whilst it cannot be clearly distinguished, dissatisfaction due to load and dissatisfaction due to QoS and load are registered for Figure 7.2.2. However, they are so low that these results can be attributed to the rare occasions that a server enters an overloaded state between opportunities to load balance.

Figure 7.2.3 gives the average dissatisfaction of the network with a loading of 75% (or 150 users). As with Figure 7.2.2, we find that average dissatisfaction increases

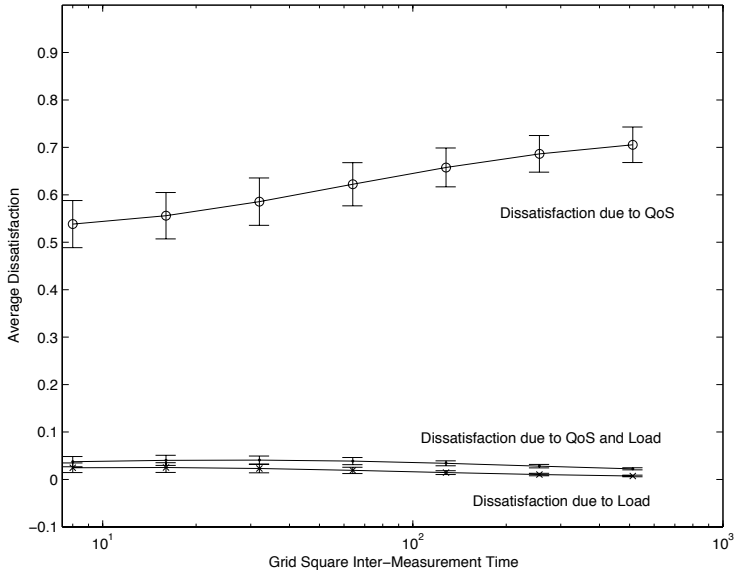


Figure 7.2.3: Average Dissatisfaction with Network at 75% loading and Server Inter-Measurement Time = 16s

as the current capacity increases. This can be attributed to the difficulty faced by both the load balancing algorithm, as well as the QoS algorithm, to find preferred servers to offload grid squares to. This becomes increasingly problematic as the current capacity of the network approaches the maximum, since the load balancing algorithm is primarily concerned with reducing excess load, with the secondary objective to offload a grid square to a preferred server. Additionally, the algorithm used in the simulation is Minimum Offload and is designed to offload smaller grid squares first. Thus, the smaller grid squares will be given preferential treatment, whilst larger grid squares that are offloaded, will be more likely to be offloaded to a server lower on their preferred server list. We also find that both the dissatisfaction due to load, and the dissatisfaction due to QoS and load, whilst larger than that found in Figure 7.2.2, remain low. However, it is interesting to note that both the dissatisfaction due to load, and the dissatisfaction due to QoS and Load decrease as the grid square inter-measurement time increases. This can be attributed to grid square rearrangement due to the QoS algorithm having a reduced impact upon the

results of rearrangement due to the load balancing algorithm. The QoS algorithm has the potential to move a grid square from a server providing low QoS but with a large amount of spare capacity to a server able to provide high QoS but only has enough spare capacity to take the grid square. Any increase in grid square size will see the server become overloaded.

