Quality Planning for Distributed Collaborative Multimedia Applications

Ph.D. Thesis

Mark Claypool
Advisor: John Riedl

University of Minnesota
Computer Science Department

September 2, 1997

Acknowledgements

There are many people who have helped me in this work, either directly by adding their work to mine or indirectly by giving me vision and support. However, above all there are two people who deserve my deepest gratitude, Professor John Riedl and my wife, Kajal. As my advisor, John has contributed to all levels of this thesis, from guidance on writing C code and English grammar rules to experimental design and data analysis to a rich, exciting vision of computer science research. Most importantly, he has provided motivation to persevere through the slow, difficult parts of research, motivation to pursue research problems that make a difference in the world, and motivation to pursue a career as a professor, myself. I hope to emulate much of John's style when I advise my future graduate students.

Equally important, Kajal has been a pillar of strength and support. She has been understanding of the fluctuations in the demands on the time of a graduate student. She has been patient through the many years she worked to support us while I was still in school, working on this thesis. She has been inspirational when I was down about progress or down about some experiment results. And her technical knowledge has provided me with nudges in the right direction when my research lacked focus. I hope I can be as supportive of her during her Ph.D. quest as she was during mine.

I would like to thank the members of my examining committee, Professors John Carlis, Shashi Shekhar, David Lilja and George Wilcox. Professors Shekhar and Lilja were also on my Masters committee, work which was later molded to this thesis. Professors Carlis and Wilcox collaborated with me on the "Flying" interface, work

which became Chapter 6 of this thesis. Professor Carlis provided valuable insights during the very last phase of this document, including writing the abstract and making my final defense a successful talk.

I also must thank those who long ago gave me the tools to achieve a Ph.D. Professors Steven Janke, David Roeder and Michael Siddoway, professors at Colorado College, taught me the elegance of academic work and gave me insights into computer science. My high school teacher, Herr Berndt, gave me the appreciation for the richness of life, a vision which helped me see the beauty in computer science, as well. Above all, my parents taught me how to set my sites high and how to succeed, as well as providing me with my first computer, a TRS-80 model III. Little did any of us know it at the time, but that computer became the reason I am writing this document today.

I would like to thank several additional people for their direct contributions to my work: Dan Frankowski, Mike Maley, Mike Stein, Vahid Mashayekhi and Joe Habermann all worked with me closely during several smaller research projects that led to this thesis.

Without the contributions of the above people, this work would never have been completed, nor even begun.

Vita

Mark Claypool was born in Washington D.C. on September 16, 1968. He attended primary and secondary school in Colorado and finished his secondary education at Nuremburg American High School in Nuremburg, Germany. In 1990, he completed his Bachelor of Arts in Mathematics at Colorado College, in Colorado Springs, CO. He entered the Computer Science program at the University of Minnesota in the fall of 1990. He received his M.S. and Ph.D. in Computer Science in the summer of 1993 and the summer of 1997, respectively.

Abstract

The tremendous power and low price of today's computer systems have created the opportunity for exciting applications rich with graphics, audio and video. These new applications promise to support and even enhance the work we do in teams by allowing users to collaborate across both time and space. Despite their exciting potential, building distributed collaborative multimedia applications is very difficult and predicting their performance can be even more difficult. Performance predictions must deal with future hardware, future software, and future users and their requirements. The rate of change in new technologies continues to accelerate, with each new generation of hardware and software introducing new capabilities. Furthermore, distributed environments have a wider variety of potential configurations than do centralized systems.

To determine the computer resources needed to meet distributed application demands we have developed a flexible model and a method for applying it that allows us to predict multimedia application performance from the user's perspective. Our model takes into account the components fundamental to multimedia applications: latency, jitter and data loss. The contributions of this thesis, to both computer scientists and computer system developers, are: 1) a multimedia application quality model and method for predicting application performance and evaluating system design tradeoffs; 2) detailed performance predictions for three distributed collaborative multimedia applications: an audioconference, a "flying" interface to a 3D scientific database and a collaborative flight simulator; 3) the effects of system improvements

on these multimedia applications; 4) a demonstration of measures of jitter that have been used by jitter researchers showing all reasonable choices of jitter metrics are statistically similar; and 5) experiment-based studies of the effects of high-performance processors, real-time priorities and high-speed networks on jitter.

In predicting the bottlenecks in quality for the above three applications, we have identified three general traits: 1) processors are the bottleneck in performance for many multimedia applications; 2) networks with more bandwidth often do not increase the quality of multimedia applications; and 3) performance for many multimedia applications can be improved greatly by shifting capacity demand from computer system components that are heavily loaded to those that are more lightly loaded.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

> *"If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100 and get a million miles per gallon..."* Robert X. Cringely, InfoWorld

Never before has industry seen the tremendous change that is transforming today's computers. There has been a tremendous growth in computer performance and reliability, accompanied by an equally dramatic decrease in price. Today, a few thousand dollars will buy more performance, memory and disk storage than a million dollar computer bought in 1965. Since 1965, the performance growth rate for supercomputers and mainframes has been just under 20% per year, and the performance growth rate for microprocssor-based computers has been about 35% per year [56]. There has been an even more explosive growth in computer networks. The number of hosts on the Internet has roughly tripled in the time from January 1994 to January 1996 [47]. And the World Wide Web has grown substantially faster than the Internet. The rate of the World Wide Web's growth has been and continues to be exponential, doubling in number of hosts in under six months [47].

With world-wide networks connecting a myriad of powerful computers, there is

a growing need for collaborative applications. For thousands of years, people have done their best work when they can interact and work together. Only recently they discovered the computer and now seek to use it to solve problems. The application of the computer to cooperative work can support and even enhance the way we work together. Computer supported distance learning provides educational opportunities to remote students [102, 2, 49]. Scientific collaboration can be enhanced when supported by computers [15] as can military training [25]. Businesses can also benefit from computer-assisted collaboration. Business transactions can be more efficient through the use of electronic cash [52]. Business costs and errors can be reduced by the use of Electronic Data Interchange applications [5, 95].

Computer-supported cooperative work (CSCW) helps to overcome time-space limitations and allows people to share information from distributed locations at the same or different times. Collaborative applications can be enhanced by multimedia. People communicate best when they can draw pictures and use voice inflections and body language rather than simply type text. Communication can more closely resemble face-to-face interaction when computers are used through the use of record/playback, on-screen speaker identification, floor control and subgroup communication [129]. Multimedia can even enable collaboration environments that do not currently exist today, such as virtual reality [9].

1.2 Problem

> *"A Picture is worth a thousand words. That's why it takes a thousand times longer to load."* Anonymous.

Despite the growing need and prevalence of distributed, collaborative multimedia applications, planning for proper computer systems to support multimedia effectively is still difficult. Multimedia has greater system requirements than traditional text-based applications. Figure 1.1 gives a coarse illustration of the number of bytes

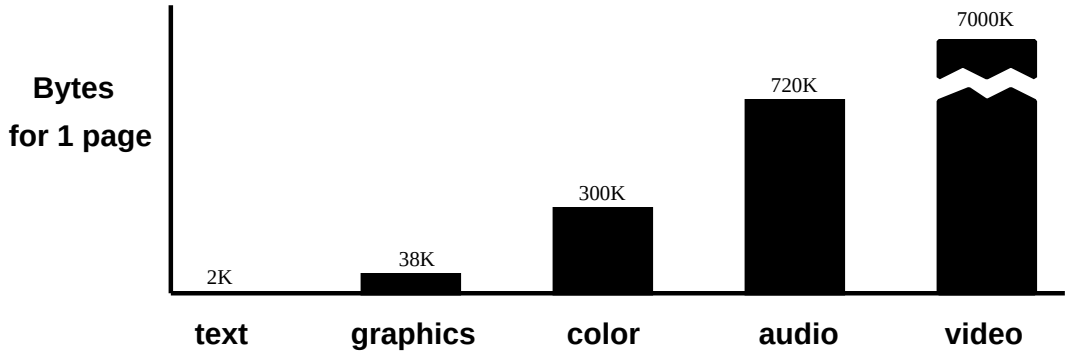

Figure 1.1: Relative Bytes Required for Multimedia. The bars indicate the number of bytes required to convey a paper-sized page of information in each type of media. We assume a standard `8.5 x 11` inch piece of paper containing 66 lines of text, 80 characters per line. Text is the number of bytes required to store the page in 8 bit ASCII characters. Graphics is the number of bytes required to render those characters in postscript. Color is the number of bytes to convey the postcript in 8-bit color. Audio is the number of bytes required to read that page with an audio sampling rate of 8000 bytes/second. Video is the number of bytes required to read that page of text over a small 3 frames/second video stream.

required to convey one page of information in various media. Each step up in media requires increased computing capabilities. Text, graphics, and color require relatively little computing capabilities. Audio is feasible on most workstations currently being marketed. Video, however, requires high-end multimedia workstations. And when multimedia applications support multi-users, the system requirements are increased even more.

Moreover, the different media each place different constraints on computer systems. For instance, human eyes can smooth over occasional glitches in a video stream more readily than human ears can smooth over breaks in an audio stream [26]. Having the computer system place greater emphasis on preserving audio data than on preserving video data might be important for user satisfaction.

In addition, the constraints for each type of media can vary from application to application. For example, the acceptable delay for an audio broadcast application

such as a radio program may be far more than the acceptable delay for an audioconference. In an audioconference, users require low latencies so that the conversation is as life-like as possible. However, in an audio broadcast program, the users do not interact, allowing a larger delay to go unnoticed. You could imagine a case where a user downloads an entire radio program overnight and then plays it back in the morning. In this case, a delay of over twelve hours might be quite acceptable.

Multimedia applications that must run in a distributed environment and support many users pose even more challenges. Network software components for multimedia environments have often been overshadowed by issues such as human interface design or graphics realism [81]. This has led to the fact that there are not many good solutions to network problems that scale to many users. Even schemes that have readily apparent benefits such as IP multicast have been slow to be adopted [37]. Even commercial systems are finding the limits to current technology in creating networked environments for large numbers of users. America Online provides low-bandwidth mltiuser games, messaging and chat services for about a million subscribers, yet its recent attempt to provide unlimted access time created too much demand for their modem servers to handle.

Collaborative applications have additional communication requirements and paradigms that need to be explored and supported. Cooperative processes among humans, involving issues of conflict, selective attention, self-interest, trust and privacy are not understood well enough, especially when computers are used to support such processes [59]. Separating time and space introduces new problems in building work-enhancing collaborative applications, such as the continued need for face-to-face meetings and support of roles in normal communication environments [83].

Planning is the first fundamental step in developing a software system. And not only are distributed, collaborative multimedia applications difficult to build, their performance is difficult to predict, also. Part of the difficulty arises from the difficulty in predicting the future. Performance predictions must deal with future hardware,

future software, and future users and their requirements. Even the users themselves often cannot predict what they want. The rate of change in new technologies continues to accelerate. Often, by the time the performance of a technology is understood it is obsolete. Furthermore, distributed environments have a wider variety of potential configurations that do centralized systems. And each new generation of hardware and software introduces new capabilities.

1.3 Our Solution

In order to build systems that will adequately support distributed collaborative multimedia applications, it is important to predict application performance as the number of users increases and evaluate performance and cost tradeoffs for different system designs. We have developed a solution to predict bottlenecks in the performance of such applications. The contribution of our solution is depicted by Figure 1.2.

Our solution consists of a general-purpose method to predict the performance of multimedia applications. At the heart our method is a flexible model, adjustable to applications with different user requirements and systems with different system designs and hardware. Our model uses a quality metric as a means of measuring the performance of an entire computer system.

One indication of the performance of an entire computer system is the users' opinions on the *multimedia quality* of the applications they run. Multimedia quality is a measure of the performance of a multimedia application based on the requirements expected by the user. If the user performance requirements are met, application quality will be acceptable. If the user performance requirements are not met, application quality will be unacceptable. We have developed a quality planning model to aid in designing systems that meet users' quality requirements for multimedia applications in the future.

Although we often think of a multimedia application as a continuous stream of

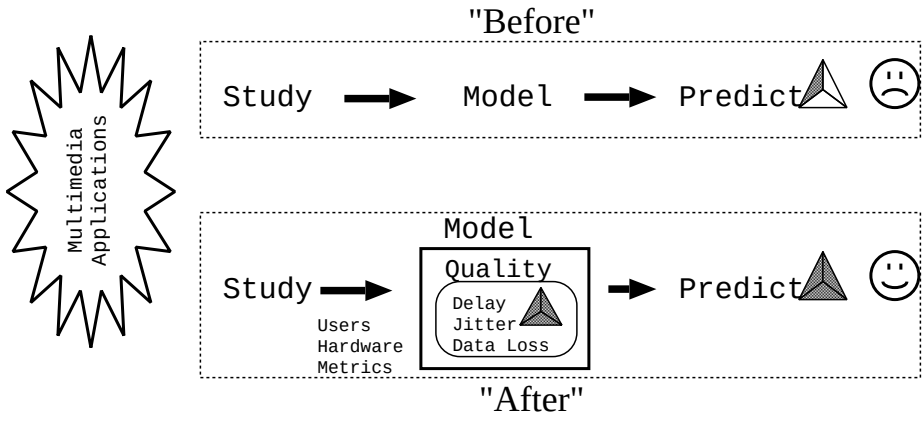

Figure 1.2: Our Contributions. The area on the left labeled "Multimedia" represents the space of future multimedia applications. "A" and "B" are two different applications. "Before" shows how the performance of A and B is predicted without the contribution of our method. "After" shows how the performance of A and B is predicted with the contribution our method. The two figures on the right represent the users of the applications. The shaded triangles represent performance parameters that the users care about.

data, computer systems handle multimedia in discrete events. An event may be receiving an update packet or displaying a rendered video frame on the screen. The quantity and timing of these events give us measures that affect application quality. Based on previous multimedia application research [114, 6, 100, 60, 92, 96, 84, 39, 75, 115, 90, 40], we have identified three measures that determine quality for most distributed multimedia applications:

- *Latency.* The time it takes information to move from the server through the client to the user we call *latency.* Latency decreases the effectiveness of applications by making them less like real-life interaction [128, 60, 100, 32].

- *Data Loss.* Any data less than the amount determined by the user requirements we call *data loss.* Data loss takes many forms such as reduced bits of color, pixel groups, smaller images, dropped frames and lossy compression [6, 88, 84, 114]. Data loss may be done voluntarily by either the client or the server in order to reduce load or to reduce jitter and/or latency.

- *Jitter.* Distributed applications usually run on non-dedicated systems. The underlying networks are often packet-switched and the workstations are often running multiple processes. These non-dedicated systems cause variance in the latency, which we call *jitter.* Jitter can cause gaps in the playout of a stream such as in an audioconference, or a choppy appearance to a video display [66, 96, 68].

The effects of latency on a user's perception of an application is well-understood and well-researched [128, 60, 100, 32]. Similarly, there is a clear relationship between data loss and application quality deterioration [6, 88, 84, 114]. Methods to ameliorate the effects of jitter have been explored by many researchers [113, 96, 39, 20]. The tradeoff between buffering and jitter has also been explored [75, 112]. We contribute to the investigation of the effects of high-performance workstations, real-time priorities and high-speed networks on jitter. Moreover, we incorporate these jitter results with latency and data loss into a general model for application quality. Lastly, we study

of the effects of different system configurations in our general model for application quality.

There may be additional measures that affect application quality that are application specific. For example, Distributed Interactive Simulation applications use a process of computing the location of other simulators through "dead reckoning" [55]. When state update packets are dropped, the accuracy of the simulation decreases [24]. The use of dead reckoning creates an additional quality measure specific to DIS applications. In the rest of this thesis, we consider only delay, jitter and data loss, except where noted, but our analysis could be similarly applied to such application specific measures.

1.4 Applications

In order to determine the quality of distributed collaborative multimedia applications we must first understand understand the user requirements and derive the system requirements. Some of the user requirements that affect quality and determine system requirements include: number of users, type of media, time of collaboration and distribution of data. Users can range from several to several thousand. Media can be any combination of text, color, audio, video and graphics. Collaboration time can at the same time (synchronous) or at different times (asynchronous). Data can reside in a central repository (client-server) or data can be communicated equally among clients (peer-to-peer). We study three emerging applications that span these characteristics for distributed collaborative multimedia applications:

1. *Audioconferences.* Audioconferences have been shown to enhance collaborative work among users on distributed workstations [98, 82, 104]. Why are audioconferences becoming so important? Hearing is one of our strongest senses. Thus, sound is one of our most powerful forms of communication. If we wish to use the flexibility and power of computers to support communication and

collaboration, then they must support audio data.

2. *Zoomable Database.* Neuroscientists from diverse disciplines plan to collaborate across distances in exploring various aspects of brain structure [15]. Their design includes a zoomable multimedia database of images of the brain tissue. High-resolution magnetic resonance imaging (MRI) shows the entire brain in a single dataset. Even higher resolution confocal microscope images are anchored to these MR images in three dimensions. The user starts a typical investigation by navigating through the MR images in a coarse 3-d model of the brain to a site of interest. The user then zooms to higher resolution confocal images embedded in the MRI landscape. This real-time navigating and zooming is called "flying." In order to be an effective collaboration tool, flying must provide high-resolution images and a high-frame rate as well as high-quality audio and video to allow neuroscientists to communicate effectively.

3. *Distributed Interactive Simulation.* Distributed Interactive Simulation (DIS) applications are designed to enable soldiers to engage in simulated combat [25]. The DIS protocol allows participation from soldiers at military bases across the country using current packet-switched networks, saving the time and trouble of traveling for combat training. Many DIS developers are designing simulators that use off-the-shelf general purpose hardware [85]. In order for the combat to be realistic, the simulators use high-quality graphics and allow communication among the soldiers with audio and video. With the high multimedia system requirements and many users, applications such as DIS applications will stress all parts of a computer system.

Many text-based applications, such as email, have simple user requirements: send and receive mail, byte for byte, in a reasonable amount of time, *i.e.* reasonable latency and no data loss. Other text-based applications, such as file transfer, have more stringent user requirements: put and get files, byte for byte, in a short amount

Application	Users	Media	Time	Distribution
Audioconference	2 to 10s	audio	synchronous	peer-to-peer
Zoomable Database	10s to 100s	color, text, video, graphics	asynchronous	client-server
Distributed Interactive Simulation	100s to 1000s	color, audio, video	synchronous	peer-to-peer

Table 1.1: Three Distributed Collaborative Multimedia Applications. We study the three applications listed in this table above. Users are the number of users who will simultaneously use the application. Media is the type of media the application provides. Time is "synchronous" if users work at the same time and "asynchronous" if users work at different times. Distribution is "client-server" if workstations access a centralized data repository or "peer-to-peer" if workstations communicate equally to other workstations.

of time, *i.e.* short latency and no data loss. Multimedia applications, such as a video conference, have more complicated user requirements: record and play video frames, keep the frames mostly clear, and deliver at a fast and reasonably steady rate *i.e.* low latency, low data loss and low jitter.

Table 1.4 summarizes the application characteristics.

The rest of this thesis is organized as follows:

Section 2 introduces a taxonomy of computer support for multimedia quality. Our taxonomy describes our quality model for multimedia applications and categorizes the means with which computer systems support multimedia.

Section 3 presents an in-depth study of jitter, one of the fundamental components of our model. We present: a comparison of measures of jitter that have been used by jitter researchers; an experimentally-based study of the effects of high-performance processors, real-time priorities and high-speed networks on jitter; and predictions on the importance of application jitter reduction techniques in the future.

Section 4 describes the method we use to apply our quality planning model to distributed, collaborative multimedia applications. We describe our model of the user, application and system and detail how our experiments are used as the basis

for our model and as a means of testing the accuracy of our predictions.

Section 5 applies our method to audioconferences. We introduce audioconferences and present our hypotheses. We then apply our model and predict audioconference performance under a variety of system configurations, including faster processors, networks and Digital Signal Processing (DSP) hardware.

Section 6 applies our method to a 3-d interface to a scientific brain database. In applying our model, we move through bottlenecks in application performance from the network, processor and disk and arrive at a means to configure the flying interface to fit on a proposed national network topology.

Section 7 applies our method to the Virtual Cockpit. We describe the Virtual Cockpit in detail, including a protocol designed to reduce network bandwidth. We then apply our model and predict the performance of the virtual cockpit under high-speed networks and high-performance processors.

> *"A man who does not read good books has no advantage over the man who can't read them."* Mark Twain

Each Chapter has a "Related Work" section that discusses research that is specifically related to that section.

Chapter 2

A Taxonomy of Computer Support for Multimedia Quality

2.1 Overview

We present a taxonomy of multimedia application quality. Our taxonomy seeks to categorize popular multimedia research topics into three main groups. It defines and clarifies the usage of several common terms used when building multimedia systems. In addition, it provides a framework on which research in application quality can build. We may use our taxonomy as a map towards finding and improving the performance of a multimedia application. In looking to improve a computer system's support for multimedia, we can use our model obtain a quantitative measure of application quality. We locate the current bottleneck in acceptable application quality and determine how appropriate system changes from the taxonomy categories affect the application quality.

A taxonomy represents a view of the sub-fields of a research area. For a rapidly developing and evolving research area such as multimedia, a taxonomy represents a snapshot of the field at one point in time. A taxonomy can change, and should change as research in the field evolves. The view presented in any taxonomy is subjective,

based on the background and experience of those developing the taxonomy.

A taxonomy can be useful for categorizing work in the field. Note that research very rarely fits neatly into exactly one category. More often, it overlaps two or three topics, so that taxonomies should often not be considered strict or binding. However, the very process of developing a taxonomy forces a focus on the differences between the defining keywords of the field, and may help clarify their usage.

This Chapter is organized as follows: Section 2.2 presents related work in taxonomy and quality. Section 2.3 introduces our model of multimedia applications that allows us to predict user satisfaction with a computer system and application configuration. Section 2.4 describes computer systems support for multimedia and presents our taxonomy and Section 2.5 summarizes the important contributions of this section, including a start at defining the relationship between our quality model and our taxonomy.

2.2 Related Work

2.2.1 Taxonomy

Heller and Martin described a media taxonomy that serves both research and development of multimedia applications. It correlated well with previous categorizations of multimedia and helped researchers better understand the impact and value added by an individual medium in a multimedia presentation. Their media taxonomy included a table with rows specifying media types: text, graphics, sound and motion; with columns specifying media expressions: elaboration, representation and abstraction.

In describing a taxonomy for music, Pope made a strong case for the usefulness of taxonomies [93]. Researchers in the field of computer music struggling with the notion of a taxonomy identified seven hierarchical areas related to composition. Their categories were: theory, acoustics, representation, synthesis, hardware, education and history.

We present a taxonomy for multimedia application quality. We examine how computer systems support multimedia applications in light of this taxonomy.

2.2.2 Quality of Service

Quality of Service (QoS) represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application. The user should be the starting point for overall QoS considerations. Some QoS research defines user limits of tolerability for application performance. The user-defined QoS parameters are converted into system-level QoS parameters. System parameters include such notions as throughput and delay.

Kalkbrenner presented an interface to map user choices into system parameters [71]. For example, users chose image size, resolution, and color for video and speech of telephone or CD quality for audio. These choices are then mapped into lower-level system parameters.

Wijesekera and Srivastava defined QoS parameters for continuity and synchronization specification of continuous media presentations [125]. They defined metrics for average frame rate, variance in average frame rate, frame losses and synchronization.

Another category of QoS research takes user-defined parameters and maps them into parameters for various system level components, such as network throughput. Naylor and Kleinrock developed a model for measuring the quality of an audioconference based on the amount of dropped frames and client-side buffering [75].

We develop a model for application quality that covers a wider scope than past QoS research. Our model provides a view of quality from the user level. Thus, our model incorporates both the workstation and network when determining application quality. In addition, our model is tunable to different networks and workstations, allowing it to be applied to systems of the future.

2.3 Quality

> *"There is an old network saying: 'Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed – you can't bribe God."'* David Clark, MIT

Although there are many different measures of computer system performance, ultimately, the users of the applications must be satisfied. A multimedia performance metric must account for components fundamental to multimedia applications. As described in Section 1, we have identified three fundamental measures that determine quality for most distributed multimedia applications: latency, jitter and data loss. The quality of a distributed multimedia application is a measure of the application's acceptability to the user. Our quality model allows us to compare performance for different system configurations from a user-centered approach.

Ideally, we would like there to be no latency, jitter or data loss. Unfortunately, on a variable delay network and non-dedicated computer this can never be achieved. To compute the application quality, we use the above quality components in a process depicted by Figure 2.1. The user requirements for the application define the acceptable latency, jitter and data loss. The system determines the predicted latency, jitter and data loss. Acceptable and projected data are fed into a *quality metric* for the application. The quality metric is a function, based on the acceptable components and dependent upon the projected components, that computes the application quality.

In order to quantitatively compare application quality for different system configurations, we need a reasonable quality metric. In the mathematical sense, given a space S with at least 3 elements (x,y,z) a metric is a real function of 2 variables `D(x,y)` such that [42]:

1. `D(x,y) = 0` iff x=y (x and y are the same elements)
2. `D(x,y) = D(y,x)` (symmetry)

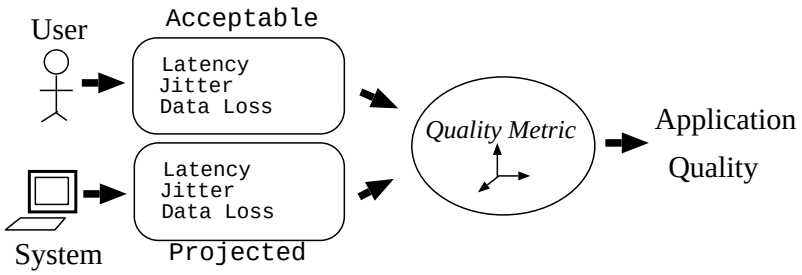

Figure 2.1: The Process for Computing Application Quality. The user defines the acceptable latency, jitter and data loss and the system determines the actual values. Based on the acceptable values specified in the user requirements, a quality metric computes the application quality from the actual values.

3. `D(x,y) >= 0` (non-negative)

4. `D(x,y) + D(y,z) >= D(x,z)` (triangle inequality, which says you cannot gain by going through an intermediate point)

A metric is sometimes called distance between 2 points in any space. We further define a "multimedia quality metric" as having several other important properties:

1. It incorporates the three fundamental multimedia quality components: latency, jitter and data loss.

2. It treats the fundamental components equally, which seems appropriate in the absence of user studies to the contrary.

3. It produces a convex region of acceptable quality. This fits our intuition about changes in quality: the measure increases total quality with any increase in quality along one axis. There are no pockets of unacceptable quality within the acceptable quality region, nor can you move from unacceptable to acceptable by any combinations of increase along the axes.

To form our quality metric, we build upon the work of Naylor and Kleinrock [75]. Naylor and Kleinrock developed a model for measuring the quality of an audioconference based on the amount of dropped frames and client-side buffering. We extend this model by using each quality component as one axis, creating a multi-dimensional quality space. We place the best quality value for each axis at the origin and scale each axis so that the user-defined minimum acceptable values have an equal weight. An instantiation of the application lies at one point in this space. The location of the point is determined by our predictions of the amount of latency, jitter an data loss that would occur with the given system configuration. In order to satisfy the mathematical properties above, we compute the application quality by taking the Euclidean distance from the point to the origin. All points inside the region defined by the user-defined minimums have acceptable quality while points outside do not.

Figure 2.2 depicts a 3-d quality space for multimedia applications. The user requirements determine a region of acceptable application quality, depicted by the shaded region. All points inside the shaded region have acceptable quality, while those outside the region do not. An instantiation of the application and the underlying computer system would lie at one point in this space.

Our metric attains all of the mathematical metric properties and multimedia metric properties listed above.

There can be many possible quality metrics for a given application. In fact, there may be many quality metrics that agree with a user's perception of the application. Mean opinion score (MOS) testing can be used to determine if a metric agrees with users' perception. The MOS is a five-point scale where a MOS of 5 indicates perfect quality and a score of 4 or more represents high quality. MOS has been used extensively in determining the acceptability of coded speech. MOS testing is beyond the scope of this paper, so we cannot be certain our quality metric fits user perceptions. However, the rest of our model is independent of the quality metric chosen. If new metrics are developed and validated with MOS testing, they can be used in place of

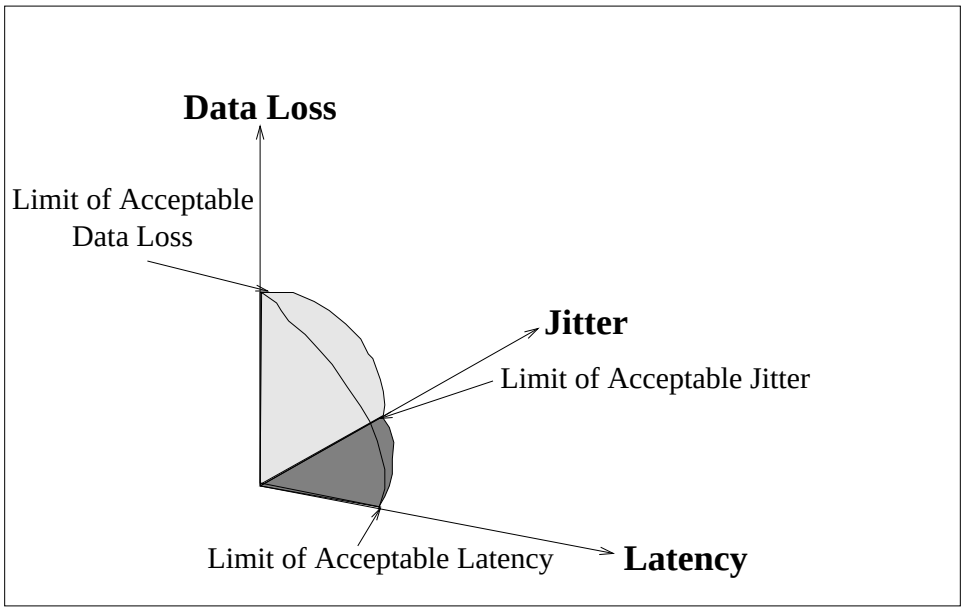

Figure 2.2: Application Quality Space. The user defines the acceptable latency, jitter and data loss. These values determine a region of acceptable application quality, depicted by the shaded region. All points inside the shaded region have acceptable quality, while those outside the region do not. An instantiation of the application and the underlying computer system would lie at one point in this space.

our quality metric.

One limitation to multimedia quality metrics is that after scaling, the upper limits on the axes have different characteristics. The "data loss" axis has a finite upper-limit of 100%, while the "latency" and "jitter" axes each have an infinite bound. Comparing application quality for two different configurations at the upper-limit of any of the axes may not match user perception. Fortunately, this limitation only arises when comparing two unacceptable configurations. The metric is most valuable for determining whether a configuration provides "acceptable" or "unacceptable" application quality and comparing configurations within the "acceptable" region.

2.4 Computer Support for Multimedia

From the time that there was more than one computer, there has been a range of computer processing capabilities. As the number of low- and high-end computers continues to grow, the size of this range continues to expand. Just as the capacity of computer systems has grown, so has the number of applications. Today, as new computer systems emerge [123], new applications are being developed [8, 31] and old applications are being enhanced [43, 33]. The growth of computers has resulted in a large space of possible system capacities, increasingly being filled by developing and emerging applications.

The ability of today's computer systems to support a given multimedia application falls into one of three categories, represented in Figure 2.3. The computer system can have abundant enough resources to always provide acceptable application performance. For example, today most computer workstations provide a graphical user interface. Or, the computer system can have insufficient resources to every provide acceptable application quality. For example, a typical workstation and network will not have enough resources to provide adequate quality for the Virtual Cockpit [24]. In between these first two scenarios lies the "Region of Scarce Resources" [4]. Within

this region, a computer system will have enough resources to adequately support the application *only* if the resources are carefully utilized. For example, workstations that can provide acceptable audio quality most of the time, may suffer from gaps in an audio stream when the audio application is competing for resources with another application.

Yesterday, computer systems could handle text, but struggled with graphics. Today, they can handle audio and video, but to do so computer resources must be carefully managed. Tomorrow, we will have GigaFLOP processors and Terabit networks so we will not have a bottleneck for audio and video, just as today there is no bottleneck for graphics. However, there will always be a region of scarce resources. As capacity and capacity demand both increase, so does the size of the region of scarce resources, enabling more applications that will require careful capacity planning. Moreover, applications continue to expand to fill (or surpass) available capacity. A recent quote from Byte Magazine summarizes this situation:

> *"... imagine a 15,000 MHz processor, 1600 MB of RAM and a 100 GB hard drive for less than 20 dollars and the new types of applications that it would spawn. I personally think the new word processor releases would have enough fatware to bring that system to its knees!"* Daren Coppock, Byte, December 1996

If Daren Coppock is correct, some multimedia applications will *always* lie in the region of insufficient or scarce resources particularly as the number of collaborative users increases. Applications and system resources must be enhanced so that applications currently in the region of insufficient resources will lie in the region of scarce resources. At the same time, current and future resources must be carefully managed so that applications in the region of scarce resources will have acceptable quality. In light of this view, we have begun developing a taxonomy for the areas of computer science research that benefit the performance of multimedia applications. Our tax-

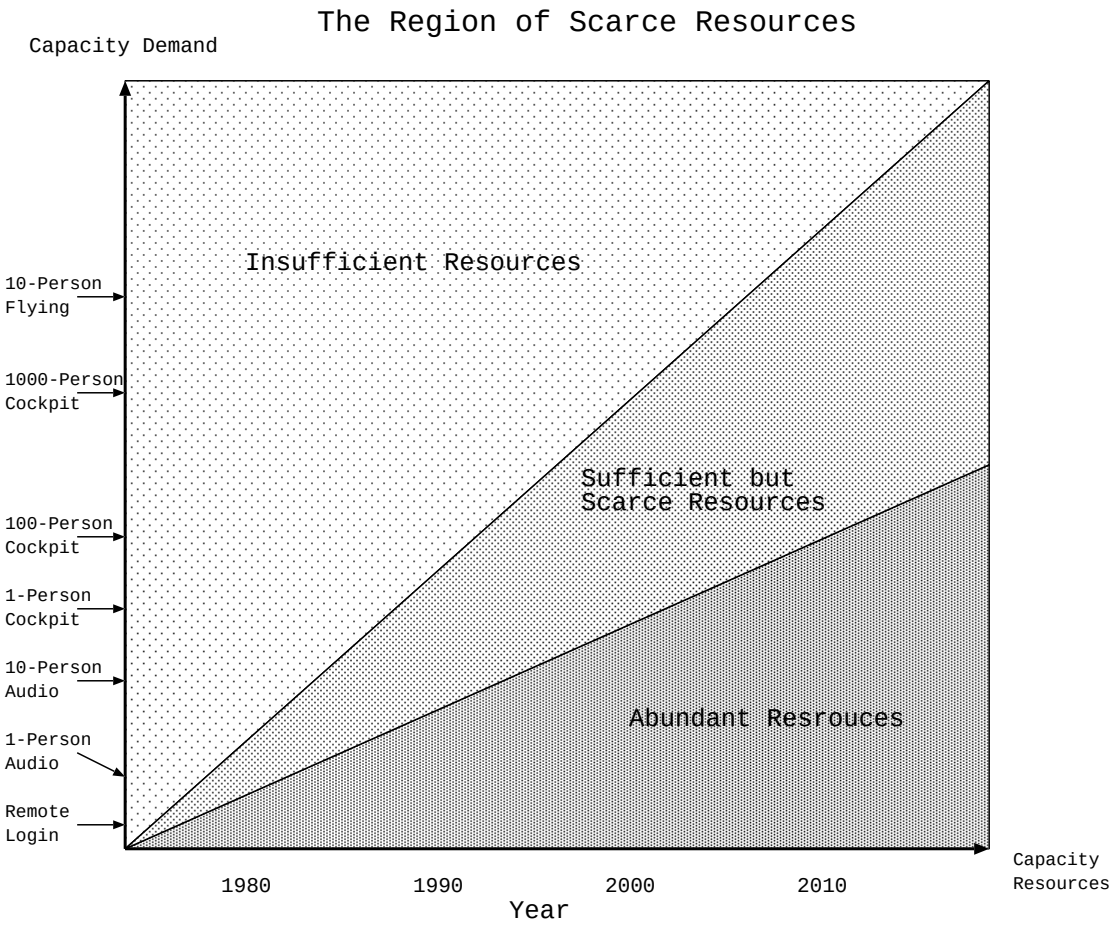

Figure 2.3: The Region of Scarce Resources. The horizontal axis represents the year and corresponding capacity increase. The vertical axis is the capacity demand. The regions represent areas in which there are insufficient resources to meet demand, sufficient but scarce resources to meet demand and abundant resources to meet demand.

onomy is a developing document and is meant to represent the current research areas for multimedia application performance.

In our taxonomy, we identify three categories of research aimed at improving multimedia application performance:

- *Capacity* research seeks to increase computer system resources to meet the performance requirements for multimedia applications. Capacity research strives to move applications currently in the region of insufficient or scarce resources into the region of abundant resources.

- *Scheduling* research seeks to use computer resources in a more effective manner for supporting multimedia applications. Scheduling research attempts to make the performance of applications inside the region of scarce resources more acceptable.

- *Reservation* research seeks methods to reserve computer resources for more consistent multimedia application performance. Reservation research attempts to improve the performance of applications inside the region of scarce and abundant resources.

These three categories are depicted in Figure 2.4. The boxes inside each category represent sub-categories of multimedia quality research. The words in italics represent specific examples of research taking place within a research sub-category. As our taxonomy will show, there is enormous breadth and depth of the research in improving multimedia application performance. Our work concentrates on carefully analyzing the performance improvements of a few specific areas: the capacity improvements of processors, networks and disks; and real-time operating system priorities.

2.4.1 Capacity

Computer system capacity is the maximum available amount of output over a given time period. Capacity is the ultimate prerequisite for handling multimedia. Neither

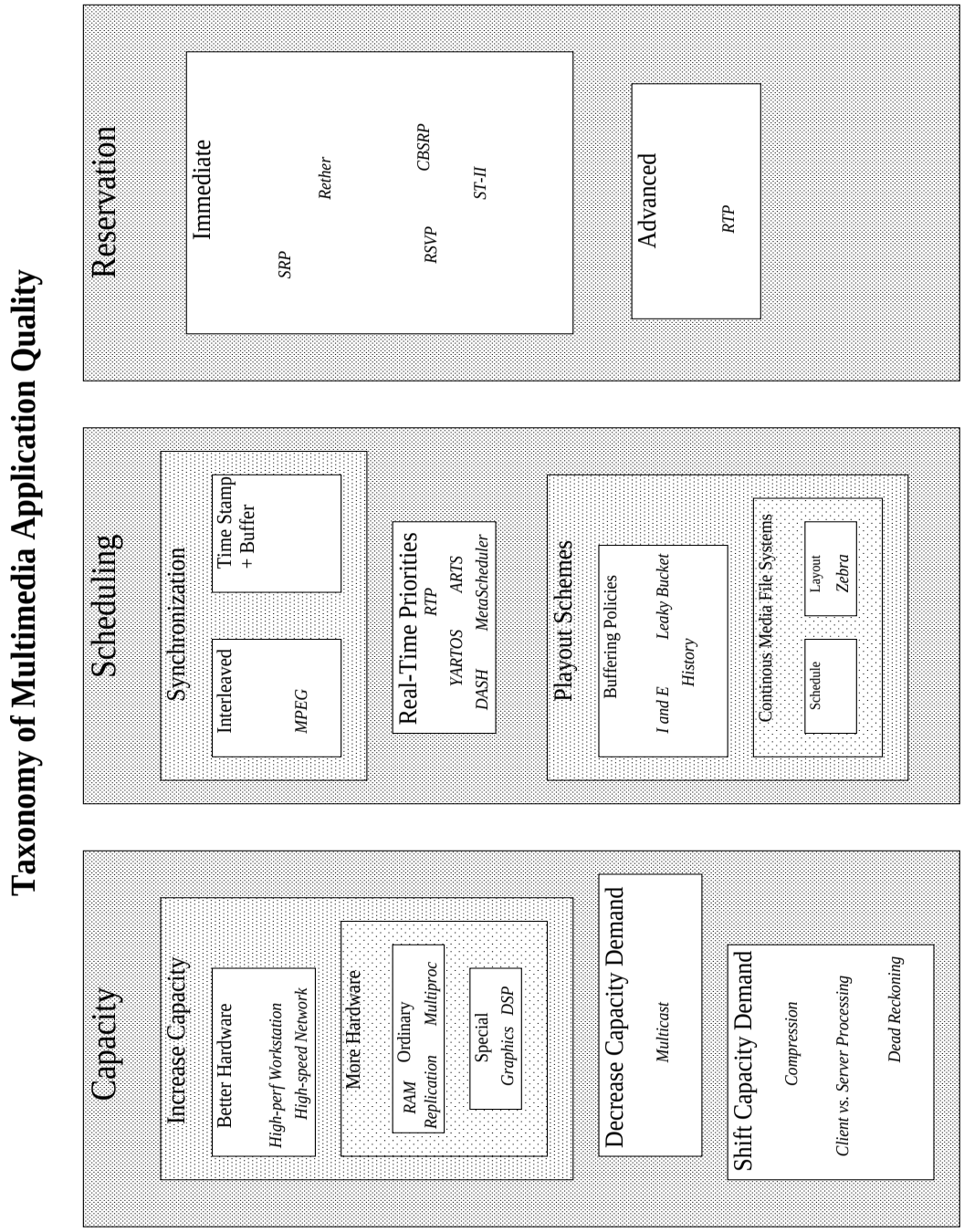

Figure 2.4: Taxonomy of Multimedia Application Quality. There are three categories depicted for improving computer system support for multimedia. Computer system capacity is the maximum available amount of output over a given time period. Scheduling is the processing of controlling computer resources so that application quality is maximized. Reservation is a method of controlling resource access.

careful scheduling nor optimal reservation schemes will help if the computer system does not have the capacity to support the multimedia application.

The computer system component with the smallest capacity will determine the total capacity. For example, it does not matter that your processor can send 50 Mbits/second of video if your network can only carry 10 Mbits/second.

Demand will also impact on capacity. Having a workstation and network that can process 50 Mbits/second of video for an application that only requires a maximum of 10 Mbits/second of video results in an excess capacity.

We consider computer systems made up of:[1]

- *Workstations:* memory, processor, disk, display, bus, audio/video cards, and other specialized hardware. Capacity for the workstation is the workstation multimedia throughput, such as frames/second or Kbits/second of audio.

- *Networks:* network cards, cables and routers (for multi-hop networks). Capacity for the network is Mbits/second.

The capacity of computer system and application can be changed by: increasing the capacity, decreasing the capacity demand from the application or shifting the capacity demand from one computer component to another.

Increasing Capacity Application quality may be improved by increasing the capacity of the supporting computer system, particularly when the current computer resources provide insufficient capacity for minimal services. Capacity may be increased either through better hardware, more hardware or specialized hardware.

- *Better Hardware.* With better hardware, either the workstation can be upgraded or the network upgraded or both. Increasing the capacity of the workstation

[1]We also consider the software that drives computer systems, such as an operating system and network protocol, as a component that affects the computer system capacity.

involves improving any of: disk, processor, memory, bus, graphics card, or monitor (or display device). Increasing the capacity of the network involves improving either the network card, network cable or network protocol.