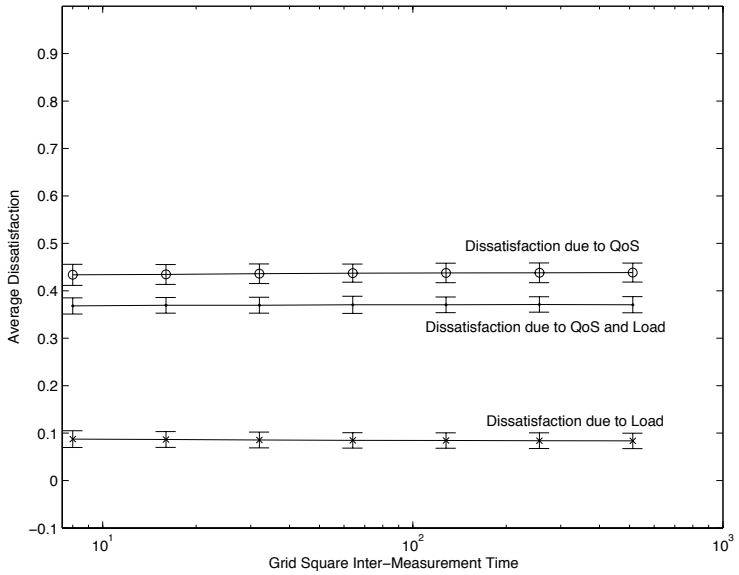


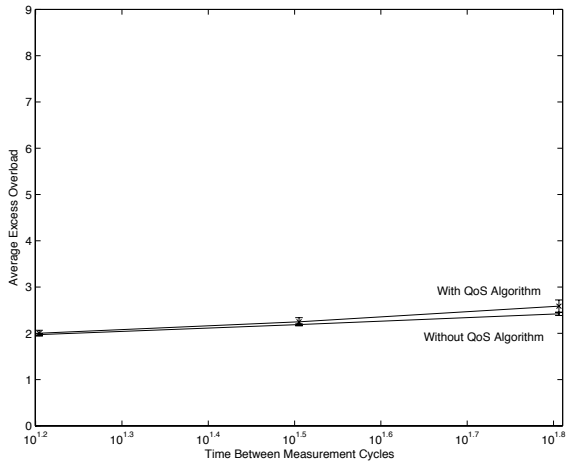
Figure 7.2.4: Average Dissatisfaction with Network at 100% loading and Server Inter-Measurement Time = 16s

Figure 7.2.4 gives the average dissatisfaction of the network with a loading of 100% (or 200 users). As with Figure 7.2.3, we find that average dissatisfaction increases as the current capacity increases. However this time, dissatisfaction due to QoS has decreased, whilst both dissatisfaction due to load and dissatisfaction due to QoS and load have increased significantly. In addition, we find that there are no longer increases in dissatisfaction with an increase in grid square inter-measurement time, but instead our results have plateaued.

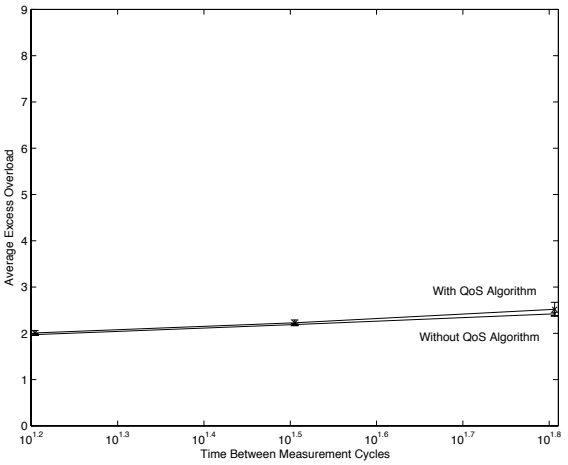
From the plots of Figures 7.2.1-7.2.4, we find that grid square inter-measurement time has a definite effect upon average dissatisfaction of the network. Some scenarios experience over a 10% increase in average dissatisfaction between the lowest grid square inter-measurement time of 8 seconds and the highest at 512 seconds. Thus it would be desirable to maintain a small grid square inter-measurement time to ensure accurate QoS information. Unfortunately, it is likely to be expensive, both computationally and in terms of bandwidth, to have such a small inter-measurement time. Our results indicate that grid square inter-measurement times of 256 and 512 seconds are not necessarily too large, however, from a human factors point of view, user QoS can be improved faster using smaller inter-measurement times. As we have said it is important to ensure that grid squares maintain up-to-date QoS information. To this end, for grid square inter-measurement time we recommend a range of 16 to 128 seconds.

Furthermore, we find, not surprisingly, that the loading of the system plays a pivotal role as to the level of dissatisfaction experienced, with greater levels of load ensuring high dissatisfaction. If possible, provisioning should be made to allow for the network to run at a loading as low as feasibly possible.

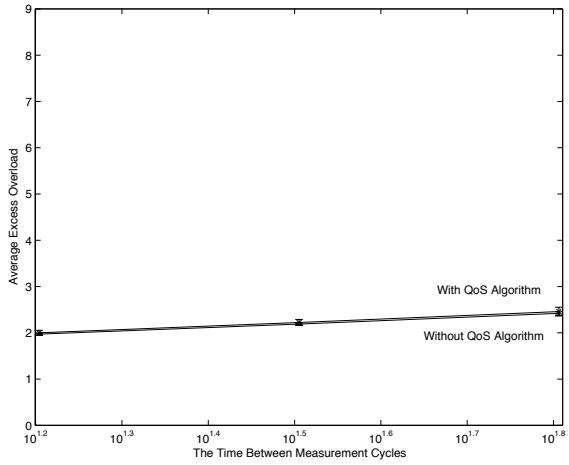
The plots of Figure 7.2.5 explore the effect the QoS algorithm has upon the average excess overload of the network, for a network at maximum capacity (200 users). Four grid square inter-measurement times were chosen (out of the seven) for examination, although all grid square inter-measurement times give similar results. As we can see from the plots, the inclusion of the QoS algorithm does increase the average excess overload of the system. However, this increase is inconsequential, although there is some variation (as depicted by error bars) as the server inter-measurement time increases. However, we deem this to be acceptable especially when we consider the improvement in user QoS that the QoS algorithm provides.



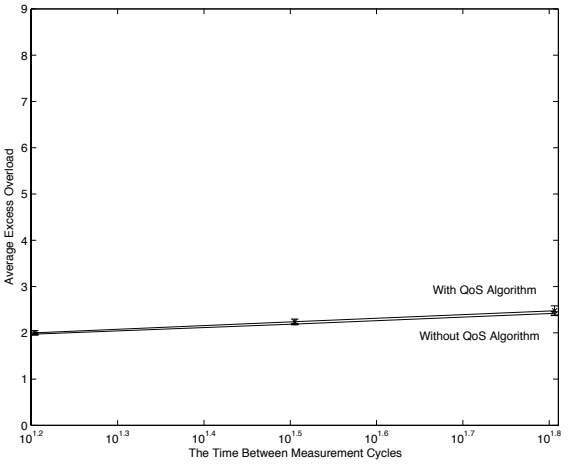
(a) Grid Square Inter-Measurement Time = 16s.



(b) Grid Square Inter-Measurement Time = 32s.



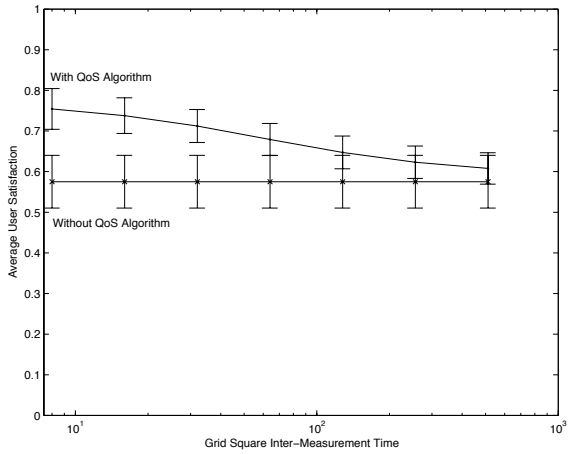
(c) Grid Square Inter-Measurement Time = 64s.