- *More Hardware.* Workstations can be improved by adding more basic hardware or specialized multimedia hardware. Some ordinary hardware that may be increased to increase workstation capacity includes:
  - *Memory.* Most workstations will have increased capacity with increased memory.
  - *Processors.* Some applications can greatly benefit from a multi-processor workstation.
  - *Replication.* Duplicating servers will often increase the capacity of a computer system, especially when there are multiuser applications.
- *Specialized Hardware.* Hardware specific to multimedia tasks will often increase system capacity. Some examples of specialized hardware includes graphics boards for video frame rendering and DSP chips for specialized signal processing algorithms such as silence deletion [98].

Decreasing Capacity Demand When a computer system does not have sufficient capacity to support the application, the application capacity requirements may be reduced. For example, with unicast routing, you do one send for each of N other receivers. With multicast routing, you do only one send for N other receivers. Multicast routing has been shown to dramatically reduce network capacity requirements [24], and to moderately reduce processor capacity requirements [22].

Alternatively, capacity demand may be decreased through voluntary data loss. As mentioned in Section 1, data loss takes many forms such as decreased frame rate, reduced bits of color, jumbo pixels, smaller images, dropped frames and lossy compression [6, 88, 84, 114].

Shifting Capacity Demand When user requirements cannot be completely satisfied, there is often a tradeoff in system choices. If part of a computer system has sufficient capacity while another part has surplus capacity, it may be possible to shift some of the application capacity requirements from the component with scarce resources to the component with surplus resources.

For example, compression can reduce the size of the data packets that are sent from server to client, decreasing the network capacity requirements. However, the compression and decompression increase the workstation capacity requirements.

Capacity requirements can often be shifted between the client and the server. Many applications have data that needs processing before it is displayed to the user [19]. This processing can be done at either the client or the server. For example, in a Geographic Information Systems application the frame computation can take place by two different methods: in remote image processing the server does the image computation and transfers only a 2D frame to the client; in local image processing the server transfers the 3D data and the client does the image computation. In remote image processing the server capacity requirements are high but the network capacity requirements are low. In local image processing, the processing capacity demand has shifted from the server to the client and the network.

Another example that shifts capacity demand is *dead reckoning.* Dead reckoning is used in Distributed Interactive Simulation (DIS) [55]. DIS is a virtual environment being designed to allow networked simulators to interact through simulation using compliant architecture, modeling, protocols, standards and databases. DIS simulators use "dead reckoning" algorithms to compute position. Dead reckoning shifts the capacity demand from the network to the simulators. See Section 7 for more information.

2.4.2 Scheduling

"Let's do smart things with stupid technology today, rather than wait and

do stupid things with smart technology tomorrow." William Buxton, University of Toronto.

Scheduling is the processing of controlling computer resources so that application quality is maximized. Traditional scheduling techniques for workstations and networks are Round-Robin or First-Come-First-Served or variants of them. While fair, these algorithms generally are not careful in ensuring that the users' quality requirements are met. Scheduling that favors multimedia data may be done in several ways:

Real-Time Priorities. Multimedia applications incorporating audio and video often have stringent delay requirements. Real-time priorities seek to reduce delay by giving multimedia processes and network data precedence over traditional processes or data.

- *Solaris.* Some modern versions of Unix have support for real-time scheduling. In particular, SunOS 5.3 (Solaris) allows for fixed-priority processes and has a fully preemptive kernel. These features produce dispatch latencies on the order of a few milliseconds [74].

- *YARTOS.* Jeffay and Stone have built an experimental real-time operating system that provides guaranteed response times to tasks [63]. It is used as a vehicle for research in the design, analysis and implementation of real-time applications.

- *RETHER.* Venkatramani and Chiueh propose and evaluate a software-based protocol called RETHER (Real-time ETHERnet) that provides real-time performance guarantees to multimedia applications without modifying existing Ethernet hardware [cite RETHER]. RETHER features a hybrid mode of operation to reduce the performance impact of non-real-time network packets.

- *Real-Time Network.* Verma, Zhang and Ferrari describe a scheme for setting up a connection on a packet switched network with guaranteed performance

parameters [121]. Such parameters include a bound on the minimum bandwidth, maximum packet delay and maximum packet loss rate. Their scheme is theoretically capable of providing a significant reduction in jitter.

- *RTP.* Schulzrinne, Casner, Frederick and Jacobson propose RTP, the real-time transport protocol, as an Internet standard [103]. RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTP and RTCP are designed to be independent of the underlying transport and network layers. The protocol supports the use of RTP-level translators and mixers.

Playout Policies. Ideally, we would like to play-out multimedia frames continuously with each successive frame right after the previous. However, this smooth playout is often not possible unless strict policies are enforced that control the play-out. There are several categories of such policies:

- *Buffering Policies* To support continuous play-out, we must compensate for jitter by having a delay buffer at the receiver end. Buffering can compensate for jitter at the expense of latency. Transmitted frames are buffered in memory by the receiver for a period of time. Then, the receiver plays out each frame with a constant latency, achieving a steady stream. If the buffer is made sufficiently large so that it can hold all arriving data for a period of time as long as the tardiest frame, then the user receives a complete, steady stream. However, it is not clear that the primary goal should be play-out with no gaps. In most multimedia applications, gaps from delay variance or even dropping frames occasionally is tolerable. In these cases, play-out with low-latency and some gaps is preferable to play-out with high-latency and no gaps. The added

latency from buffering can be disturbing [90], so minimizing the amount of delay compensation is desirable.

Another buffering technique to compensate for jitter is to discard any late frame at the expense of data loss. Discarding frames causes a temporal gap in the play-out of the stream. Discarding frames can keep play-out latency low and constant, but as little as 6% gaps in the playout stream can also be disturbing [75]. In the case of audio speech, the listener would experience annoying gaps, somewhat similar sounding to static, during this period. In the case of video, the viewer would see the frozen image of the most recently delivered frame.

Naylor and Kleinrock describe two policies that make use of these buffering techniques: the E-Policy (for Expanded time) and the I-Policy (for late data Ignored) [75]. Under the E-policy, frames are never dropped. Under the I-policy, frames later than a given amount are dropped. Since it has been observed that using a strict E-Policy tends to cause the playout latency to grow excessively and that dropping frames occasionally is tolerable [18, 113], we use the I-Policy as a means of examining needed jitter compensation for a multimedia stream.

Queue monitoring uses a policy for decreasing display latency based on observing queue lengths over time at the receiver [58, 113]. If the queue length becomes sufficiently long compared with previous queue length, we can conclude that display latency can be reduced without causing gaps by discarding a frame.

Frankowski and Riedl define a heuristic solution for choosing the buffer time called *History* [41]. The buffer times are based on an ordered list of past interarrival times. They find History to have some benefits over buffering methods that use the standard deviation or absolute deviation of past interarrival times for choosing delay buffer size.

As opposed to policies of receiver-side buffering, Ferrari presents a scheme for buffering data at the network nodes between the sender and receiver [39]. His buffering scheme is capable of providing a significant reduction in jitter, with no accumulation of jitter along the path of a channel. This form of buffering significantly reduces the buffer space required in the network.

Jeffay presents a method of buffering at the side of the sender, instead of the receiver [115]. He argues that a sender based buffering scheme may be more effective than are receiver based schemes.

- *Continuous Media File Systems.* Continuous media file systems seek to improve multimedia application quality by decreasing either the time to access continuous multimedia data on disk and decreasing the variance in this access time. Continuous media file systems are most used to improve the quality of asynchronous multimedia applications as synchronous multimedia application generally do not rely on multimedia data on a disk. There are two ways multimedia disk access may be improved:

  - *Scheduling.* Under scheduling, access to the disk occur at the time when the multimedia data is needed.

  - *Layout.* One approach to providing continuous media is to organize disk layout to allow efficient access to continuous data. For example, Zebra is a network file system that stripes file data across multiple servers for increased file throughput [54]. Rather than striping each file separately, Zebra forms all the new data from each client into a single stream, which it then stripes. This provides high performance for reads and writes of large files and also for writes of small files. Zebra also writes parity information in each stripe in the style of RAID disk arrays; this increases storage costs slightly but allows the system to continue operation even while a single storage server is unavailable. A prototype implementation of Zebra, built

in the Sprite operating system, provides 5-8 times the throughput of the standard Sprite file system or NFS.

Synchronization. Many multimedia applications have synchronized renditions of audio and video. The granularity of synchronization may differ from application to application. For example, one application may require tight lip synchronization between a speaker and his voice, while another application may only require spoken words displayed with a changing background. There are two kinds of synchronization:

- *Interleaved.* With interleaved synchronization, the audio data is interleaved with the video data with which it is to be played. The client, upon receiving the data, separates the audio and video data and plays it, without performing any manual synchronizing. An example of interleaved synchronization is the standard for video from the Motion Picture Experts Group (MPEG) [92].

- *Timed.* Synchronization of multimedia streams can be achieved by the use of timestamps. The sender timestamps each sample and the receiver uses the timestamps to restore the inter-sample spacing. An example is `vat`, a voice-conferencing system developed by Van Jacobson in experimental use on the Internet.

2.4.3 Reservation

"Reservations recommended." Ristorante Luci, St. Paul, MN.

The best scheduling method is useless if the requirements imposed on a resource exceeds the system capacity. Reservation is a method of controlling resource access. Setting aside resource capacity implies some form of reservation. We have identified two categories of reservation schemes:

Immediate. Most reservation services for real-time communication assume that real-time channels are requested for an indefinite duration. Clients are not asked how long such changes will be alive and are assumed to be for an indefinite duration. This approach is similar to the model of a telephone call, where you do not have to specify how long you are going to talk.

Anderson, Herrtwich and Schaefer describe the Session Reservation Protocol (SRP) which allows communicating peer entities to reserve the resources, such as CPU and network bandwidth, necessary to achieve given delay and throughput [3]. The immediate goal of SRP is to support 'continuous media' (digital audio and video) in IP-based distributed systems. SRP is based on a workload and scheduling model called the DASH resource model. This model defines a parameterization of client workload, an abstract interface for hardware resources, and an end-to-end algorithm for negotiated resource reservation based on cost minimization.

Yau and Lam present a framework for scheduling multimedia applications that allows processes to reserve CPU time to achieve guarantees [127]. It provides firewall protection between processes such that the progress guarantee to a process is independent of how other processes actually make scheduling requests.

The Experimental Internet Stream Protocol (ST-II) is a network layer protocol providing multicast services [120]. It provides facilities to negotiate and server resources for packet size and data rate.

Advanced. Some important multimedia applications require that advance reservations be possible. For example, users wanting a multi-person videoconference might desire to reserve resources in advance so that a meeting can be held at the appointed time. Advanced reservation involves setting aside resources in advance such that at a specified future time the resources will be available. Advanced reservations have start time and duration. The crucial difference between immediate reservations and advanced reservation is in the *duration*.

The coarseness of the "time granule" for reservations slots has a large impact on the amount of processing time needed to determine the acceptance of an advanced reservation. Coarse time granules will require less processing time. Limiting the duration of advanced reservation requirements will allow more sharing of resources.

Ferrari, Gupta and Ventre describe the Tenet Real-Time Protocol Suite 2, a suite being developed for multi-party communication, which will offer advance reservation capabilities to its clients [38].

2.5 Summary

This section presents a taxonomy of computer support for multimedia quality. Our taxonomy may be useful for categorizing related research in multimedia applications and it can provide a framework on which research in application quality can build.

At the heart of our taxonomy is our model for determining distributed, collaborative multimedia application quality. Our quality model can be used to compare the effects of different system configurations on application quality. Using our model, we can find the bottlenecks in application quality and determine which categories under our taxonomy improve quality the most. In addition, we can use our model to evaluate the potential performance benefits from expensive high-performance processors and high-speed networks before installing them. We can even investigate possible performance benefits from networks and processors that have not yet been built. In order to improve multimedia quality, system configurations can be altered, affecting one of the three categories in our taxonomy: capacity, scheduling and reservation.

After determining application quality, we may seek to improve it by making a change in the underlying computer system. The change is reflected by tuning the system configuration part of our model. The potential benefit from the system change can improve latency, jitter and/or data loss. We have begun to identify how system changes from each of the taxonomy categories affects latency, jitter or data loss.

Since capacity improvements to computer systems will continue independent of other research in multimedia, it is important to understand what impacts larger capacities will have on multimedia application performance. In the rest of this thesis, we concentrate on the the effects that changes in capacity have on multimedia application performance. Capacity improvements usually improve multimedia quality by reducing latency, jitter and data loss. As shown in Section 5.4, capacity improvements reduce latency because faster hardware can respond faster. As shown in Section 3.4, capacity improvements reduce jitter because faster hardware also reduces variance in response time. And as described in Section 5.6.4, larger capacities reduce the amount of data loss from system saturation.

In order to support current and future multimedia application requirements, it will always be crucial to use available resources in an effective matter. In Section 3, we take an in-depth look at how real-time scheduling affects jitter, a fundamental component of multimedia application quality. We demonstrate that some methods of scheduling improve jitter more than do capacity improvements.

Reservation has the potential to improve multimedia application quality by reducing latency, jitter and data loss. However, this thesis focuses on the effects of capacity and scheduling on multimedia application quality.

Synchronization is an interesting case. Synchronization does not improve latency; synchronized data arrives no sooner and may even arrive later than non-synchronized data. Synchronization does not improve jitter; synchronization does not improve the variance in the multimedia data. And synchronization does not improve data loss. Synchronization points out the need for some applications to have an additional axis to determine application quality in addition to the three fundamental axes. We may choose to call the additional axis the "accuracy" axis. For example, the quality of Distributed Interactive Simulation applications includes an "accuracy" axis (see Section 7). Previous quality research by Jeffay et al. has evaluated videoconference quality based on audio and video synchronization [66].

A computer system's change in one category might require a change in another. For example, to implement a buffering scheme (a scheduling change), you might need more memory (a capacity increase). Or, to carry out a network reservation scheme, you might lose some capacity from inefficiency, requiring an increase in network bandwidth (another capacity increase).

In Sections 5, 6 and 7, we apply our quality model to three applications: audio-conferences, a 3-d interface to a scientific database and a distributed, multi-person combat simulator. We vary system configurations in order to predict the effect of capacity and scheduling changes on application quality.

Chapter 3

Jitter

3.1 Overview

In a distributed multimedia application, a multimedia stream is generated at the sending workstation and sent over a network to the receiving workstation. The data is generated and sent in fixed-sized quantities called frames. The end-to-end frame delay is the time between a frame's generation on the sender and the time it is processed by the receiver. Variation in this end-to-end delay we call *jitter*. We have observed jitter on the order of a few hundred milliseconds when sending a multimedia stream using unloaded workstations on a quiet Ethernet local area network.

How does jitter affect a multimedia stream? In the absence of jitter, the frames can be played as they are received, resulting in a smooth playout, depicted by Figure 3.1. However, in the presence of jitter, interarrival times will vary, as depicted in Figure 3.2. In Figure 3.2 the third frame arrives late at r*2*. In the case of audio speech, the listener would experience an annoying pause during this period. In the case of video, the viewer would see the frozen image of the most recently delivered frame.

If we decrease jitter, we will have a less choppy playout and better application quality. In seeking to decrease jitter, we can tune the application, the operating system or the underlying hardware. This gives us a hierarchy of possible jitter reducing

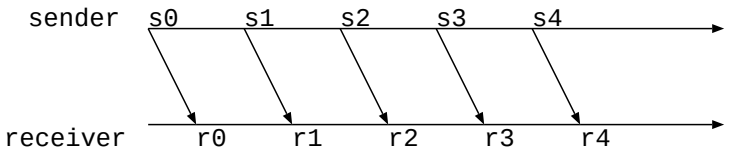

Figure 3.1: A Jitter-Free Stream. The above figure is a model of a jitter-free stream. Each si is the time at which the send process initiates the transmission of frame i. Each ri is the time at which the receiving process plays frame i.

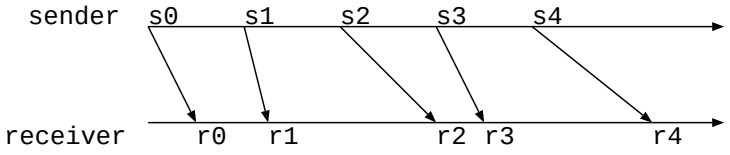

Figure 3.2: A Stream with Jitter. The above figure is a model of a stream with jitter. Each si is the time at which the send process initiates the transmission of frame i. Each ri is the time at which the receiving process plays frame i.

techniques. Table 3.1 depicts this hierarchy. Application techniques for reducing jitter include application tuning and buffering. System techniques for reducing jitter include disk layout/scheduling, alternate network protocols and operating system priorities. Hardware techniques for reducing jitter include hardware improvements, such as high-performance processors, high-speed networks and fast disks. We consider the merits of conducting research on each jitter reducing technique:

Application Tuning and Buffering. Application-level techniques for reducing jitter are done in the context of compensating for the jitter produced by the underlying computer system. There has been extensive research done on application techniques [35, 104, 81]. Controlled frame playout through buffering, in particular, has been well studied [75, 96, 115, 113, 39]. With buffering, transmitted frames are held in memory by the receiver for a period of time. Then, the receiver plays out each frame with a

Level	Possible Jitter Reducing Technique
Application	Application Tuning
	Buffering
System	Disk Layout/Scheduling
	Alternate Network Protocols
	Operating System Priorities
Hardware	High-Performance Processors
	High-Speed Networks
	Fast Disks

Table 3.1: Hierarchy of Possible Jitter Reducing Techniques

constant latency, achieving a steady stream. If the buffer is made sufficiently large so that it can hold all arriving data for a period of time as long as the tardiest frame, then the user receives a complete, steady stream. However, the added latency of data can be disturbing [90]. So there is a tradeoff between ameliorating the effects of jitter and minimizing the amount of latency due to buffering.

Computer systems continue to get faster while human perceptions remain the same. Future system improvements may remove enough of the underlying jitter such that application-level jitter reduction techniques are unnecessary. How far in the future will this be? In this work, we seek to answer this question by experimentally measuring the effects of system improvements on jitter. Then, based on the rate of hardware improvements, we predict when the jitter levels contributed by the underlying system will drop below human perception.

Disk Layout/Scheduling and Fast Disks. Fast disks, and disk layout/scheduling strategies may be crucial for reducing jitter for some multimedia applications. In this work, none of the three multimedia applications we study, audioconferences, the scien-

tific brain database and the virtual cockpit, access multimedia data on disks directly, so we do not consider the effects of disk layout or scheduling on jitter further.

Alternate Network Protocols. Alternate network protocols may reduce jitter over traditional network protocols. There has been a lot of research into network protocols designed for handling multimedia data [79, 120, 100]. Our work experimentally measures the effects on jitter of two high-speed network protocols: Fibre Channel and HIPPI.

Operating System Priorities. There is evidence to suggest that a significant source of jitter in the transmission of multimedia may be found in the operating system of the sending and receiving workstations [46]. The principal deficiency in process scheduling in modern operating systems is that user-level processes are not preempted while in kernel mode. As a result, increased system activity can increase response time without bound. In addition, modern operating systems allow priority inversions to occur whereby a low priority process excludes high-priority processes from accessing time-critical data, thus causing the high-priority processes to miss deadlines. These deficiencies can result in latency on the order of 100 milliseconds [74]. Real-time scheduling allows the operating system to accurately time events and allocation of memory to the process and provide for priority scheduling. This feature can reduce latency to the order of a few milliseconds [74]. Our work experimentally compares the benefits of real-time scheduling to normal scheduling to determine if real-time scheduling does reduce jitter.

High-Performance Processors and High-Speed Networks. High-performance processors have higher throughput and a faster context switch time than typical processors resulting in better application response time. High-speed networks deliver frames from the sender to the receiver faster than typical networks, reducing the network transmission time. Together, high-performance processors and high-speed

networks networks will reduce application latency. The reduced latency should be accompanied by a reduction in latency variation, or jitter. We ran experiments on SGI Challenge workstation clusters and Fibre Channel and HIPPI networks under both light and heavy load to determine the effects that hardware improvements have on jitter.

3.1.1 Hypotheses

Given the above discussion, we propose the following hypotheses:

1. *High-performance processors reduce jitter.*

2. *Real-time operating system priorities reduce jitter.*

3. *High-speed networks reduce jitter.*

Processor performance approximately doubles every year [56]. As processor performance increases, latency decreases. This decrease in latency may be accompanied by a decrease in jitter. However, in order to significantly improve application quality, any jitter reduction from high-performance processors needs to be large compared to the total jitter contributed by the network and operating system. In other words, the processor may not be the bottleneck in reducing jitter.

Since real-time operating system priorities have been shown to decrease latency [74], it seems natural to assume that they may decrease jitter, also. If real-time priorities do prove to significantly reduce jitter, application quality may be improved without expensive hardware upgrades or time-consuming application tuning.

Under heavy loads, high-speed networks should deliver multimedia frames faster than traditional networks. By delivering frames faster, high-speed networks will reduce contention for the network, decreasing the latency of frames waiting to be delivered by the network. We would also expect this reduced contention to decrease

jitter under such conditions. However under light loads, the reduced latency from high-speed networks may not significantly improve application quality.

In order to test our hypotheses, we looked at three possible performance evaluation techniques: analytic modeling, simulation and experimental measurement. Frankowski and Riedl found that analytically modeling jitter is very difficult, even on a quiet, single-hop network [41]. The contributions to jitter from the operating systems on both the sender and receiver and the contribution to jitter from the network are difficult to capture mathematically. Stein and Riedl had some success in using simulation to evaluate the effects of jitter on audioconference quality [112]. However, according to Jain, simulations are most effective when they are based on previous measurement [62]. Unfortunately, to the best of our knowledge, careful measurements of the contributions to jitter from high-performance processors, real-time priorities and high-speed networks have not been made. Simulating the effects of such components on jitter may give rise to inaccurate or misleading results. We use experimental measurement to test our hypotheses about the effects of high-performance processors, priorities and high-speed networks on jitter reduction. Future research may be able to use our performance measurements as a basis for simulations.

Although our hypotheses are simply stated, the answers to these hypotheses are not quite as simple as "true" or "false." The rest of this chapter explains why. Section 3.2 lists related work in teleconferencing systems, delay buffering and real-time performance. Section 3.3 details the experimental design components that were common to all experiments. Section 3.4 explores the effects of processor performance on jitter. Section 3.5 examines the effects of real-time priorities on jitter. Section 3.6 looks at the effects of high-speed networks on jitter. And Section 3.7 summarizes the results from this section.

3.2 Related Work

3.2.1 Teleconferencing Systems

Several experimental teleconferencing systems have been designed to explore teleconferencing performance issues such as jitter.

Riedl, Mashayekhi, Schnepf, Claypool and Frankowski developed SuiteSound [98]. SuiteSound attempted to integrate support for multimedia into the Suite programming environment. They performed experiments to determine the network and CPU load of the SuiteSound tools, including the effects of an algorithm that removes silence from digitized speech.

Hopper developed the Pandora system to investigate the potential for creating and deploying a desktop multimedia environment based on advanced digital video and audio technology [58]. Pandora offered a "tool box" to those analyzing and working on information by providing a flexible communications system that allowed effective interaction between a number of users.

Jeffay, Stone and Smith developed a transport protocol that supports real-time communication of audio/video frames across campus-area packet switched networks [65]. They demonstrated the effectiveness of their protocol by measuring the performance of their protocol when transmitting audio and video across congested networks.

Teleconferencing systems attempt to provide quality audio and video to groups of users. One component to teleconferencing quality is jitter. We provide experimental results on the effects that system improvements will have on reducing jitter. We also present a quality model that can be used to evaluate system configuration tradeoffs in predicting teleconferencing quality from the users' perspective.

3.2.2 Delay Buffering

Research in delay buffering has looked at ways to ameliorate the effects of jitter by controlling the playout and delivery of frames at either the sender or the receiver.

Ramjee, Kurose, Towsley and Schulzrinne compared the effects of four different buffering algorithms for adaptively adjusting the playout delay of audio packets over a wide area network [96]. They found that an adaptive algorithm which explicitly adjusts to the sharp, spike-like increases in packet delay achieved the lowest rate of lost packets.

Stone and Jeffay presented an empirical study of several buffering policies for managing the effect of jitter on the playout of audio and video in computer-based conferences [113]. They evaluated a particular policy called queue monitoring by comparing it with two policies from other literature. They showed that queue monitoring performs as well or better than the other policies over the range of observed network loads.

As opposed to the two papers above that use receiver-side buffering, Ferrari presented a scheme for buffering data at the network nodes between the sender and receiver [39]. He studied the feasibility of bounding jitter in packet-switched wide area networks with a general topology. He presented a buffering scheme that is capable of providing a significant reduction in jitter, with no accumulation of jitter along the path of a channel, and demonstrated that jitter control significantly reduces the buffer space required in the network.

Talley and Jeffay presented a method of buffering at the side of the sender, instead of the receiver [115]. They presented a framework for transmission control that describes the current network environment as a set of sustainable bit and packet transmission-rate combinations. They empirically demonstrated the validity of adapting both packet and bit-rate using simple adaptation heuristics.

Naylor and Kleinrock developed a model for measuring the quality of an audioconference based on the amount of dropped frames and client-side buffering [75]. They used their model to investigate two adaptive receiver-side buffering schemes which may be used to achieve a smooth playout. Method E expands the buffer to preserve all incoming frames. Method I ignores all late frames in order to preserve timing.

We explore alternative means to delay buffering to reduce jitter. Specifically, we examine when hardware improvements might make delay buffering techniques unnecessary. We extend the work of Naylor and Kleinrock to develop a more general model for multimedia application quality.

3.2.3 Real-time Performance

Govindan and Anderson conjectured that the traditional operating system goals of fairness, maximum system throughput and fast interactive response may conflict with the needs of real-time applications such as continuous media and proposed a new processor scheduling algorithm [46]. Jeffay, Stone and Smith made similar remarks and provided experimental evidence [65]. Jeffay and Stone and went so far as to build an operating system that supports real-time multimedia [63]. Khanna, Serbree and Zonowsky discussed the design, implementation and performance of the SunOS 5.0 operating system as a real-time system [74]. Gerhan presents the new real-time enhancements to Microsoft Windows NT [48].

We experimentally measure the effects of Solaris real-time priorities and the Unix nice facility on jitter. We compare the effects of using operating system priorities with processes run under default priorities.

3.3 Shared Experimental Design

> *"The fundamental principle of science, the definition almost, is this: the sole test of the validity of any idea is experiment."* Richard P. Feynman

We ran a series of experiments to test our hypotheses. This section details the design components that were common to all experiments.

Our experiments were all conducted on a single-hop LAN. In our first set of experiments, we attempted to measure jitter contributions on a WAN. However, getting tight confidence intervals on WAN jitter proved extremely difficult. Perhaps in the

future, our LAN measurements may be used in simulations to explore the effects of a WAN on jitter.

Each experiment simulated the transmission of a multimedia stream under various conditions and measured the amount of jitter. We used pairs of user-level processes that sent and received UDP datagrams using Berkeley socket I/O. The `send` process used an interval timer to initiate frame delivery. The `receive` process took timestamps using the `gettimeofday()` system call. These timestamps are used to measure the amount of jitter.

A jitter measure must reflect the extent to which the interarrival times vary. There are several classical statistical measures of variation that have been used in jitter research:

- *Range.* The simplest measure of variability is the maximum delay between any two consecutive frames. Generally speaking, more variability is reflected in a larger range. The range also provides a maximum delay variance for providing jitter guarantees to multimedia applications. For this reason, range has been used in some past jitter research [121].

- *Variance.* The primary measure of variability determines the extent to which each frame deviates from the mean frame interarrival time. The standard way to prevent values below the mean interarrival time from negating values above the mean interarrival time is to square them. Dividing by the number of frames gives the average squared deviation. This is the standard definition of variance and is used by several jitter researchers [39, 96]. The formula for variance is [30]:

$$Variance = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{(n-1)}$$

- *Standard Deviation.* By taking the square root of the variance, we get the same units as the frame delay. This is the standard deviation, the classical

measure of spread. Several researchers have used the standard deviation of packet interarrival times as a measure of jitter [112, 68, 7]. The formula for standard deviation is [30]:

$$StandardDeviation = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{(n-1)}}$$

- *Absolute Deviation.* Variance and standard deviation have proven to be very sensitive to outliers. Since human eyes and ears can smooth over an occasional glitch in an audio or video stream, we do not want our jitter measure to be especially sensitive to outliers. For this reason, some researchers have used absolute deviation as a measure of jitter because it is less sensitive to values far from the mean [104]. The formula for absolute deviation is :

$$Absdev = \sum_{i=1}^{n} |x_i - \overline{x}|$$

In addition, there has been jitter research that has incorporated derived measures of jitter:

- *Gaps.* When playing out a multimedia stream, a late frame will cause a gap in the smooth playout. The total number of gaps and the number of gaps per second can provide a measure of how much jitter there is in the stream. Gaps as a measure of jitter has been by several jitter researchers [113, 41].

- *Jitter Compensation.* In addition to determining buffer size, the jitter compensation curve can be used to measure jitter. The jitter compensation curve is a measure of the amount of delay buffering needed to achieve a smooth multimedia stream (see Section 5.6.4 for a complete description). The more jitter in a multimedia stream, the larger the area under the jitter compensation curve. This jitter measure has been used by [51].

	Range	Variance	Stddev	Absdev	Gaps	Area
Range	1.00	0.34	0.34	0.33	0.21	0.34
Variance	0.34	1.00	0.96	0.89	0.85	0.95
Stddev	0.34	0.96	1.00	0.95	0.73	0.99
Absdev	0.33	0.89	0.95	1.00	0.66	0.96
Gaps	0.21	0.85	0.73	0.66	1.00	0.72
Area	0.33	0.95	0.99	0.96	0.72	1.00

Table 3.2: Correlation Among Jitter Measures. This table depicts the correlation coefficients for 9 different jitter measures. Range is the maximum interarrival time. Variance is the variance in interarrival times. Stddev is the standard deviation of interarrival times. Absdev is the absolute deviation of interarrival times. Gaps are the number of playout gaps per second with 250,000 microseconds of buffering. Area is the area under the jitter compensation curve. The table is symmetric about the diagonal.

There are many more possible measures of jitter: interquartile range, mean length of gaps, median length and even second-order statistics such as variance on the mean length, etc. However, such measures are less likely to depict the amount of jitter meaningful to the multimedia application than the more direct methods we detailed above. Because of this, to the best of our knowledge, other researchers have not used them to measure of jitter. We do not consider these methods further.

However, we did want to compare the equivalence of jitter measures that *had* been used in previous research. We recorded interarrival times for all experiments in Sections 3.4 through 3.6 and computed jitter values for range, variance, standard deviation, absolute deviation, gaps and jitter compensation. We then computed the Pearson's correlation coefficient [30] for each pair of jitter measures. For the *gaps* jitter measure, we assumed all late data is ignored and that there is a buffer of 250,000 microseconds, values used by other researchers [75]. Table 3.2 gives the correlation coefficient for each jitter measure pair.

Range does not correlate well with any of the other jitter measures. Range measures variation as the distance between the two most extreme values. Variation depends on more than just the extreme values as we can see from these two samples: {10,15,15,15,20} and {10,10,15,20,20} both have the same range but there is less dispersion in the first sample.

Gaps does not correlate well with Absolute Deviation and it correlates only slightly well with Area. The number of gaps per second depends upon the initial buffer chosen, so different buffer sizes might have different correlation results. The rest of the jitter measures correlate well (0.72) to to extremely well (0.99) with each other. With the exception of range, the measurement of jitter chosen does not not appear to matter. We choose variance as our measure of jitter because it is easy to understand and compute.

Audio and video have very different bandwidth needs. The Sun audio device records audio at a rate of 8000 bytes/second [1]. Video acquisition hardware typically generates 30 frames-per-second and compressed frames with a resolution of 256x240 require around 2 megabits-per-second of bandwidth [18]. We wanted to see if either packet size or packet rate changed the frequency and/or magnitude of the interarrival times.

Pairs of packets tend to reflect about the mean packet arrival time. If a packet arrives late, the next packet usually arrives early by the same time that the previous packet was late. When interarrival times become small, the packet following a late packet is unable to arrive an equal amount early, being unable to arrive in fewer than zero microseconds. Figure 3.3 depicts packet reflection. There are two multimedia streams shown. The top stream has a mean interarrival time of 160,000 microseconds. The lower stream has a mean interarrival time of 30,000 microseconds. When the packet first packet arrives late in the 160,000 stream, the subsequent packet arrives an equal amount early. However, when a packet arrives late in the 30,000 microsecond stream, the subsequent packet arrives almost immediately, being unable to reflect

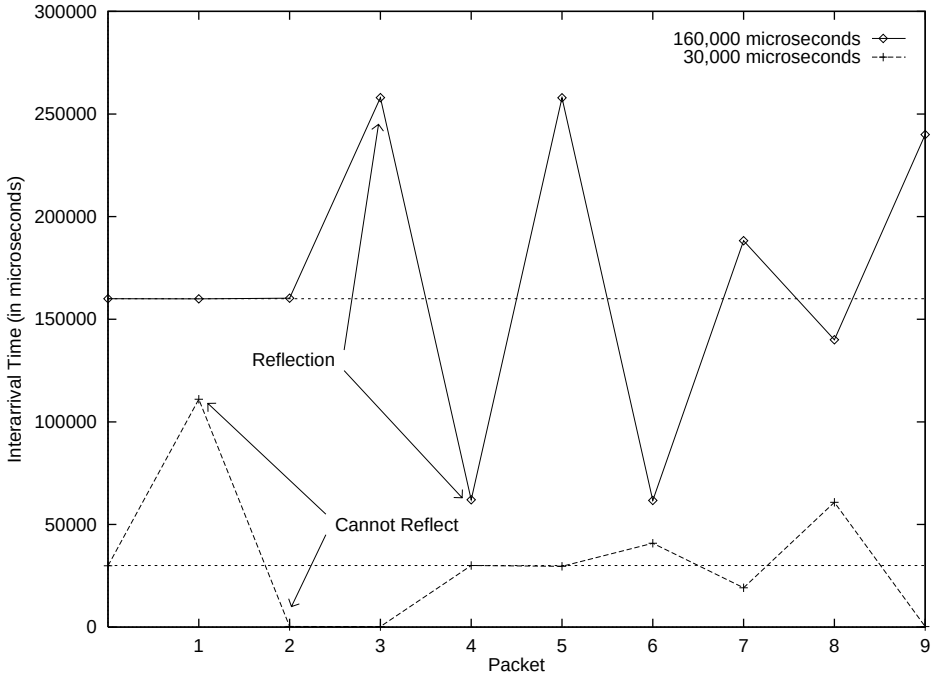

Figure 3.3: The Reflection Effect. The horizontal axis is the packet number. The vertical axis the interarrival time in microseconds. The zig-zag lines represent two multimedia streams with the top having a mean interarrival time of 160,000 microseconds and the bottom having a mean interarrival time of 30,000 microseconds.

an equal amount. Mathematically, the top stream will have a larger variance, even though they both have an equal chance of a having a packet arrive late. This reflection effect is an artifact of the environment and does not accurately indicate the frequency and magnitude of late packets.

We performed three mini-experiments to test if either packet size or packet rate changed the frequency and/or magnitude of the interarrival times. We subtract the mean from each interarrival time and drop all points that are below zero. This avoids the reflection effect and allows us to compare the frequency and magnitude of late packets. In the first experiment, we sent packets of three different sizes at three different rates: audio-rate (160,000 microseconds) and audio-size (1280 bytes); video-rate (33,000 microseconds) and video-size (6000 bytes); and mid-rate (100,000 microsec-

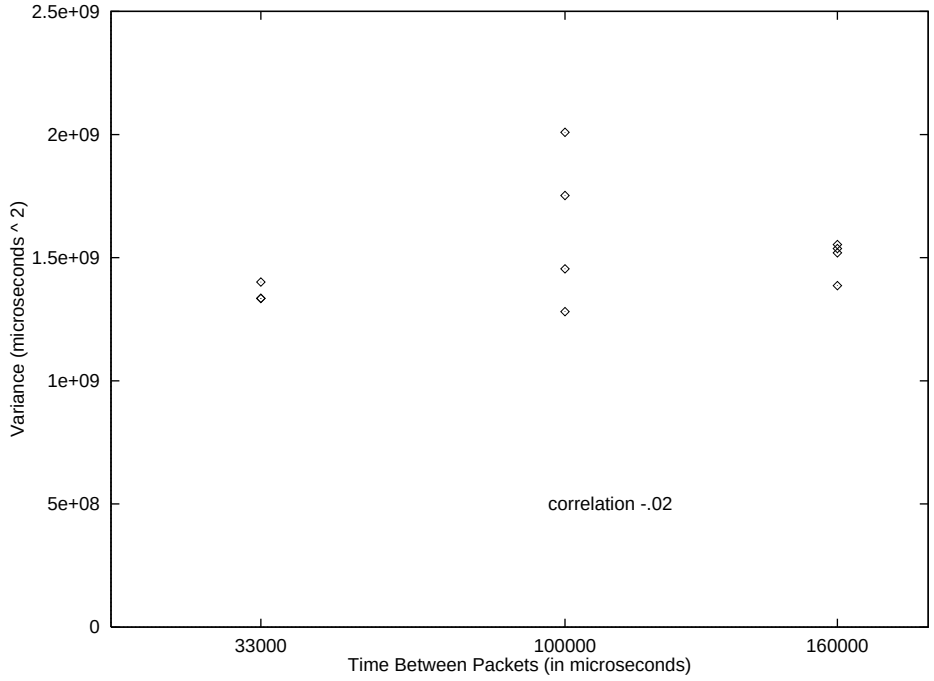

Figure 3.4: Jitter versus Packet Rate. The horizontal axis is the time between packets. The vertical axis is the variance. Each point represents an independent experiment. The correlation is -0.02.

onds) and mid-size (4000 bytes). Figure 3.4 depicts the results. The correlation between packet rate and variance is an extremely low -0.02.

In the second experiment, we sent packets all of the same size (1280 bytes) at four different rates: 30,000 microseconds, 70,000 microseconds, 110,000 microseconds, and 160,000 microseconds. Figure 3.5 depicts the results. The horizontal axis is the time between packets. The vertical axis is the variance. The correlation between packet rate and variance is an extremely low 0.09.

In both experiments, the correlation between packet rate and jitter was extremely low. This indicates that the amount of jitter, in terms of number and magnitude of late packets, we would expect to see in a high-rate video stream would be the same as the jitter from a lower-rate audio stream. For all subsequent experiments, we used the audio rate in order to avoid the reflection effect and to keep network and processor

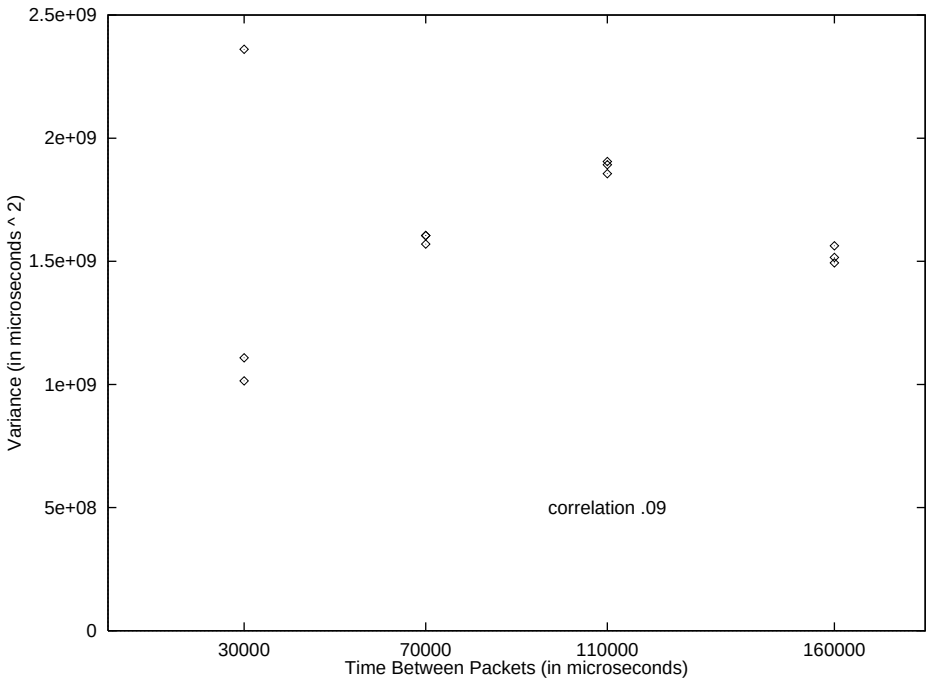

Figure 3.5: Jitter versus Packet Rate for 1280 Byte Packets. The horizontal axis is the time between packets. The vertical axis is the variance. Each point represents an independent experiment. The correlation is 0.09.

Workstation	MHz	SPECint92
SLC	20	8.6
IPC	25	13.8
IPX	40	21.8
Sparc 5	85	64.0

Table 3.3: Workstations used in Processor Experiments.

loads low. We can then measure the effects that increased network and processor load had on jitter. To simulate the transmission of audio, we sent 1280-byte datagrams every 160,000 microseconds.

3.4 Processor Experiments

Processor performance approximately doubles every year [56]. These high-performance processors will probably have a huge effect on improving the quality of current multimedia applications and may enable new multimedia applications. We explore the effects of processor performance on jitter, one component in multimedia application quality.

3.4.1 Specific Experimental Design

We used four classes of Sun processors: SLC, IPC, IPX and Sparc 5. Table 3.3 summarizes the workstation attributes. The workstations were connected to an 10 Mbits/second Ethernet. We used a process that increments a long integer variable in a tight loop (a "counter process") to induce processor load.

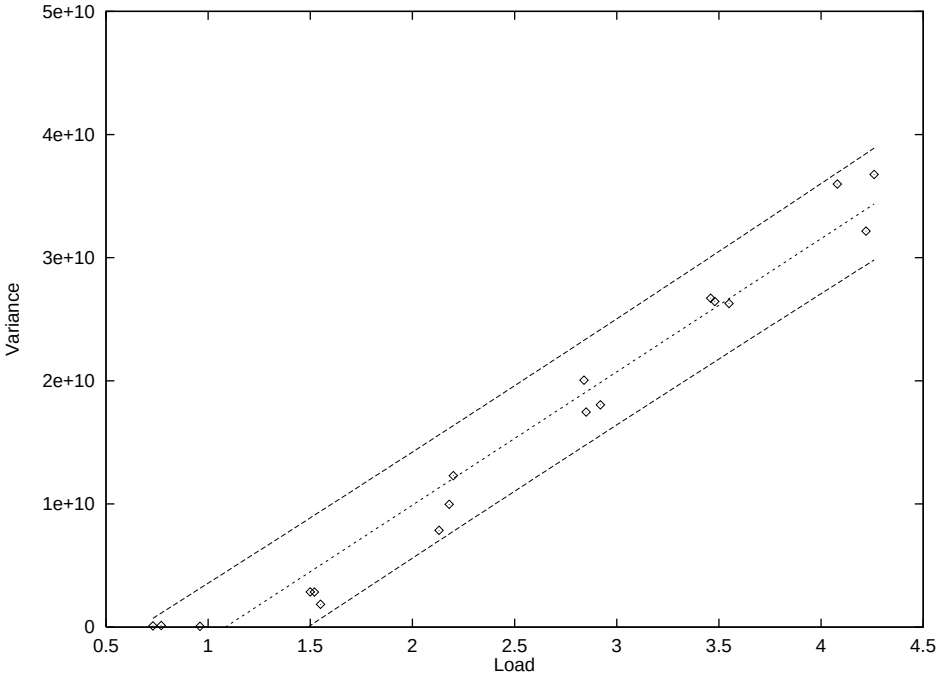

Figure 3.6: Sun SLC Jitter versus Receiver Load. The horizontal axis is the processor load as reported by the Unix `w` command. The vertical axis the the variance in interarrival times. The middle line is the least squares line fit. The outer two lines form a 95% confidence interval around the line. The correlation coefficient is 0.98.