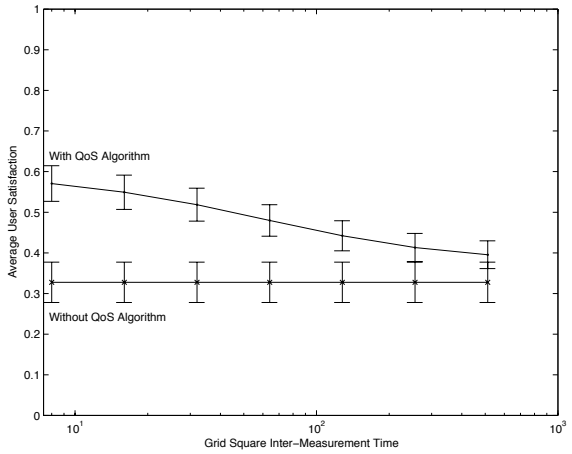
(d) Grid Square Inter-Measurement Time = 128s.

Figure 7.2.5: Comparison of Average Excess Overload With and Without QoS Algorithm.

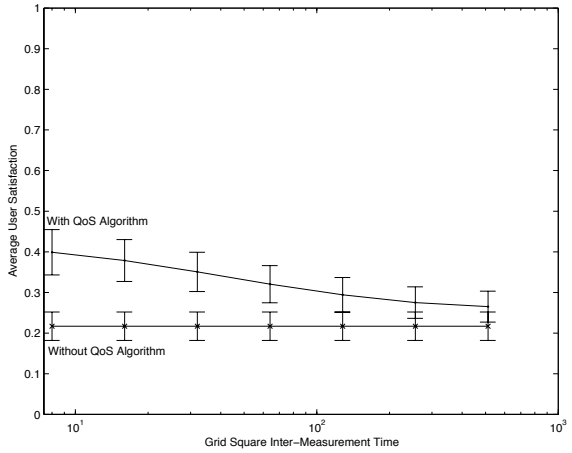
Finally, Figure 7.2.6 examines user satisfaction, comparing the effect that the integrated architecture has upon the system at different figures of loading. For each plot, the simulation is run with a server inter-measurement time of 16 seconds. As



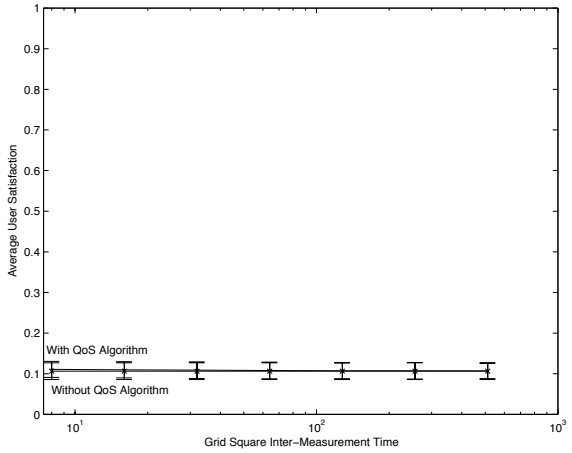
(a) Network at 25% loading.



(b) Network at 50% loading.



(c) Network at 75% loading.



(d) Network at 100% loading.

Figure 7.2.6: Comparison of User Satisfaction With and Without QoS Algorithm.

we can see, the integrated architecture has a definite positive effect upon the user satisfaction of the network for a network of up to 75%. As can be expected, the user satisfaction of the system drops as the grid square inter-measurement time increases. However, some of this reduction in user satisfaction can be attributed to the rate at which users move through the network. Figure 7.2.6 examines the effect of the

integrated architecture for a network at 100%, and demonstrates that the algorithm is unable to cope with such a heavily loaded system. Even still, the results show that the integrated architecture did increase performance, even though the increase was marginal.

7.3 Summary

Through the integration of the load balancing architecture described in Chapter 4 with the QoS algorithm developed in Chapter 6, we have been able to develop an effective architecture for dealing with the real-time conversations of the IACMMG and DICE projects. Through simulation we have been able to gauge the effectiveness of the architecture over a number of loadings of the network, using a variety of server and grid square inter-measurement times.

From our simulation results, we have been able to conclude that the load of the network has a significant effect upon the dissatisfaction experienced and thus must be kept balanced and below acceptable limits to ensure that user QoS remains high. In addition, we found that the effect the QoS algorithm has upon excess overload is negligible, making any improvement in QoS worth the small compromise of integration.

Chapter 8

Conclusion

To conclude this thesis we give a brief summary of the technologies we have explored as well as the conclusions drawn in our investigation. We began with a discussion of Massively Multiplayer Online Games, investigating what they are able and not able to provide end users. In particular, we were interested in the inability of current MMOGs to provide a realistic 2 dimensional audio system that includes voice communication. This led to the introduction of the Immersive Audio Communications for Massively Multiplayer Games and Dense Interactive Communications Environment projects which are currently being developed by the Smart Internet Technologies Cooperative Research Centre (SITCRC). The aim of the projects is to provide a generic platform for audio scene generation for MMOGs. Having investigated the general design of the IACMMG and DICE projects, we found that they face a number of challenges that must be solved in order to create an effective technology. Two such challenges were addressed in this thesis. The first was concerned with the finite resources of each of the servers and was thus a question of how to effectively balance load, in the form of users, over the available audio servers. The second, was with regards to ensuring that users receive an acceptable level of audio Quality of Service (QoS).

To address the first problem a load balancing architecture was designed and four

load balancing algorithms were implemented. Of the four algorithms one was centralised, whilst the others were distributed. The difference between the two types is that the centralised algorithm requires greater overhead in that a central controller must be introduced, whose job it is to manage load throughout the network. The problem with having a central controller is that it has the potential to become a performance bottleneck, and a single point of failure, of the network. We found that the centralised algorithm performed far better than any of the distributed algorithms. However, the algorithm also proved costly in terms of the number of grid square movements performed whenever load balancing was required. Thus, the centralised algorithm was not considered as a candidate load balancing scheme due to the negatives of the algorithm type, and the costly nature of the algorithm. Of the three distributed algorithms, Minimum Offload proved to be the statistically superior algorithm. Thus further analysis was restricted to this algorithm. However, being a distributed algorithm, the Minimum Offload algorithm, is significantly affected by the distribution of server state information. As the frequency of distribution of load information decreases, load balancing errors are encountered as servers then rely upon stale state information with which to perform load balancing. The result of which is that a server that was underloaded but is now overloaded will receive load from misinformed servers in the network. However, by measuring intelligently, ensuring that load balancing is performed between servers that possess the current state information of the entire network, results close to that of a system with complete information can be obtained.

We also investigated, the ordering of server load balancing, giving our servers an ordering to prevent the *herd effect* that can occur through simultaneous load balancing. The three orderings we chose were; forward, backward and random. We found that whilst they all gave similar results, a random ordering would allow for fairer distribution of load throughout the network. In addition, we recommended that the frequency of server measurement be somewhere in the range of less than a

minute. If the server inter-measurement time is smaller, then servers are overloaded less often, but at the cost of a potential increase in bandwidth. On the other hand, if it is any larger, then servers become overloaded, with a potential impact upon the QoS received by users.

To address the second problem, we first explored how QoS should be defined in an MMOG context. The IACMMG and DICE projects are VoIP based and hence QoS for audio communication is primarily dependent upon packet delay and loss. We explored techniques for the gathering of such information from packet traffic and used it along with the ITU-T E-Model to determine perceived audio QoS. This led to the development of a QoS algorithm designed to measure the QoS of conversations within the virtual environment of the MMOG, reacting to the perceived level of QoS, and moving users to servers capable of providing higher levels of QoS should this be necessary. We concluded that by measuring the QoS of the users of a grid square more often resulted in a higher average audio QoS for end users, whilst a lower frequency of grid square measurement will naturally enough result in a lower average audio QoS for users. However, QoS measurement does require significant resources in terms of both bandwidth and processing power, which cannot be justified for the smaller grid square inter-measurement times. Thus we recommended that the frequency of grid square measurement be somewhere around one minute.