3.4.2 Jitter versus Processor Load

We first test whether increasing processor load increases jitter. We ran an increasing number of counter processes on the receiver and obtained the processor load from the Unix `w` command at the end of each experiment run. The `w` command displays a summary of the current activity on the system, including the average number of jobs in the run queue over the last 1, 5 and 15 minutes. Lastly, we did a least squares line fit for the load versus variance and computed the correlation coefficient. Figure 3.6 shows the results of our experiments on the Sun SLC.

Figure 3.7 shows the results of our experiments on all the Sun workstations listed in Table 3.3. Again, we did a least-squares line fit for the load versus variance, for

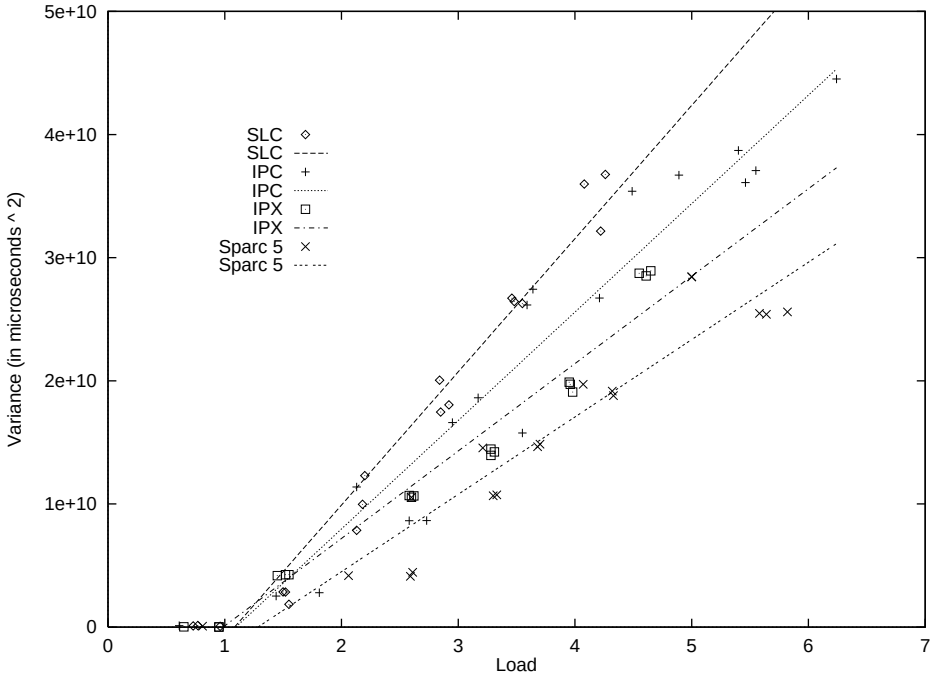

Figure 3.7: Jitter versus Receiver Load. The horizontal axis is the processor load as reported by the Unix `w` command. The vertical axis the the variance in interarrival times. The lines are the least squares line fits. The correlation coefficients range from 0.97 to 0.99.

each class of processor. The slopes for all processors are positive, meaning that an increase in load results in an increase in jitter no matter how powerful the processor. In addition, the slopes of the least squares line fits decrease as the processors get more powerful. This means that under loaded conditions, more powerful processors will contribute less jitter to a multimedia stream than will less powerful processors.

3.4.3 Jitter versus Processor Power

We observed that more powerful processors decrease jitter under high loads, but what happens under conditions of more normal load?

In Figure 3.7, the lines come to nearly the same point at a processor load of 1. At a load of 1, we could show no correlation between jitter reduction and processor

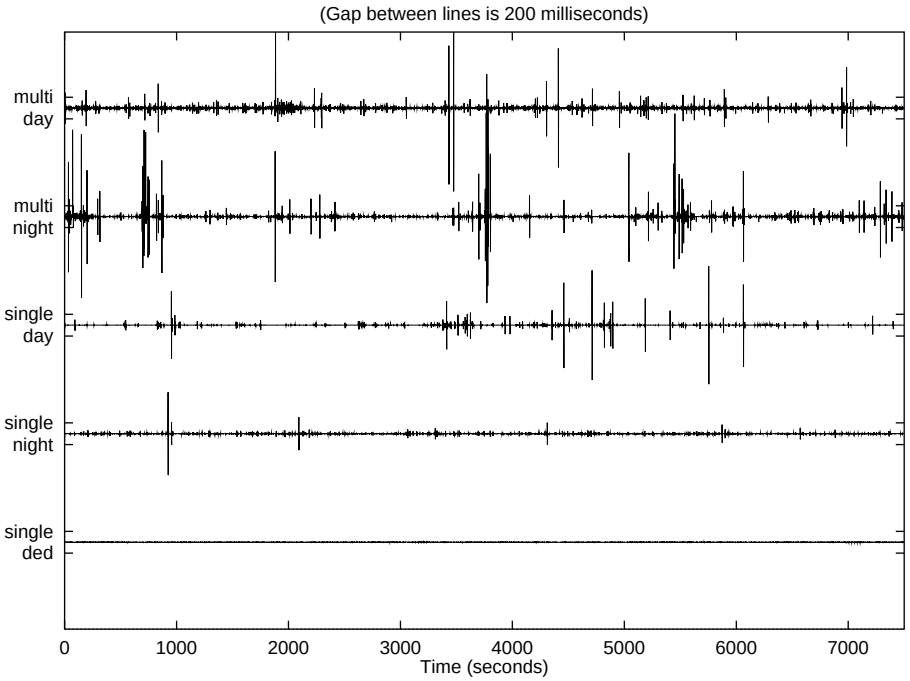

Figure 3.8: Effects on Jitter. There are 5 multimedia streams represented in this picture, each by a horizontal line depicting the interarrival times. "Single" indicates an experiment run in single-user mode. "Multi" indicates an experiment run in typical multi-user mode. "Day" indicates an experiment run in the middle of the day. "Night" indicates and experiment run at night. "Ded" indicates an experiment run on a dedicated network. Each multimedia stream is off-set from the one below it by 200 milliseconds. The horizontal axis is the time in seconds.

power, so we sought to isolate our experimental environment from the outside world to better observe the effects of the processor. Figure 3.8 depicts a series of attempts to do this. The top line in the picture represents the interarrival times for an experiment run in normal multi-user processor mode during the day.

We wanted a quiet network to reduce effects of network on jitter. We assumed that most users do their work in the day, so there should have been less network traffic at night. We recorded interarrival times at night to see if they appeared significantly different than those recorded during the day. These times are depicted by the 2nd line from the top of Figure 3.8. The amount of jitter at night appears no better than

the amount of jitter during the day. We attribute this to the academic environment in which we tested, in which some students and faculty members are likely to work in the early morning hours. In addition, many system backups are scheduled to be done at night in order to interfere less with the majority of the users during the day.

We also wanted to reduce the effects of other workstation processes on jitter. We ran an experiment in single-user mode during the day and recorded the interarrival times, depicted by the 3rd line from the top of Figure 3.8. Still, the amount of jitter in single-user mode appears only slightly better than the amount of jitter in multi-user mode. We could still show no correlation between jitter reduction and processor power.

We then tried a combination of single-user mode and nighttime. These interarrival times are depicted by the 2nd line from the bottom of Figure 3.8. The amount of jitter in this case is noticeably less than the amount of jitter from the multi-user mode experiment run during the day. However, we could still show no correlation between jitter reduction and processor power.

In our last effort to remove as much jitter as possible, we physically disconnected our workstations and subnet from the surrounding networks and ran our experiments in single-user mode. Success! These interarrival times are depicted by the bottom line in Figure 3.8. This nearly flat line represents a multimedia stream that is almost jitter-free.

Once we isolated the machines on a quiet network and ran our experiments in single user mode, we were able to observe the effects different power processors have on jitter under light loads. Figure 3.9 depicts these results. There is a strong correlation between increased processor power and decreased jitter. However, having dedicated workstations and networks is implausible in many real-world cases and probably impossible for wide-area networks such as the Internet.

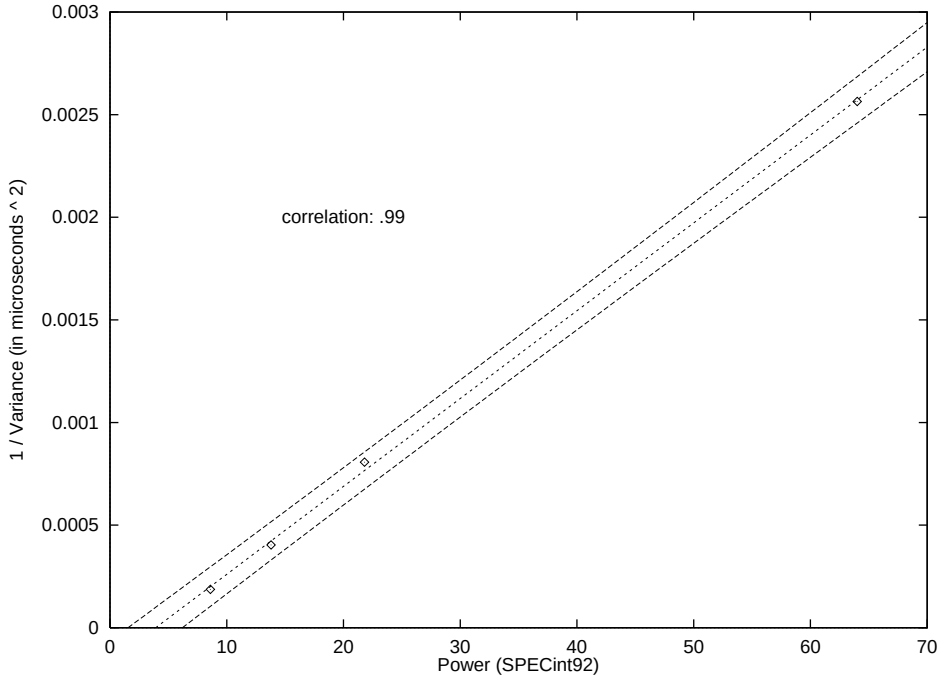

Figure 3.9: Jitter versus Power. The horizontal axis is the SPECint92 result of the workstation. The vertical axis is 1 / variance in packet interarrival times. The middle horizontal line is the least squares line fit. The outer two lines form a 95% confidence interval around the line. The correlation coefficient is 0.98.

3.4.4 Section Summary

Jitter correlates highly with processor load. Since multimedia applications are often processor intensive, they often force processors to run at a heavy load. Moreover, in the past, applications have tended to expand to fill (or surpass) available system capacity, making heavy-load conditions likely in the future. These heavily loaded workstations will contribute to the jitter in multimedia applications.

Fortunately, under heavy loads, faster processors reduce jitter compared to slower processors. However, under normal load, the reduction in jitter from the faster processors is overshadowed by the variance in average processor and network loads. To fully determine the benefits of faster processors to jitter, it is imperative to determine if future multimedia application will, in fact, push processor capacities to the limit. If so, future multimedia application quality will benefit from faster processors. If not, application quality should be improved by means other than processor improvement.

We can use our model from Section 2.3 to determine if current and future processor capacities are being consumed by multimedia applications. In addition, as we show in Sections 5, 6 and 7, our model allows us quantitatively determine the benefit to application quality from high-performance processors for today's and tomorrow's multimedia applications.

3.5 Real-time Priority Experiments

From the previous experiments, single-user mode and a dedicated network can nearly eliminate jitter. However, in a multi-user system a dedicated processor is seldom available, and a dedicated network is even more rare. It seems unlikely that the above results alone will help users in most cases. We turn to real-time priorities to see how they compare to the benefits of a single-user mode processor under realistic load conditions.

3.5.1 Specific Experimental Design

We used Sun SPARC Classics running SunOS 5.4, a Unix variant. In Unix, a processes priority determines how much CPU time the process gets [87]. The kernel will decrease the priority of processes that have accumulated what it considers "excessive" CPU time. The priority of a process is not the only thing that determines when a process will be run. To determine which process should be run next, the scheduling mechanism in the kernel uses a formula that takes into account each process's priority, how much CPU time each process has gotten recently, and how long it has been since each process has run. SunOS 5.4 extends Unix process scheduling with real-time support. A real-time process runs in a separate scheduling class than normal processes. In the default configuration, a runnable real-time process runs before any other process. This gives real-time processes the highest priority.

Even without real-time extensions, Unix provides the `nice` facility. Nice allows you to change the default priority of a process. Using nice, it is possible to improve the user-mode priority of a process allowing it to run ahead of other eligible runnable user-mode processes. Processes using the nice facility may still suffer long dispatch latencies, however. Any process that is suspended waiting for I/O will, upon being re-activated, have a higher priority than any process that is in user-mode, regardless of its nice value [78].

We varied the process priority from Default (processes were run without enhanced priority), Niced (processes were invoked using the `setpriority(-19)` system call as an offset to determine the final user-mode priority of the process) and Real-Time (processes were given fixed priorities in the SunOS real-time scheduling class using `priocntl()` system call).

As in Section 3.4, we used a process that increments a long integer variable in a tight loop to induce processor load. We ran separate experiments with one through six of these counter processes.

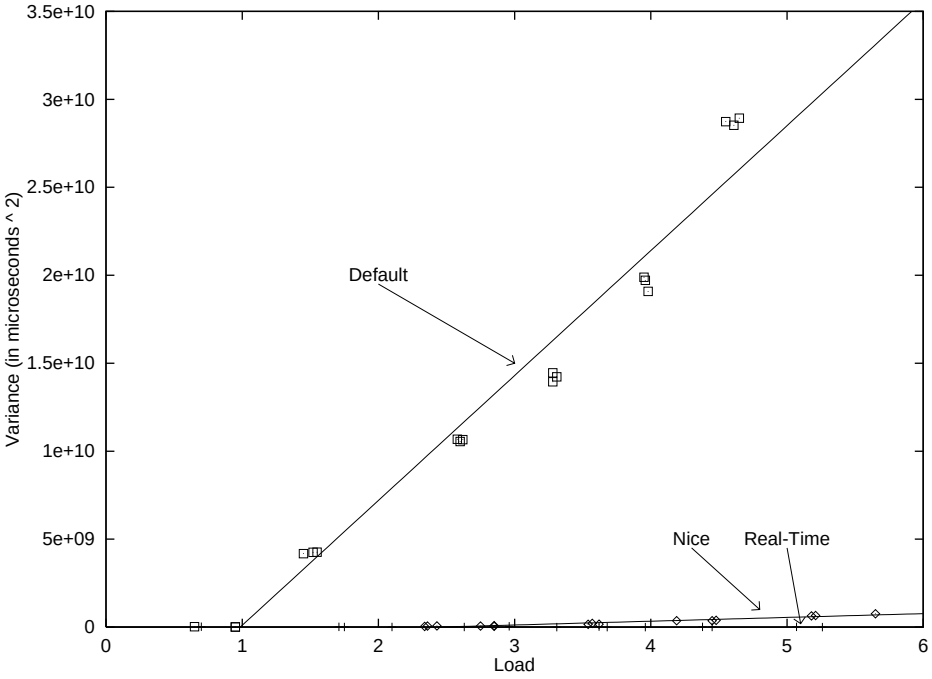

Figure 3.10: Jitter versus Receiver Load. The horizontal axis is the processor load as reported by the Unix `w` command. The vertical axis the the variance in interarrival times. The lines are the least squares line fits for default priorities, niced priorities and real-time priorities, as indicated.

3.5.2 Results and Analysis

Figure 3.10 compares the jitter for real-time, nice and default priorities as processor load increases. The slopes of the lines indicate how sensitive a process running under the given priority is to the effects of increases in processor load. The steeper the slope, the more jitter the process contributes as processor load increases. Both nice and real-time priorities have *significantly* gentler slopes than default priority, indicating that nice and real-time processes do not suffer nearly as much jitter as do as default priority processes as processor load increases.

Figure 3.11 is a close-up of the nice and real-time priorities in Figure 3.10. The most steeply sloped line is for default priority. The next most steeply sloped line is nice priority and the horizontal line is real-time priority. Real-time priority has nearly

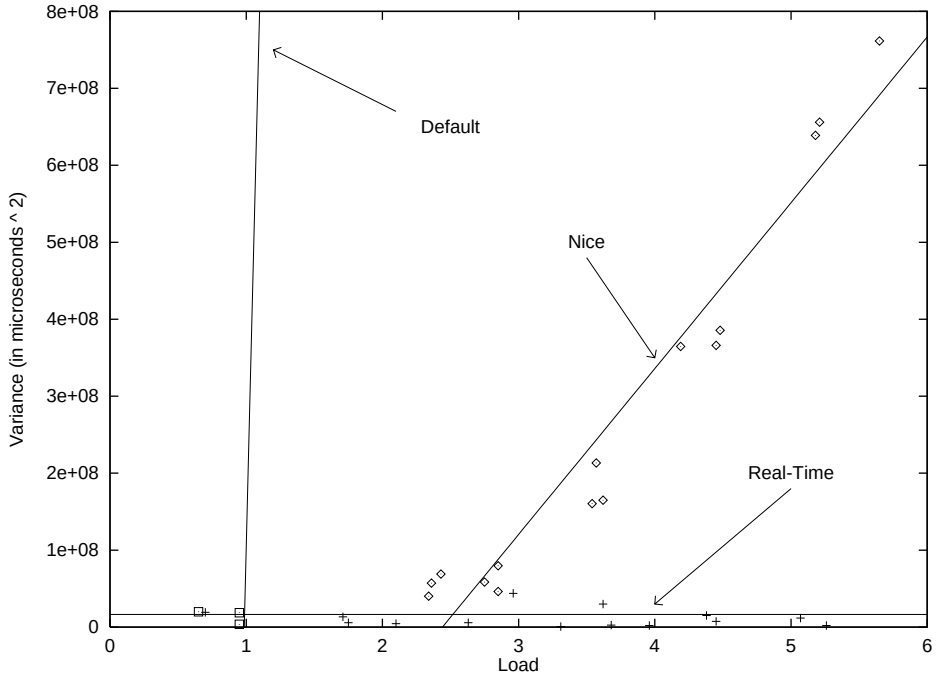

Figure 3.11: Zoom of Jitter versus Receiver Load. This graph is a close-up of Figure 3.10 by changing the upper bound on the vertical axis from 3.5e+10 to 8e+08. The horizontal axis is the processor load as reported by the Unix `w` command. The vertical axis the the variance in interarrival times. The lines are the least squares line fits for niced priorities and real-time priorities, as indicated.

a flat slope, indicating that real-time processes show almost *no* increase in jitter as processor load increases. Nice priority processes, however, do suffer from increased jitter with increased processor load, as indicated by its steep slope. Notice that the line for nice priority intersects the x-axis at about a load of two, as opposed to default priority which intersects the x-axis around a load of one. This indicates that even under light loads, nice priority processes reduce jitter as opposed to default priority processes.

3.5.3 Section Summary

Real-time priorities significantly reduce jitter as processor load increases. The severity of jitter that was observed when real-time priorities were used was extremely light compared to the jitter without real-time priorities. Real-time priorities show almost *no* increase in jitter as processor load increases. Real-time processes have priority over other processes, allowing them to respond to multimedia data with less jitter than do nice or default priority processes. Note, however, that real-time priorities must be used carefully. A real-time process can starve other process of CPU time, including critical operating system processes. We found this out the hard way when we ran a counter process in real-time mode. The counter process consumed *all* of the CPU cycles, not even allowing enough CPU time to kill the process as the super-user. We were forced to reboot our computer, which displeased our system administrator and other users as our computer was a server. In addition, if many processes are run with real-time priorities the benefits in jitter reduction will most certainly be reduced compared with having just one process with real-time. To picture this, assume that *every* process ran in real-time mode. All processes would compete for CPU time in the real-time queue and scheduling would occur just as it does in the usual case. The benefits of having a real-time priority mode would be lost.

Nice priorities significantly reduce jitter as processor load increases. Under heavy loads, nice priority processes had much less jitter than did default priority processes. Even under light loads, nice priorities significantly reduced jitter versus default priorities. The improved user-mode priority of a niced process causes it to run ahead of other eligible runnable user-mode processes, allowing the niced process to respond to multimedia data with less jitter. However under heavy processor loads, nice priority processes did suffer from some increased jitter while jitter under real-time processes remained nearly constant. Even a nice priority process will run after any newly-activated process, regardless of its nice value, causing the niced process to suffer from more jitter than would a real-time process. The newly-activated process will complete

its time-slice before the CPU reschedules and activates the niced process. When there are more processes, this becomes more likely, causing the niced process to suffer from more jitter.

The amount of jitter reduction from nice or real-time priorities depends upon the processor load. We can use our model in Section 2.3 to determine the processor load from multimedia applications. As we show in Sections 5, 6 and 7, with our model we can also explore how the reduction in jitter from nice and real-time priorities benefits the application quality as future multimedia applications push processors to the limit.

3.6 Network Experiments

From the experiments in Section 3.4 and Section 3.5, we know that high-performance processors and real-time priorities affect jitter. We next examine the effects that high-speed networks have on jitter.

3.6.1 Specific Experimental Design

We used 4 processor SGI Challenge L workstations. Each workstation was connected to three networks: a 10 Mbit/s Ethernet; a 266 Mbit/s Fibre Channel; and a 800 Mbit/s HIPPI.

We induced network load using `netperf` [69]. Netperf is a benchmark that can be used to measure the performance of many different types of networks. It provides tests for both uni-directional throughput and end-to-end latency.

Both HIPPI and Fibre Channel are point-to-point networks, unlike Ethernet which is a shared medium network. Inducing network load between two workstations that were not the sender and receiver did not increase jitter because they did not share the same physical wire. Instead, we ran the `netperf` and `netserver` processes on the same workstations as the sender and receiver, respectively. We ran the netperf processes on different processors than the sender and receiver by using the `runon`

command. This minimized the jitter that might be contributed by the processor load induced by the `netperf` processes.

3.6.2 Results and Analysis

Figure 3.12 depicts the experiment results that show the interarrival times for the three networks under two different conditions of network load. The "no load" runs are the experiments on a quiet network. The "load" runs are the experiments run with `netperf`. The three "no load" lines are almost indistinguishable, indicating no correlation between jitter and network bandwidth. However, under high network load there is a noticeable difference in the interarrival times for the three networks, indicating there may be a correlation between jitter and network bandwidth.

Figure 3.13 shows the relationship between jitter and network bandwidth for loaded networks. We expect that jitter decreases as network bandwidth increases, so we graphed the theoretical network bandwidth versus 1/variance. There are three data points plotted, one for each of Ethernet, Fibre Channel and HIPPI. The line represents a least squares line fit through the three data points. The correlation coefficient is 0.98, indicating that there is a strong inverse relationship between jitter and network bandwidth. In other words, as network bandwidth increases, jitter decreases. Note that although the correlation coefficient is a high 0.98, the fact that there are only three data points is evident in the width of the 95% confidence intervals around the least squares line fit. It would be nice to have more data points in order to strengthen the statistical significance of our results. To do this, however, we need to have new networks on which to experiment.

3.6.3 Section Summary

Under low network loads, high-speed networks do not significantly reduce jitter. Under low loads, the network contributes little variation to the interarrival times for the multimedia frames. Most of the variance is caused outside the network, rendering

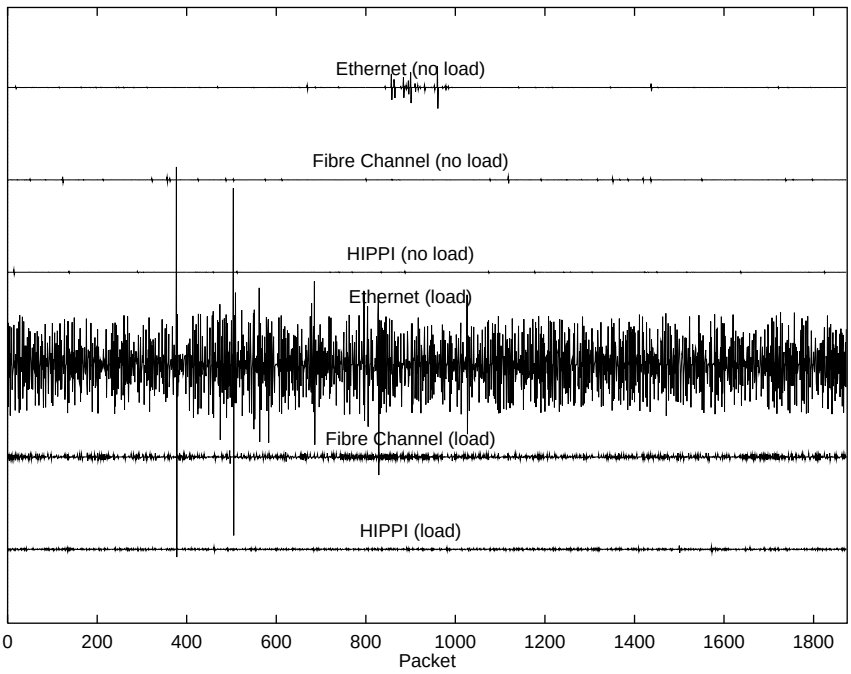

Figure 3.12: Interarrival Times for Different Networks and Network Loads. There are 6 multimedia streams represented in this picture, each by by a horizontal line depicting the interarrival times on the indicated network. The network was either either loaded or unloaded. Each multimedia stream is off-set from the one below it by 750,000 microseconds. The horizontal axis is the packet number.

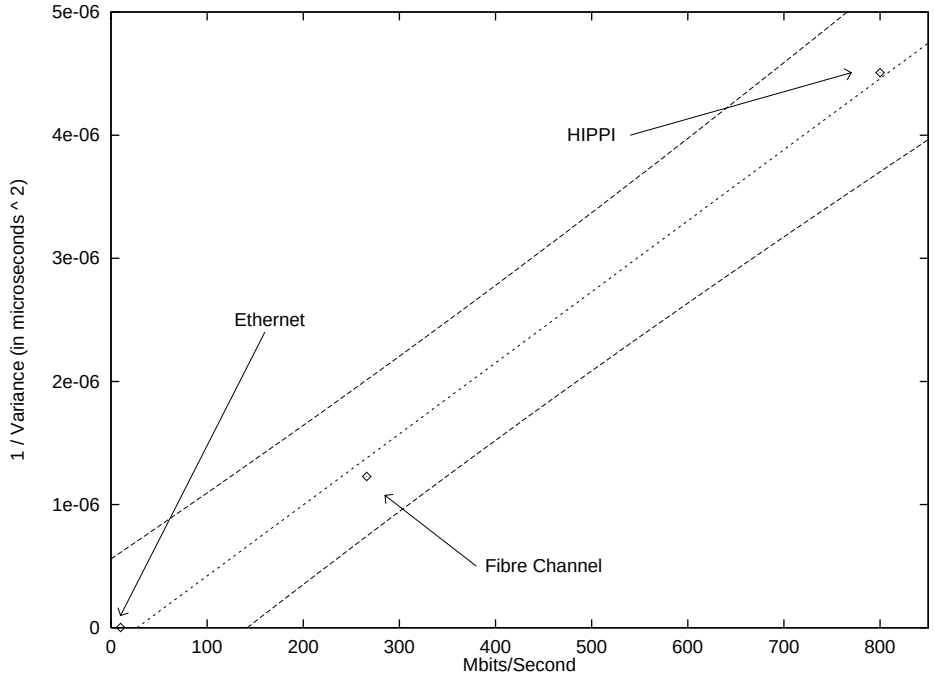

Figure 3.13: Jitter versus Network Bandwidth. The horizontal axis is the network bandwidth. The vertical axis is 1 / variance. The line is a least squares line fit of jitter versus theoretical network bandwidth for three networks: Ethernet, Fibre Channel and HIPPI. The correlation coefficient is 0.98.

any jitter reduction from the high-speed networks insignificant.

However, under heavy network loads, high-speed networks significantly reduce jitter. Under heavy loads, network contention causes multimedia frames that are ready to be delivered to be delayed. Because network traffic is often bursty, the delays are non-deterministic, causing increased variation in the interarrival times for the multimedia frames. High-speed networks reduce the amount of time to deliver the interfering network traffic, decreasing the variation in the multimedia interarrival times.

The amount of jitter reduction from high-speed networks depends upon the network load. We use our model from Section 2.3 to determine the network load from multimedia applications as the number of users increases. Then, using the results from this section, we can determine what effect jitter reduction from high-speed networks has on the application quality.

3.7 Summary

Jitter hampers computer support for multimedia. Jitter is the variation in the end-to-end delay for sending data from one user to another. Jitter can cause gaps in the playout of a stream such as in an audioconference, or a choppy appearance to a video display for a videoconference. There are several techniques that can be used to reduce jitter. In this work, we have experimentally measured the effects of three jitter reduction techniques: high-performance processors, real-time priorities and high-speed networks.

Jitter researchers have used a variety of metrics to measure the amount of jitter in a multimedia stream. However, with the exception of range, all previously-used jitter metrics are statistically similar. Range is probably not an appropriate measure of jitter because it is very susceptible to outliers. While outliers do create glitches in an audio or video stream, they can often be smoothed over and ignored by human

eyes and ears.

We find high-performance processors, real-time priorities and high-speed networks all significantly reduce jitter under conditions of heavy load. Multimedia applications, tending to be resource intensive, are likely to push processors capacities to the limit, making conditions of heavy load likely. As the growth in distributed collaborative applications continues, multimedia applications will push network bandwidths to the limits, also. Thus, high-performance processors, real-time priorities and high-speed networks will all be effective in reducing jitter.

Computer systems continue to get faster while human perceptions remain the same. Future system improvements may remove enough of the underlying jitter such that application-level jitter reduction techniques are unnecessary. How far in the future will this be?

In Section 5.6.4, we see that as the area under the jitter compensation curve decreased, the required buffer decreased. From Figure 5.22, if we had an area of 75,000 square microseconds or less, we would require virtually no buffering in order to achieve acceptable quality. From Figure 5.23, we can predict that this will happen if jitter is 10^9 square microseconds or less. From our results in Sections 3.4, 3.6 and 5.6.4, we can predict how powerful hardware must become in order to achieve this low jitter rate. We can determine when this will be if we assume both processor power and network bandwidth double each year, as has been the trend in the past [56]. Figure 3.14 depicts these predictions.

For the next 5 years, hardware improvements alone will not reduce the effects of jitter low enough to eliminate the need for application buffering techniques. However, if we implement real-time priorities in scheduling our multimedia stream, we can reduce jitter enough to eliminate the need for application buffering today.

However, improving jitter alone is not sufficient to guarantee improving application quality. In addition to jitter, the application quality also depends upon *latency*, which decreases the application realism, and *data loss*, which reduces the application

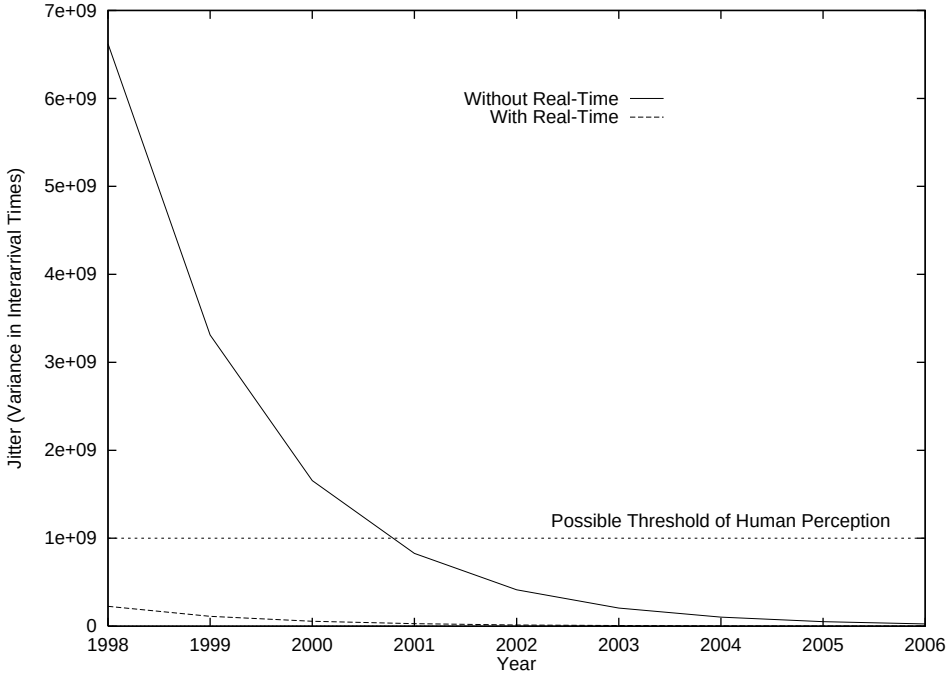

Figure 3.14: Jitter versus Years. The horizontal axis the year. The vertical axis is the amount of jitter. There are two sets of predictions. The top, slightly curved line depicts the amount of jitter in a system without real-time priorities. The lowest, slightly curved line depicts the amount of jitter in a system with real-time priorities. The horizontal line represent the threshold below which application buffering would not be needed.

resolution. We have developed a model that allows us to evaluate the effects of jitter reduction in the larger context of a user's perception of the multimedia application. Our model allows us to show how advances in networks and processors will improve application quality without tuning the operating system or the application.

Chapter 4

Application Method

4.1 Overview

In order to plan for the quality of computer enhancements, we have developed a method to apply our quality planning model to distributed collaborative multimedia applications. We start by studying the application from the perspective of the user. The user describes a distributed collaborative multimedia application and defines a set of requirements that need to be fulfilled if the application performance is to be acceptable. We model the user, application, computer system and quality metric. We perform some detailed experiments, called *micro experiments*, to measure the fundamental components of the application. We test the accuracy of an analytic model based on the micro experiments through larger experiments, called *macro experiments*. We use these macro experiments to calibrate the model. Lastly, we predict the application quality as various model components change. Figure 4.1 depicts our method.

We present the methods we use in applying our quality planning model to multimedia applications. Our model is intended to locate the bottlenecks in the performance of distributed collaborative multimedia applications. There are three classes of solutions to this problem[62]:

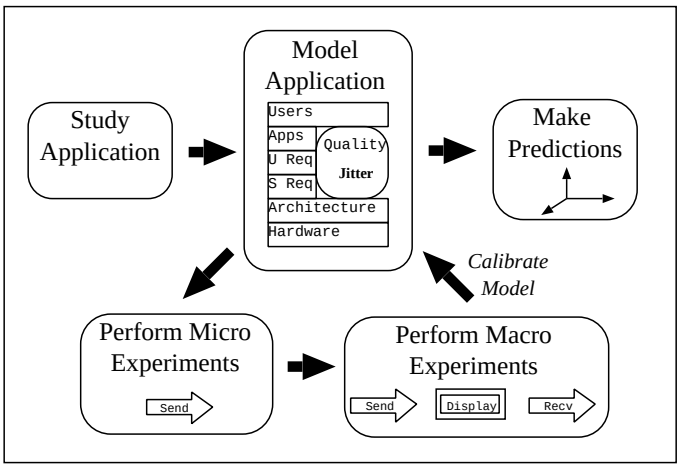

Figure 4.1: Quality Planning Method. We have developed a method for applying our model to distributed multimedia applications. We start with an application, develop our model, perform micro and macro experiments and make quality predictions.

1. *Analytic Model.* An analytic model is a mathematical representation of a computer system. Analytic models are fast and flexible, making them inexpensive to develop and to run. Analytic models allow easy variation of system parameters, which provides insights into the effects of various parameters and their interactions.

2. *Simulation.* A simulation is a software representation of a computer system that may include application, operating system, processor and more. Simulations often capture more system details than do analytic models, but are often less useful for comparing trade-offs among different parameters. Simulation measurements can often take a long time, either to build the simulation or to run it. However, simulation time can often compress wall-clock time that is on the order of days or weeks. In these cases, simulation runs can be much faster than measurement.

3. *Measurement.* Measurement is possible only if something similar to the pro-

posed system already exists. When planning for future application performance on future systems this is not possible. When they are possible, measurements usually take longer than analytic models, but shorter than building and running a simulation. However, there is greater variability in the time required to perform measurements than in the time required for simulation or analytic modeling. Measurements are often the most believable to users, which also makes them useful for validating simulation or analytic models.

We use all three solutions where appropriate. At the heart of our method is an analytic model. Analytic models are most effective when they are based on actual measurements [62]. We base our models on careful measurements of fundamental application components in our micro experiments. One of the concerns with analytic models is that they abstract away too many details to be useful for real-world application predictions. In order to address this concern, we perform macro experiments that compare our model predictions to real-world application performance in an attempt to test the accuracy of our models. If the real-world application we wish to study is not available, either because the users are not at hand or the application has not yet been build, we simulate the application performance in our macro experiment and compare the measured simulation results to our model predictions.

This Chapter is organized as follows: Section 4.2 presents related work in benchmarks and capacity planning. Section 4.3 describes the first step of our method, how we study the application to obtain information for the model. Section 4.4 details our model of the user, application and computer system. Section 4.5 introduces the experiments we use to measure fundamental application components. Section 4.6 introduces the experiments we use to test the accuracy of predictions we make based on the fundamental components. Section 4.7 briefly mentions our methods for predicting application performance under different system configurations. And Section 4.8 summarizes the important points of this chapter.

4.2 Related Work

4.2.1 Benchmarks

> `benchmark` (v. trans). *To subject (a system) to a series of tests in order to obtain prearragned results not available on competitive systems.*
> S. Kelly-Bootle, "The Devil's DP Dictionary"

The process of performance comparison for two or more systems by measurements is called *benchmarking*, and the workloads used in the measurements are called *benchmarks*. Standard measures of performance provide a basis for comparison, which can lead to improvements and predict effectiveness under different applications. If computer users ran the same programs day in and day out, performance comparisons would be straight-forward. However, as this will never be the case, systems must rely on benchmarks to predict performance of estimated workloads.

As early as 1971, Lucas provided a survey categorizing benchmarks [70]. His work divided benchmarks into seven categories:

Timings. Early computer systems were evaluated based on a comparison of processor cycle times and add times.

Mixes. The instruction mix is an attempt to provide a broader range of evaluation than timings. A frequency of execution is specified and timings given appropriate weights. An example of a previously used instruction mix is *Gibson* [56].

Kernels. A kernel program is a typical program that has been partially coded and timed. The kernel includes more parameters than mixes however they generally do not include adequate I/O considerations. The *Sieve of Eratoshenes* is a program sometimes used as a kernel.

Models. An analytic model is a mathematical representation of a computer system. The performance of the slotted Aloha network protocol is a well-known example

of an analytic model [116].

Benchmarks. A benchmark is a specific program that is coded in a specific language and executed on the machine being evaluated. The `Livermore Loops` are one of the oldest, widely-used benchmarks [86]. `LINPACK` has extensive use even up until today [34]. `HINT` promises to be a benchmark that measures workstation capacity in a realistic manner [50].

Synthetic Programs. A synthetic program is a representation of a real-world program that is coded and executed on the machine being evaluated. Unlike benchmarks, synthetic programs are intended to represent an application that will be used on the machine. Two popular synthetic programs are `Whetstone` [29] and `Dhrystone` [124].

Simulation. Simulation uses software to represent some or all of a computer system, including the application and hardware. Simulations can be trace-driven, event-driven or probabilistic, such as in a Monte Carlo simulation [62].

Monitor. Monitoring is a method of collecting data on the performance of an existing system. `Gprof` is a method of collecting profile data on a running program. `Etherfind` is a method of collecting network information from an active network.

Figure 4.2 depicts our layout of his categories and the scope of our quality planning models.

Jain, Hennessey and Patterson break benchmarks into four categories [56, 62]:

Real Programs. Real programs are applications that users will actually run on the system. A system under such programs can be observed to get a good indication of performance under normal operations. Examples are language compilers such as `gcc`, `acc` and `f77`, text-processing tools like `TeX` and `Framemaker`, and CAD tools like `Spice`.

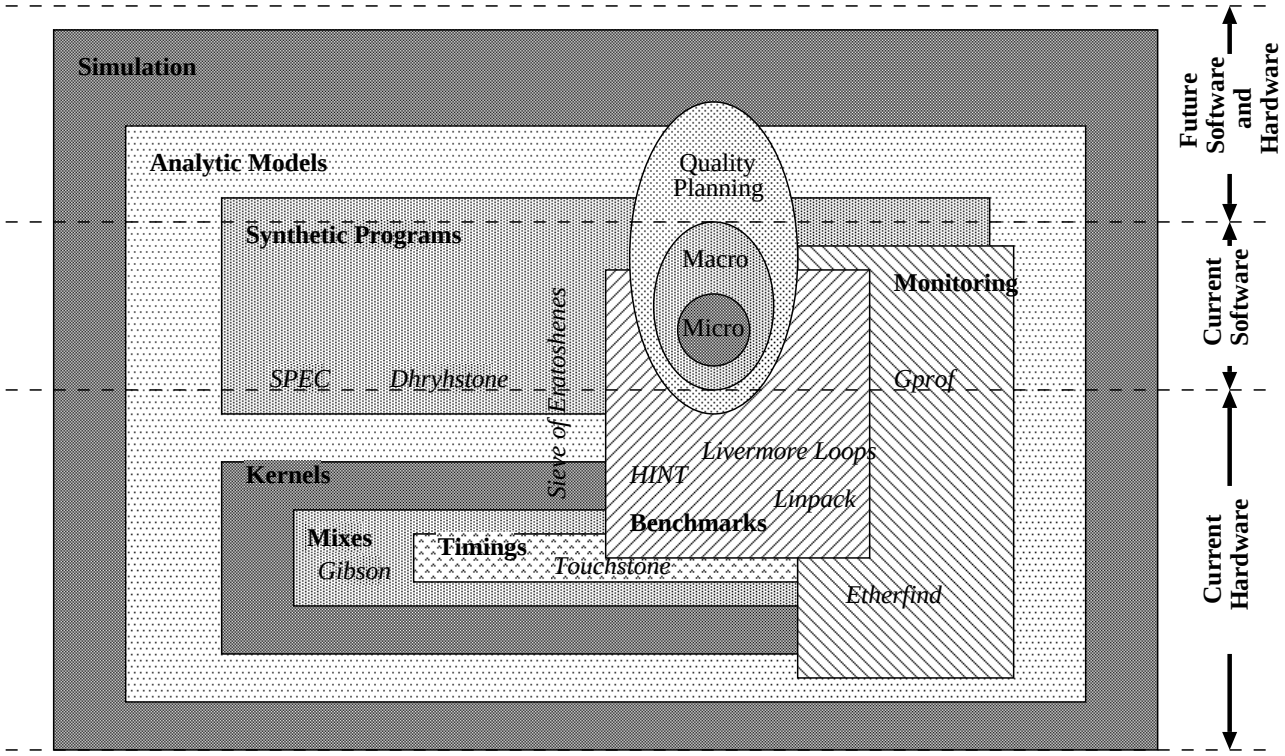

Figure 4.2: Scope of Quality Planning Models. This picture depicts the scope of our quality planning models in relation to benchmarks. Included as a subset of our quality planning models are our macro and micro experiments. The vertical axes depict approximate ranges that the various benchmark categories span.

Kernels. Kernels are small, key pieces from real programs. Unlike real programs, no user runs kernel programs, for they exist only for performance measurements. They are best used to isolate performance of key system features. Examples are `LINPACK` [34] and the `Livermore Loops` [86].

Synthetic Benchmarks. Synthetic have characteristics similar to those of a set of real programs and can be applied repeatedly in a controlled manner. They required no real-world data files (that may be large and contain sensitive data), and can be easily modified without affecting operation. They often have built-in measurement capabilities. Examples are the Byte benchmarks [109], `Whetstone` [29] and `Dhrystone` [124].

Toy Benchmarks. Toy benchmarks are small pieces of code that produce results known before hand. They are small, easy to type, and can be run on almost any computer. They may be used in some real programs, but often are not. Examples include the `Sieve of Eratosthenes` and `Quicksort`.

SPEC, the Standard Performance Evaluation Corporation, has sought to create an objective series of synthetic benchmarks [28]. SPEC is a non-profit corporation formed by leading computer vendors to develop a standardized set of benchmarks. Founders, including Apollo/Hewlett-Packard, DEC, MIPS and Sun, have agreed on a set of real programs and inputs that all will run. The benchmarks are meant for comparing processor speeds. The SPEC benchmark numbers are the ratio of the time to run the benchmarks on a reference system and the system being tested. We use SPEC results to make predictions in our quality planning model. Performance results from our research may also be useful to other benchmark researchers.

4.2.2 Capacity Planning

"Do not plan a bridge's capacity by counting the number of people who swim across the river today." Heard at a presentation

The study of computer resources needed to meet expected computer demand is called *capacity planning*. Many experts agree that the main goal of capacity planning is to maintain a balance between business growth and needs and explicit or implicit service level objectives of computing support. In other words, capacity planning is a method used for projecting computer workload and planning to meet the future demand for computing resources in a *cost-effective* manner.

Tim Browning applies the concepts of capacity planning to computer systems with a blend of measurement, modeling and analytic methods. He provides business-oriented forecasts based on service level objectives, chargeback and cost control [13]. However, capacity planning is a difficult task because no clear economic framework exists to do a cost-benefit analysis of information technology. There have been several general frameworks established in an attempt to manage the growth of information technology [61, 67]. There are also many specific capacity planning solutions designed for specific platforms and intended to plan for specific workloads [126]. Generally, it is still easier to find that a configuration will *not* support a specific level of service than to predict it *will*.

Snell describes several commercial products that are designed for capacity planning on the Internet [111]. Most of the tools she describes are high in cost and difficult to deploy. She cites integration of the the tools into a company's environment and filtering the data as major obstacles.

We develop a form of capacity planning that emphasizes the quality of the application as perceived by the user, enabling designers to tradeoff application performance and system cost.

4.3 Study Application

Our method begins by studying the application to obtain information on the users and their requirements. We start with the application users. People interact with touch,

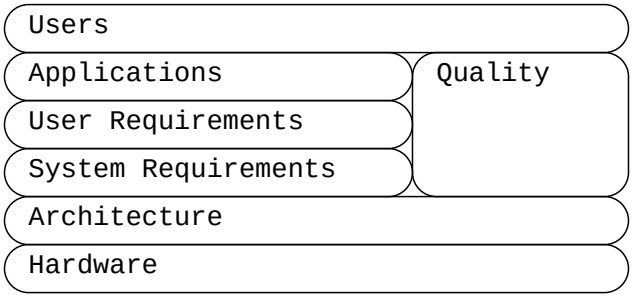

Figure 4.3: Quality Planning Model. Our model of application performance incorporates users, applications, user requirements, system requirements, architecture and hardware. In addition, we include a measure of application quality as perceived by the user.

sight and hearing. We would like computers to come as close to real-life interaction as possible, even enhancing personal interaction by allowing it across both time and space. The application is founded on a set of user requirements that need to be fulfilled for the application to be effective for the user. These include information such as frame rate and frame size, acceptable latency and jitter and tolerance of data loss.

4.4 Model

We use the information about the users and their requirements in our model. Our model for the quality of a distributed multimedia application incorporates the user, application and hardware. Figure 4.3 depicts our model.

Users The users of the application are those we used during the "Study Application" phase of our method as described in Section 4.3. Applciation users would like to interactively view a 3D brain database in Sweden while working in Minnesota; synchronously train as a fighter pilot in a virtual battlefield with a thousand other soldiers spread across the country; or discuss research with a half-dozen scientists in

a computer-mediated meeting.

Applications The applications are the software programs the users will run. For the examples under **Users** above, these would be: a "flying" interface to a 3D brain database; a DIS flight simulator; and an audioconference.

User Requirements The user requirements are the user's interface to our model. The requirements they specify may drive the selection of the underlying system in order to make the application acceptable for the user.

System Requirements The user requirements impose a series of requirements on the system. Some of these include network bandwidth, disk throughput and processor power. The method to determine the system requirements from the user requirements depends upon the application and, to some extent, its implementation. For instance, the workstation can make rendering frames the highest priority, possibly forsaking sending and receiving data to do so. Or, sending and receiving data can be the top priority at the expense of a lower display frame rate. Data packets can be compressed before sending, reducing network bandwidth but possibly increasing processor load from compression and decompression.

Architecture Architecture is the structure of the distributed program which determines the location of data and the distribution of the processing. Architecture can greatly affect the application. For example, a multimedia application that supports all users on one central mainframe would perform differently than one in which each user had a dedicated workstation connected by a network.

Hardware Given the system requirements and architecture, the hardware needed to support the application can be determined. Hardware might range from a low-end workstation with a T1 network up to a high-performance workstation with a HIPPI network.

Quality The variations in hardware, architecture, system requirements, user requirements and the application all effect the application quality as perceived by the user. The acceptability of the application to the user is determined by how closely the application performance matches the user requirements. We are developing a quantitative measure of the difference between application performance and user requirements. We call this the application quality. See Section 2.3 for more details.

As an brief example of the application of our model, suppose we wish to predict the performance of a proposed voice mail system that will allow a group of software engineers browse their archived voice-mail [12]. We first determine the quality of the audio required by the users, either by MOS user testing or by an analogy to similar applications. The audio quality determines the user requirements. The system requirements are derived from the user requirements, with key system components used to examine tradeoffs. For example, we might vary the number of users, the amount of compression or the network protocol. We choose an architecture and hardware on which to analyze the system. For example, we might pick Sun Sparc 5 workstations connected via a 10-base T Ethernet cable. As described in Section 2.3, we build a quality model based on the user requirements. The system requirements, architecture and hardware are all used in the quality model to determine if the proposed configuration is acceptable to the users. We can then iterate on our method, modify the component parameters and determine a new application quality.

4.5 Micro Experiments

Experiments that measure processor performance of the fundamental components of an application we call *micro experiments*. We do micro experiments to allow us to predict the effects of systems on applications built with those components. Some fundamental components for many multimedia applications include:

- *Record:* data from the microphone or video codec.

- *Play:* data to the speakers.
- *Render:* a frame to be displayed.
- *Display:* a frame to the screen.
- *Read:* data from a disk.
- *Write:* data to a disk.
- *Compress:* data.
- *Decompress:* data.
- *Send:* a data packet to a client.
- *Receive:* a data packet from a server.

We use a process that increments a `long integer` variable in a tight loop to measure the processor load for the individual components. The use of this *counter process* is depicted in Figure 4.4. To obtain a baseline for our counter, we run the counter process on a quiet machine. This gives the processor potential for the machine, depicted by the single column on the far left. For example, a Sun IPX will count to 1.7 million per second. We then run the counter process with each component in the model. The difference in the bare count and the component count is the component-induced load. For example, the middle two columns show the count obtained if two counter processes were run simultaneously. On a Sun IPX, each counter would count to about 0.85 million per second. In the last two columns, the processor load of the `send` process would be the count obtained on a bare machine less the count obtained by the counter process. If the Sun IPX were sending data at a rate of a 1280 byte packet every 160 ms, the count would be about 1.69 million in one second. The difference between the 1.7 million barecount and the 1.69 million send count can be converted into milliseconds of processor load, resulting in a send load of about 1 millisecond per second.

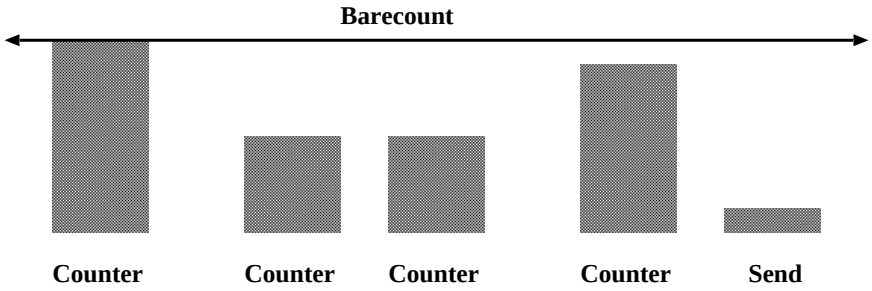

Figure 4.4: The Counter Process. Our model of application performance incorporates users, applications, user requirements, system requirements, architecture and hardware. In addition, we include a measure of application quality as perceived by the user.

To verify that the counter process does indeed accurately report loads of processor-bound processes with which it runs concurrently, we ran the counter process with 1, 2, 3 and 4 other counter processes. Since we assume the counter processes will have same priorities, we expect the counter value to be `(count on bare machine) x 1/(N+1)`, where N is the number of other counters running. Figure 4.5 shows the results of this experiment. The predicted values were within the confidence intervals for all the measured values.

After carefully measuring the processor load of each component, we can predict the processor load of an application built with those components. Changes in application configuration or changes in hardware are represented by modifying the individual components and observing how that affects performance.

4.6 Macro Experiments

Experiments that measure performance of applications built with micro experiment components we call *macro experiments*. We do macro experiments to test the accuracy of micro experiment-based predictions of application performance.

For example, assume we have a two-person audioconference that lasts for three

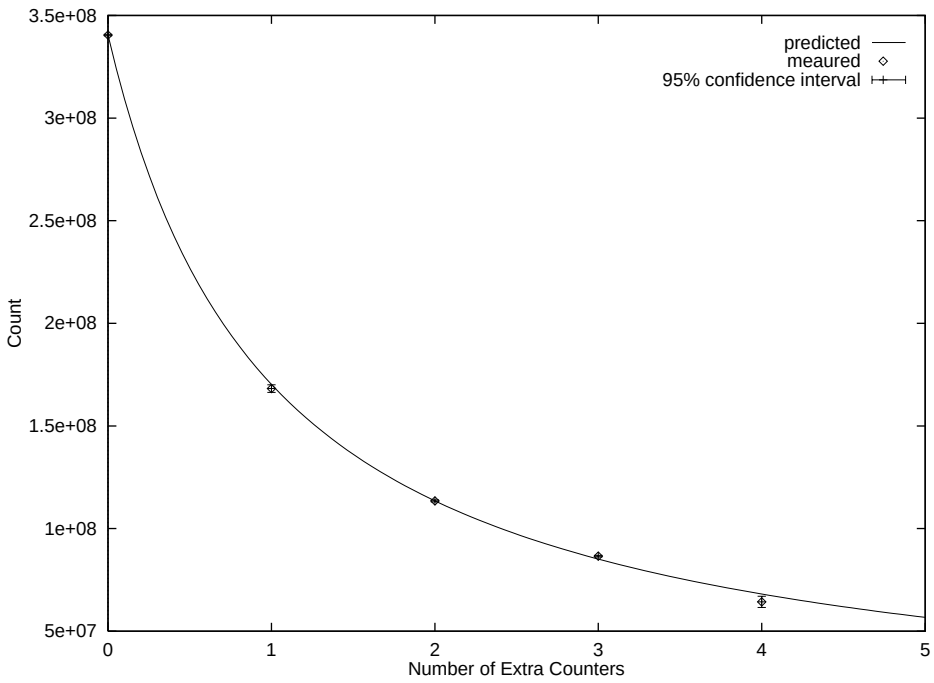

Figure 4.5: Count versus Number of Counters. The vertical axis is the count obtained by the counter process. The horizontal axis is the number of additional counters running. The curve represents the predicted value. All points are shown with 95% confidence intervals. The largest interval is 8% of the measure value. The curving line is the predicted value. The measurements were taken on a Sun Sparc IPX.

minutes. Each component of the audioconference, record send, receive and play, processes the three minutes of audioconference data. We predict the total processor load from our micro experiment measurements of the record, send, receive and play loads. In addition, we predict the network load based on the audio data rate of the workstations. In our macro experiments, we run a two-person audioconference and carefully measure the processor and network load. We then compare these measured values to the predicted values in an attempt to test the accuracy of our predictions methods.

4.7 Predictions

By modifying the fundamental application components, we can predict performance on alternate system configurations. This allows us to evaluate the potential performance benefits from expensive high-performance workstations and high-speed networks before installing them. Moreover, we can investigate possible performance benefits from networks and workstations that have not yet been built.

Our approach for evaluation of each alternative system is the same: We modify the parameters of our performance model to fit the new system, then evaluate the resulting model to obtain performance predictions. These analyses are intended to provide a sense of the relative merits of the various alternatives, rather than present absolute measures of their performance.

Our micro and macro experiments are done on only a handful of platforms. However, we would like our predictions to be accurate for untested platforms, and even future, as yet unbuilt hardware. In order to attempt these extrapolations we rely on research in benchmarks that compare the performance among systems and alternate system configurations. In particular, we rely upon SPEC benchmarks results to predict the performance of application components on untested workstations.[1] We rely

[1]Many SPEC benchmark results can be found at the Performance Database Server (PDS). The

upon landmark studies in network and disk performance to predict performance on alternate networks [11, 80, 79, 106, 101].

4.8 Summary

The strengths in our model and implementation method include:

1. *Tradeoffs:* A flexible way to compare the tradeoff in performance for alternate system configurations.

2. *Users:* A measure of performance from the point of view of the application users.

3. *Validation:* Experimental confirmation that our methods of predicting processor, disk and network throughput (the basis of our model) are accurate.

In the next three sections, we apply our model to three distributed, collaborative multimedia applications:

1. *Audioconferences:* Multi-person, synchronous audio conferencing.

2. *Flying:* 3-d, asynchronous, scientific database browsing.

3. *Virtual Cockpit:* Distributed, virtual-reality flight-combat training.

URL for the PDS is http://netlib2.cs.utk.edu/performance/html/PDStop.html

Chapter 5

Audioconferences

5.1 Overview

There is an increasing interest in the use of audio for computer-based communication applications.

- Electronic mail includes audio along with text [118]. Multimedia editors enhance text documents with audio annotations [17]. World Wide Web radio stations spread audio across the world. Movies, containing audio in addition to video, are starting to grace consoles everywhere [99]. And audioconferences synchronously link workstations [98, 82, 104].

- General-purpose workstations can have advantages over the use of specialized hardware: corporate and academic environments have ready-access to necessary hardware; and teleconferencing can be enhanced when computers are used through the use of record/playback, on-screen speaker identification, floor control and subgroup communication.

- Audio and video streams similar to those in a teleconference are often integrated into larger distributed multimedia applications. For example, a shared editor may allow several users to simultaneously collaborate on a document from

separate workstations. Audio and video links coupled with the shared editor enhance the editing process by making it more like face-to-face collaboration.

Audioconferences frequently support other collaborative tasks that are themselves processor-intensive. Therefore, efficient processor use is essential. Our goal is to identify improvements that reduce audioconference processor load. The major contribution of this section is an experimentally-based comparison of the processor performance benefits of five potential audioconference improvements:

- *Faster processors.* How much do faster processors benefit audioconference processor load?
- *Faster communication.* How much do more efficient network protocols and faster network speeds benefit audioconference processor load?
- *Better compression.* How much do improved compression techniques benefit audioconference processor load?
- *Hardware support.* How much does Digital Signal Processing (DSP) hardware reduce audioconference processor load?
- *Silence deletion.* How much does removing the silent parts from a conversation reduce audioconference processor load?

We focus particularly on a comparison of the first four areas to silence deletion. Without silence deletion, in a unicast network environment network bandwidth will increase $O(N^2)$ where N is the number of audioconference participants. However, in most conversations, only one person speaks at a time. Silence deletion detects silence, only transmitting the sound of the person who is presently speaking, reducing the network bandwidth increase to $O(N)$. Although silence deletion algorithms take additional processing time, they may yield a net savings by reducing the amount of data that must be communicated from $O(N^2)$ to $O(N)$. Therefore, we hypothesize

silence deletion will improve the scalability of audioconferences more than any of the other four improvements.

Our analysis can help direct research in networks, multimedia and operating systems to techniques that will have a significant impact on audioconferences. In addition, our approach may generalize to video and other forms of multimedia.

This Chapter is organized as follows: Section 5.2 of this section describes related work in audioconference experiments, measuring processor load, analyzing UDP, using silence deletion and audio compression. Section 5.3 introduces our model of an audioconference. Section 5.4 details micro experiments that measure the processor load of each component. Section 5.5 describes macro experiments that test the accuracy of the predictions based on the micro experiments. Section 5.6 analyzes the experiments and projects the results to future environments. And Section 5.7 summarizes the important contributions of this section.

5.2 Related work

5.2.1 Audioconference Experiments

There has been a variety of experimental audioconference work. Casner and Deering performed a wide-area network audioconference using UDP multicast [16]. They found disabling silence suppression increases average bandwidth and eliminates the gaps between packets that give routers a chance to empty their queues. They recognized that experimenters need better tools to measure audioconference performance. Our model may be one of the tools they seek. They conjectured that ubiquitous multicast routing support can greatly reduce network and processor loads. In addition, they described several on-going experiments in which readers can participate.

Terek and Pasquale implemented an audioconference with an Xwindow server [117]. They described the structure and performance of their system. In particular, they described a strategy for dealing with real-time guarantees.

Gonsalves predicted that without software or protocol overhead, a three Mbps Ethernet could support 40 simultaneous 2-way 64Kbps conversations [44]. Thus, if our results show what is needed to enable the processors to handle the conversation loads, the networks can.

Riedl, Mashayekhi, Schnepf, Frankowski and Claypool measured network loads of audioconferences using silence deletion [98]. They found silence deletion significantly reduces network loads. We analyze how silence deletion affects processor loads.

Frankowski and Riedl study the effects of delay jitter on the delivery of audio data [41]. They developed a heuristic for managing the audio playout buffer and compare it to several alternative heuristics. They found none of the heuristics is superior under all possible arrival distributions.

5.2.2 Measuring Processor Load

Jeffay, Stone and Smith discussed a real-time kernel designed for the support of multimedia applications [64]. They achieved some real-time guarantees through utilizing close to eighty percent of the processor. Our results may indicate methods that can trim audioconference processor loads, while achieving the same guarantees.

Lazowska parameterized queuing network performance models to assess the alternatives for disk-less workstations [77]. He used a counter process to measure processor loads. We use a similar process (see Section 5.4.1 of the present document).

Riedl, Mashayekhi, Schnepf, Frankowski and Claypool measured the processor load of an audio conference with no floor control and no silence deletion using a counter process [98]. They showed the processor loads quickly become prohibitive under increasingly large audioconferences. We provide further analysis and a component breakdown of the processor load.

5.2.3 Analyzing UDP

Cabrera measured throughput for UDP and TCP for connected Sun workstations [14]. In analyzing the call-stack for UDP in detail, he determined the checksum at the receiving end takes the most processor time; this checksum may not be needed for adequate quality sound.

Kay and Pasquale measured delay at the processor end for sending UDP packets for a DEC 5000 [73]. They found checksumming and copying dominated the processing time for high throughput applications.

Bhargava, Mueller and Riedl, divided communications delay in Sun's implementation of UDP/IP into categories such as buffer copying, context switching, protocol layering, Internet address translation, and checksum implementation [10]. They found socket layering and connection were the most expensive categories. We use their analysis of their kernel buffering techniques in defining our experiment.

5.2.4 Using Silence Deletion

Rabiner viewed voice as a measure of energy and presented an algorithm for discovering the endpoints of words [94]. Henning Schulzrinne implemented an audioconferencer with silence deletion [104]. We adapt and build upon their view of voice and silence deletion algorithms.

5.2.5 Digital Audio Compression

Pan surveyed techniques used to compress digital audio signals [89]. He discussed *u-law*, Adaptive Differential Pulse Code Modulation (ADPCM) and Motion Picture Experts Group (MPEG), compression techniques which are not specifically tuned to human voice. Kroon and Swaminathan described techniques to improve performance of Code Excited Linear Prediction (CELP) Type coders, compression techniques tuned to human voice [76]. We predict the processor load of audioconferences

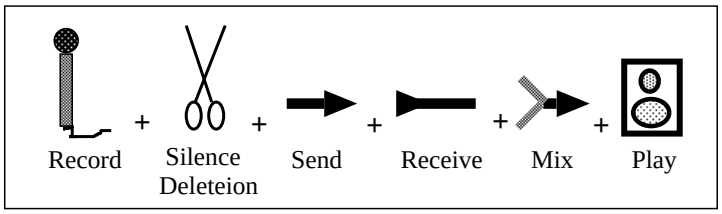

Figure 5.1: Audioconference Model. Our model of audioconference processor load, that includes recording from the audio device, silence deletion, sending and receiving packets, mixing sound packets that should be played simultaneously and writing to the audio device.

using such compression techniques.

5.3 Model

Figure 5.1 depicts our model of an audioconference processor load. Our model is based on the components of recording, silence deletion, sending, receiving, mixing and writing. *Recording* is the processor load for taking the digitized sound samples from the audio device. *Silence deletion* is the processor load for applying one of the deletion algorithms to the record sample. *Sending* is the processor load for packetizing the sample and sending it to all other stations. *Receiving* is the processor load for processing all incoming packetized samples. *Mixing* is the processor load for combining sound packets that arrive simultaneously. *Writing* is the processor load for delivering the incoming samples to the audio device. Our model asserts we can predict audioconference processor load from the sum of the above components.

Silence deletion removes silent parts from speech. Experiments have shown that silence deletion substantially reduces network load for two reasons [98]:

1. In a typical `N` person conversation, at any given time one person is talking and `N-1` are silent. With silence deletion, only the talking person's packets are sent; each workstation must send only 1/N of the packets on average. Without silence

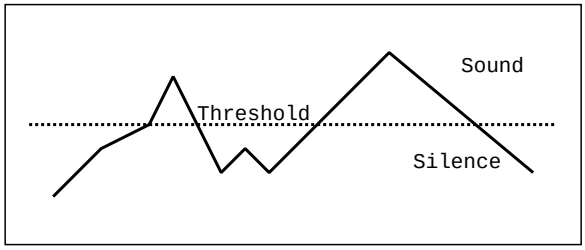

Figure 5.2: Absolute Silence Deletion Algorithm. The zig-zap line represents the energy of each byte. The horizontal line represents the sound-silence threshold. Bytes with energy above the threshold are treated as sound. Bytes with energy below the threshold are treated as silence.

deletion, all packets are sent; each workstation sends `N` of the packets. A linear increase in participants results an N^2 increase in network load.

2. Pilot tests suggest about 1/3 to 2/3 of the digitized speech data can be identified as pauses between words or sentences. Silence deletion may remove these pauses.

Although silence deletion algorithms themselves take additional processing time, they may yield a net savings in total processor load by decreasing the processor costs of the communication.

We consider four common silence deletion algorithms [21]. All four algorithms use the energy contained in each byte:

1. *Absolute* uses the average energy over chunks of bytes to determine silence. It removes all chunks with energy less than a threshold. See Figure 5.2 for a visual representation of the Absolute algorithm.

2. *Ham* (an adaptation of an algorithm used by Ham radio) removes chunks with enough consecutive byte energies below a threshold. Ham decreases a counter for each byte with energy below the threshold. When the counter reaches zero, the chunk is not sent on. Any byte above the threshold resets the counter to a maximum value. See Figure 5.3 for a visual representation of the Ham algorithm.

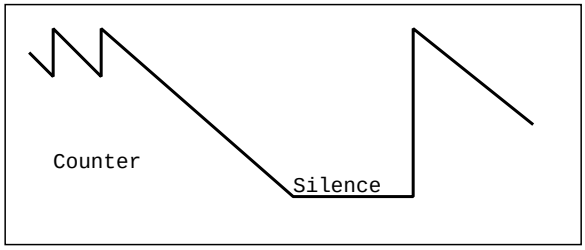

Figure 5.3: Ham Silence Deletion Algorithm. The zig-zap line represents the counter used to determine sound or silence. When the counter reaches zero the sound bytes are treated as silence.

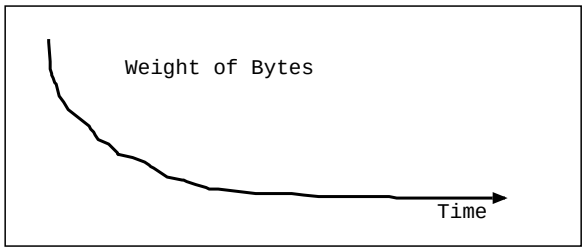

Figure 5.4: Exponential Silence Deletion Algorithm. The curved line represents the weighting of each byte in determining sound or silence. The more recent bytes (to the left) have a greater weight than older bytes (to the right).

3. *Exponential* also uses the energy in each byte. It removes all exponentially weighted byte energies that are below a threshold. The most recent bytes are weighted most heavily. Exponential decreases a counter value according to the decay value. When the counter drops below the threshold, the bytes are not sent on. The decay determines the exponential change. When decay is small, the counter fluctuates quickly. When decay is large, the counter fluctuates slowly. See Figure 5.4 for a visual representation of the Exponential algorithm.

4. *Differential* uses the changes in energy of each byte to determine silence. All chunks with changes below a threshold are interpreted as silence. The algorithm keeps a counter of the number of non-changes. If there are too few changes (the

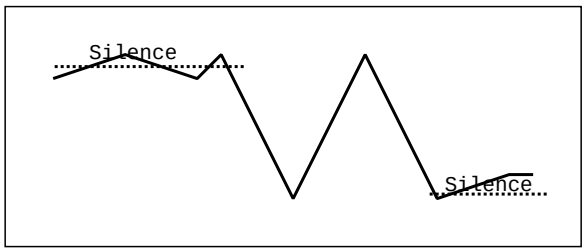

Figure 5.5: Differential Silence Deletion Algorithm. The zig-zap line represents the energy of each byte. The horizontal lines represent periods where there was little change in the energy of consecutive bytes. Little or no change in sound energy is treated as silence. Significant change in sound energy is treated as sound.

counter gets higher than a threshold), then differential does not send the chunk on. When a change is encountered, the counter is reset to zero. See Figure 5.5 for a visual representation of the Differential algorithm.

In our pilot tests, the Absolute and Exponential algorithms often yielded poor sound quality. Absolute is effective when signal-to-noise ratio is very high. This may occur in a recording studio or with very high fidelity magnetic tape. However, it is not practical in real-world situations [94]. Further MOS testing should be done to verify that Absolute and Exponential do, indeed, yield poor sound quality. In the meantime, we concentrate on the performance of the Differential and Ham silence deletion algorithms.

5.4 Micro Experiments

5.4.1 Design

Our micro experiments were designed to measure the processor load of audioconference components. We chose two Sun workstations, the 20 MHz SLC and the 40 MHz IPX, to test if the components of the audioconference scale with processor speed.

To obtain the processor load for each component of the model, we ran the counter process described in Section 4.5 in conjunction with a process for each of the components of the audioconference (see [21] for more information):

- *Read.* The read process takes sound from the audio device. User level processes access Sun's audio device driver `/dev/audio` through `open()` and `read()` calls. The audio device delivers sound at a rate of 8000 bytes per second. By default, it delivers it in 1K chunks, but this can be changed by modifying kernel variable ${}_{a}udio_{7}9C30_{b}size$. The audio device compresses 13 bit sound samples into 8 bit *u-law*.

  The packet size determines the read frequency. The read process sets a `signal()` and `alarm()` to read from the audio device at the appropriate frequency. The process reads until killed by the counter process.

- *Deletion.* The deletion process measures the load of the deletion algorithms. Since we want to measure the load of the algorithms only, we need to eliminate reads from the disk. Thus, the deletion counter must follow a different paradigm than the other pieces.

  The deletion process `forks()` a counter process which waits until given the `kill()` signal to start. Deletion then reads 200 seconds of sound into RAM and records the number of page faults from the `getrusage()` command. It then sends a signal to the counter to start it counting. Deletion applies the deletion algorithms described above to the 200 second sound buffer. After repeating processing the appropriate number of iterations (see Section4.1.2.2 on page16), deletion kills the counter process and records the page faults. The page faults are recorded to be sure that paging activity is not measured along with the deletion.

- *Send.* The send process sends UDP packets. Send opens and binds to a socket. It sets a `signal()` and `alarm()` to send a packet every 5 milliseconds. Send

delivers packets until killed by the counter.

A shell script starts the send process and dummy receive process to pull the packets off the network.

- *Receive.* The receive process receives UDP packets. Receive opens and binds to a socket. It blocks on the socket until a packet arrives. Receive listens on the socket until killed by the counter.

- *Add.* In an audioconference with more than two participants, sound recorded from two different stations may need to be played simultaneously. To do this, the *u-law* sound-bytes must be converted to linear form, added, and converted back to *u-law*. We use an efficient table lookup to do both conversion.

  Add repeatedly adds two arrays of sound together byte by byte. The two bytes to be added are converted from *u-law* to linear, added, and converted back to *u-law*. When killed by the counter, add reports the number of bytes it added.

- *Play.* The play process delivers sound to the audio device. The packet size determines the write frequency. The read process sets a `signal()` and `alarm()` to read from the audio device at the appropriate frequency. The process reads until killed by the counter.

5.4.2 Data Collection

Since the counter process measurements are sensitive to other processes, we performed the experiments on machines in single user mode. In single user mode, the processor runs a bare minimum of system processes and no other user processes.

We ran the counter process for 200 seconds to amortize start-up costs. To determine the number 200, we ran the counter process for increasing times and recorded the standard deviation of its counts. The standard deviations level out just before 200. At this point, the standard deviation is only 0.001% of the mean. Thus, we chose

one 200 second counter run as one iteration. Pilot tests indicated that five iterations for each machine at each packet size were needed to achieve reasonable confidence intervals.

The process we used to measure the deletion algorithm was memory intensive to avoid I/O costs. In order to avoid measuring unwanted paging, we recorded the total page faults during the experiments.The number of page faults during the deletion iterations is almost always three, which we accepted as the baseline case. We decided to discard cases that had more than three page-faults, as they incur extra processor load from paging. In our experimental runs, this situation never occurred.

The sound chunk size and the threshold levels determine how the deletion algorithms perform. We tuned them such that they deleted about 1/3 to 2/3 of live, audioconference speech (in accordance to the sound pilot tests mentioned above). We confirmed that the speech with the sound deleted was still very understandable.

The kernel changes between large and small buffers at various packet sizes has a direct influence on packet sending and receiving times [10, 21]. User messages are copied into buffers in kernel space. The kernel may use one large buffer and one copy for certain messages, while it uses several smaller buffers and several copies for a slightly smaller message. While the kernel is using the small buffers, there may be a steady continuous increase in time per message as the message sizes increase. When the kernel stops using the smaller buffers in favor of the larger buffers, there may be a break in the linear continuity. To avoid these possible nonlinearities, we collected data on 515 to 2000 byte packets, which we assume covers most audioconference packet sizes.

5.4.3 Results

The 200 second counts on bare machines are shown in Table 5.1.

Figure 5.6 shows the line equations obtained from the counter measurements for the deletion algorithms on the IPX. We have similar graphs for the SLC and for other

Machine	Bare Count	Left Endpoint	Right Endpoint
SLC	123924595	123920808	123928381
IPX	340539937	340204150	340875723

Table 5.1: Bare Counts for SLC and IPX with 95% Confidence Intervals (Indicated by the Left and Right Endpoints).

components of the model [21], but to avoid redundancy we do present them here.

Table 5.2 shows the values for the line equations for each of the audioconference components for each machine type.

The per-packet and per-byte terms above pertain to the equations: `Load(component) = per-packet + per-byte * bytes`. The equations are the processor costs for each component of an audioconference from which we can project the cost of a complete audioconference (see Section 5.6).

5.5 Macro Experiments

5.5.1 Design

In order to test the accuracy of our model in predicting audioconference processor loads, we measured the performance of a simple audioconferencer, *Speak*. Speak is two person, uses UDP, can employ any of the five deletion algorithms (Absolute, Differential, Exponential, Ham or None), and has little extra user-interface overhead.

We used Internet Talk Radio (ITR) files rather than real conversants. This made our experiments more reproducible and gave us a large conversation sample space from which to choose. Since the ITR files have one person speaking most of the time, the silence deletion algorithms typically deleted only 10% of the packets. As the number of audioconference participants increased, the one person speaking in the ITR audio would reflect the group communication characteristics less and less.

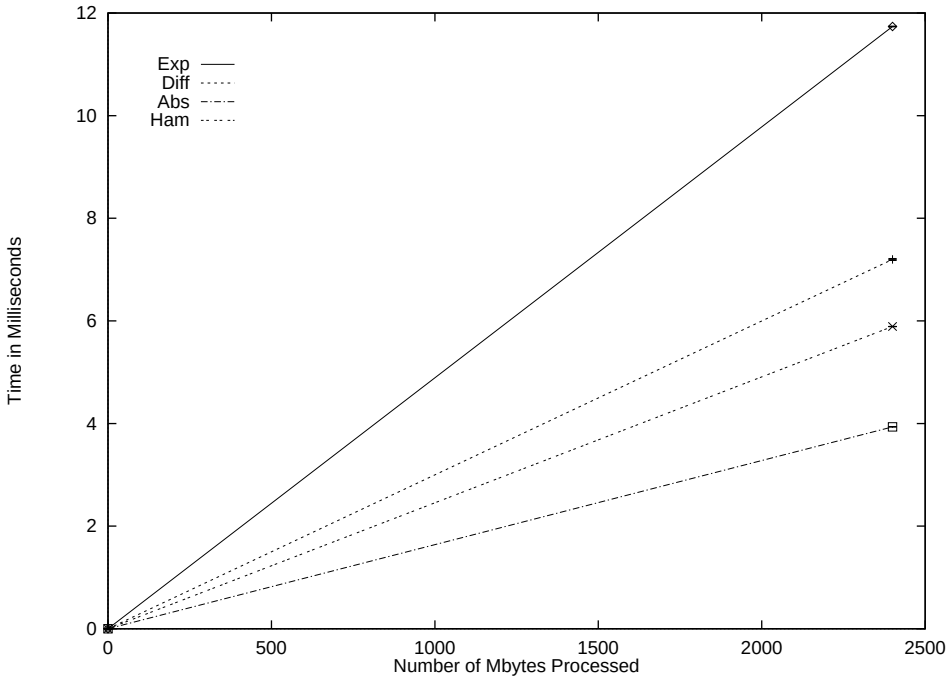

Figure 5.6: Processor Time for Deletion Algorithms on the Sun IPX. The four deletion algorithms are shown for their time to process 300 seconds worth of sound. All points are shown with 95% confidence intervals.

Operation	SLC per-packet	SLC per-byte	IPX per-packet	IPX per-byte
Record	0.810	0.0145	0.597	0.00169
Absolute	0.00	0.00302	0.000	0.000164
Differential	0.00	0.00563	0.000	0.00300
Exponential	0.00	0.0130	0.000	0.00489
Ham	0.00	0.00454	0.000	0.00245
Send	0.807	0.000194	0.210	0.000100
Receive	0.910	0.000129	0.187	0.000103
Mix	0.00	0.00546	0.000	0.00245
Play	1.26	0.0137	0.726	0.00103

Table 5.2: Values for Sun SLC and Sun IPX Line Fits for audioconference components. Units are in milliseconds.

However, the actual audio data used in these experiments does not matter, since our model is parameterized by the amount of silence deleted.

We did experiments on the five possible silence deletion methods on the SLC and two such methods on the IPX. A shell script initiated a remote Speak process and a local Speak process. One two hundred second conversation was one data point. We repeated each data point 5 times. We predict the load from the speak processes by using the micro experiment results. From the conversation length, the record size and the sample rate, we calculate the total packets read. By profiling the sound files with the deletion algorithms, we know the number and size of the packets sent, received and written. Because sound only arrives from one other Speak process, there is no mix component.

5.5.2 Results

Table 5.5 shows the results of the seven macro experiments.

Machine	Algorithm	Predicted	Actual	Left Endpoint	Right Endpoint
SLC	Absolute	45.55	44.84	44.76	44.91
SLC	Differential	54.92	49.02	48.93	49.11
SLC	Exponential	54.08	50.00	49.91	50.09
SLC	Ham	49.34	48.45	48.45	48.46
SLC	None	49.92	46.49	46.44	46.55
IPX	Differential	10.63	12.50	12.49	12.56
IPX	None	6.96	8.81	8.78	8.85

Table 5.3: Predicted and Measured Loads from the Macro Experiments with 95% Confidence Intervals (Indicated by Left and Right Endpoints). Loads are in seconds. Maximum load is 200 seconds.

The discrepancy in predicted and actual values may be due to the unforeseen costs that occur when putting micro components together. In most cases, the predicted values are within 10% of the actual values. We therefore consider the projected results presented in Section 5.6 to be significant only if the differences are larger than 10%.

5.6 Predictions

The processor loads in the macro experiments compared to the processor loads predicted by adding the components of the micro experiments suggests that our model may be a reasonable way to predict the processor load of audioconferences. It is difficult and expensive to run experiments with many people on a large number of workstations. Instead, from our micro experiments we extrapolate our results to conversations that have more silence and audioconferences that have more participants. Furthermore, we adjust the pieces of our model to compare the performance benefits of four potential audioconference improvements: faster processors, faster communi-

cation, better compression, and Digital Signal Processing (DSP) hardware. We also compare the benefits of better silence deletion algorithms.

All analysis can be done for both the SLC and IPX and with any of the four silence deletion algorithms. To avoid repetition, we do our extrapolations on one machine type (IPX) with one deletion algorithm (Differential). We use the fastest processor we studied to improve the quality of our extrapolations to even faster machines. Our results are largely independent of the type of silence deletion.

5.6.1 Increasing Participants

What happens when we have an increasing number of audioconference participants? We can extrapolate the loads at each workstation to an N person audioconference. The load depends upon the number of participants, the total conversation time, the packet size, the percentage of packets deleted and the percentage of sound removed from the remaining packets. The load without silence deletion does not have the deletion algorithm component. The load with silence deletion does not have the mix component since we assume only one person speaks at a time and the deletion algorithm removes all sound of those who are not speaking.

Figure 5.7 shows the predicted load for an audioconference without silence deletion and an increasing number of participants. The audioconference components are displayed, each line representing the sum of the particular piece and the pieces below it. The total load is the value of the line labeled play since it is the sum of all the components. The communication components, send, receive and mix, all increase as the number of participants increases. The mix component increases the fastest, accounting for approximately 70% of the processor load with 8 participants.

Figure 5.8 shows the predicted load for an audioconference with silence deletion and an increasing number of participants. Again, the audioconference components are all displayed. There is no mix component since we assume only one participant is speaking at a time. The communication components are only approximately 2% of

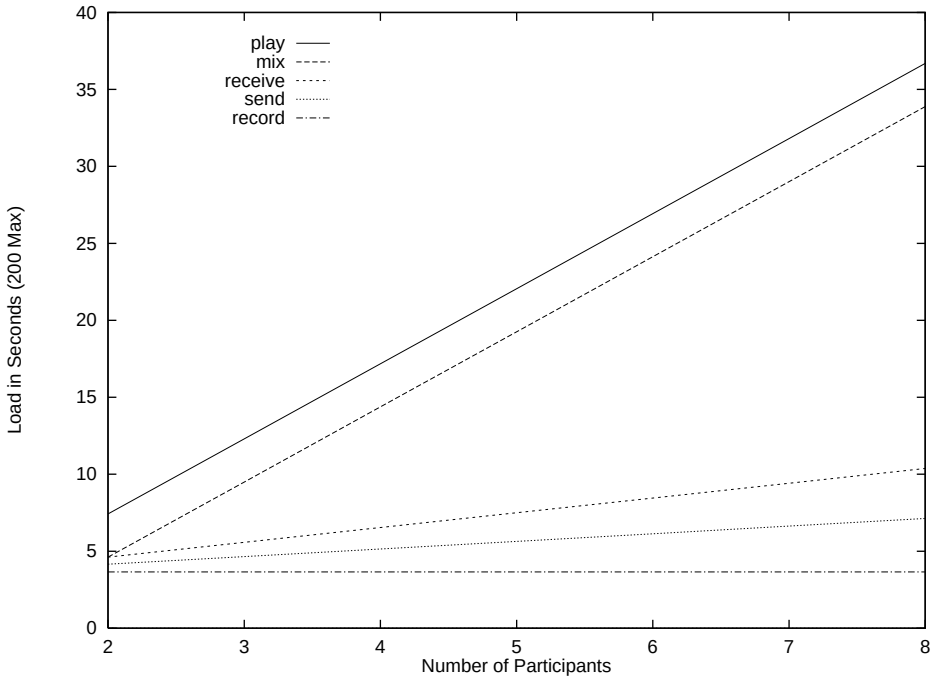

Figure 5.7: Sun IPX Processor Load Without Silence Deletion. The graph reads from the bottom. Each piece is the sum of the pieces below it. Thus, the total load is indicated by the play piece at the top. The maximum load is 200 seconds.

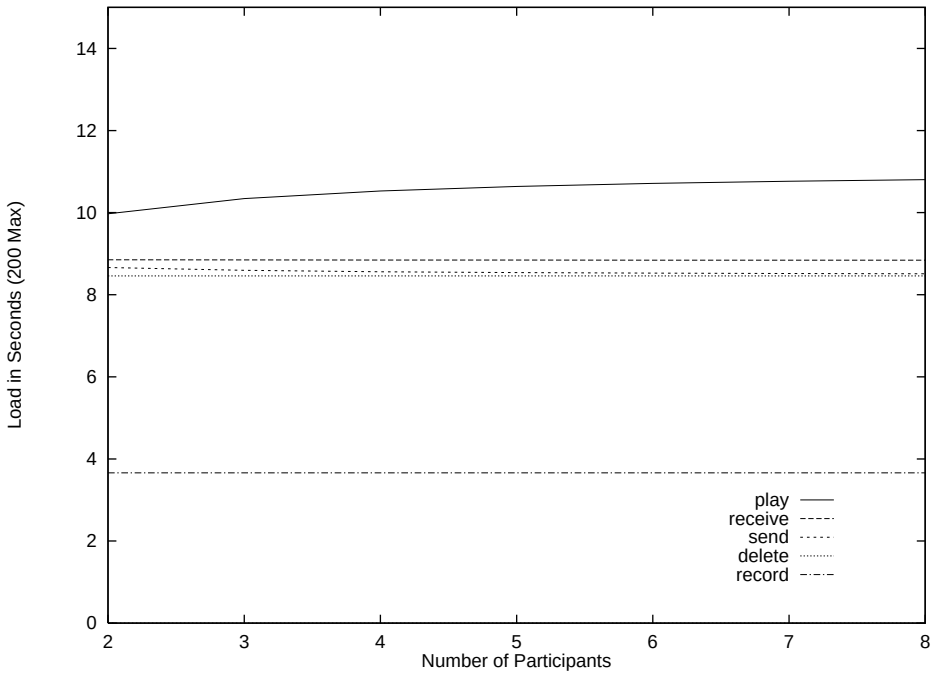

Figure 5.8: The IPX processor load with silence deletion. The graph records from the bottom. Each piece is the sum of the pieces below it. Thus, the total load is indicated by the play piece at the top. Since only one person sends packets at a time, there is no mix component. The maximum load is 200 seconds.

the total load for all audioconferencing. The record and play from the audio device account for about 50% of the audioconference load, which seems disproportionately high compared to the communication components. The silence deletion component is also large, accounting for just under 50% of the audioconference load. Section 5.6.3 investigates the effects of reducing this component through software optimization or DSP hardware.