In order for a complete solution to be realised, the two architectures need to be merged together. To do this, we modified both the load balancing algorithms and the QoS algorithm to be aware of the other and to take advantage of information the other may provide. The integration of the two systems was carried out with emphasis placed upon minimising server overload as a primary objective, with maximising user QoS a second. The rationale behind this was that server overload has the potential to have an impact upon the audio QoS of all users processed by that server, whilst the poor QoS encountered on a particular grid square is localised to

only the users of that grid square. We found that the effectiveness of the load balancing architecture suffered negligibly in the integrated environment in exchange for significant QoS improvement. We also found that changes in the inter-measurement times of grid square QoS has only a marginal effect upon overall QoS.

The contribution to the field made by this work is that we have been able to combine two disparate systems; a load balancing architecture, and a QoS management architecture, to form a complete system that is currently absent from the literature.

Future Work and Research

The most pressing future work present for the research we have undertaken would be to integrate the load and QoS measurement architecture within the real system that is currently under development. This would not only allow for the inclusion of an integral component but also allow for the validation of our simulated results. However, a number of simplifications and assumptions were made to the measurement architecture during its development that we felt would not affect the underlying structure of the system, these would need to be addressed in order for integration within the real system could be undertaken.

For example, the current measurement architecture assumes that each user within the audio environment is equivalent to a single unit of load. Whilst this is a reasonable assumption to make since a unit of load can represent a later established maximum load a user may contribute, it does not take into consideration users contributing smaller loads. The load a user contributes is made up of the number of audio streams that the user sends and receives. This includes both the streams within the grid square, as well as the streams used for background noise. The benefit of undertaking further research would be to allow for greater accuracy in the provisioning of resources of the IACMMG and DICE [43] projects.

Additional assumptions were placed upon the audio codecs we were investigating. Whilst we did not explore the audio codecs to be used with much depth, we assumed that they did not employ Voice Activity Detection (VAD). Voice Activity Detection allows for a significant reduction in total size of the audio stream by detecting silence during speech and not transmitting silent packets. This has the potential to significantly reduce the number of packets sent within a conversation, as it is custom for participants to take turns when conversing. This may in turn reduce the load placed upon the network and the measurement architecture, and it is for these reasons that further research would be beneficial.

Whilst we feel that we have developed a comprehensive measurement architecture, we found that a number of interesting research avenues arose that could not be explored in much depth and we present them here as possible topics for future research. One possible topic for future research would be in the investigation of the ITU-T E-Model as being suitable for assessing the quality of multi-party VoIP. The work undertaken in this thesis acknowledged that whilst the E-Model was untested for the purposes of multi-party VoIP, and thus may have been unsuitable, it was the only industry standard method for audio quality assessment, and was thus the best method found within the literature. Therefore, it would be interesting to test the abilities of the E-Model to determine if it can be used successfully for quality assessment of multi-party VoIP. If it were found to be non-suitable, what modifications could be made to allow for its use.

Another possible research avenue would be to reinvestigate methods of measurement gathering within data networks. Our investigation examined briefly a number of passive and active methods for gathering measurements. Whilst we finally settled upon the use of active measurement, we did conclude that neither passive nor active were entirely suitable for our measurement purposes. Moreover, the meth-

ods of active measurement examined have the added problem of requiring a large number of probe packets for measurement. Thus, it would be beneficial to conduct further research into other measurement techniques that could be used for the architecture and then to test the suitability of the varying methods of measurement.

Some other research avenues that would be interesting to investigate would including investigating the validity of using a fixed threshold of quality. Instead of using a fixed threshold, a more flexible approach may be more appropriate. From R values, a proportion of users that consider the quality to be Good or Better (GoB), or Worse or Poor (PoW) can be established [24]. This proportion may prove a better decider than a fixed threshold. Another research path would be to look at the effect jitter buffers have upon the quality of the environment. Within VoIP systems, jitter buffers are used to delay packets so that they may arrive in time to be played. They make a compromise between increasing the delay of the audio transmission and the loss of audio data. Investigations into this compromise between further delay and loss may improve predictions of quality. Finally, the movement of users between grid squares of the IACMMG and DICE [43] projects, as described in Chapter 2, has parallels with the movement of users within cellular mobile telephone systems. It is therefore a possibility that analysis techniques found in cellular mobile telephone networks [5] could be adapted for use within MMOG systems.

Bibliography

[1] J. Andren, M. Hilding, and D. Veitch. Understanding end-to-end internet traffic dynamics. In *Globecom '98*, pages 1118–1122, November 1998.

[2] J.-C. Bolot. End-to-end packet delay and loss behavior in the internet. In *Journal of High Speed Networks*, volume 2, pages 289–298, 1993.

[3] F. Bonomi and A. Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers*, 39:1232–1250, October 1990.

[4] F. W. Burton and M. R. Sleep. Executing functional programs on a virtual tree of processors. In *Conference on Functional Programming Languages and Computer Architectures*, pages 187–194, 1981.

[5] G. Cao. Integrating distributed channel allocation and adaptive handoff management for QoS-sensitive cellular networks. *Wireless Networks*, 9(2):131–142, March 2003.

[6] J. Case, M. Fedor, M. Schoffstall, and C. Davin. RFC 1157: Simple network management protocol (SNMP), May 1990.

[7] S. T. Chanson, W. Deng, C.-C. Hui, X. Tang, and M. Y. To. Multidomain load balancing. In *8th IEEE International Conference on Network Protocols*, pages 315–324, November 2000.

[8] Cisco Systems. Echo analysis for voice over IP. *http://www.cisco.com/univercd/cc/doc/cisintwk/intsolns/voipsol/ea_isd.%pdf*, 2001.

[9] D. E. Comer. *Internetworking With TCP/IP Volume 1: Principles Protocols, and Architecture*. Prentice-Hall International, Inc, 2000.

[10] L. Cottrell. Comparison of some internet active end-to-end performance measurement projects. *http://www.slac.stanford.edu/comp/net/wan-mon/iepm-cf.html*, 1999.

[11] J. Davidson and J. Peters. *Voice over IP Fundamentals*. Cisco Press, 2000.

[12] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. In *Proceedings of the Workshop on Passive and Active Measurements, PAM*, pages 556–567, April 2001.

[13] D. Gupta and P. Bepari. Load sharing in distributed systems. In *National Workshop on Distributed Computing*, January 1999.

[14] P. Hudak and B. Goldberg. Experiments in diffused combinator reduction. In *Proceedings of the ACM Symposium on Lisp and Functional Programming*, pages 167–176, 1984.

[15] G. S. Hura, S. Mohan, and T. Srikanthan. On load sharing in distributed systems: A novel approach. In *Transactions of the Society for Design and Process Science*, volume 6, pages 59–81, March 2002.

[16] K. Hwang, W. J. Croft, G. H. Goble, B. W. Wah, F. A. Briggs, W. R. Simmons, and C. L. Coate. A unix-based local computer network with load balancing. In *IEEE Computer*, volume 15, pages 55–66, April 1982.

[17] ITU-T. Recommendation G.723.1. *Dual rate speech codec for multimedia communications transmitting at 5.3 and 6.3 kbits/s*, 1996.

[18] ITU-T. Recommendation G.729. *Coding of speech at 8 kbits/s using conjugate-structure algebraic-code-excited linear-prediction*, 1996.

[19] ITU-T. Recommendation P.800. *Methods for subjective determination of transmission quality*, 1996.

[20] ITU-T. Recommendation G.108. *International telephone connections and circuits - General definitions*, 1998.

[21] ITU-T. Recommendation G.711. *Pulse code modulation (PCM) of voice frequencies*, 1998.

[22] ITU-T. Recommendation G.109. *Definition of categories of speech transmission quality*, 1999.

[23] ITU-T. Recommendation G.113. *International telephone connections and circuits - General Recommendations on the transmission quality for an entire international telephone connection*, 2001.

[24] ITU-T. Recommendation G.107. *The E-Model, a computational model for use in transmission planning*, 2002.

[25] ITU-T. Recommendation G.113 Appendix I. *Provisional planning values for the equipment impairment factor Ie and packet-loss robustness factor Bpl*, 2002.