Figure 5.9 shows the total loads with silence deletion and the total loads without silence deletion. Here and in all subsequent graphs, only the total loads are displayed. For three or more participants, silence deletion reduces processor load compared to not using silence deletion.

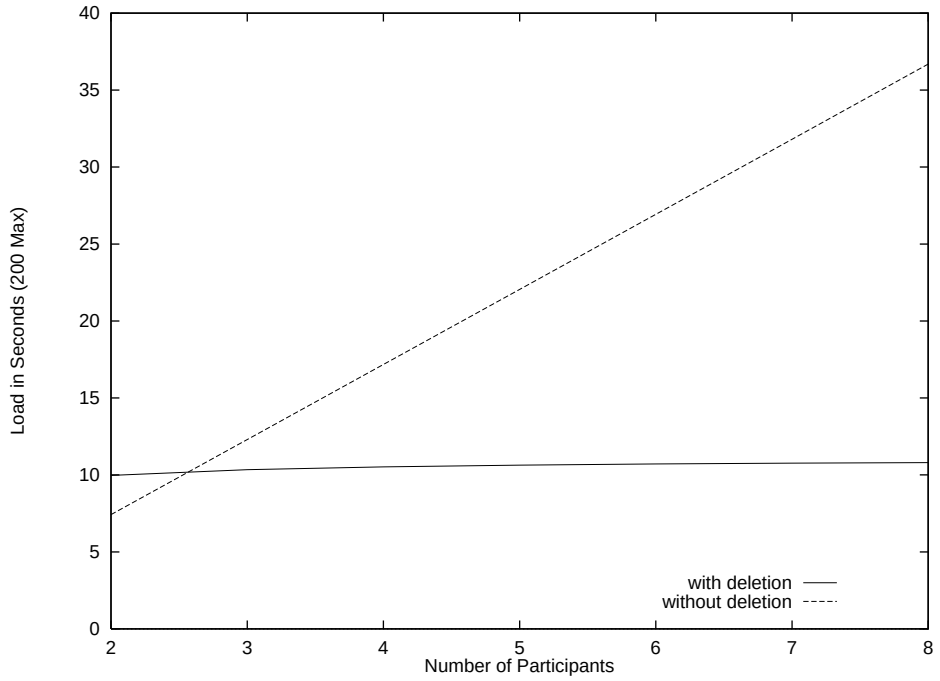

Figure 5.9: Comparison of processor loads on a Sun IPX with and without silence deletion. Only the total load processor is indicated. The maximum load is 200 seconds.

5.6.2 Increasing Silence

What happens when the conversation has more silence in it? The amount of silence will vary, as conversation characteristics such as speakers and topics change. The percentage of bytes in packets after deletion has an inconsequential effect on processor load because most cost is per-packet [10], but the percentage of packets after deletion may have a greater effect.

Figure 5.10 shows the result of our extrapolations to an increasing amount of silence. For a two person audioconference, the processor load without silence deletion is always less than the load with silence deletion. Thus, for a two person audioconference, there is never a processor benefit from using silence deletion, even when 100% of the packets are removed. This is because it takes less processor time to send a packet once than it does to remove silence from it. For 4 and 8 person audioconferences, the load without silence deletion is always greater than the load with silence deletion. Thus, for audioconferences of 3 or more, any silence deletion results in a reduction in processor load over no silence deletion. The largest part of this effect is that there is only one person speaking at a time, and even basic silence deletion algorithms delete all of the speech of silent participants.

5.6.3 Potential Audioconference Improvements

We adjust the components of our model to compare the performance benefits of four potential ways technology could be used to improve audioconference performance: faster processor, faster communication, better compression, and Digital Signal Processing (DSP) hardware.

Faster Processors What happens when the processors become faster? Faster processors decrease the per-byte times and the per-packet times of all components. Figure 5.11 shows the effects of faster processors for a 6-person audioconference. Notice that even the conversations without silence deletion use less than 5% of the processor

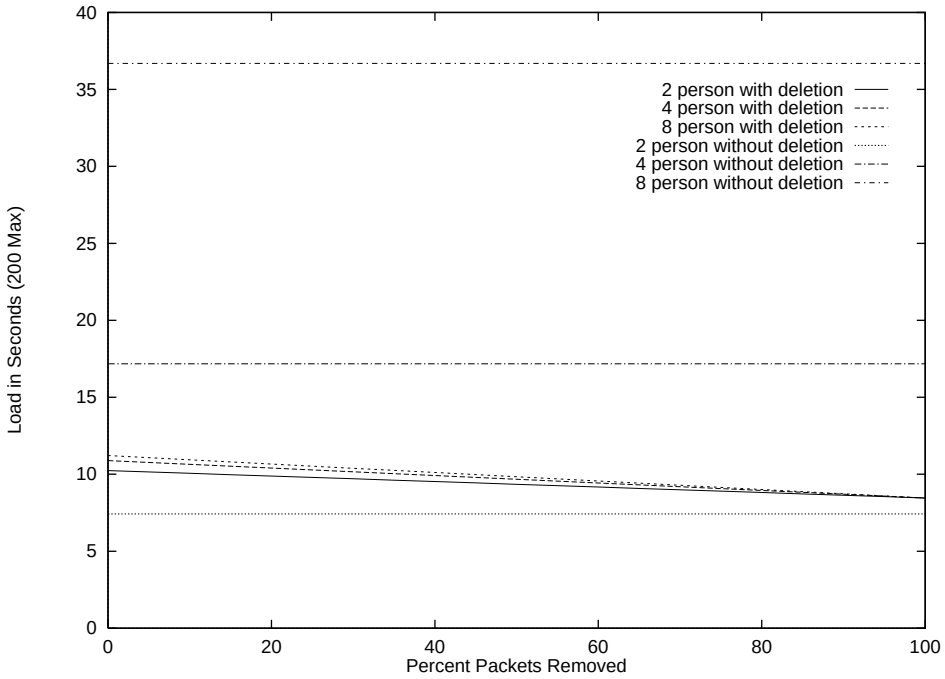

Figure 5.10: Affect of percentage of packets removed on conversations with silence deletion for 2, 4, and 8 participants. For comparison, conversations without silence deletion are also shown, depicted by the horizontal lines.

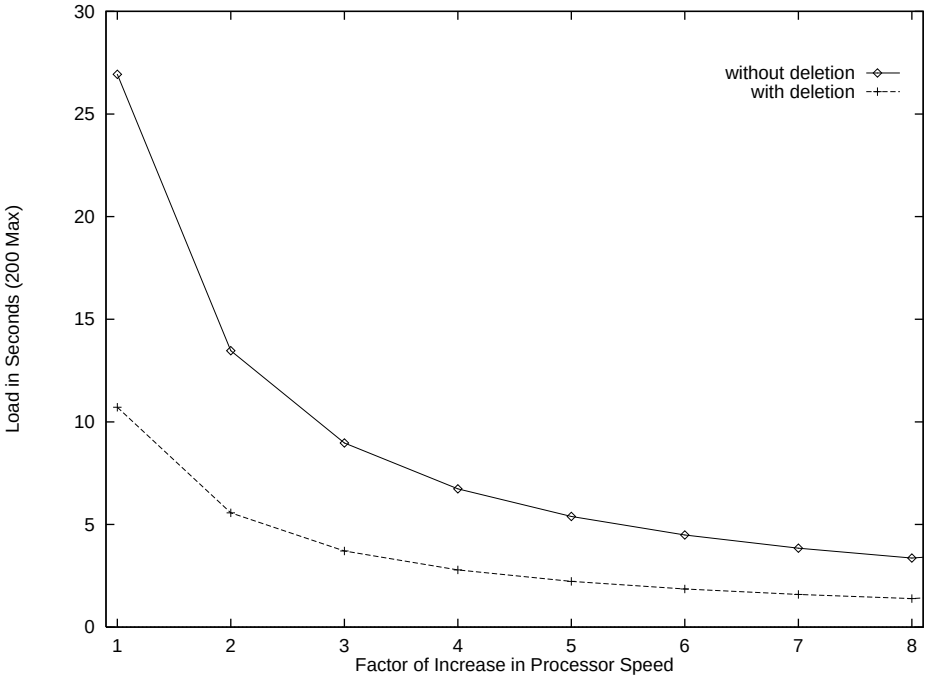

Figure 5.11: The effects of faster processors on audioconferences with and without silence deletion. The number of participants is fixed at 6. Maximum load is 200 seconds.

with an 8 times faster processor. With a fast enough processor, the load from audioconferences may be insignificant with or without silence deletion. Similar results will hold for 4 to 8 participants.

Faster Communication What happens when the network protocols become more efficient? A more efficient network protocol decreases the amount of processor time required to send and receive the sound packets. Figure 5.12 shows the predicted effect of an 8 times faster network protocol. Since the processor load of the network protocol is so slight, the audioconference processor load does not improve much despite a significant decrease in network protocol load.

What happens when multicasting is used in place of unicasting? With multicasting, only one send is required for each sound packet regardless of the number of participants. Thus, the send component decrease by a factor of N-1. However, the

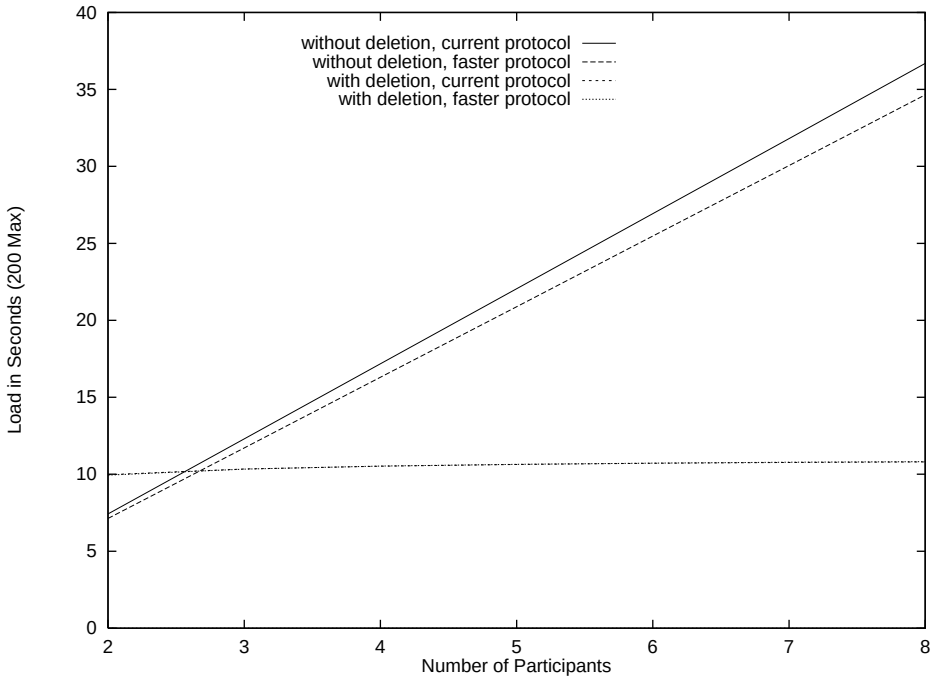

Figure 5.12: Audioconference processor load with and without silence deletion with an 8 times faster network protocol. The processor loads under the current network protocol is shown for comparison. With silence deletion, the two lines overlap. Maximum load is 200 seconds.

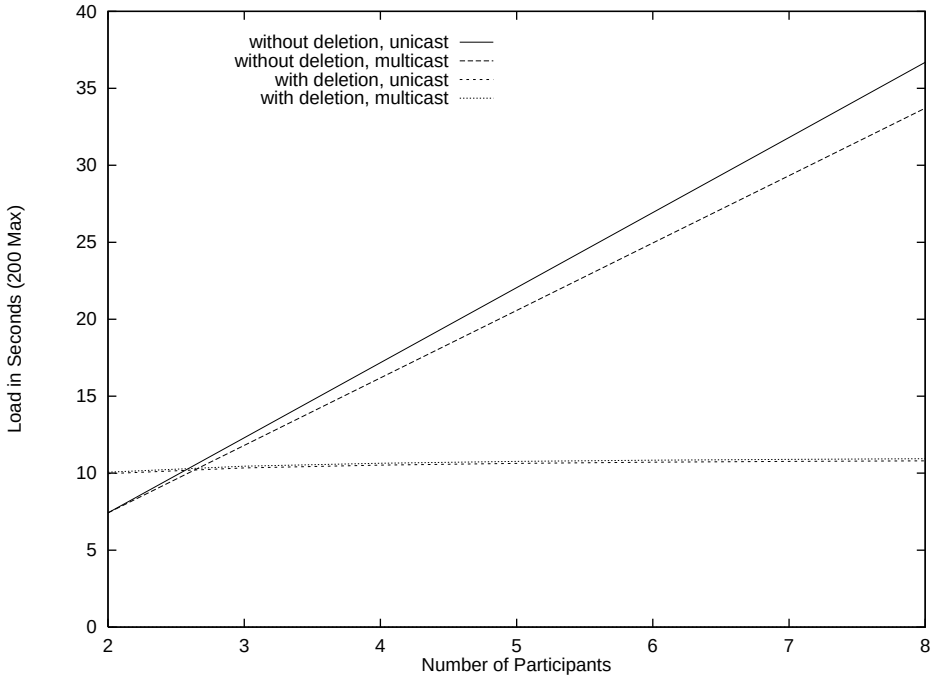

Figure 5.13: Audioconference processor load with and without silence deletion for multicasting. The loads under unicasting are shown for comparison. With silence deletion, the two lines overlap. Maximum load is 200 seconds.

receive, mix and play components remain unchanged. Figure 5.13 shows the predicted effect of multicasting on audioconferences with and without silence deletion. Since the processor load for sending packets is small compared to the mix and play components, multicast routing reduces processor load only slightly.

Better Compression How do forms of compression other than silence deletion affect audioconference processor loads? Compression reduces the packet size but not the number of packets. The silence deletion algorithms we used are fairly simple. We assume they are a lower bound on the processor complexity of most compression and decompression algorithms. We estimate the processor load of compression from the processor load of silence deletion and use a range of values for the amount audio is compressed. Audioconference loads with other forms of compression will have

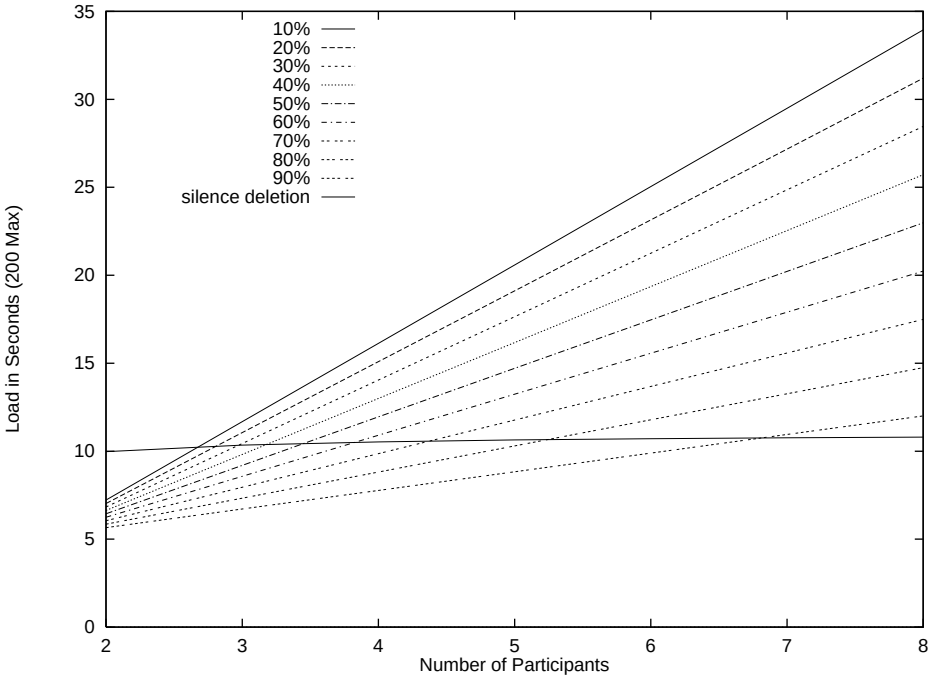

Figure 5.14: The effects of other forms of compression on audioconference processor load. For comparison, processor load with silence deletion is displayed. The maximum load is 200 seconds.

full-sized packets for recording and compressing, and reduced packets for sending, receiving, mixing and writing. In addition, they will have an additional uncompressing component.

Figure 5.14 shows the predicted effects of other forms of compression. Only algorithms that compress sound bytes by 70% or more perform better than silence deletion for more than 4 people. It appears unlikely that compression algorithms better than this can be expected to run in realtime. Audioconference processor load using compression scales worse than audioconference processor load using silence deletion in every case.

Digital Signal Processing Hardware In Section 5.6.1, we observe that the silence deletion component accounts for almost half the processor load. A DSP chip might completely remove the load of silence deletion from the processor. Such hard-

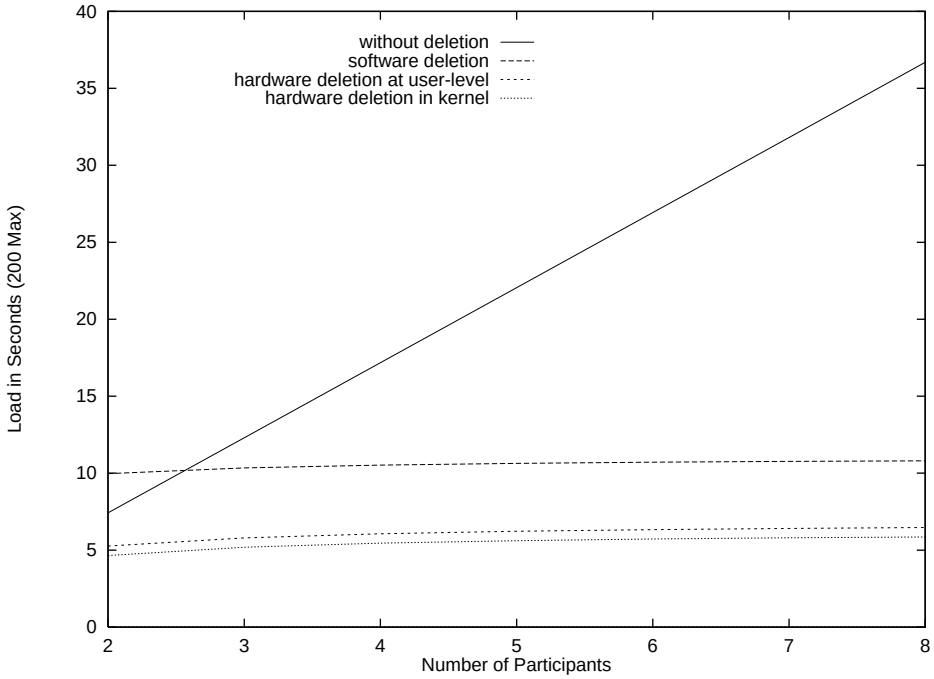

Figure 5.15: Audioconference processor load with silence deletion done in hardware, compared with software and without silence deletion. The two different hardware deletion designs, having a zero-cost user level call, or having the hardware in the kernel, are presented. The maximum load is 200 seconds.

ware silence deletion could be available at the user level through memory mapping, performing silence deletion at near-zero processor cost. Or, the chip could be in the kernel, removing silence before the user makes the record call, reducing the number of bytes in each record call. Figure 5.15 shows the predicted effects of the two hardware silence deletion designs. Hardware silence deletion always reduces processor load compared to no silence deletion. In addition, hardware silence deletion halves processor load compared to software silence deletion. There is no significant difference between the two hardware deletion implementations.

What happens when we have both compression and silence deletion in hardware? DSP chips could completely remove the load of silence deletion and compression from the processor. Since we observed that a zero-cost user level call to the DSP or the

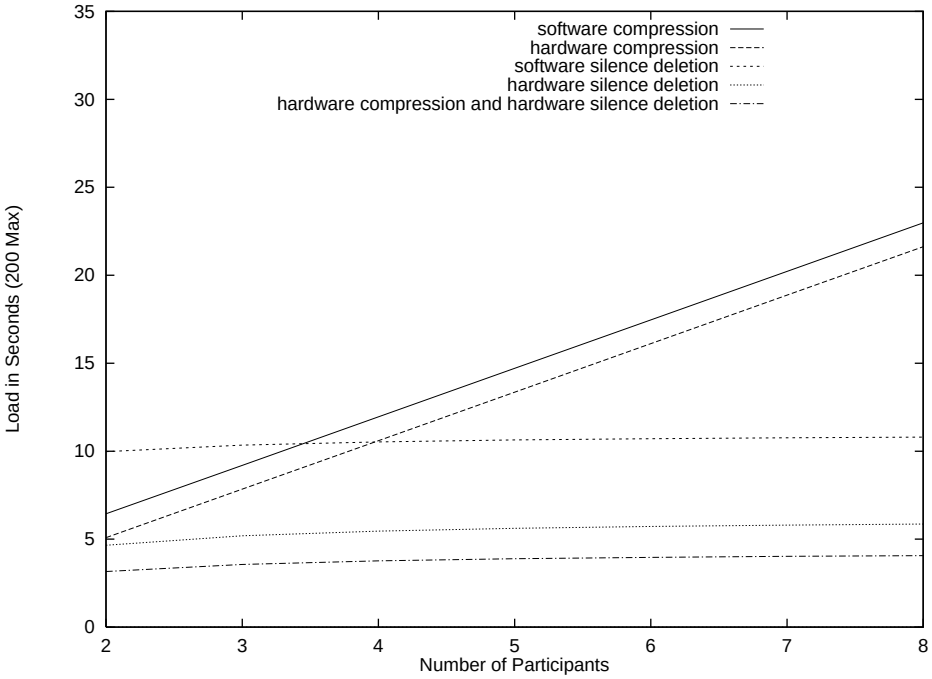

Figure 5.16: Audioconference processor load with hardware silence deletion and hardware compression. For comparison, software and hardware silence deletion and software and hardware compression are shown. All hardware implementations have the DSP chip in the kernel. Maximum load is 200 seconds.

DSP in the kernel perform similarly (Figure 5.15), we only present data for the DSP in the kernel. Figure 5.16 shows the processor load for hardware silence deletion and compression. Having both silence deletion and compression in hardware decreases total processor load by 60%.

In Section 5.6.1, we noted that in conversations without silence deletion the mixing component accounts for over 70% of the processor load. Hardware mixing can be done with a DSP chip or on a multi-channel audio-board. When mixing is done in hardware, the processor does not have to mix. The processor plays all incoming sound packets to the audio device, so the load of writing then scales with the number of participants. Figure 5.17 shows the predicted effects of hardware mixing. Hardware mixing significantly reduces non-silence deletion audioconference processor

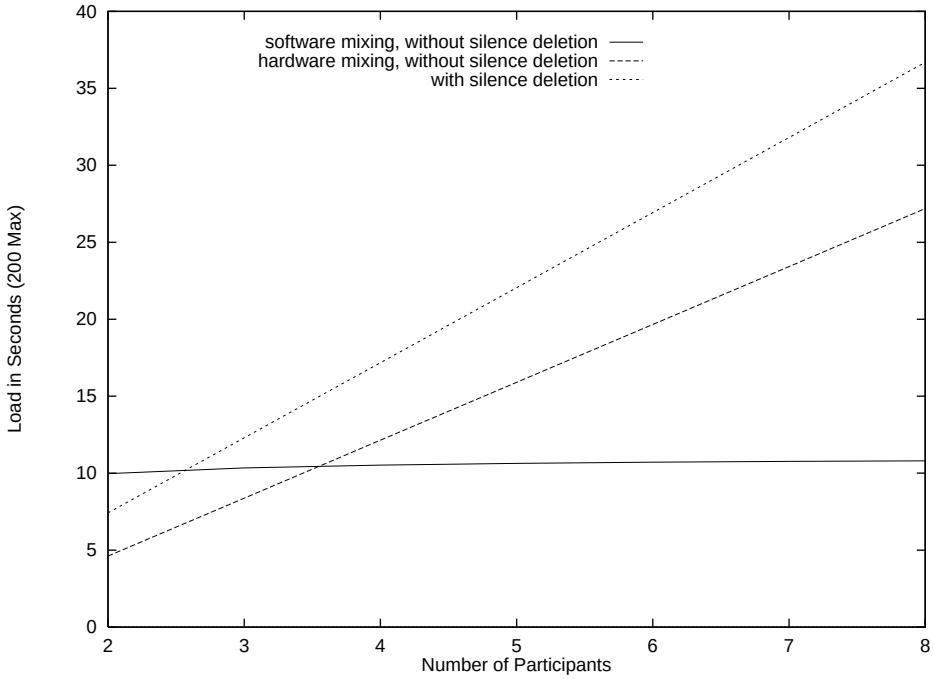

Figure 5.17: Audioconference processor load with packet combining done in hardware compared with packet combining done in software. The silence deletion conversations are unaffected by hardware mixing. Maximum load is 200 seconds.

loads. However, such loads still increase rapidly with the number of participants, becoming larger than the loads under silence deletion at four or more people.

We observed in the micro experiments (Section 5.4) that the audio component appeared comparatively large. Measurement of three classes of Suns' audio devices show that the audio device efficiency is improving disproportionately to processor speed (Figure 5.18). Perhaps the audio device will eventually be made to record and play as efficiently as sending and receiving packets (a 10 fold increase). Figure 5.19 shows the processor load for an 8 times faster audio device. For audioconferences with silence deletion, a faster audio device can reduce total Sun processor load by almost one-half.

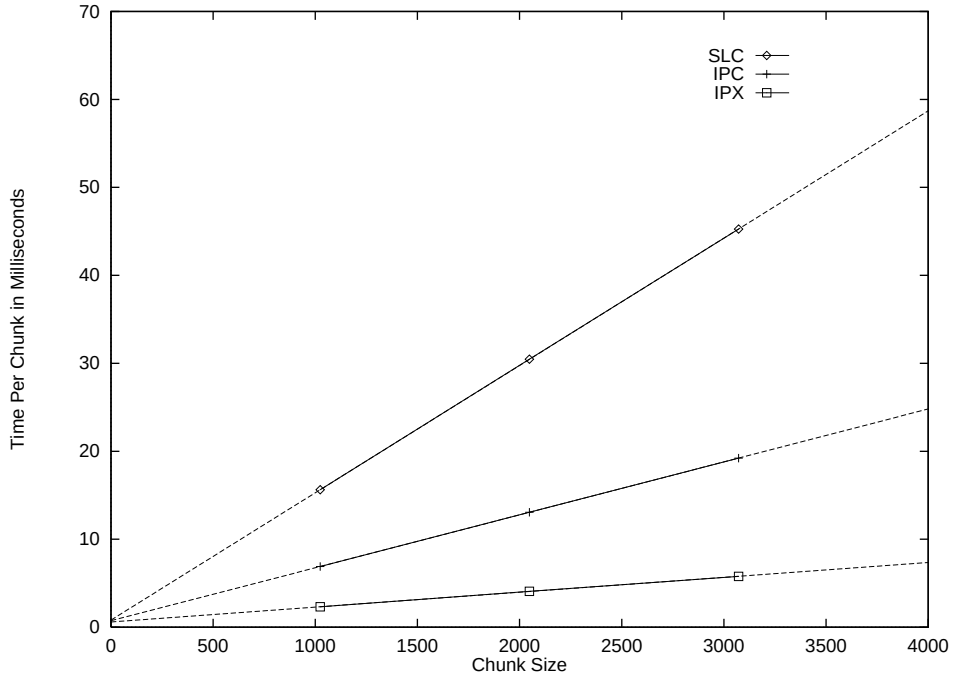

Figure 5.18: Record times per byte for different classes of machines. We would expect the slopes to scale according to the 20-25-40 MHz clock speeds but they actually scale in about a 20-50-170 ratio. The improved performance on the IPC and IPX may be due to better audio hardware. The dashed lines represent predictions beyond the measured data.

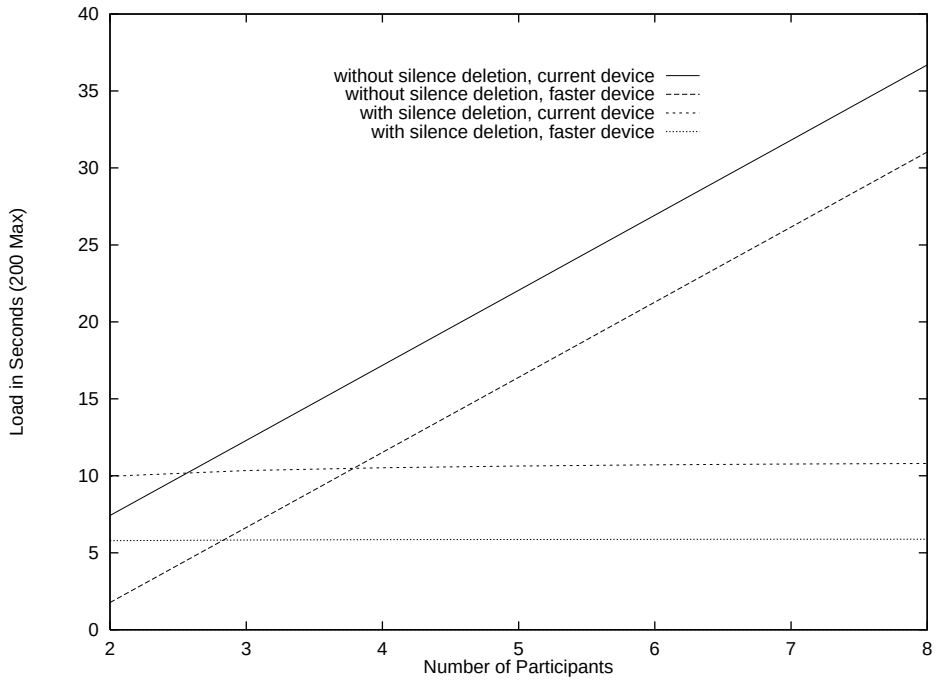

Figure 5.19: Effects of an 8 times faster audio device on processor loads. For comparison, loads under the current device are shown. Maximum load is 200 seconds.

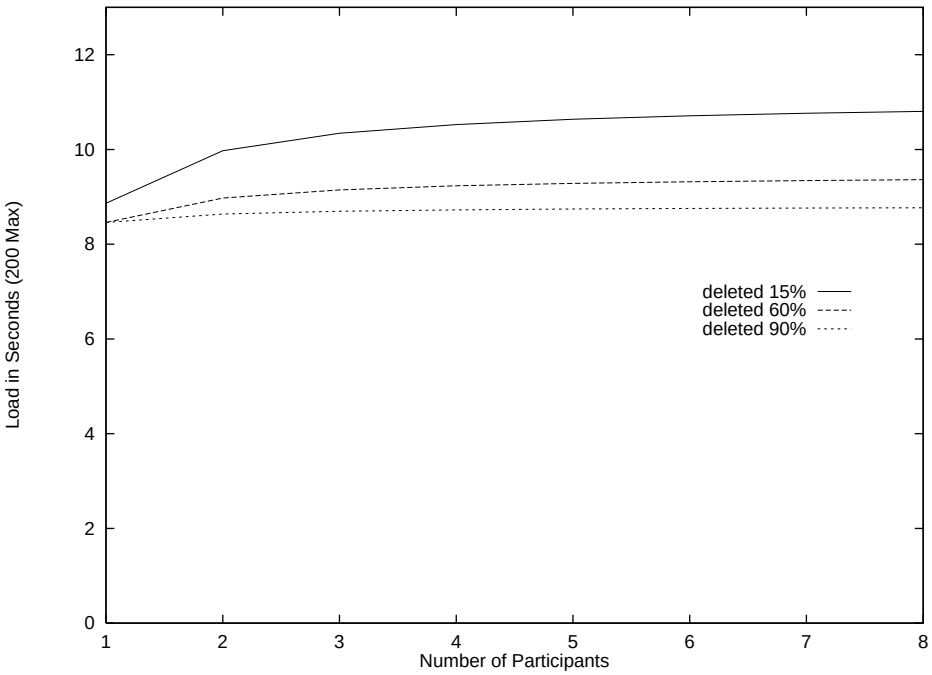

Figure 5.20: Audioconference processor load with different amounts of silence deleted. The 15% is based on a fairly poor algorithm. 60% is the maximum observed during our pilot tests. We consider 90% deletion to be an upper bound on silence deletion algorithms. The maximum load is 200 seconds.

Improved Silence Deletion Algorithms The above experiments and equations all used a silence deletion algorithm that removed only 15% of the packets. Our pilot tests suggest that, theoretically, deletion algorithms could remove as much as 60% of speech packets. Figure 5.20 shows the predicted effects of increasing the amount of silence deleted from the speaker's speech. Improving the deletion algorithm to remove as much as 90% of the packets improves the processor load by only approximately 10%. The biggest reduction in load is in removing the silence of the non-speakers; doing better than that improves processor load only a little.

5.6.4 Quality

The above predictions are all concerned with the processor load of an audioconference. However, having an adequate processor capacity to meet the predicted audioconference load does not guarantee that the user will find the audioconference acceptable. How does the user perceive the audio? What is the audioconference *quality*?

In order to apply our quality model to a audioconference under various system configurations, we must: 1) determine the region of acceptable audioconference quality; 2) predict jitter; 3) predict latency; and 4) predict data loss.

The Region of Acceptable Audioconference Quality To determine the region of acceptable audioconference quality, we need to define acceptable limits for audioconferences along each of the latency, jitter and data loss axes.

According to [40], fewer than 6% gaps in an audio stream playout and 230 milliseconds or less of delay resulted in acceptable audio quality. Audioconference quality is then the Euclidean distance from the origin to a point represented by delay milliseconds normalized over 230 and the percentage of audio gaps normalized over 6%. Any quality value under 1 is considered acceptable. But how do we predict latency and data loss?

The presence of jitter often presents an opportunity for a tradeoff among latency and data loss. Buffering, an application-level technique for ameliorating the effects of jitter, can compensate for jitter at the expense of latency. Transmitted frames are buffered in memory by the receiver for a period of time. Then, the receiver plays out each frame with a constant latency, achieving a steady stream. If the buffer is made sufficiently large so that it can hold all arriving data for a period of time as long as the tardiest frame, then the user receives a complete, steady stream. However, the added latency from buffering can be disturbing [90], so minimizing the amount of delay compensation is desirable.

Another buffering technique to compensate for jitter is to discard any late frame

at the expense of data loss. Discarding frames causes a temporal gap in the play-out of the stream. Discarding frames can keep play-out latency low and constant, but as little as 6% gaps in the playout stream can also be disturbing [75]. In the case of audio speech, the listener would experience an annoying pause during this period. In the case of video, the viewer would see the frozen image of the most recently delivered frame.

Naylor and Kleinrock describe two policies that make use of these buffering techniques: the E-Policy (for Expanded time) and the I-Policy (for late data Ignored) [75]. Under the E-policy, frames are never dropped. Under the I-policy, frames later than a given amount are dropped. Since it has been observed that using a strict E-Policy tends to cause the playout latency to grow excessively and that dropping frames occasionally is tolerable [18, 113], we use the I-Policy as a means of examining needed jitter compensation for a multimedia stream.

The I-policy leads to a useful way to view the effects of jitter on a multimedia stream. Figure 5.21 depicts the tradeoff between dropped frames and buffering as a result of jitter. We generated the graph by first recording a trace of interarrival times. We then fixed a delay buffer for the receiver and computed the percentage of frames that would be dropped. This represents one point in the graph. We repeated this computation with buffers ranging from 0 to 230 milliseconds to generate the curved line. The graph can be read in two ways. In the first, we choose a tolerable amount of dropped frames (the horizontal axis), then follow that point up to the line to determine how many milliseconds of buffering are required. In the second, we choose a fixed buffer size (the vertical axis), then follow that point over to the line to determine what percent of frames are dropped. In Figure 5.21, if we wish to restrict the amount of buffering to 100 ms, then we must drop about 2% of the frames since that is how many will be more than 100 ms late, on average. For an 8000 KBps audio stream consisting of 6.25 1280-byte frames per second, this equates to dropping one frame every 8 seconds. On the other hand, if we wish to not drop any frames, we

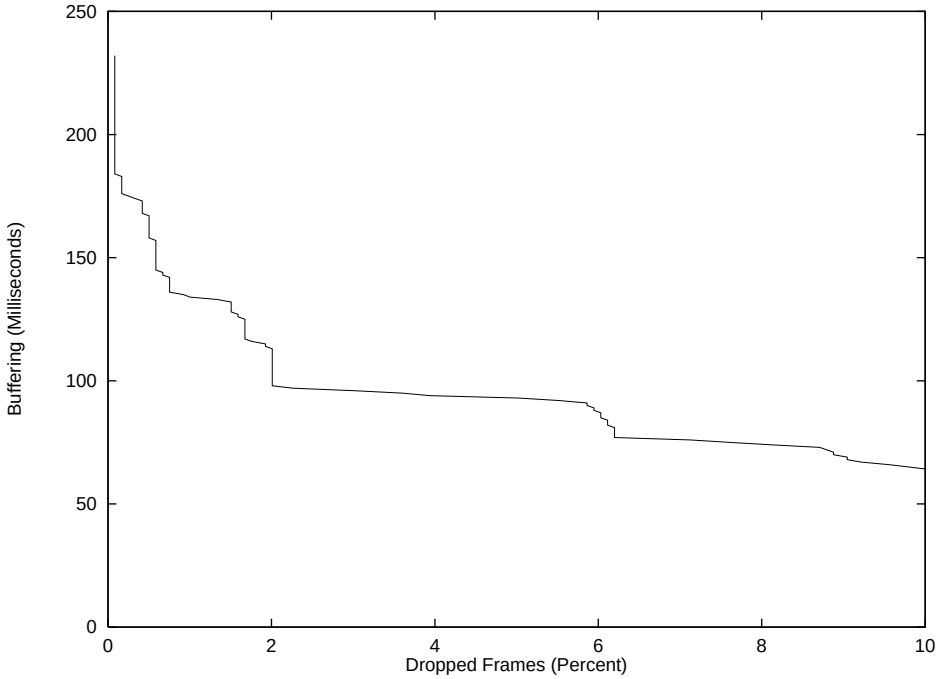

Figure 5.21: Jitter Compensation. This picture depicts the amount of buffering needed for a given number of dropped frames. The horizontal axis is the percentage of dropped frames. The vertical axis is the number of milliseconds of buffering needed.

have to buffer for over 200 ms.

How much buffering should we choose? We normalize the axes from 6% gaps to 230 ms buffering. The best quality value in the jitter compensation curve is the closest point to the origin along the curve. In practice, there is no way for an application to determine exactly what buffer will give it this point. However, there are heuristics that consider past jitter in determining the most appropriate buffer size for the future jitter[41]. We assume a heuristic can provide an application with a near-optimal buffer size. We would like to know how much latency is added from buffering at this point. It seems natural to assume that as the area under the jitter compensation curve gets larger, the amount of buffering at the closest point along the curve gets larger. We hypothesize that there is a strong correlation between the area

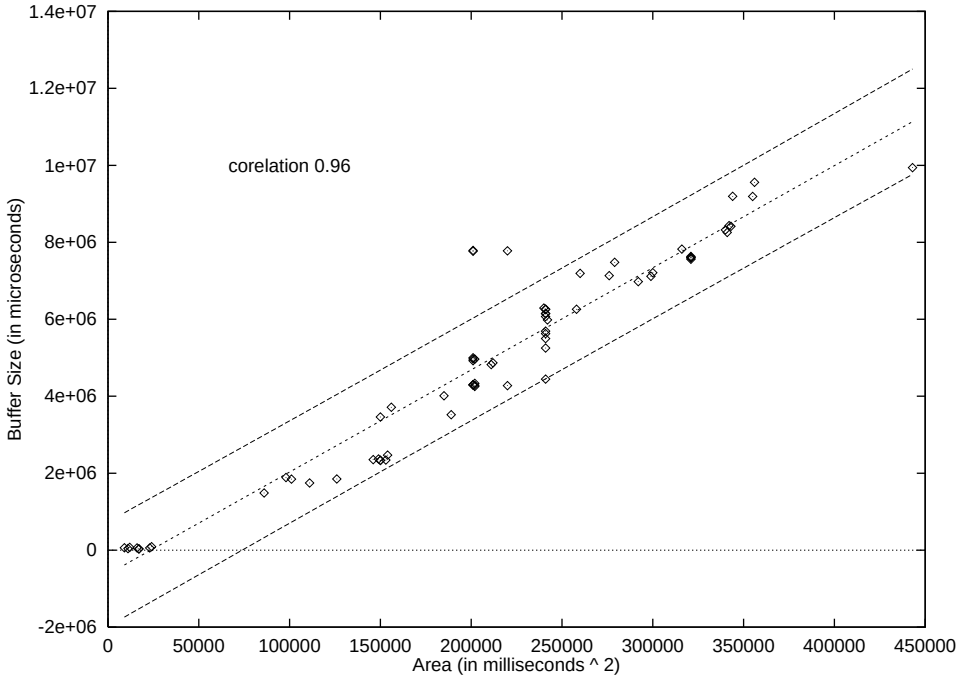

Figure 5.22: Jitter Compensation Area versus Buffer Size. The horizontal axis is the area under the jitter compensation curve. The vertical axis is the buffer size in microseconds. The points are each a separate experiment run. The middle line is the least squares line fit. The outer two lines form a 95% confidence interval around the line. The correlation coefficient is 0.96.

under the jitter compensation curve and the buffer size. If this hypothesis is true, we can determine the latency attributed to buffering from the jitter compensation area. We tested our hypothesis by generating jitter compensation curves for all data points from the experiments detailed in Section 3.4, Section 3.5 and Section 3.6. We then computed the area under each curve. We computed the buffer size by normalizing the axes as described in Section 2.3 and finding the lowest Euclidean distance to the origin along the curve. We plotted buffer size versus area and computed the correlation coefficient. Figure 5.22 depicts these results. There is a high correlation between jitter compensation area and buffer size.

We can predict the optimal amount of buffering if we know the area under the jitter compensation curve. How can we determine the area under the jitter compen-

sation curve? As the amount of jitter experienced by the system gets larger, more buffering should be required to alleviate the effects of jitter and more gaps should appear in the multimedia stream. Thus, the area under the jitter compensation curve should get larger as jitter increases. We hypothesize that there is a strong correlation between the area under the jitter compensation curve and the variance in the packet interarrival times. If this hypothesis is true, then we can predict the area under the jitter compensation curve from the amount of jitter. Then, we can predict the buffer size from the jitter compensation area. We tested our hypothesis by computing jitter from all data points from the experiments detailed in Section 3.4, Section 3.5 and Section 3.6. We then compared these jitter values to the areas under the jitter compensation curves from our previous hypothesis. We plotted area versus jitter and computed the correlation coefficient. Figure 5.23 depicts these results. There is a high correlation between jitter and compensation curve area. Note that streams with the most jitter (the right-most points) are all outside the confidence interval curves. Further work might be needed to determine the relationship between high-jitter multimedia streams and the area under the jitter compensation curve. Fortunately, our predictions deal almost exclusively with streams with far less jitter than the streams represented by those right-most points.

From the above graphs, we have:

$$Buf. = 26.5 \times Area - 624,000$$

$$Area = 0.000198 \times Jitter + 1,590,000$$

Substituting the equation for Area into the equation for Buffer, we have:

$$Buf. = 0.00525 \times Jitter + 41,500,000$$

This last equation allows us to accurately estimate the optimal buffer size for an application given the amount of jitter.

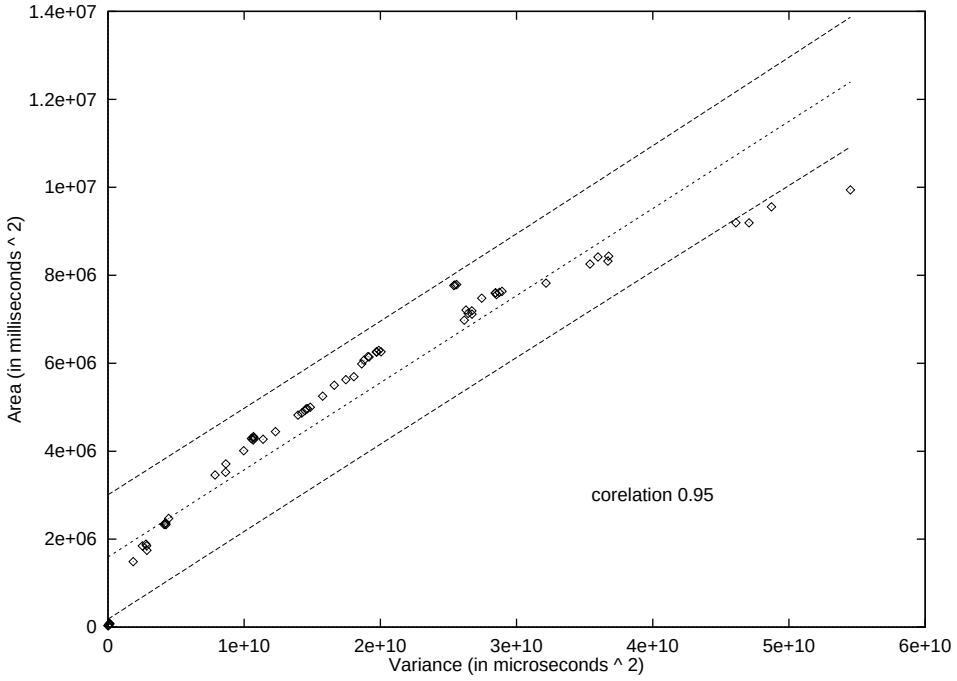

Figure 5.23: Jitter versus Jitter Compensation Area. The horizontal axis is the jitter (variance in packet interarrival times). The vertical axis is the area under the jitter compensation curve. The points are each a separate experiment run. The middle line is the least squares line fit. The outer two lines form a 95% confidence interval around the line. The correlation coefficient is 0.95.

Predicting Jitter From our results in Sections 3.4 and 3.6, we know the relationship between load and jitter for faster processors and networks. We hypothesize that a high network load along with a high processor load increases jitter by the sum of the jitter from the network and processor. While perhaps seeming obvious, our hypothesis will be false if processor jitter and network jitter are not independent. If some of the jitter attributed to the processor is actually due to jitter from the network and/or some of the jitter attributed to the network is actually due to jitter from the processor, adding the jitter from each component will result in more jitter than the system actually experiences. However, if our hypothesis is true, we can then add the predicted jitter from each component in predicting jitter for systems with both components.

We tested our hypothesis with an experiment. We first computed the jitter we would predict on a system with a loaded processor and loaded network. This prediction is based on the amount of jitter attributed from the processor as obtained in results from Section 3.4 and from the network as obtained in results from Section 3.6. We next experimentally measured the jitter with a loaded processor and a loaded network for each of the Ethernet, Fibre Channel and HIPPI networks. We then compared the predicted results to the actual results. Table 5.4 depicts this comparison. The predicted jitter values are within 7% of the actual jitter values. It seems appropriate to add the jitter attributed to processor load alone with the jitter attributed to network load alone to predict the jitter attributed to processor load and network load together.

Predicting Latency We can predict the amount of latency from the jitter compensation buffer by using predictions on the amount of jitter. In addition to the buffering latency, there is the additional latency from the sender processing, the network transmitting and the receiver processing. From our micro experiments, we know the latency from recording and playing audio and the latency attributed to sending

Network	Processor	Network	Predicted	Actual
Ethernet	564.63	411.37	976.00	913.72
Fibre Channel	590.60	58.33	648.93	641.14
HIPPI	501.15	28.62	529.77	538.41

Table 5.4: Predicted versus Actual Jitter. This table depicts total jitter predictions based on the amount of jitter contributed by the processor and network. "Processor" is the amount of jitter contributed by a loaded processor. "Network" is the amount of jitter contributed by a loaded network. "Predicted" is the sum of the "Processor" and "Network" values. "Actual" is the amount of jitter measure during our experiments.

and receiving packets [22]. We can compute the latency from the network based on the frame size and network bandwidth. To predict the total latency, we add the latencies from: recording the audio frame; sending the audio frame to the client; receiving the audio frame from the receiver; buffering in the jitter compensation curve; and playing the audio frame.

Predicting Data Loss In order to predict data loss, we need to identify what form data loss may take and when data loss may occur. In general, data loss can take many forms such as reduced bits from encoding, dropped frames and lossy compression. For an audioconference, we assume data loss only in the form of dropped frames or reduced frame rate. We assume data loss under three conditions:

- *Voluntary.* As described in Section 5.6.4, an application may chose to discard late frames in order to keep playout latency low and constant and we assume the audioconference uses heuristics to discard enough frames to achieve the best quality.

- *Saturation.* When either the network or the processor do not have sufficient capacity to transmit data at the required frame rate, data loss occurs. For

example, if the network has a maximum bandwidth of 5 Mbps and the audioconference required 10 Mbps there will be a 50% data loss. We can compute when systems reach capacity based on our previous work measuring processor capacities [22, 24] and theoretical network bandwidths [27, 119, 45]. Although the actual network bandwidth can often be less than the theoretical network bandwidth [106, 80], using the theoretical bandwidth gives us an upper bound on network utilization. If future work measures the actual network bandwidth appropriate for an application, in can be used in place of the theoretical network bandwidths we use.

- *Transmission Loss.* In our previous experiments, we found that typically about 0.5% packets on the average are lost when the network is running under maximum load [24]. We assume a maximum lost data rate of about 0.5% due to network transmission.

Predicting Quality At last! We have built and tested the accuracy of an experiment-based model that will allow us to explore audioconference quality under different system configurations. We can quantify how effectively today's computer systems support multi-person audioconferences. We can predict when today's systems will fail due to too many users or too much load on the processors or networks. We can see how much using real-time priorities will help audioconference quality. We can evaluate the benefits of expensive high-performance processors and high-speed networks before installing them. We can even investigate possible performance benefits from networks and processors that have not yet been built. Let's go exploring!

We predict application quality for three scenarios: 1) high-performance processors and high-speed networks; 2) increasing users; and 3) increasing load.

High-Performance Processors and High-Speed Networks Our results in Sections 3.4 and 3.6 showed that both high-performance processors and high-speed net-

works reduce jitter. However, which reduces jitter more? And more importantly, which improves application quality more?

We assume we have twenty audioconference participants. In Section 5.6.4, we use our model to evaluate quality for a variable number of users, but here we evaluate a possible audioconference configuration that has interesting quality predictions. We compute quality under two different scenarios. In the first, processor load remains constant while the network bandwidth increases. In the second, network bandwidth remains constant while processor power increases. Figure 5.24 shows these predictions. For twenty users, increasing the processor power to a SPECint92 of 40 or greater results in acceptable audioconference quality. At no time does increasing the network bandwidth result in an acceptable quality. In this scenario, we conclude that processor power influences audioconference quality more than does network bandwidth.

Number of Users While today's computer systems may struggle to support even twenty audioconference participants, tomorrow's processor improvements promise to support more and more users. But how many more? How do more and more audioconference users affect application quality? Figure 5.25 depicts the predicted effects of increasing users on audioconference quality. We predict audioconference quality for three different audioconference configurations: a low-end workstation with a typical network (Sun IPX and Ethernet), a mid-range workstation with a fast network (Sun Sparc 5 and Fibre Channel), and a high-performance workstation with a high-speed network (DEC Alpha and HIPPI). As we saw in Section 5.6.4, today's typical workstations and networks have trouble supporting a small number of audioconference participants. However, workstations such as Sun Sparc 5s connected by fast networks such as a Fibre Channel can support up to 30 users. Very high-performance workstations such as a DEC Alpha connected by high-speed networks such as a HIPPI can support over 70 users.

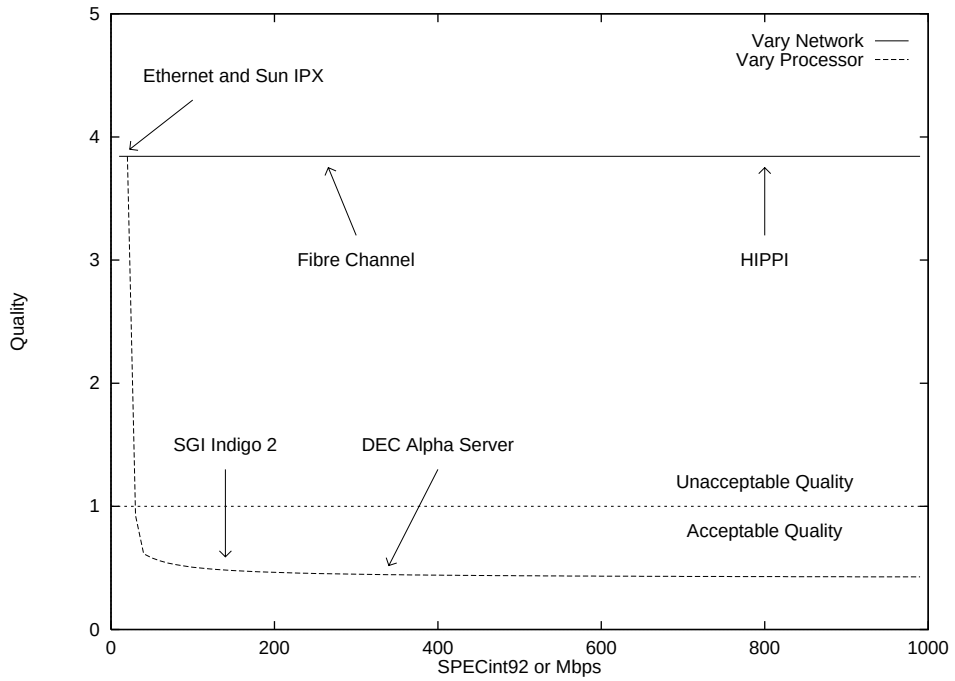

Figure 5.24: Audioconference Quality versus Processor or Network Increase. The horizontal axis is the SPECint92 power of the workstation or the network Mbps. The vertical axis is the predicted quality. There are two scenarios depicted. In the first, the processor power is constant, equivalent to a Sun IPX (SPECint92 = 22), while the network bandwidth increases. This is depicted by the solid curve. In the second scenario, the network bandwidth is constant, equivalent to an Ethernet (10 Mbps), while the processor power increases. This is depicted by the dashed curve. The horizontal line marks the limit between acceptable and unacceptable audioconference quality.

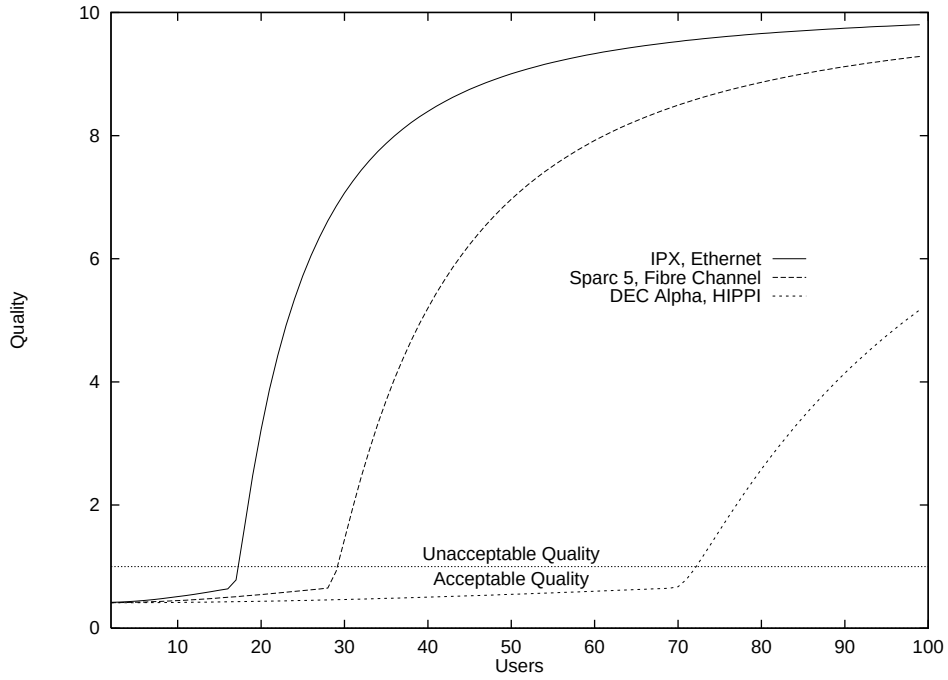

Figure 5.25: Audioconference Quality versus Users. The horizontal axis is the number of users. The vertical axis is the predicted quality. There are three scenarios depicted. In the first, the processors are Sun IPXs connected by an Ethernet. In the second, the processors are Sun Sparc 5s connected by a Fibre Channel. In the third, the processors are DEC Alphas connected by a HIPPI. The horizontal line marks the limit between acceptable and unacceptable audioconference quality.

Processor and Network Load Audioconferences with many users are resource intensive, forcing processors and networks to run at a heavy loads. In addition, audioconference streams are often integrated into larger distributed multimedia applications. In the past, applications have tended to expand to fill (or surpass) available system capacity. As system capacities increase, audioconference users will demand higher frame rates and better resolution, making heavy-load conditions likely in the future. We predict the effects of increasing load on audioconference quality.

Figure 5.26 depicts the predicted effects of load on audioconference quality. There are three classes of systems depicted. A traditional system has Sun IPXs connected by an Ethernet. A mid-range system has Sun Sparc 5 connected by a Fibre Channel. A high-end system has DEC Alphas connected by a HIPPI. The predictions for audioconference quality are almost identical for the three systems. We saw in Section 5.6.4 that the processor is more crucial than network for audioconference quality. Increasing processor load has a larger effect on decreasing audioconference quality than does improving the network speed and processor power.

Figure 5.26 also depicts Sun IPXs connected by an Ethernet but using real-time priorities instead of default priorities, shown by the bottom line. With real-time priorities, audioconference quality does not suffer from increased jitter from the processor as processor load increases. For conditions of increasing load, real-time priorities have a greater effect on improving quality than do faster processors and faster networks.

5.7 Summary

Our goal is to identify improvements that reduce audioconference processor load. We developed a model for audioconference processor load and presented experimentally-based analysis on the effects on processor load from improving each component of the model. In addition, we explored the affects of system improvements on audioconference quality.

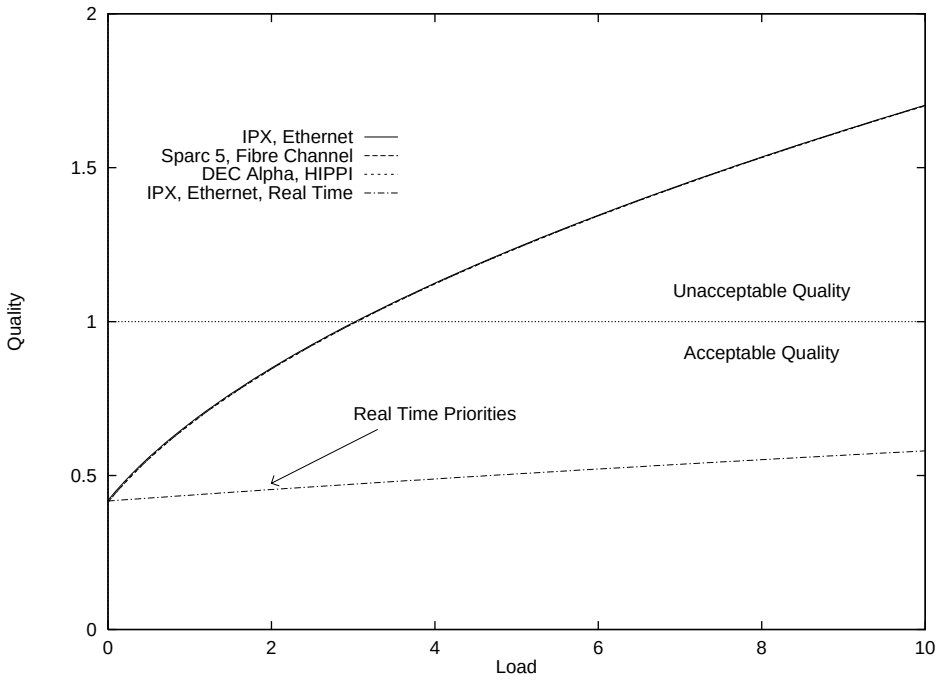

Figure 5.26: Audioconference Quality versus Load. The horizontal axis is the processor load. The vertical axis is the quality prediction. There are four system configurations depicted. In the first, the processors are Sun IPXs connected by an Ethernet. In the second, the processors are Sun Sparc 5s connected by a Fibre Channel. In the third, the processors are DEC Alphas connected by a HIPPI. In the fourth, the processors are again Sun IPXs connected by an Ethernet, but they are using real-time priorities instead of default priorities. The upper horizontal line marks the limit between acceptable and unacceptable audioconference quality.

Based on our analysis of individual components, silence deletion appears to improve the scalability of audio more than all other improvements. This result holds even when using compression and even for ten times faster processors, networks and Digital Signal Processing (DSP) hardware. Further, the simple silence deletion algorithms we studied achieve 90% of the potential benefit from silence deletion for audioconference processor load. Simple silence deletion algorithms are sufficient because the biggest reduction in processor load from silence deletion is in removing the silence of the non-speakers which even the basic algorithms do well.

Sending and receiving packets are already relatively low-cost, so reducing the cost further has relatively low benefit. Better network protocols, faster networks and even multicasting reduce load only slightly, with or without silence deletion. Techniques based on DSP hardware alone do not scale as well as silence deletion alone. However, DSP based silence deletion and compression together scale better than any other technique. Silence deletion done in a DSP chip can reduce processor load by 50%. Compression and silence deletion in a DSP chip can reduce processor load by an additional 30%.

In applying our quality model we found that for some configurations, high-performance processors will improve audioconference quality more than will high-speed networks. However, the typical Ethernet network is the bottleneck in application quality as the number of users increases. Thus, even high-performance processors will not sufficiently improve application quality as the growing number of users saturate existing networks. The traditional Ethernet network will become the bottleneck in approximately a year when workstation performance has doubled and may even be the bottleneck for networks of today's high-performance workstations.

When multimedia applications are running under conditions of increasing load, real-time priorities have a greater effect on improving quality than do faster processors and faster networks. In fact, hardware improvements alone will not reduce jitter enough to eliminate the need for application buffering techniques. However,

for multimedia on a Local Area Network (LAN), real-time priorities can reduce jitter enough to eliminate the need for application buffering today. On a Wide Area Network (WAN) especially the Internet, real-time priorities will not be available on all routers, reducing the effectiveness of real-time priorities in reducing. In this case, buffering techniques may still be needed.

Chapter 6

Flying through the Zoomable Brain Database

6.1 Overview

Gradually, in laboratories around the world, neuroscientists from diverse disciplines are exploring various aspects of brain structure. Since there is so much research to be done on the nervous system, neuroscientists must specialize. An undesirable result of this specialization is that it is difficult for individual neuroscientists to investigate how their results fit together with results from other scientists. Moreover, they sometimes duplicate research efforts since there is no easy way to share information.

To enhance the work of neuroscientists, we propose a zoomable database of images of the brain tissue. We begin with the acquisition of 3-d structural maps of the nervous system using high-field, high-resolution magnetic resonance imaging (MRI). The MR images show the entire brain in a single dataset and preserve spatial relationships between structures within the brain. However, even high resolution MRI cannot show individual cells within the brain. We therefore anchor confocal microscope images to these 3-d brain maps. Because of their higher resolution, the confocal images are of smaller regions of the brain. Many such images are montaged into larger images

by aligning cellular landmarks between images.[1] These montages are then aligned with structural landmarks from the MR images, so the high-resolution images can be anchored accurately within the MR image. In addition to the image data, other types of brain data can be linked to the 3-d structure.

The brain database will be embedded in three dimensions. The user starts a typical investigation by navigating through the MR images in a coarse 3-d model of the brain to a site of interest. The user then zooms to higher resolution confocal images embedded in the MRI landscape. This real-time navigating and zooming we call "flying."

The scientific value of the data and the distributed nature of the database impose a series of user-level requirements that the flying interface should satisfy:

- *Wide Spread Use.* We estimate the number of users who may be interested in using the database to be 10,000 (half the number of members of the Society for Neuroscience in 1994), the average day to be 10 work hours, the average work week to be 5 days and the average work month to be 4 weeks. We can predict the average number of simultaneous database users based on some possible usage amounts:

Uses the Database	Simultaneous Database Users
1 hour/day	1000
1 hour/week	200
1 hour/month	50

- *Twenty-four Bit Color.* As often as possible, the database will express experimental data in its purest form, so expensive raw data can be analyzed and reanalyzed by researchers worldwide. Confocal microscope images include up

[1] We have created a WWW server that includes an example of a montage of confocal micro-graphs at http://www.cs.umn.edu/Research/neural/

to three layers of eight bit gray-scale, with each layer representing one brain chemical. The final images include 24 bits of color, all important to the scientists who view it.

- *Three Dimensions.* Since the anchoring MR images are three dimensional, flying must be allowed in all three dimensions. This necessitates computing two dimensional frames from the three dimensional images. The computation can take place by two different methods:

  Remote Flying: In remote image processing the server does the image computation and transfers only a 2-d frame to the client. We estimate Remote Flying will require sending 24 Mbits (a mega-pixel) of data per frame. Remote flying shifts the image processing load from the client to the server.

  Local Flying: In local image processing the server transfers the 3-d data and the client does the image computation. We estimate Local Flying will require sending 384 Mbits (the 3-d region of the brain) of data per frame. Local flying shifts the image processing load from the server to the client.

- *Smooth Navigation.* Flying must be very smooth even over a varied, non-dedicated network. This necessitates a motion picture quality rate of 30 frames per second, and jitter control to compensate for network variance.

- *Adaptability.* Flying must adapt to a wide variety of resources, including varying processor and disk types and non-dedicated variable bandwidth networks.

This Chapter is organized as follows: Section 6.2 of this section describes related work in scientific visualization, neuroscience, compression, network performance and disk performance. Section 6.3 introduces our model of an audioconference. Section 6.4 details micro experiments that measure the processor load of each component. Section 6.5 analyzes the experiments and projects the results to future environments. And Section 6.6 summarizes the important contributions of this section.

6.2 Related Work

6.2.1 Scientific Visualization

Hibbard, Paul, Santek, Dyer, Battaiola and Voidrot-Martinez designed an interactive, scientific visualization application [57]. They were seeking to bridge the barrier between scientists and their computations and allow them to experiment with their algorithms.

Elvins described five foundation algorithms for volume visualization in an intuitive, straight-forward manner [36]. His paper included references to many other papers that give the algorithm details.

Singh, Gupta and Levoy showed that shared-address-space multiprocessors are effective vehicles for speeding up visualization and image synthesis algorithms [107]. Their article demonstrated excellent parallel speedups on some well-known sequential algorithms.

We investigate performance of an application that will use techniques developed in other scientific visualization applications. In particular, we use performance results from Singh, et al. and may implement some of the algorithms that Elvins describes.

6.2.2 Neuroscience

Carlis, et al, present the database design for the Zoomable Brain Database [15]. We have developed data models for novel neuroscience. One model focuses on MR and Confocal microscopy images, the connections between them, and the notions of macroscopic and microscopic neural pathways in the brain. We have implemented this "connections" data model in the Montage DBMS, and have populated the schema with neuroscience data.

Kandel, Schwartz and Jessel discussed in detail the fundamentals behind neural science [72].

Slotnick and Leonard had an extensive photo atlas of an albino mouse forebrain

[108]. The ideas of a zoomable, digitized rat brain came from atlases such as this one.

The Zoomable Brain Database is based on neuroscience fundamentals. At its heart is a digital brain atlas. Our work presents network, processor and disk performance analysis in accessing the digital images.

6.2.3 Compression

Wallace presented the Joint Photographic Experts Group (JPEG) still picture standard [122]. He described encoding and decoding algorithms and discusses some relations to the Motion Picture Experts Group (MPEG).

Patel, Smith and Rowe designed and implemented a software decoder for MPEG video bit-streams [92]. They gave performance comparisons for several different bit rates and platforms. They claimed that memory bandwidth was the bottleneck for the decoder, not processor speed. They also gave a cost analysis of three different machines for doing MPEG.

We expand compression research by providing careful processor load measurements of JPEG compression and decompression on a Sun IPX. In addition, we predict the effects of compression on network, processor and disk performance.

6.2.4 Network Performance

Hansen and Tenbrink investigated gigabyte networks and their application to scientific visualization [53]. They explored various system topologies centered around the HIPPI switch. They analyzed network load under some possible visualization applications and hypothesize on the effects of compression.

Lin, Hsieh, Du, Thomas and MacDonald studied the performance characteristics of several types of workstations running on a local Asynchronous Transfer Mode (ATM) network [80]. They measured the throughput of four different application programming interfaces (API). They found the native API achieves the highest throughput, while TCP/IP delivers considerably less.

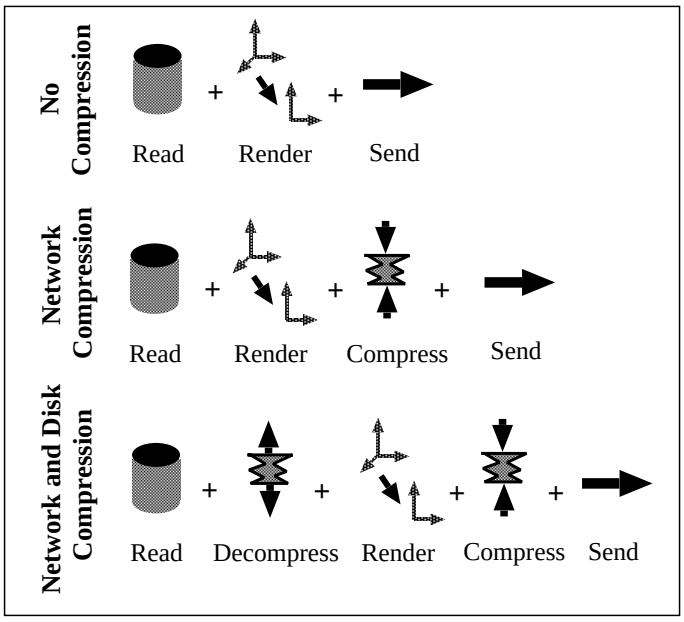

Figure 6.1: Flying Model. Our model of the flying software used by the server, that includes reading, rendering and sending. Compression reduces the data before sending. Decompression expands the data before rendering.

6.2.5 Disk Performance

Ruwart and O'Keefe describe a hierarchical disk array configuration that is capable of sustaining 100 MBytes/second transfer rates [101]. They show that virtual memory management and striping granularity play key roles in enhancing performance.

6.3 Model

We have three different models for the Flying software depicted in Figure 6.1:

1. *No Compression.* Flying with no compression has three processor load components: *read* is the processor load for reading the image from the disk; *render* is the processor load for computing the 2-d frame from the 3-d image; and *send* is the processor load for sending the frame to the user.

2. *Network Compression.* Flying with network compression has four processor load components: *read* is the processor load for reading the image from the disk; *render* is the processor load for computing the 2-d frame from the 3-d image; *compress* is the processor load for compressing the frame; and *send* is the processor load for sending the compressed frame to the user.

3. *Network and Disk Compression.* Flying with network compression and disk compression has five processor load components: *read* is the processor load for reading the compressed image from the disk; *decompress* is the processor load for decompressing the image; *render* is the processor load for computing the 2-d frame from the 3-d image; *compress* is the processor load for compressing the frame; and *send* is the processor load for sending the compressed frame to the user.[2]

6.4 Micro Experiments

We can predict the processor throughput required for flying by using the processor load for individual components as in Section 5.4. We obtain the flying throughput for a 40 MHz Sun IPX to obtain a baseline for predictions to faster machines. We re-use the appropriate micro-experiment results from Section 5.4 and [21] that were used for the audioconference application and run experiments to measure the components new to the flying application. The components we re-use are: `send`, `receive`, `read` and `write`. The new components are: `compress` and `decompress`.

We obtained the processor load for doing JPEG compression and decompression. We modified the source code for *cjpeg* and *djpeg*[3] to perform compression and de-

[2] A clever image computation algorithm might do the calculation on the compressed images [110].

[3] The "official" archive site for this software is ftp.uu.net (Internet address 137.39.1.9 or 192.48.96.9). The most recent released version can always be found there in directory graphics/jpeg. The particular version we used is archived as jpegsrc.v4.tar.Z.

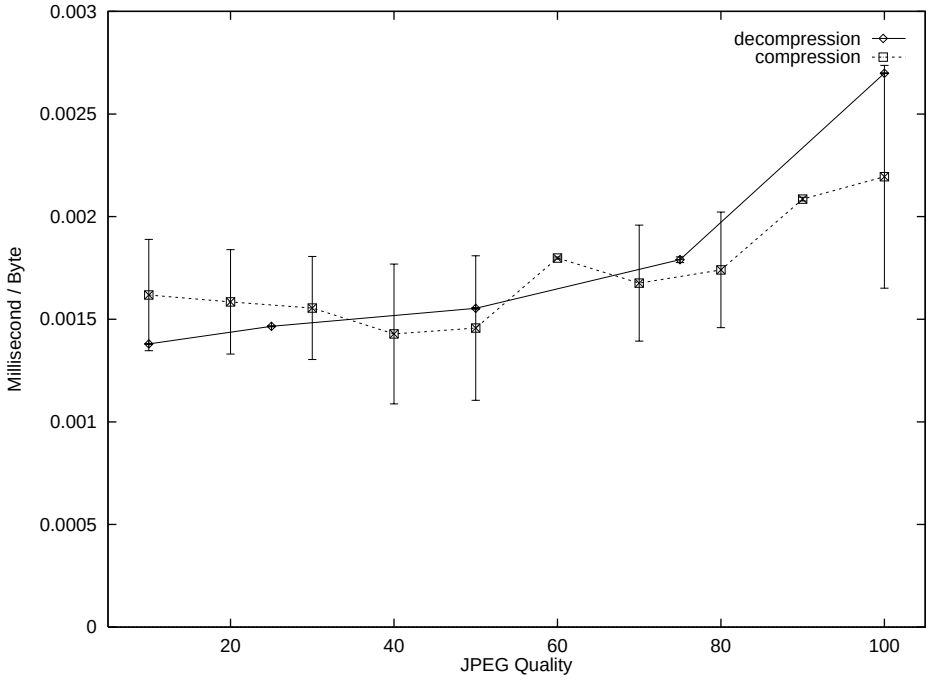

Figure 6.2: Processor load for JPEG Compression and Decompression versus Compressions Quality. The vertical axis is processor time in milliseconds/byte. The horizontal axis is the JPEG quality setting. All points are shown with 95% confidence intervals.

compression in separate processes. We followed experimental techniques that were identical to those used to obtain the processor loads for the previous components. As in Section 5.4, we used a counter process that incremented a double variable in a tight loop to measure the processor load of the JPEG components. We compressed and decompressed images to and from PPM files (see the footnote in section 6.5.2).

The independent variable in our measurements was the JPEG compression quality. Figure 6.2 depicts the processor load for compression and decompression versus quality. All points are shown with 95% confidence intervals.

Using linear regression, we can derive the JPEG processor load in milliseconds per bit of the original PPM image:

JPEG	msec/bit
compression	0.000338
decompression	0.000341

6.5 Predictions

6.5.1 Networks

Since the database is distributed, the above user-level requirements determine the network requirements. For one user, we can predict the network load that flying will induce under the image processing types described in Section 6.1:

Flying Type	Mbits/second
Remote	720
Local	11520

We can determine the minimal network required for flying using projected network bandwidths. The following table lists the Optical Carrier (OC) network rates:

Protocol	Mbits/second
OC-1	52
OC-3	155
OC-12	622
OC-24	1243

We infer from the above two tables that we need more bandwidth than even an OC-12 network can deliver to support just one remote flying user. Since local flying appears to be very difficult under all projected network bandwidths, we will assume remote flying in the rest of the flying predictions. We also assume a flying rate of one hour per week. The network bandwidth will increase further with additional simultaneous users. Figure 6.3 shows the load predictions for a variable number of users.

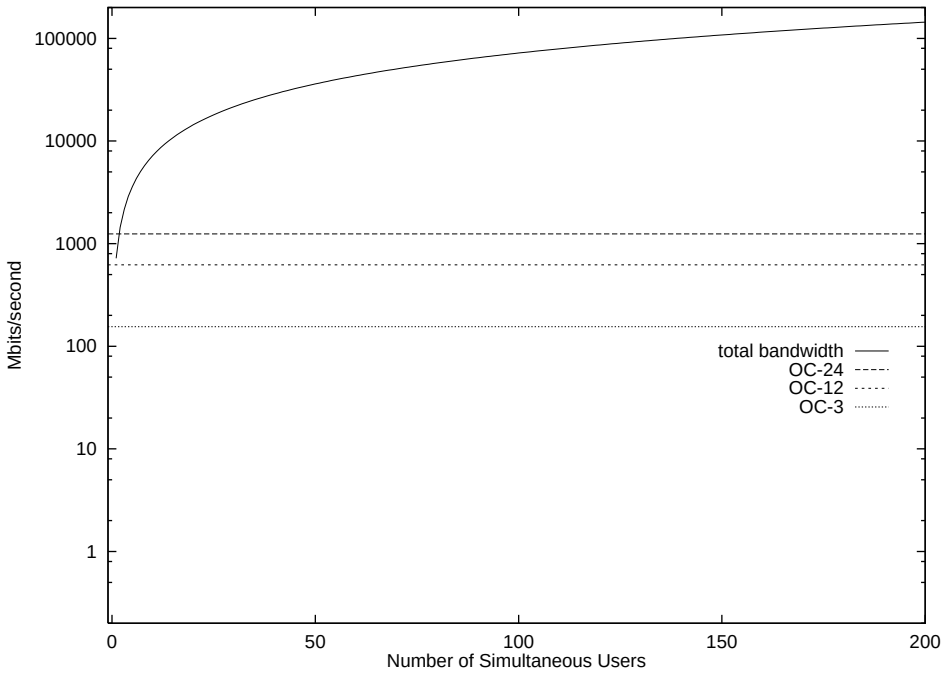

Figure 6.3: Bandwidth versus Number of Simultaneous Users. The curve is the total bandwidth required. The horizontal lines are various Optical Carrier (OC) network bandwidths. On this, and subsequent graphs, the Mbits/second axis is in log 10 scale.

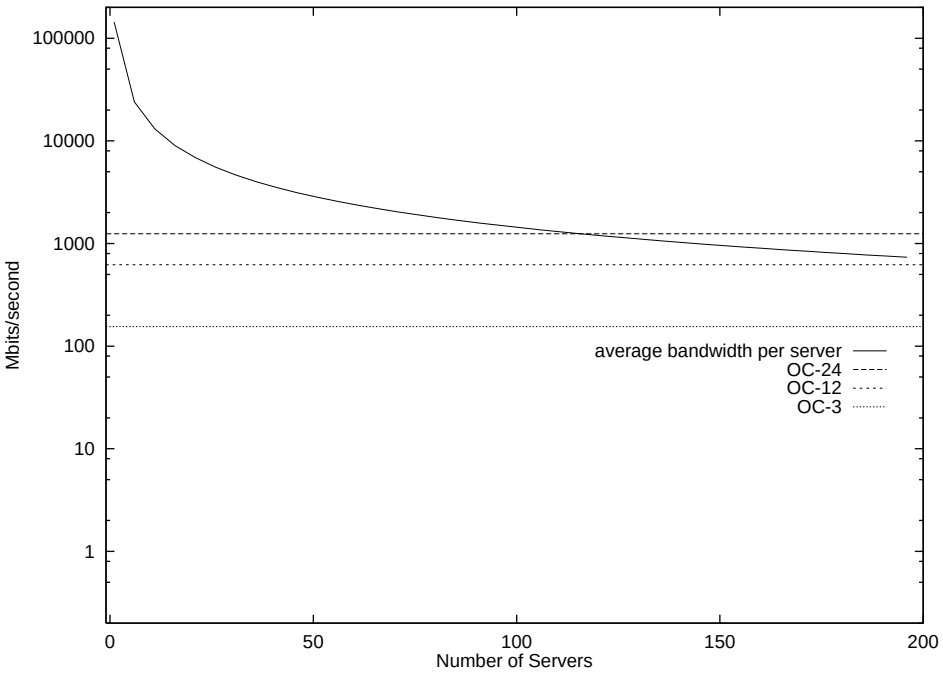

Figure 6.4: Bandwidth versus Servers. The curving is the average bandwidth per server. The horizontal lines are OC network bandwidths.

To counteract the huge bandwidth requirements, we can increase the number of servers. If we assume that each user flies through the database for one hour per week, we have an average of 200 simultaneous users (see section 6.1). Figure 6.4 shows the load predictions for a variable number of servers.

With 200 servers, there is, on average, one server per connected user. We take 200 to be the upper limit on the number of servers. But with 200 servers, each connected to clients by OC-12s, each server can only support one user. Clearly, there is a need to find ways to reduce network load.

6.5.2 Compression

We turn to compression to reduce network bandwidth. If each frame is compressed before sending, the network bandwidth will be reduced.

We assume JPEG compression for flying. The Joint Photographic Experts Group (JPEG) have been working to establish the first international compression standard for continuous-tone still images, both grayscale and color [122]. Although JPEG is intended as a still picture standard, it has greater flexibility for video editing than does MPEG, and is likely to become a "de facto" intraframe motion standard as well.

The quality factor in JPEG lets you trade off compressed image size against quality of the reconstructed image: the higher the quality setting, the closer the output image will be to the original image and the larger the JPEG file.[4] A quality of 100 will generate a quantization table of all 1's, eliminating loss in the quantization step (but there is still information loss in subsampling, as well as roundoff error).

Since the viewed images are to be as close to the original data as possible, we assume a quality of 100 must be used for flying compression. Pilot tests show that JPEG with a quality of 100 reduces the size of PPM[5] images by about 70%. In all subsequent predictions, we assume a compression ratio of 70%.

Figure 6.5 shows our predictions of the effects of compression versus the number of simultaneous users. Figure 6.6 shows our predictions of the effects of compression versus the number of servers.

With compression and 75 servers, we can satisfy the user-level network bandwidth requirements for 200 simultaneous users. However, doing compression and decompression induces additional processor load (see Section 6.5.4).

---

[4] Quality values below about 25 generate 2-byte quantization tables, which are considered optional in the JPEG standard. Some commercial JPEG programs may be unable to decode the resulting file.

[5] Jef Poskanzer's PBMPLUS image software can be obtained via FTP from export.lcs.mit.edu (contrib/pbmplus*.tar.Z) or ftp.ee.lbl.gov (pbmplus*.tar.Z).

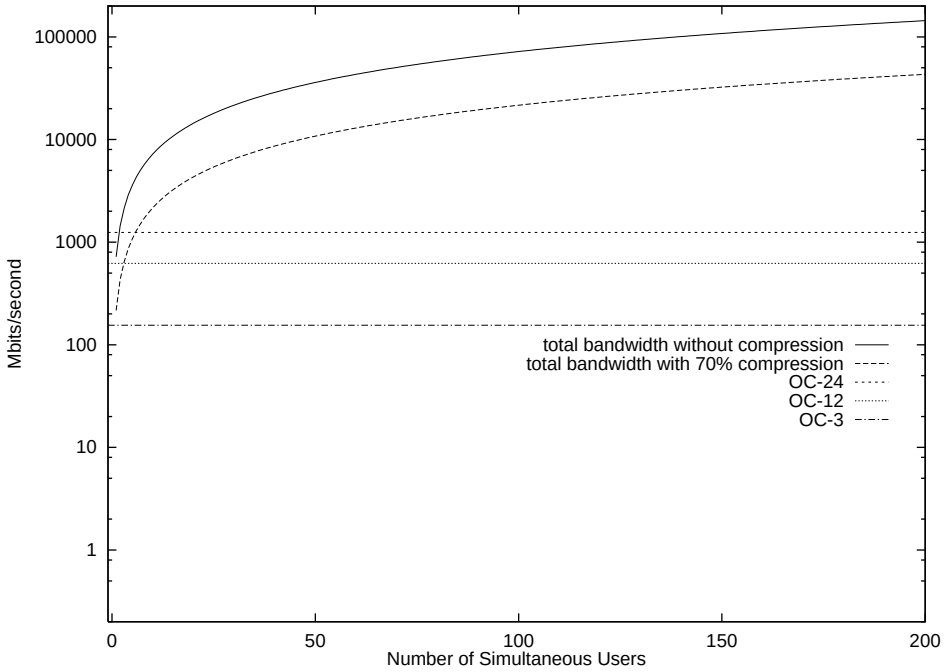

Figure 6.5: Bandwidth versus Simultaneous Users. The top curve depicts the total bandwidth without compression. The lower curve depicts the total bandwidth with 70% compression. The horizontal lines are OC network bandwidths.

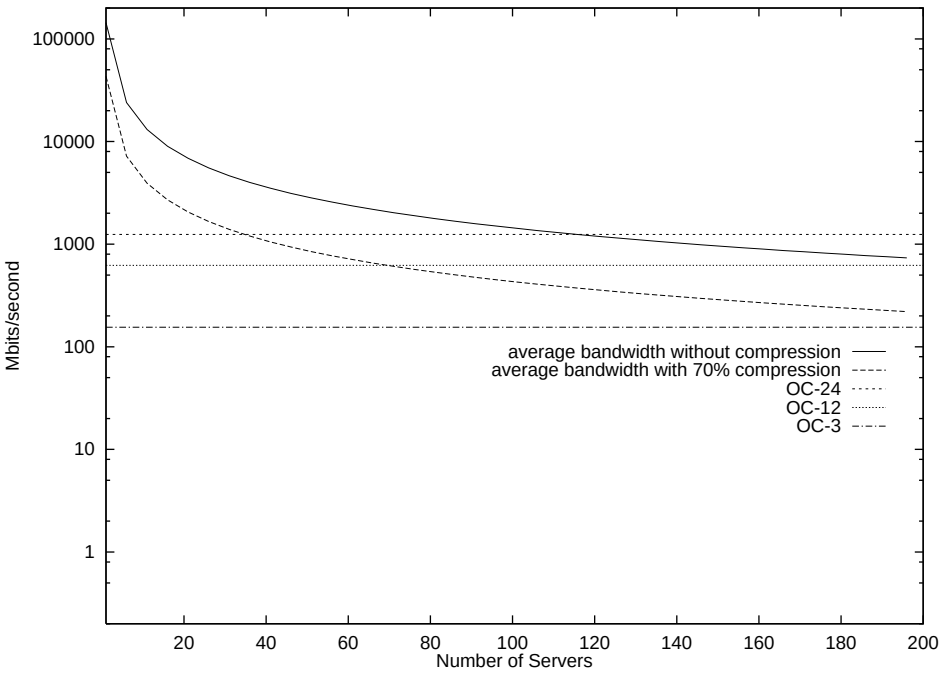

Figure 6.6: Bandwidth versus Servers. This graph depicts the average bandwidth per server for a variable number of servers. The top curving line represents bandwidth without compression. The lower curving line represents bandwidth with compression. The horizontal lines are OC network bandwidths.