[26] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Communication Review*, pages 75–88, July 2002.

[27] K. Y. Kabalan, W. W. Smari, and J. Y. Hakimian. Adaptive load sharing in heterogeneous systems: Policies, modifications, and simulation. In *International Journal of Simulation: Systems, Science & Technology*, volume 3, pages 89–100, June 2002.

[28] H. D. Karatza and R. C. Hilzer. Load sharing in heterogeneous distributed systems. In E. Yucesan, C. H. Chen, J. L. Snowdon, and J. M. Charnes,

editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 489–496, 2002.

[29] L. Kencl and J.-Y. L. Boudec. Adaptive load sharing for network processors. In *INFOCOM'02*, pages 545–554, June 2002.

[30] H. Kuchen and A. Wagener. Comparison of dynamic load balancing strategies. Technical Report 90-05, Technical University of Aachen (RWTH Aachen), 1990.

[31] F. C. H. Lin and R. M. Keller. The gradient model load balancing method. In *IEEE Transaction on Software Engineering*, volume 13, pages 32–38, 1987.

[32] P. K. K. Loh, W. J. Hsu, C. Wentong, and N. Sriskanthan. How network topology affects dynamic load balancing. In *IEEE Parallel & Distributed Technology*, pages 25–35, September 1996.

[33] A. Markopoulou, F. Tobagi, and M. Karam. Assessing the quality of voice communications over internet backbones. *IEEE/ACM Transactions On Networking*, 11(5):747–760, 2003.

[34] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Analysis of the effects of delays on load sharing. In *IEEE Transactions on Computers*, volume 38, pages 1513–1525, 1989.

[35] M. Mitzenmacher. How useful is old information? In *IEEE Transactions On Parallel and Distributed Systems*, volume 11, pages 6–20, January 2000.

[36] S. Moller. *Assessment and Prediction of Speech Quality in Telecommunications*. Kluwer Academic Publishers, USA-Boston, 1991.

[37] A. Mondal, K. Goda, and M. Kitsuregawa. Effective load-balancing of peer-to-peer systems. In *Proceedings of Data Engineering Workshop (IEICE DEWS) DBSJ Annual Conference*, March 2003.

[38] B. Northcote. Real-time audio composition load balancing. Technical report, Smart Internet Technologies Cooperative Research Centre, 2004.

[39] P.-O. Osland and P. J. Emstad. Performance evaluation of load scheduling policies of intelligent networks. In *Teletraffic Engineering in a Competitive World*, volume 3a, pages 283–292. Elsevier, June 1999.

[40] A. Pasztor and D. Veitch. A precision infrastructure for active probing. In *Proceedings of Passive and Active Measurements PAM workshop*, pages 33–44, 2001.

[41] J. Postel. User datagram protocol. RFC 768. Internet Engineering Task Force, August 1980.

[42] M. Roussopoulos and M. Baker. Practical load balancing for content requests in peer-to-peer networks. cs.ni/0209023, Stanford University, 2003.

[43] F. Safaei, P. Boustead, C. D. Nguyen, J. Brun, and M. Dowlatshahi. Latency-driven distribution: Infrastructure needs of participatory entertainment applications. *IEEE Communications Magazine*, 43(5):106–112, May 2005.

[44] D. Sanghi, A. K. Agrawala, and B. Jain. Experimental assessment of end-to-end behavior on internet. In *IEEE INFOCOM '93, San Fransisco*, pages 867–874, March 1993.

[45] M. Schaar, K. Efe, L. Delcambre, and L. Bhuyan. Load balancing with network cooperation. In *IEEE International Conference on Distributed Computing Systems*, pages 328–335, 1991.

[46] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications RFC1889.

[47] A. Takahashi. Opinion model for estimating quality of VoIP. *Proc. IEEE ICASSP '04*, pages 1072–1075, 2004.

[48] J. Watts and S. Taylor. A practical approach to dynamic load balancing. In *IEEE Transactions on Parallel and Distributed Systems*, volume 9, pages 235–248, 1998.