6.5.3 Disks

Compression may also be used for reducing the size of images on disk. The most obvious effect is the decrease in storage space. A fully-mapped mouse brain will take approximately 24 terabits of data, while a rat brain will take approximately 80 terabits of data. A 70% compression rate would reduce the required storage space to about 7.2 terabits and 24 terabits respectively.

Compressed disk images will also increase the disk flying throughput. With an image compression of 70%, disk drives can supply over 3 times more frames/second. Using the disk bandwidths obtained from [15], we can determine the disks required for meeting the requirements for flying throughput. Recent studies at the Army High Performance Computing Research Center at the University of Minnesota (AHPCRC) have measured the performance of single disk arrays [101]. When going through a file system, normal I/O runs at about 6 Mbits/second. Direct I/O which bypasses the file system buffer cache and beams data directly to the user's application buffer, runs at 14.5 Mbits/second on a non-fragmented file. Direct I/O and striping across multiple disk arrays can achieve up to N times 14.5 Mbits/second where N is the number of arrays striped over. The AHPCRC currently stripes over 4 or 8 arrays, producing transfer rates around 40 Mbits/second for 4 arrays. In the 3-5 year time frame they expect to see the single array speed go to 100 Mbits/second and possibly 1 Gbits/second for large file transfers. If the physical design matches users' needs effectively, the database retrieval rate may approach the maximum disk rate, but applications cannot exceed this retrieval speed.

Figure 6.7 shows the disk drive throughputs compared to the increase in bandwidth as simultaneous users increase. At least 8 direct I/O disk arrays are required to meet the flying requirements for one user. However, with compression, one user's flying need can be met with 4 direct I/O disk arrays.

Figure 6.8 shows the disk rates compared to the decrease in bandwidth as servers increase. A 1 disk array will not satisfy even a single user's flying requirements.

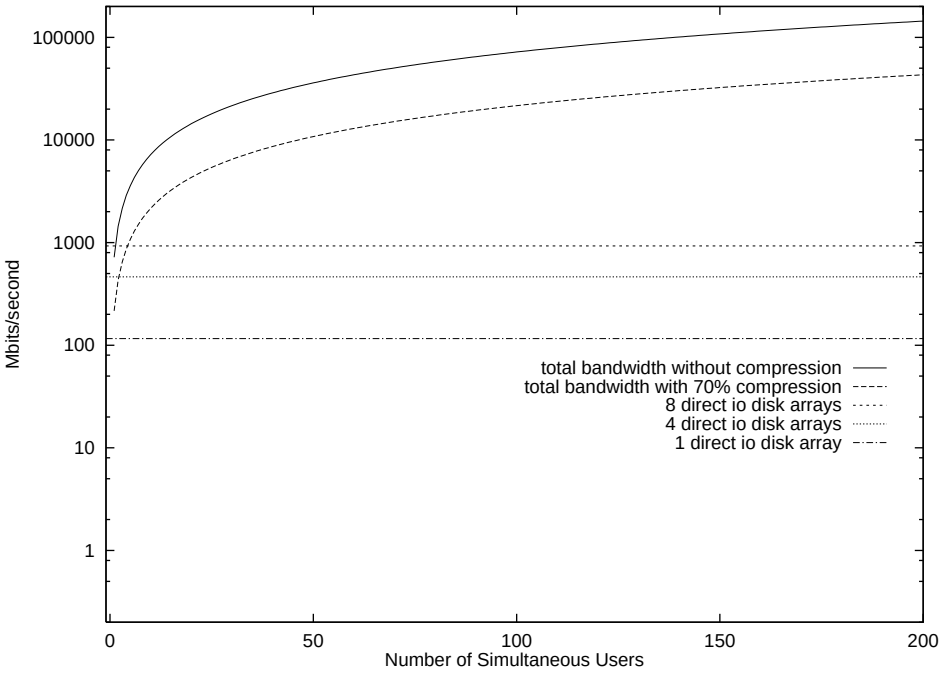

Figure 6.7: Bandwidth versus Simultaneous Users. The Mbits/second axis is a log 10 scale. The increasing curves are bandwidths without compression and with compression. The horizontal lines are all disk throughput rates.

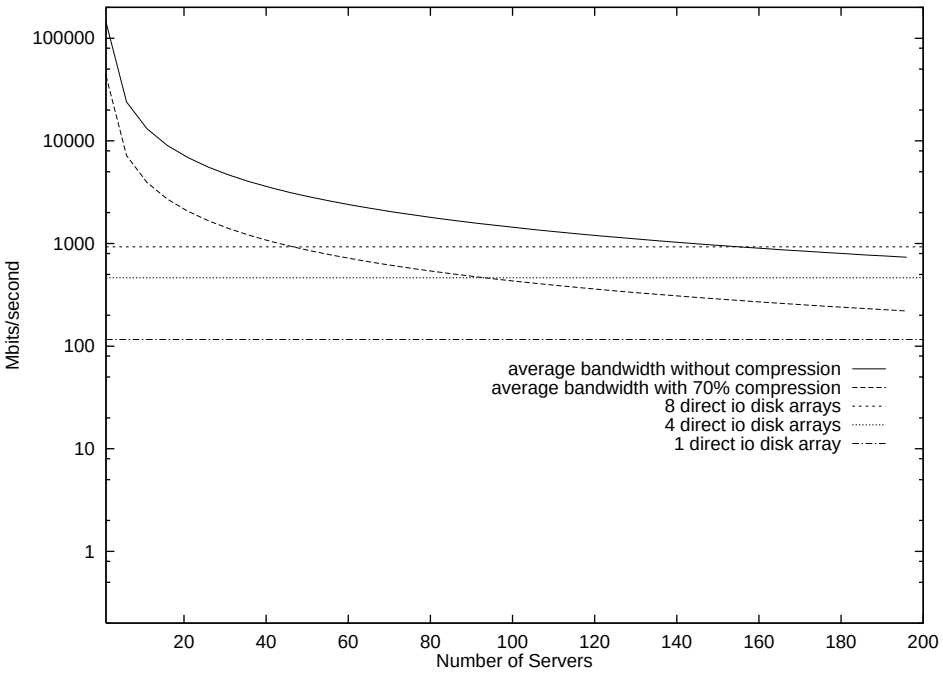

Figure 6.8: Bandwidth versus Number of Servers. The decreasing curves are bandwidths with compression and without compression. The horizontal lines are disk rates.

An 8 disk array will satisfy user flying requirements only with compression and 40 servers. Without compression, an 8 disk array will need 140 servers to satisfy user flying requirements. A 4 disk array will need compression and 100 or more servers to meet flying requirements.

A drawback of compression is that it will decrease processor frame throughput. Compressed images on disk must be uncompressed before the processor can perform image computation to generate the 2-d frame. And compressing before sending takes additional processor time. What are the processor requirements for flying?

6.5.4 Processors

To estimate the effects of new high-performance graphical workstations on the processor load for image calculation, we use the results reported in [107]. They report

Compression	Operations	Seconds/frame	Mbits/second
None	read render send	32.6	0.74
Network	read render compress send	40.4	0.59
Network and Disk	read decompress render compress send	170	.14

Table 6.1: Flying Throughput. This table shows the predicted Sun IPX processor loads with three ways to use compression for flying.

that image calculation from a `256x256x256` voxel volume data set (the same size we assume for our 3-d region) takes about 5 seconds per frame on a 100 MHz Silicon Graphics Incorporated (SGI) Indigo 2 workstation using Levoy's ray-casting algorithm and about a second per frame using a new shear-warp algorithm. We will assume a processor load of one second per frame for an Indigo workstation.

We compare the processing power of the Sun IPX to the SGI Indigo by comparing their performance under the Systems Performance Evaluation Cooperative (SPEC) benchmark integer suite. The SPECint value for a Sun IPX is 21.8 and the SPECint value for the 150 MHz Indigo 2 is 92.2. Roughly, the Indigo 2 is 4 times faster than IPX, so we assume the IPX takes 4 seconds to do the 2-d image calculation from the 3-d region.

We can now predict the flying throughput for the Sun IPX processor server. Table 6.1 gives the Sun IPX processor throughput predictions for the above 3 methods for the server to provide flying described in Section 6.3:

Network compression slightly reduces processor frame throughput. Disk compression, however, reduces processor frame throughput by more than 80%. Most importantly, it would take a processor *over nine-hundred times faster* than a Sun IPX to satisfy the flying requirements for even one user! Even an SGI Indigo 2 with 20 processors would still only have a flying throughput of 59.2 Mbits/second, not even enough for 1 user! Clearly, there is a need to find ways to increase processor flying

Component	Software Load	Hardware Load
read uncompressed image	28.2 seconds	0.03 seconds
read compressed image	8.5 seconds	0.009 seconds
decompress image	131 seconds	-
compress frame	8.2 seconds	-
render frame	4 seconds	0.03 seconds
send uncompressed frame	3.8 seconds	0.006 seconds
send compressed frame	1.1 seconds	0.0002 seconds

Table 6.2: Flying Component Loads. This table shows Sun IPX processor loads induced by various flying components when done in software and hardware.

throughput.

One such method may be specialized hardware. There are co-processors that perform JPEG compression and decompression. Similarly, many computers have specialized graphics rendering hardware. To estimate the processor load for using specialized hardware, we assume that accessing specialized hardware is equivalent to one kernel call and that calls can take place in block sizes of 100 Kbytes. We also assume that all hardware is sufficiently fast to keep up with the processor. Table 6.2 analyzes the Sun IPX processor improvements from using such hardware for each flying component operating on one frame. Table 6.3 gives the processor throughput predictions for an SGI Indigo 2 workstation using hardware support for the flying components.

We can now predict the processor throughput required for flying. Figure 6.9 shows the load predictions versus simultaneous users and Figure 6.10 shows the load predictions versus servers. Flying throughput for a Sun IPX would be at the very bottom of these graphs. Even a 20 processor 100 MHz SGI Indigo 2 using compression cannot satisfy user flying requirements for even one user. We therefore consider the

Compression	Operations	Seconds/frame	Mbits/second
None	read render send	0.0170	1416
Network	read render compress send	0.0150	1608
Network and Disk	read decompress render compress send	0.0066	2400

Table 6.3: Hardware Flying Throughput. Possible SGI Indigo 2 processor loads induced by the three forms of flying when the processor is equipped with specialized flying hardware.

use of specialized hardware and kernel support for reading, compressing, calculating and sending.

6.5.5 Quality

So far we have explored the necessary network, disk, and processor requirements to completely satisfy our user requirements. But when user requirements cannot be completely satisfied, there is often a tradeoff in system choices. For example, if more compression is applied, bandwidth requirements for network and disk are reduced, but the processor may become overloaded with the added decompression cost. We apply the model presented in Section 2.3 to predict the quality for flying on different system configurations. In order to apply our quality model, we must: 1) determine the region of acceptable flying quality; 2) predict jitter; 3) predict latency; and 4) predict data loss.

To determine the region of acceptable flying quality, we need to define acceptable limits for flying along each of the latency, jitter and data loss axes. According to Jeffay and Stone, delays of 230 milliseconds or under are acceptable for a videoconference [64]. We assume this is a lower bound on the acceptable latency for flying. For data loss, research in remote teleoperator performance has found that task performance is virtually impossible below a threshold of 3 frames per second [84]. We also assume fewer than 20 Mbits/second of data provides too little data to be useful, and more

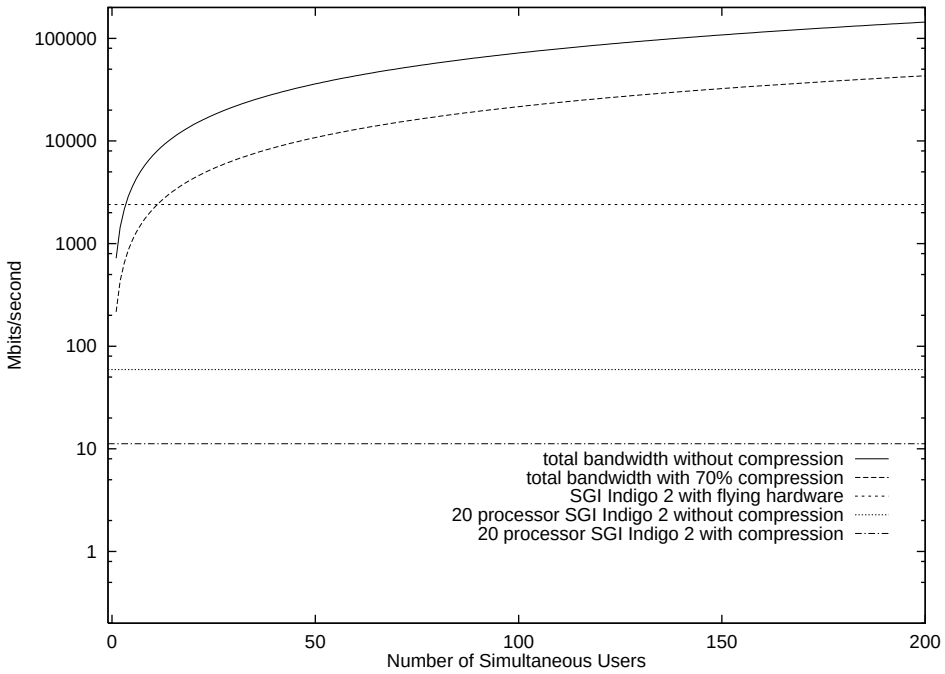

Figure 6.9: Bandwidth versus Simultaneous Users. The upward sloping curves are total bandwidths with compression and without compression. The horizontal lines are processor flying throughput rates. The top horizontal line is the predicted flying throughput of a 2 processor SGI Indigo 2 with specialized flying hardware.

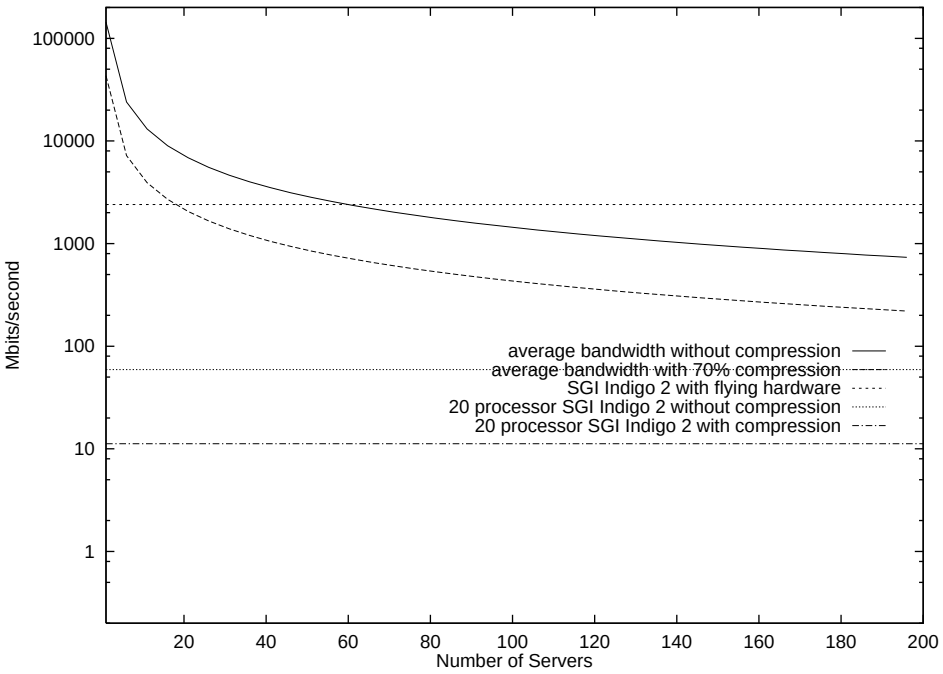

Figure 6.10: Bandwidth versus Number of Servers. The upward sloping curves are the average bandwidths per server with compression and without compression. The horizontal lines are processor flying throughput rates. The top horizontal line is the predicted flying throughput of a SGI Indigo 2 with specialized flying hardware.

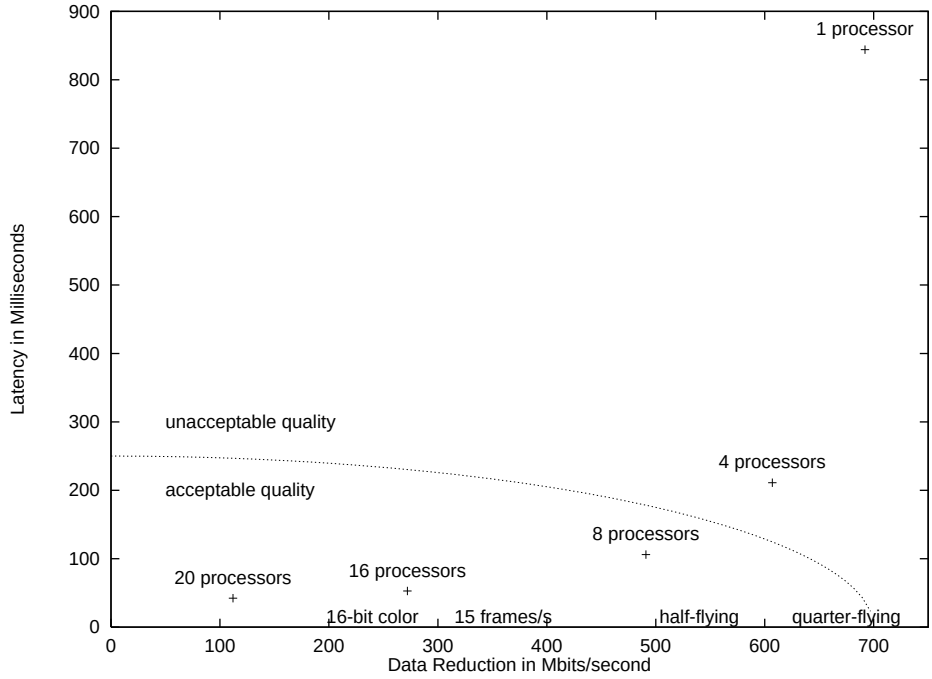

Figure 6.11: Client Application Quality. This graph depicts a measure of quality for the client. The horizontal axis is the number of Mbits/second of data reduction received by the client. The vertical axis is the latency added by the client. The points are SGI Indigo 2's clients with different numbers of processors. The curve represents an acceptable level of quality; all points inside the curve will have acceptable quality while points outside will not. Note that the clients are not equipped with any special flying hardware.

than 720 Mbits/second (30 frames/second and 24 bits of color) provides no more useful data. As described in Section 5.6.4, the presence of jitter often presents an opportunity for a tradeoff among latency and data loss. We predict jitter, latency and data loss as in Section 5.6.4, Section 5.6.4 and Section 5.6.4, respectively.

To compute application quality, we use the methods described in Section 2.3. Figure 6.11 depicts our quality predictions. The individual points are all SGI Indigo 2 clients with a different number of processors. In this figure, we have not assumed any specialized flying hardware. We assume that the servers will be able to provide the bandwidth requested by all the clients to simplify the computation.

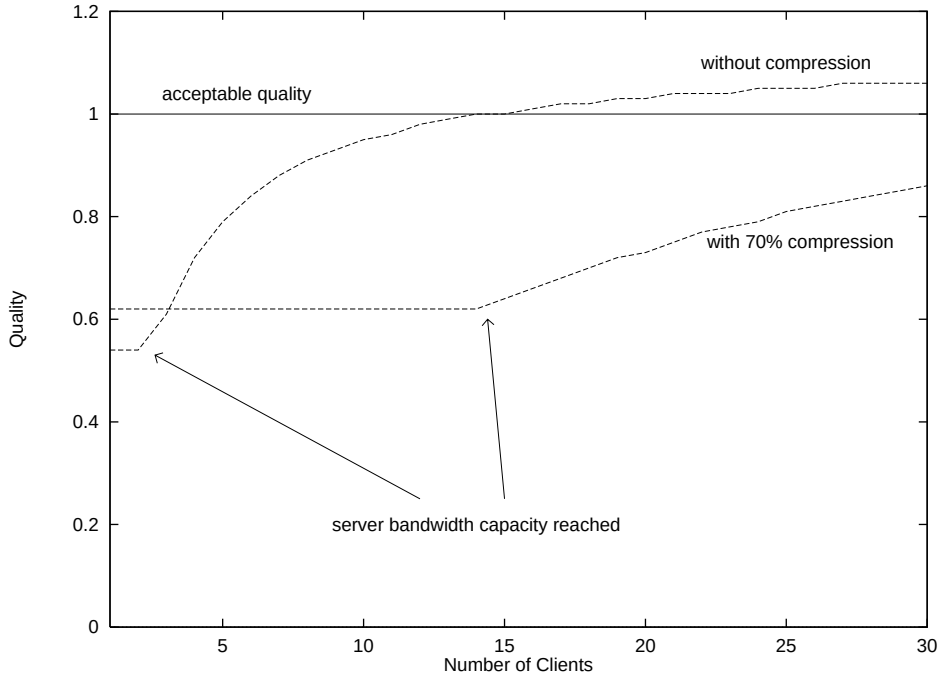

Figure 6.12: Quality versus Clients. Clients are 100 processor SGI Indigo 2 workstations with no specialized hardware. The server is an SGI Indigo 2 workstation with specialized hardware (see Section 6.5.4). The arrows indicate points at which the server can no longer keep up with the bandwidth requests by the clients. At this point, perceived application performance decreases, as depicted by the increasing quality values.

Figure 6.12 depicts quality versus the number of clients for both flying with compression and without. Clients are assumed to be 20 processor Indigo 2's without specialized hardware. The server is assumed to be an SGI Indigo 2 workstation with specialized hardware (see Section 6.5.4). The arrows indicate points at which the server can no longer keep up with the bandwidth requests by the clients. At this point, application performance decreases, as depicted by the increasing quality values. Thus, for fewer than 4 clients, compression decreases client-side quality, mostly because of the latency increase from the clients decompressing the images. However, for 5 or more clients, compression increases application quality because the server can meet the bandwidth requirements of more clients.

Note that this graph depicts the quality tradeoffs with one particular system configuration. Different system configurations may have different quality tradeoffs.

6.6 Summary

Since the users of the Zoomable Brain Database will be distributed across the country, the system stress from flying will be spread over the underlying network. If the network topology appears as a backbone with six Network Access Points (NAP's), like the proposals for the national data highway [97], we can determine the bandwidth required for each link.

We assume:

- An equal distribution of users among servers,
- 50 servers, each directly connected to the backbone, and
- 70% compression.

We can use the analysis from previous sections to determine the system requirements:

Component	Mbits/second
Backbone:	43200
NAP's:	7200
Individual Connections:	216
50 Processor Servers:	4680

Figure 6.13 depicts the network for the constructed Zoomable Brain Database. Unfortunately, some of the proposals for the national data highway have bandwidths like those in figure 6.14.

We can only hope to fly on such a network by reducing the user-level flying requirements. [15] describes a number of modifications that can be done to the user

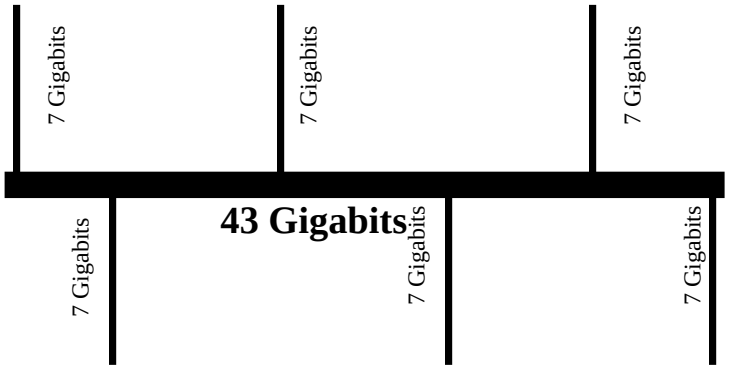

Figure 6.13: Bandwidth requirements for the Zoomable Brain Database system on a topology similar to a possible national data highway [93].

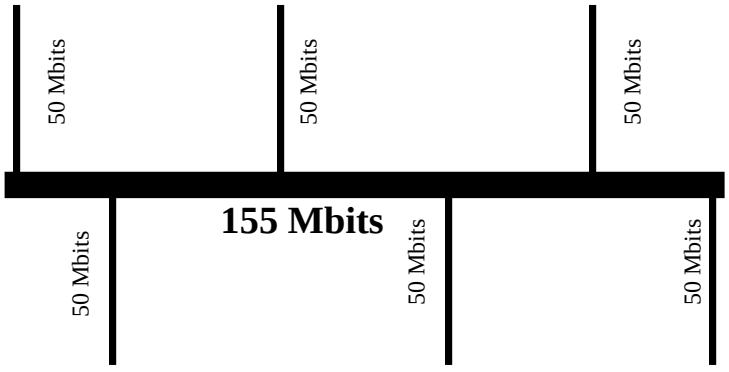

Figure 6.14: A proposed national data highway network. In order to allow flying in this environment, user-level requirements must be relaxed.

requirements that will ease the system requirements. The server can reduce the frame resolution in each dimension by one-half (half-flying) to a one-fourth (quarter-flying), reducing the data needed by 1/8th and 1/64th respectively. Likewise, fewer bits of color decrease the data needed for each pixel. The table below summarizes the system benefits of several of these modifications:

	Reduction		
Modification	I/O	Processor	Network
8 bit color	0	2/3	2/3
half-flying	0	0	3/4
quarter-flying	0	0	15/16
3 frames/sec	9/10	9/10	9/10

Processing and sending 8-bit color reduces processor and network requirements by 2/3. Yet many neuroscientists may find scientific analysis difficult under such conditions. In half- or quarter-flying, one frame pixel represents 4 or 16 display pixels, respectively. Thus, network data rates are reduced by 3/4 and 15/16 respectively, but at the expense of image quality. Sending fewer frames per second reduces all system components. However, motion displayed at 3 frames/second has a much rougher flow than does motion displayed at 30 frames/second.

The data reduction from the above modifications can also be combined. If we modify the user-requirements to allow 8 bit color, 3 frames/second and 1/2 flying, the system requirements for the Zoomable Brain Databases would appear as in Figure 6.15. These reductions in user requirements make it possible to use the proposed infrastructure to achieve some of our goals, but will not be suitable for some neuroscience analysis.

All of our predictions have done the image computation at the server (*remote flying*) as opposed to image computation done at the client (*local flying*). But remote flying, while inducing significantly less bandwidth than local flying, increases the amount of server computation. With many database users, the server computa-

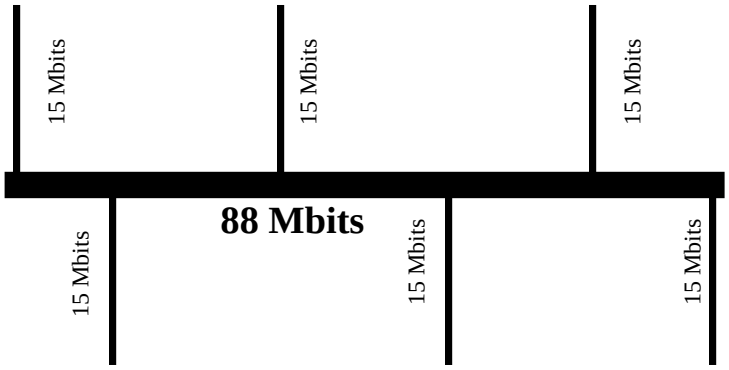

Figure 6.15: Bandwidth requirements for the Zoomable Brain Database system on a topology similar to a proposed national data highway. User requirements have been reduced to ease system requirements.

tion may become prohibitive. As client workstations become increasingly powerful, they may be able to readily perform the needed computations at a sufficient flying rate, reducing server load. In other environments, server load has proven critical for performance [77].

Advances in networking, such as those described in this paper, are needed to enable a new era of neuroscience. Researchers will be able to directly access high-quality pristine data from all areas of the brain. Hypotheses will be formulated, evaluated, and scientifically tested through interaction with data collected by scientists on the other side of the world. As we have shown, realizing this dream fully will require gigabits per second of sustained throughput per user, and terabits per second of aggregate bandwidth.

Chapter 7

The Virtual Cockpit

7.1 Overview

We have applied our model to the specifications and preliminary performance results of a flight simulator called the Virtual Cockpit [85]. The Virtual Cockpit is a low-cost, manned flight simulator of an F-15E built by the Air Force Institute of Technology. A soldier flies the Virtual Cockpit using the hands-on throttle and stick, while the interior and out-the-window views are viewed within a head-mounted display. The Virtual Cockpit research was undertaken to build an inexpensive system for use as a tactics trainer at the squadron level that could participate in a Distributed Interactive Simulation (DIS).

DIS is a virtual environment being designed to allow networked simulators to interact through simulation using compliant architecture, modeling, protocols, standards and databases. The DIS system must be flexible and powerful enough to support increasingly large numbers of simulators. The Advanced Research Projects Agency (ARPA) estimates the need for exercises with 100,000 simulators [25].

In addition to latency, jitter and lost data, DIS simulators have an additional application-specific measure that affects application quality. DIS simulators use "dead reckoning" algorithms to compute position. Each simulator maintains a simplified

representation of other simulators and extrapolates their positions based on their last reported states. When a simulator determines that other simulators cannot accurately predict its position within a pre-determined threshold, it sends a state update packet. The state update contains the correct position and orientation as well as velocity vectors and other derivatives that the other simulators can use to initiate a new prediction. Figure 7.1 depicts the difference between the actual flight path of a simulator and the dead reckoning flight path computed by the other simulators.

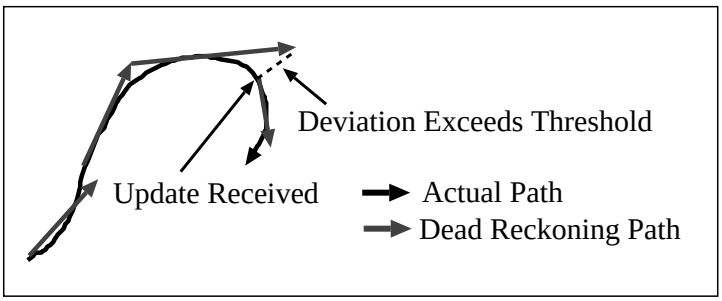

Figure 7.1: Actual Path versus Dead Reckoning Path. The solid line represents the actual flight path of the simulator. The grey line represents the flight path as computed by the other simulators using a simple dead-reckoning algorithm based only on direction and velocity. The dashed line represents a time when the perceived path deviated from the actual path by more than a pre-set threshold. At this time, a packet updating position, orientation and direction is sent and the dead reckoning resumes from this new posture.

Dead reckoning creates an additional quality measure specific to DIS applications:

- *Missed Updates.* If a simulator is unable to send the update or process an incoming update, the accuracy of the simulation decreases.

Figure 7.2 shows the effects of missed updates on accuracy. The horizontal line is the position threshold, set at 1 meter. When the inaccuracy surpasses the threshold, an update is sent, bringing inaccuracy back to 0. Inaccuracy increases one meter on average for each update missed. These missed updates are shown by the taller

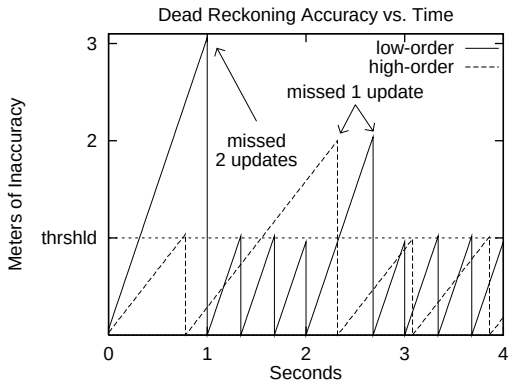

Figure 7.2: Dead Reckoning Accuracy. The horizontal line is the position threshold, set at 1 meter. The triangular-shapes represent areas of inaccuracy. When they reach the threshold, an update is sent, bringing inaccuracy back to 0. Inaccuracy increases by as much as a meter on the average for each update missed. These missed updates are shown by the taller triangles.

triangles. The total inaccuracy for the simulation is represented by the sum of the areas of the triangles. The larger the sum, the more inaccurate the simulation. As we would expect, consecutive missed updates affect accuracy more than non-consecutive missed updates. There are two different simulations depicted in the graph. One uses a dead reckoning algorithm that does more computation (high-order) while the other uses a dead reckoning algorithm that does less computation (low-order). The total accuracy is independent of the dead reckoning algorithm used; changing the dead reckoning algorithm only affects the number of updates sent and not the total accuracy.

Our objective is to enhance our understanding of the performance bottlenecks that arise in the design of a multi-person, distributed multimedia application. Among the factors we investigate are the performance of the Virtual Cockpit on existing networks and processors and the effects of high-speed networks and high-performance processors on Virtual Cockpit quality.

This Chapter is organized as follows: Section 7.2 of this section describes related work in Distributed Interactive Simulation and geographic information systems. Sec-

tion 7.3 introduces our model of the Virtual Cockpit. Section 7.4 details micro experiments that measure the processor load of each component. Section 7.5 describes macro experiments that test the accuracy of the predictions based on the micro experiments. Section 7.6 analyzes the experiments and projects the results to future environments. And Section 7.7 summarizes the important contributions of this section.

7.2 Related work

7.2.1 Distributed Interactive Simulation

Distributed Interactive Simulation (DIS) is a virtual environment being designed to allow networked simulators to interact through simulation using compliant architecture, modeling, protocols, standards and databases [25]. DIS exercise involve the interconnection of a number of simulators in which simulated entities are able to interact within a computer generated environment. The simulators may be present in one location or be distributed geographically and the communications are conducted over the network. The DIS system must be flexible and powerful enough to support increasingly large numbers of simulators. The Advanced Research Projects Agency (ARPA) estimates the need for exercises with 100,000 users.

We study the performance of a flight simulator, the Virtual Cockpit, designed to participate in DIS exercises. In particular, we examine the network and processor requirements as the number of users scales to 100,000.

7.2.2 Geographic Information Systems

A Geographic Information System (GIS) is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information, i.e. data identified according to their locations. GIS are often processor and network inten-

sive. GIS research includes ways to effectively distribute processing time on high performance parallel computer systems while obtaining enough fidelity to support realistic simulations [105], and developing network software architectures to support large scale virtual environments [81].

GIS are the basis for many vehicle simulators. We use performance numbers from the GIS used by the Virtual Cockpit, Software Systems' MultiGen, in our experiments.

7.3 Model

The fundamental components for the Virtual Cockpit are: send a dead reckoning update packet, receive update packets, update dead reckoning status, read GIS information from the database, perform dead reckoning, render the frame and display the frame.[1] The fundamental components of the Virtual Cockpit are depicted in Figure 7.3. *Send* is the processor load for sending updates to the other soldiers. *Receive* is the processor load for receiving updates. *Update* is the processor load for updating dead reckoning status structures. *Read* is the processor load for reading from a file. *Reckon* is the processor load for doing a dead reckoning computation to determine the position of another soldier. *Render* is the processors load for generating a frame to be displayed. *Display* is the processor load for displaying the frame on the screen. *Send* is done once for each update packet sent. Receive and Update are done once for each update packet received. Read, Render and Display are done once per frame. Reckon is done once per soldier per frame.

7.4 Micro Experiments

After carefully measuring the processor, network and disk loads induced by each component, we can predict the load of an application built with those components.

[1] We assume that the user interface contributes an insignificant amount of processor load.

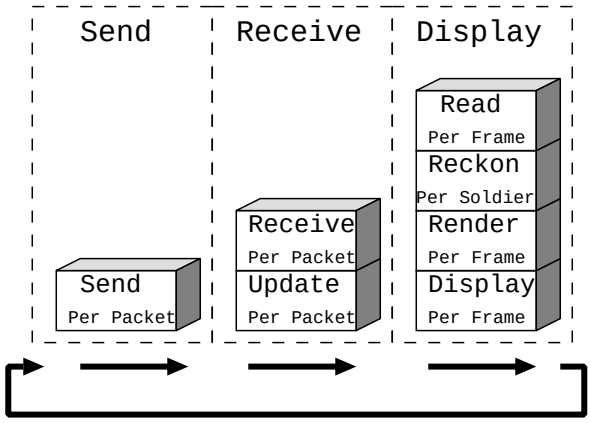

Figure 7.3: Virtual Cockpit Model. The 3-d boxes above all contain a fundamental component of the Virtual Cockpit. The larger, dashed boxes place the fundamental components into conceptual groups.

Changes in application configuration or changes in hardware are represented by modifying the individual components and observing how that affects the application performance.

We re-use the appropriate micro-experiment results from Section 5.4, Section 6.4 and [21] that were used for the audioconference and flying applications and run experiments to measure the components new to the virtual cockpit. The components we re-use are: `send`, `receive`, `read` and `write`. The new components are: `update`, `reckon`, `render` and `display`.

We ran our experiments on a dedicated network of SGI Personal Iris workstations. Each workstation had a 20 MHz IP6 R3000 processor, 16 Mbytes RAM and 96 Kbytes cache.

We use a high-order dead reckoning algorithm as an upper bound on the processor load required (see Section 7.1).

We compare the load for sending and receiving unicast packets to that of multicast so we could better analyze the benefits of multicast in Section "Predictions" below. The update packet size is 144 bytes, as defined by the DIS standard [81].

We followed experimental techniques that were identical to those used to obtain the processor loads for the previous components. As in Sections 5.4 and 6.4, we used a counter process that incremented a double variable in a tight loop to measure the processor load for the model components.

Table 7.1 gives the processor times for the Virtual Cockpit components. All points are shown with a 95% confidence interval. The times for multicast send and unicast send are indistinguishable at the 95% confidence level. The time for multicast receive is significantly more than the time for unicast receive.

Although we present only the measurements of the processor load, in the past, workstation performance has scaled with processor performance [56]. We assume that processor performance will continue to reflect the workstation performance.

7.5 Macro Experiments

We ran macro experiments for the Virtual Cockpit model for 2-15 simulated soldiers. Each soldier ran a Virtual Cockpit on an SGI Personal Iris, one workstation per soldier. The workstations were connected by an Ethernet. Each Virtual Cockpit sent 3 update packets per second, the average for most vehicles in a typical DIS exercise [81].

Our micro experiments measured the processor load for the fundamental components of the Virtual Cockpit. Using these measurements, we can predict the number of occurrences of each of the fundamental components in the macro experiments. For example, in a 10 second, two-soldier simulation, we would expect each soldier to send and receive 30 update packets. Table 7.2 gives the predicted versus actual count for all the Virtual Cockpit components for an experiment with 9 soldiers. "Predicted" are the predicted numbers of times each component occurred per second based on the performance of the individual components. "Actual" are the actual numbers of times each component was recorded per second in the experiment. Other macro ex-

Component	msec	Low	High	Per Unit
Display	0.194	0.193	0.195	frame
Read	0.000327	0.000325	0.000329	byte
Render	305	304	306	frame
Update	0.00813	0.00808	0.00818	soldier
Compute	0.949	0.940	0.958	soldier
Unicast Receive	0.579	0.230	0.928	update
Multicast Receive	1.12	1.11	1.13	update
Unicast Send	1.58	1.57	1.59	update
Multicast Send	1.57	1.56	1.58	update

Table 7.1: Processor load breakdown for the fundamental components of the Virtual Cockpit. Send is the processor load for sending updates to the other soldiers. Receive is the processor load for receiving updates. Update is the processor load for updating dead reckoning status structures. Read is the processor load for reading from a file. Reckon is the processor load for doing a dead reckoning computation to determine the position of another soldier. Render is the processors load for generating a frame to be displayed. Display is the processor load for displaying the frame on the screen. "Low" and "High" are the lower and upper bounds respectively from the 95% confidence interval.

Component	Predicted	Actual
Receive	27	25
Send	3	3
Update	27	25
Compute	27	28
Render	3	3
Display	3	3
Read	6265	6354

Table 7.2: Component predictions for a Virtual Cockpit simulation with 9 Soldiers. "Predicted" are the predicted performance numbers based on the performance of the individual components. Actual are the actual performance numbers reported in the experiment. All predicted values are for one second.

periments with 2-15 soldiers had similar results, but we do not show them here to avoid redundancy.

All of the predicted performance results are within 10% of the actual performance results. We regard future comparisons of predicted performance results that combine the Virtual Cockpit components into alternate configurations to be significantly different if they vary by more than 10%.

Our micro experiments give us the performance results for the fundamental components of the Virtual Cockpit. Our macro experiments test the accuracy of combining the values from our micro experiments into accurate Virtual Cockpit predictions.

7.6 Predictions

By modifying the fundamental Virtual Cockpit components, we can predict performance on alternate system configurations. This allows us to evaluate the potential performance benefits from expensive high-performance processors and high-speed net-

works before installing them. Moreover, we can investigate possible performance benefits from networks and processors that have not yet been built.

Our approach for evaluation of each alternative system is the same: We modify the parameters of our performance model to fit the new system, then evaluate the resulting model to obtain performance predictions. These analyses are intended to provide a sense of the relative merits of the various alternatives, rather than present absolute measures of their performance.

7.6.1 Networks

We first investigate the network bandwidth requirements for a DIS simulation. Network bandwidth requirements depend upon the frequency with which each Virtual Cockpit broadcasts its position. In the absence of dead reckoning, each simulator would have to send an update every frame, for a maximum of 30 updates per second. However, with dead reckoning, the average rate is 3 updates per second [81], with a maximum of 17 updates per second [55].

Figure 7.4 depicts the predicted network bandwidth versus the number of soldiers in a Virtual Cockpit exercise. The upward sloping lines are the predicted network bandwidth for updates sent at the frame rate without dead reckoning, the maximum update rate with dead reckoning, and average update rate with dead reckoning. The three lines with the lower slopes all use multicast routing, as specified by the DIS standard [25]. The line with the steepest slope represents the network bandwidth required for a unicast simulation at an average update rate. The horizontal lines are the maximum bandwidths for a 1.5 Mbps T1, 10 Mbps Ethernet and 155 Mbps ATM network.

Without dead reckoning, a T1 network becomes saturated at about 50 soldiers and an Ethernet at about 200 soldiers. With dead reckoning and the maximum update rate, a T1 can support nearly 100 soldiers and an Ethernet can support nearly 400 soldiers. With dead reckoning and the average update rate, a T1 can support about

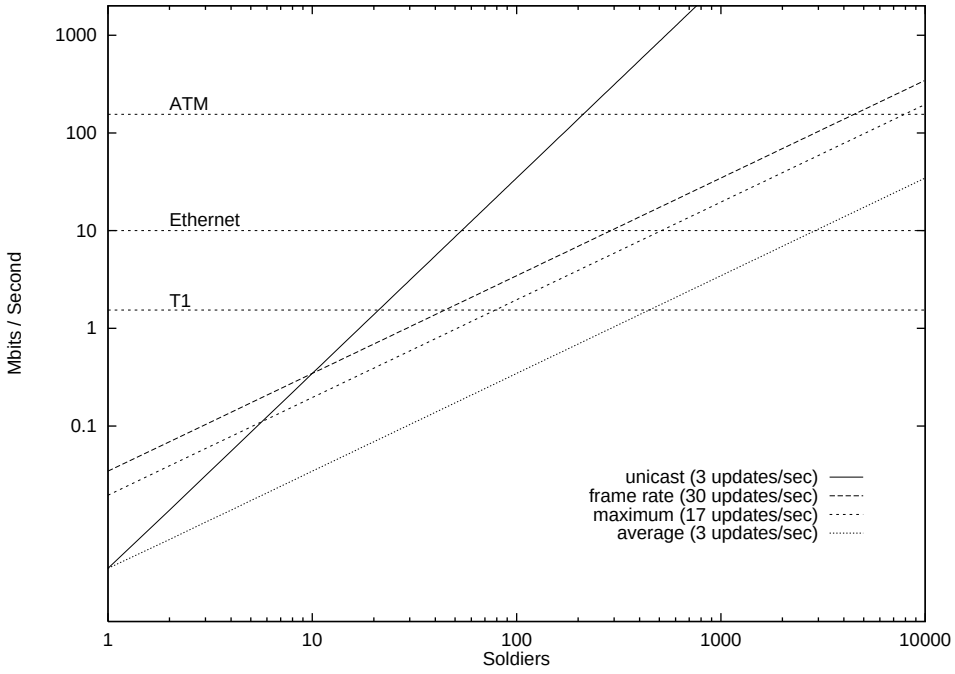

Figure 7.4: Network Bandwidth versus Soldiers. This graph shows the predicted network bandwidth as the number of soldiers increases. The upward sloping lines are the predicted bandwidth for the frame rate, maximum rate, and average update rate. The steeply sloped line is the network bandwidth required for unicast with an average update rate. The horizontal lines are the maximum bandwidths for a T1, Ethernet and ATM network. Both the horizontal axes and vertical axes are in log scale.

400 soldiers and an Ethernet can support over 2000 soldiers. We use the maximum update rate for situations in which we are concerned about peak network bandwidth. We use the average update rate in situations in which we are concerned about network throughput.

Multicast is crucial for soldier scalability. Even at the average update rate, a Virtual Cockpit exercise using unicast rapidly exceeds the bandwidth available on T1 and Ethernet networks.

7.6.2 Processors

Existing networks are capable of supporting many soldiers, but what about existing processors? We investigate the performance results Virtual Cockpit exercises with more powerful processors.

We compare the performance of the SGI Personal Iris to other processors by comparing performance results from the Systems Performance Evaluation Cooperative (SPEC) benchmark integer suite. In past work, we have found SPEC results correlate inversely with execution times for processor-intensive tasks. Since the largest Virtual Cockpit component, render, is largely processor-intensive, we assume that the Virtual Cockpit processor performance will correlate well with SPEC results. The SPEC int92 value for a SGI Personal Iris is 22.4 and the SPEC int92 value for the 150 MHz SGI Indigo 2 is 92.2. Roughly, the Indigo 2 is 6.5 times faster than the Personal Iris, so we assume it does all the Virtual Cockpit components measured in Section 7.4 6.5 times faster.

Swartz et al. did experiments to measure the effects of frame rate on on the ability to perform tasks through Unmanned Aerial Vehicles (UAVs) [114]. UAVs are used to conduct a variety of reconnaissance missions with human operators interpreting the transmitted imagery at ground stations. The experiments found that human performance with 4 frames per second is significantly better than 2 frames per second, but not consistently worse than 8 frames per second. Therefore, we assume a rate of

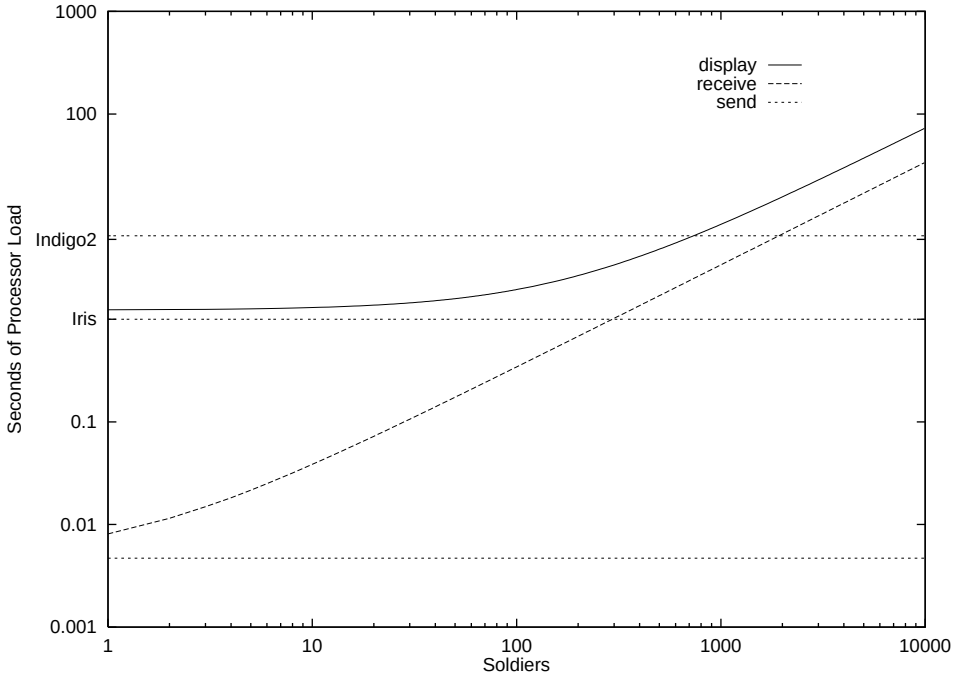

Figure 7.5: Processor Load versus Soldiers. This graph depicts the predicted processor load in seconds on a SGI Personal Iris versus the number of soldiers. The graph reads from the bottom. The load of each piece the sum of the pieces below it. Thus, the total processor load is indicated by the "display" line at the top. The horizontal lines are the maximum processor throughputs for an SGI Personal Iris and an SGI Indigo 2. Both axes are in log base 10 scale.

4 frames per second in our processor load predictions.

We break the processor load into three pieces: send, receive and display (see Figure 7.3 for the components in each piece). Figure 7.5 depicts the predicted processor load in seconds on a SGI Personal Iris versus the number of soldiers. The graph reads from the bottom. The load of each piece is the sum of the pieces below it. Thus, the total processor load is indicated by the "display" line at the top. The horizontal lines are the maximum processor throughputs for SGI Personal Iris' and SGI Indigo 2s, as indicated.

The processor is the bottleneck. The SGI Personal Iris is unable to support even the minimal frame rate for any soldiers. Processors significantly more powerful than

SGI Personal Iris' are needed to realize the army's goals of hundreds and thousands of distributed soldiers interacting in a simulation. Indigo 2s, powerful workstations, can support adequate frame rates while communicating with 1000s of other soldiers.

7.6.3 Quality

If we have a system with the more powerful processors required for acceptable frame rate, we can investigate the predicted user perception of the Virtual Cockpit – what is the *application quality*? In order to apply our quality model, we must: 1) determine the region of acceptable quality; 2) predict jitter; 3) predict latency; and 4) predict data loss.

To determine the region of acceptable quality, we need to define acceptable limits along each of the latency, jitter and data loss axes.

- Latency is the time from when an update is sent until it is processed and displayed by the other simulators. The DIS steering committee had defined latency as 100 to 300 milliseconds as acceptable for DIS applications [25]. We use 300 milliseconds as the maximum acceptable latency for the Virtual Cockpit.

- Jitter is the variance in the latency. We assume 10% jitter is the maximum acceptable for the Virtual Cockpit.

- For data loss, we note that research in remote teleoperator performance states that task performance is virtually impossible below a threshold of 3 frames per second [84]. We use 3 frames per second as the minimum acceptable frame rate. The conventional rate of 30 frames per second would provide a more realistic simulation, possibly influencing training effectiveness. We assume that frame rates above 30 frames per second provide no more useful data.

- For missed updates, we assume the same thresholds of 1 meter position and 3 degrees of accuracy used in SimNet [55]. We assume the simulation must be 95% accurate to be acceptable.

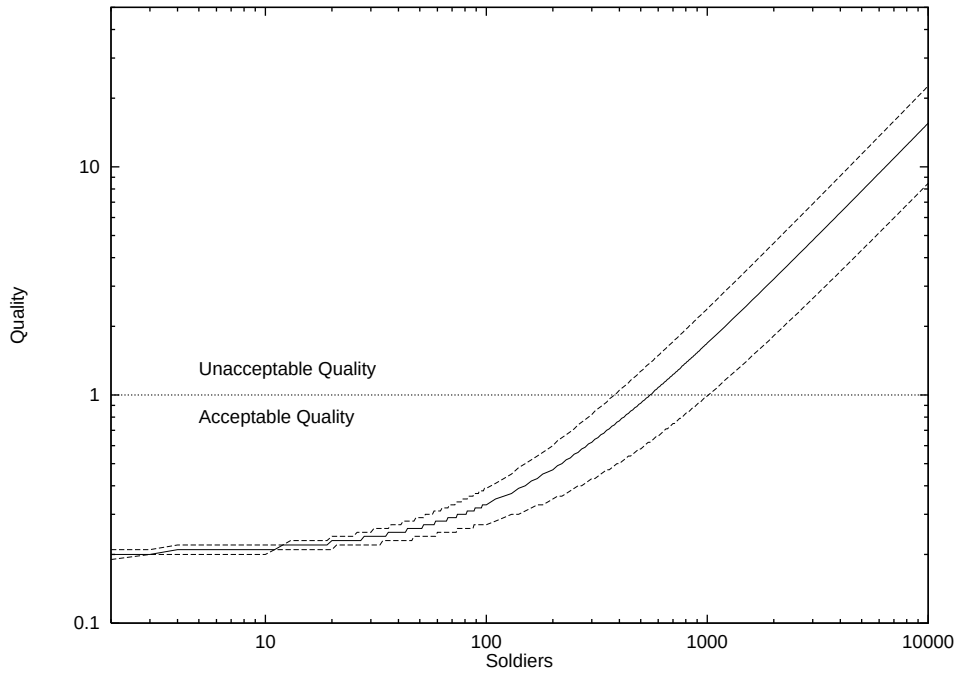

Figure 7.6: Virtual Cockpit Quality versus Soldiers. The middle line represents predicted application quality as the number soldiers increases. The upper and lower lines represent best and worst case quality scenarios respectively. The horizontal line marks the acceptable/unacceptable quality limit.

We predict jitter, latency and data loss as in Section 5.6.4, Section 5.6.4 and Section 5.6.4, respectively.

To compute application quality, we use the methods described in Section 2.3. We examine the application quality for Virtual Cockpit exercises with SGI Indigo 2s because SGI Personal Iris' were incapable of achieving the minimum acceptable frame rate. Figure 7.6 depicts our quality predictions.

SGI Indigo 2s provides acceptable quality for Virtual Cockpit exercises with about 500 soldiers. Notice that in Figure 7.5, we had predicted that the Indigo 2s could support 1000 soldiers when we analyzed the frame rate alone. Application quality is determined by latency, jitter and missed updates as well as frame rate. Latency, jitter and missed updates all increase as the number of soldiers increase.

High-Performance Processors The quality of the Virtual Cockpit was improved by using high-performance processors. Is there further benefit from higher-performance processors?

We assume the network has sufficient bandwidth to handle all necessary updates in order to minimize the effects of the network. We compare the quality of the Virtual Cockpit with SGI Personal Irises and SGI Indigo 2s to the quality of the Virtual Cockpit with processors 15 times more powerful than the Indigo 2.[2]

Figure 7.7 shows the quality predictions for the Virtual Cockpit with different processors. The top curve is an SGI Personal Iris. The second curve is an SGI Indigo 2. The bottom curve is a processor 15 times more powerful than the Indigo 2. The horizontal line represents the acceptable quality limit. The "knee" in the curve for the 15x processor is where the processor decreases the frame rate in order to handle the updates from the other soldiers.

High-performance processors are crucial for acceptable Virtual Cockpit quality. SGI Personal Iris' are unable to deliver acceptable application quality. More powerful SGI Indigo 2s can deliver acceptable application quality for up to 500 soldiers. 15x's provides better application quality than Indigo 2s and can deliver acceptable application quality for up to 7000 soldiers.

High-Speed Networks With the Virtual Cockpit running on processor 15 times more powerful than the SGI Indigo 2, a T1 network will become saturated while supporting just 100's of soldiers. How much quality benefit will then be gained from a high-speed network? We compare the quality of the Virtual Cockpit with an Ethernet to that of the Virtual Cockpit with an ATM network. The ATM network transmits the update packets faster (155 Mbits/second versus 10 Mbits/second for an Ethernet). Past work has found jitter and missed updates in the ATM network

[2]Processor performance has approximately doubled every year for the last 5-10 years [56]. If this trend continues, the 15x processor will come along in about 8 years.

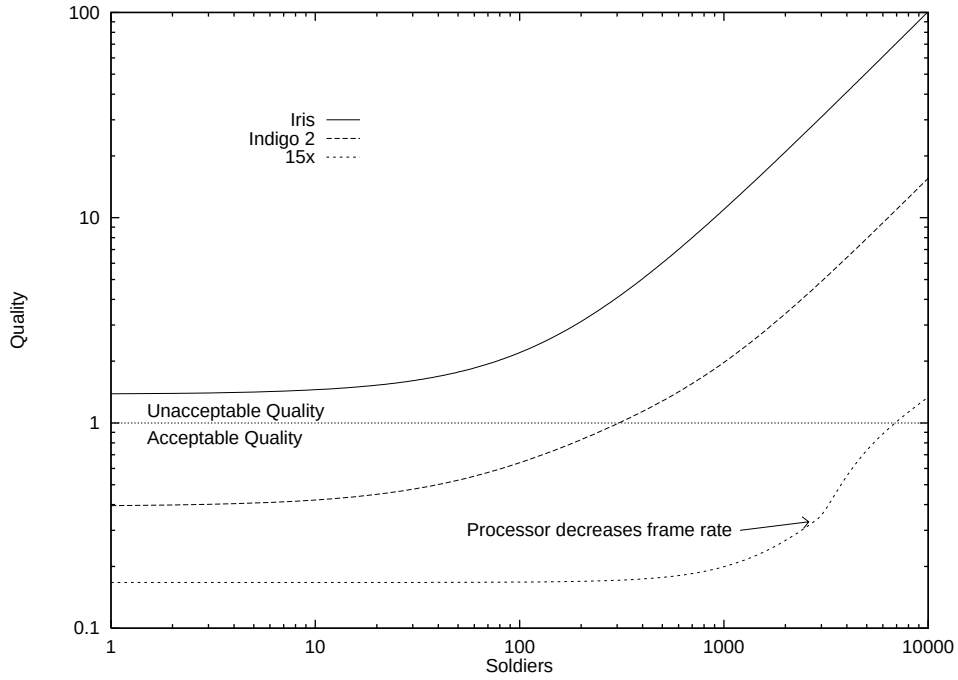

Figure 7.7: Virtual Cockpit Quality versus Soldiers. The three curves represent the quality predictions for three different processors. The top curve is an SGI Personal Iris. The second curve is an SGI Indigo 2. The bottom curve is a processor 15 times more powerful than the Indigo 2. The horizontal line represents the acceptable quality limit. Both the horizontal and vertical axes are in log scale.

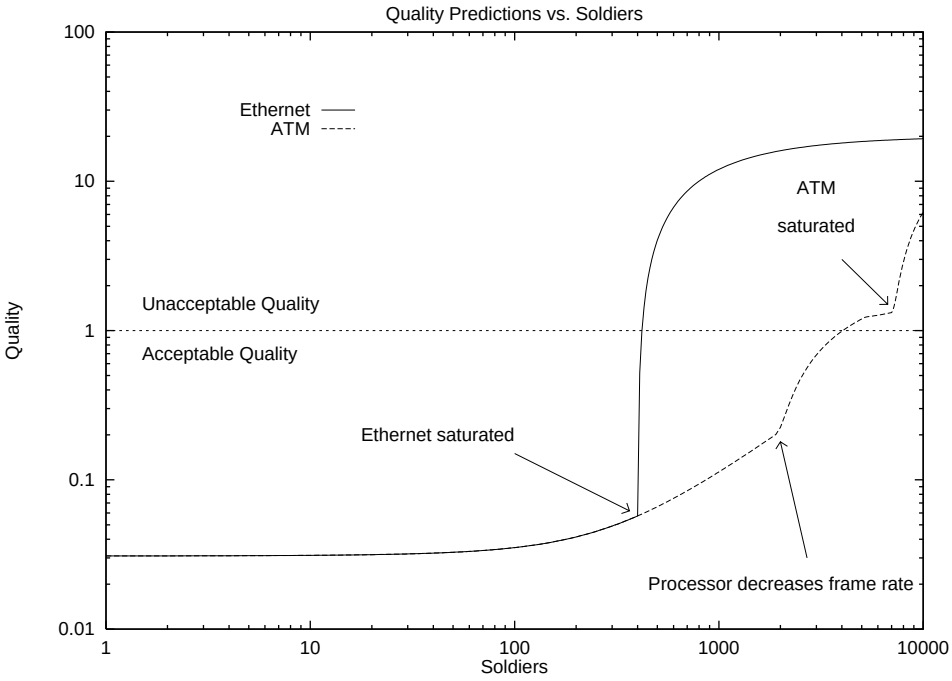

Figure 7.8: Virtual Cockpit Quality versus Soldiers. The two curves represent the quality predictions for an Ethernet and an ATM network. The horizontal line represents the acceptable quality limit. Both the horizontal and vertical axes are in log scale.

are the same as jitter and missed updates in the Ethernet [51]. We assume jitter and missed updates remain the same in high-speed networks. In order to make the network bandwidth more significant, we assume the maximum 17 updates per second.

Figure 7.8 shows the quality predictions for the Virtual Cockpit with different networks. The top curve is the quality predictions for an Ethernet. The lower curve is the quality predictions for an ATM. The steep increase in the Ethernet curve occurs when the Ethernet becomes saturated. At this point, the Virtual Cockpit begins to increasingly miss updates. The first bend in the ATM curve occurs when the processor must decrease the frame rate in order to process all updates. The second bend in the ATM curve occurs when the ATM becomes saturated.

High-speed networks are unimportant for the Virtual Cockpit quality until existing

networks reach saturation. The quality prediction curves for the Ethernet and the ATM are indistinguishable until the Ethernet becomes saturated. At this point, the ATM network greatly increases scalability. The network as the present bottleneck in application quality has been removed by the DIS use of dead reckoning and multicast.

7.7 Summary

Multi-person distributed multimedia applications stress all parts of a computer system. We have developed a quality planning model for distributed multimedia applications that allows us to investigate potential bottlenecks in application quality. We have applied our model to the Virtual Cockpit, a flight simulator used in Distributed Interactive Simulation (DIS). DIS is a virtual environment within which simulators may interact through simulation at multiple networked sites. In order for DIS to be effective for military training, simulation exercises must support up to 100,000 soldiers.

In applying our model to the Virtual Cockpit, we discovered:

- High-performance processors are crucial for acceptable Virtual Cockpit quality. Low-end processors are unable to deliver acceptable application quality. More powerful processors can deliver acceptable application quality for 1000's of soldiers.

- High-speed networks are unimportant for Virtual Cockpit quality until existing networks reach saturation. T1 and Ethernet networks will saturate after 2-3 generations of processor improvement. The network as the present bottleneck in application quality has been removed by the DIS use of dead reckoning and multicast.

- Dead reckoning greatly benefits the Virtual Cockpit quality, primarily due to its reduction of network load. The small update message size plus the infrequent

number of updates reduces the network load over a scheme where all simulators update their position with each frame. With dead reckoning, a T1 can support 10's of simulators, an Ethernet can support 100's of simulators and an ATM can support 1000's of simulators.

- Multicast is a crucial to maintain reduced network bandwidth. Even at the average update rate, a Virtual Cockpit exercise using unicast rapidly exceeds the bandwidth available on T1 and Ethernet networks.

We depict the above conclusions in Figure 7.9. The figure shows our measure of quality for the Virtual Cockpit. The user defined acceptable latency, jitter, data loss and missed updates determine a region of acceptable application quality, depicted by the shaded region. Each point is an instantiation of the application and the underlying computer system. All points inside the shaded region have acceptable quality, while those outside the region do not. There are four 100-soldier Virtual Cockpit instantiations shown: SGI Personal Iris' with Ethernet, SGI Personal Iris' with ATM, SGI Indigo 2s with Ethernet and SGI Personal Iris' with Ethernet and unicast. Only the Indigo 2s with Ethernet have acceptable application quality. The ATM does not significantly improve the quality of the Virtual Cockpit with the Personal Iris'. Using unicast instead of multicast greatly decreases the application quality for the Personal Iris' with Ethernet. Note that the graph does not show the fourth axis of quality, missed updates, because of the difficulty in depicting four dimensions. We chose to eliminate the missed updates axis because the predicted number of missed updates is constant for all system configurations.

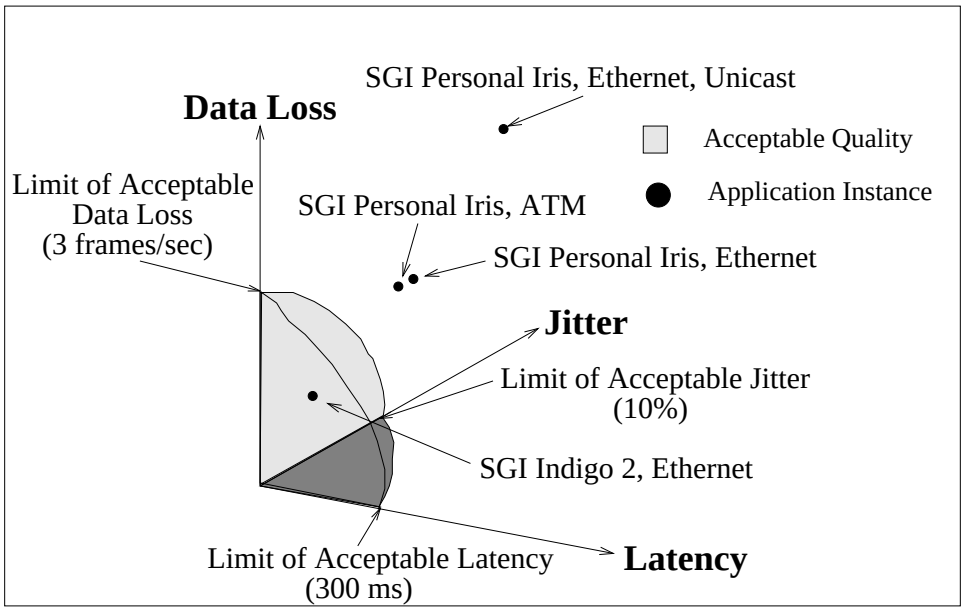

Figure 7.9: Virtual Cockpit Quality. The user defines the acceptable latency, jitter and data loss. These values determine a region of acceptable application quality, depicted by the shaded region. All points inside the shaded region have acceptable quality, while those outside the region do not. Four Virtual Cockpit instantiations are shown as points in this space. Note that the graph does not show the fourth axis of quality, missed updates, because of the difficulty in depicting four dimensions.

Chapter 8

Conclusions

Most of the work people do is in groups. People communicate best when they can draw pictures and use voice and body language. Computers supporting collaborative work can provide realistic person-to-person communication by using multimedia. Multimedia applications can also provide collaboration in synthetic environments through virtual reality. Today's explosive growth in fast networks and powerful workstations has the potential to support and even enhance group work through multimedia.

There are at least four reasons why multimedia is an important area for computer science research:

- *Teleconferences.* Multimedia has greatly enhanced the richness of computer-based conferencing. In the beginning, there was only email as a form of Internet communication. Soon after, there was the `tty`-based `talk` where two users could type synchronously on a split computer screen. Emoticons were developed to allow users to express their emotions in symbols in addition to words [91]. Today, audio and video enable people to use voice inflection, facial expressions and gestures to assist communication, making teleconference more life-like.

- *Environments.* Multimedia can enable environments that are not available in the real world. For instance, imagine a virtual reality environment that allows

scientists to explore the surface of Mars through multimedia data fed back from a robot on the planet's surface. And, as shown in Section 7, soldiers may soon train for combat on a virtual battlefield, reducing the costs and risks associated with "live" combat training.

- *Communication.* Multimedia brings the opportunity for communication with computers where previously communication was impossible. For instance, multimedia can enable people with visual impairment to operate with a computer through gesture recognition and voice synthesis.

- *Fun.* Multimedia makes computers fun. Multimedia applications have much more widespread appeal than do traditional text-based applications. Users who would otherwise not deal with computers will be attracted by multimedia bringing money into computer industry and more researchers into computer science.

Despite the real and potential benefits of multimedia, there are several obstacles to overcome in designing multimedia applications and systems. Multimedia and multi-user applications are more resource intensive than traditional text-based, single-user applications. In addition, multimedia applications have different performance requirements than do text-based applications. Text-based applications are sensitive to delay and loss, while multimedia applications are sensitive to delay and jitter. The bottlenecks to text-based application performance might lie in those components that induce delay, while the bottlenecks to multimedia applications might lie in the those components that induce the jitter. New techniques must be developed to identify bottlenecks in multimedia application performance.

We have developed a quality planning method for distributed collaborative multimedia applications that allows us to investigate potential bottlenecks in application quality. At the heart of our method is a model that allows us the predict the application performance from the user's perspective. Our model takes into account the components fundamental to multimedia applications: delay, jitter and data loss. Our

model allows us to investigate application bottlenecks by being adjustable to:

- *Number of Users.* Explicit support for collaboration in our computer tools has the potential to support and even enhance traditional group work. Our model allows us see how quality changes as the number of collaborating users increases.

- *Applications.* Our model can analyze new distributed multimedia applications by altering the user requirements, often without further micro or macro experiments.

- *Hardware and Architectures.* Our model can be adapted to hardware and methods of processing that are not available for experimentation either because of prohibitive costs, or because they are not yet built. Changing the model architecture and hardware components does yield less accurate predictions than predictions based on our micro experiments. However, additional micro and macro experiments can be incrementally performed as new hardware or architectures are explored.

- *Quality Metrics.* The quality metric we have presented in Section 2.3 has several useful characteristics: it has the properties of a linear mathematical metric; and it incorporates the three fundamental multimedia quality components: latency, jitter and data loss. However, there can be many possible quality metrics for a given application. Fortunately, the rest of our model is independent of the quality metric chosen. If new metrics are developed, they can be used in place of our quality metric.

When computing the amount of jitter in a multimedia stream, jitter researchers have used a variety of metrics. However, as our work in Section 3.3 shows, nearly all previously-used jitter metrics are statistically similar. Thus, jitter researchers can use a jitter metric most suitable to their research without fear of different result being obtained from an alternate metric.

Using the model in our method, we can explore the performance tradeoffs for a variety of multimedia applications as the underlying computers systems change. Among the system changes we can examine are changes in capacity, such as faster processors, networks or disks and scheduling, such as real-time priorities or delay buffering. We have applied our model to three emerging applications: audioconferences, flying and the virtual cockpit. There are three general trends that we have identified in analyzing these applications:

1. *Processors are King.* Processors are the bottleneck in performance for many multimedia applications. Audioconferences, Flying and the Virtual Cockpit all saw a dramatic increase in application performance with an increase in processor power.

2. *Networks are Queen.* Networks with more bandwidth often do not increase the quality of multimedia applications. For Audioconferences, a faster network did very little to improve application quality. In Flying, more network bandwidth did increase the performance for one user, but once that users' requirements were met, there was little benefit from more bandwidth. For the Virtual Cockpit, more bandwidth did not noticeably affect application quality at all, but it did allow more simultaneous users to train for combat when existing networks became saturated. Networks with more bandwidth do not benefit few-person multimedia applications but serve only to increase the scalability of the applications by allowing more simultaneous users.

3. *Share the Burden.* Application capacity requirements are not equally distributed across computer systems. Performance for many multimedia applications can be improved greatly by shifting capacity demand from computer system components that are heavily loaded to those that are more lightly loaded. And shifting capacity demand is *crucial* as the number of application users increases. For Audioconferences, silence deletion transfered load from the network to the pro-

cessor. While this decreased application quality for two audioconference users, it greatly increased application quality for three or more users. For Flying, application performance was totally unacceptable unless capacity demand was shifted from the processor to specialized hardware. Local flying, which would have shifted capacity demand from the server to the clients, required far too much network bandwidth to be feasible in the foreseeable future. For the Virtual Cockpit dead reckoning shifted capacity demand dramatically from the network, enabling current networks to support the tens of thousands of soldiers required for effective combat training.

Our objective in identifying application bottlenecks is to understand the system limits that will prevent applications from meeting users' needs. After identifying each bottleneck, we explore ways to reduce the effect of the bottleneck through improving system resources. We then explore the new bottlenecks that arise in the enhanced system. Our analysis at each stage is likely to overstate system performance, because we assume maximum possible performance of each system component. However, the bottlenecks we identify are likely to be bottlenecks in practice, and the design principles suggested by the analysis should ameliorate these bottlenecks in practice.

To conclude, the major contributions of this thesis are:

- A multimedia application quality model and implementation method that enables the prediction of application performance and evaluation of system design tradeoffs.

- Detailed performance predictions for three distributed collaborative multimedia applications: an audioconference, a "flying" interface to a 3D scientific database and a collaborative flight simulator called the Virtual Cockpit.

- The effects of system improvements on the performance of these multimedia applications.

- A demonstration of measures of jitter that have been used by jitter researchers showing all reasonable choices of jitter metrics are statistically similar.
- Experiment-based studies of the effects of high-performance processors, real-time priorities and high-speed networks on jitter.
- Predictions on the importance of application jitter reduction techniques in the future.

Chapter 9

Future Work

The research in this thesis focuses on a new, rapidly expanding area of computer science. As such, it paves the way into an exciting new area of computer science, allowing access to several new areas for future research.

The taxonomy given in Section 2 is an ongoing document. Moreover, there are several areas of the current taxonomy that support multimedia application performance research that we have not had the opportunity to examine. Most notably, we would like to see a more extensive investigation of the effects of scheduling and reservation on multimedia application performance. In addition, our taxonomy can possibly support a cost analysis of the tradeoffs between network bandwidths, disk data rates and processor throughputs. Such an economic analysis could determine what system configuration will be the most economical for improving computer support for multimedia.

For some applications, there is potential for interaction effects among the quality events. For example, 3-d graphics applications have multiple factors affecting users' perception of objects and different combinations of requirements may yield satisfactory results. Such applications may even have a non-convex region of acceptable quality. Future research into new quality metrics appropriate for these applications may be required.

Applications that have changing user requirements present another challenge. For example, users doing remote problem-solving via a video link, may want to maximize frame rate at the expense of frame resolution while they are identifying the location of the problem. Once the problem is located, they may want to maximize frame resolution at the expense of frame rate (perhaps even wanting a still image) to best identify the problem. As presented, our model does not allow specification of dynamic user requirements. One possible solution would be to apply a separate quality planning model to each set of user requirements specified. The model that had the poorest quality for a given system configuration could then be examined more closely to determine the application quality bottleneck within.

Systems that need to support more than one simultaneous application present another challenge. Our model currently assumes the computer system is dedicated to the application we are analyzing. We might further develop our techniques to determine where potential bottlenecks might arise when several applications are running at once [23].

For the Virtual Cockpit, an interesting processing decision arises in the processing of dead reckoning updates. Dead reckoning allows a tradeoff among three factors: network bandwidth, processor load and simulation accuracy. After a dead reckoning computation, a simulator will have the perceived position of other simulators. This perceived position will be inaccurate up to a pre-set threshold. Lower thresholds provide a more accurate representation of other simulators, but increase the number of updates required, causing more network load. Higher order dead-reckoning decreases the number of updates sent, but increases processor load. Our model could be used to determine the effect on application quality for the various dead reckoning parameters, possibly recommending a specific accuracy and algorithm to use.

For our predictions, we assume that processor performance will continue to reflect workstation performance. If other workstation components, in particular the data bus, do not continue to scale with processor performance, future work would include

extending our model to discover which workstation components are bottlenecks to application quality. In addition, for some predictions, especially those for Flying (see Section 6), we assumed specialized hardware to shift capacity demand from the processor. A careful study of the effects of flying hardware will determine bottlenecks in current proposed flying systems.

We studied the effects of real-time priorities on jitter when used at both the sender and receiver. Real-time priorities may not significantly reduce jitter when used at only the sender or at only the receiver. If this is true, real-time priorities may only be effective in reducing jitter for a local area network, without an intervening router. On a Wide Area Network, especially the Internet, real-time priorities may not be available on all routers, reducing the effectiveness of real-time priorities on the sender and receiver. In this case, buffering techniques will be needed.

Another possible potential jitter reduction technique would be a real-time network protocol. Future work includes experiments to measure the effects of real-time network protocol on jitter. However, network delivery of data within strict jitter bounds does not significantly help the sender and receiver. They must still worry about internal jitter due to queuing in the operating system. Stamping out jitter in the network does not eliminate the need for jitter management code in the hosts.

In Section 3.7 we made predictions on the magnitude of jitter on a LAN in the future. MOS studies or other appropriate techniques can be used to determine the actual threshold of human perception of jitter. The threshold for human perception of jitter in an audio stream is probably different than the threshold of human perception of jitter in a video stream as the eye is better able to smooth over occasional glitches in video than the ear is to account for glitches in audio.

The layout of multimedia data on disks and disk scheduling strategies may be crucial for reducing jitter for some multimedia applications. Database management systems (DBMS) that manage multimedia data may further impact the amount of jitter seen by an application that accesses that data. Future work would involve

exploring the effects of different disk strategies and DBMS techniques for reducing jitter.

This thesis presents a method for determining application quality given user requirements and system configurations. Although the analysis presented here took many years to complete, we can dream of packaging our knowledge into a fast, flexible tool usable by application developers and visionaries. Imagine a our model wrapped up in a smooth, sexy graphical user interface. The user enters application performance parameters at an appropriate level of detail, perhaps by choosing a bit rate or the frame rate, frame size and color, or even by choosing the most pleasing picture from a multimedia sample. The user enters system parameters with similar ease, specifying raw processing power and network bandwidth, or selecting a workstation and network from a list of computer system hardware. Our model, the heart of the tool, determines the quality that the application would have given the specified information. The output results appear as rich and varied as the graphs shown in this thesis, with users able to quickly and easily identify the potential bottlenecks in application performance by varying the hardware, number of users and application requirements. The benefits of packaging our model in this way would be enormous. Our model would save developers countless amounts of time and money by quickly being able to identify potential performance bottlenecks before the undertaking of expensive, time-consuming system design and implementation. More importantly, our model would enable new forms of interaction for all computer users, including ourselves. We would immerse ourselves in computer applications rich with graphics, audio and video, and interact with each other across both time and space, finally realizing the full potential of computers and multimedia.

Bibliography

[1] Sparcstation audio programming. Technical report, Sun Microsystems, 1991.

[2] Richard C. Allen, Chris Bottcher, Phillip Bording, Pat Burns, John Conery, Thomas R. Davies, James Demmel, Chris Johnson, Lakshmi Kantha, William Martin, Geoffrey Parks, Steve Piacsek, Dan Pryor, Tamar Schlick, M.R. Strayer, Verena M. Umar, Robert Voigt, Jerrold Wagener, Dave Zachmann, and John Ziebarth. *The Computational Science Education Project (CSEP)*. January 1997. The URL for this book is http://csep1.phy.ornl.gov/csep.html. This was, in 1993, the first complete textbook available on the World Wide Web.

[3] David P. Anderson, Ralf Guido Herrtwich, and Carl Schaefer. SRP: a resource reservation protocol for guaranteed-performance communication in the Internet. Report UCB/CSD 90-562, University of California, Berkeley, Computer Science Division, Berkeley, CA, USA, February 1990.

[4] David P. Anderson, Shin-Yuan Tzou, Robert Wahbe, Ramesh Govindan, and Martin Andrews. Support for continuous media in the DASH system. In *Proc. 10th Intl. Conf. Distributed Computing Systems (ICDCS-10)*, Paris, France, May 28-June 1 1990. IEEE.

[5] Sally Apgar. New weapon in retailing: Technology. *Minneapolis Star Tribune*, August 1994.

[6] Ronnie T. Apteker, James A. Fisher, Valentin S. Kisimov, and Hanoch Neishlos. Video acceptability and frame rate. *IEEE Multimedia*, pages 32 – 40, Fall 1995.

[7] Barberis and Pazzaglia. Analysis and optimal design of a packet voice receiver. *IEEE Transactions on Communication*, February 1980.

[8] John Bates and Jean Bacon. Supporting interactive presentation for distributed multimedia applciations. *Multimedia Tools and Applications*, 1(1), March 1995.

[9] Salvador Bayarri, Marcos Fernandez, and Mariano Perez. Virtual reality for driving simulation. *Communications of the ACM*, 39(5), May 1996.

[10] Bharat Bhargava, Enrique Mafla, and John Riedl. Communication in the Raid distributed database system. *International Journal on Computers and ISDN Systems*, 21:81–92, 1992.

[11] David R. Boggs, Jeffrey C. Mogul, and Christopher A. Kent. Measured capacity of an Ethernet: Myths and reality. In *Proceedings of the SIGCOMM Conference*, August 1988.

[12] M. Borwn, J. Foote, G. Jones, K. Sparck Jones, and S. Young. Open-vocabulary speech indexing for voice and video mail retrieval. In *Proceedings of the Fourth ACM International Multimedia Conference*, Boston, MA, November 1996. ACM.

[13] Tim Browning. *Capacity Planning for Computers.* Academic Press, 1995.

[14] Luis-Filipe Cabrera, Edward Hunter, Michael J. Karels, and David A. Mosher. User-process communication performance in networks of computers. *IEEE Transactions on Software Engineering*, 14(1), January 1988.

[15] J. Carlis, J. Riedl, A. Georgopoulos, G. Wilcox, R. Elde, J. H. Pardo, K. Ugurbil, E. Retzel, J. Maguire, B. Miller, M. Claypool, T. Brelje, and C. Honda. A

zoomable DBMS for brain structure, function and behavior. In *International Conference on Applications of Databases*, June 1994.

[16] Stephen Casner and Stephen Deering. First IETF Internet audiocast. *ACM SIGCOMM*, pages 92 – 97, 1992.

[17] Todd Cavalier, Ravinder Chandhok, James Morris, David Kaufer, and Chris Neuwirth. A visual design for collaborative work: Columns for commenting and annotation. In *Proceedings of IEEE HICSS*, 1990.

[18] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. *Computer Communication Review*, 22(4), July 1992.

[19] M. Claypool, J. Riedl, J. Carlis, G. Wilcox, R. Elde, E. Retzel, A. Georgopoulos, J. Pardo, K. Ugurbil, B. Miller, and C. Honda. Network requirements for 3D flying in a zoomable brain database. *IEEE JSAC Special Issue on Gigabit Networking*, 13(5), June 1995.

[20] Mark Claypool, Joe Habermann, and John Riedl. The effects of high-performance processors, real-time priorities and high-speed networks on jitter in a multimedia stream. Technical Report TR-97-023, University of Minnesota Department of Computer Science, June 1997.

[21] Mark Claypool and John Riedl. Silence is golden? The effects of silence deletion on the CPU load of an audio conference. Technical Report TR-93-81, University of Minnesota Department of Computer Science, 1993.

[22] Mark Claypool and John Riedl. Silence is golden? The effects of silence deletion on the CPU load of an audio conference. In *Proceedings of IEEE Multimedia*, Boston, May 1994.

[23] Mark Claypool and John Riedl. Analysis of processor power versus network bandwidth. Technical Report TR-95-042, University of Minnesota, 1995.

[24] Mark Claypool and John Riedl. A quality planning model for distributed multimedia in the virtual cockpit. In *Proceedings of ACM Multimedia*, pages 253 – 264, November 1996.

[25] The DIS Steering Committee. The DIS vision - a map to the future of distributed interactive simulation. Technical report, Institute for Simulation and Training, May 1994.

[26] American Technology Corporation. *HyperSonic Sound System (HSS) – (A New Method of Sound Reproduction).* July 1997. The URL for this document can be found at http://www.atcsd.com/HTML/sound2.html.

[27] Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet: A local area network data link layer and physical layer specification, September 1980.

[28] Standard Performance Evaluation Corporation. SPEC primer. July 1994. The SPEC primer is frequently posted to the newsgroup comp.benchmarks. SPEC questions can also be sent to spec-ncga@cup.portal.com.

[29] H.J. Curnow and B.A Wichmann. A synthetic benchmark. *The Computer Journal*, 19(1), 1976.

[30] Jay Devore and Roxy Peck. *Statistics – The Exploration and Analysis of Data.* Wadsworth, Inc., second edition edition, 1993.

[31] Stanford Diehl. Forging a new medium. *Byte*, July 1996.

[32] Spiros Dimolitsas, Franklin L. Corcoran, and John G. Phipps Jr. Impact of transmission delay on ISDN videotelephony. In *Proceedings of Globecom '93* –

IEEE Telecommunications Conference, pages 376 – 379, Houston, TX, November 1993.

[33] Dick Dobson. Make access and the web work together. *Byte*, October 1996.

[34] Jack J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee, February 1994. To obtain a postscript copy, send email to netlib@ornl.gov with message body: send performance from benchmark.

[35] Chip Elliot. High-quality multimedia conferencing through a long-haul packet network. In *Proceedings of the First ACM International Conference on Multimedia*, pages 91 – 98, New York, NY, 1993.

[36] T.T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194 – 201, August 1992.

[37] Hans Eriksson. MBONE: The multicast backbone. *Communications of the ACM*, 37(8):54, August 1994. A FAQ on mbone can be found at ftp://venera.isi.edu/mbone/faq.txt.

[38] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. *Lecture Notes in Computer Science*, 1018, 1995.

[39] Domenico Ferrari. Delay jitter control scheme for packet-switching internetworks. *Computer Communications*, 15(6):367–373, July 1992.

[40] J.W. Forgie and C.W. McElwain. Some comments on NSC note 78 'effects of lost packets on speech intelligibility.'. Technical Report Network Speech Compression Note 92, M.I.T., Lincoln Laboratory, March 1976.

[41] Daniel Frankowski and John Riedl. Hiding jitter in an audio stream. Technical Report Technical Report 93-50, University of Minnesota Department of Computer Science, 1993.

[42] Krzysztof Frankowski. *Definition by Professor Krzysztof Frankowski.* Computer Science Department, University of Minnesota, May 1997.

[43] Steve Gillmor. Notes opens up to the web. *Byte*, October 1996.

[44] Timoth A. Gonsalves. Packet-voice communication on an ethernet local computer network: an experimental study. *Communications of the ACM*, pages 178–185, 1983.

[45] Walter Goralski and Gary Kessler. *FIBRE CHANNEL: Standards, Applications, and Products.* December 1995. The URL for this document can be found at http://www.hill.com/personnel/gck/fibre_channel.html.

[46] R. Govindan and D. Anderson. Scheduling and IPC mechanisms for continuous media. *ACM Operating Systems Review*, 25(5), October 1991.

[47] Mathew Gray. *Internet Statistics – Growth and Usage of the Web and the Internet.* 1997. The URL for this document can be found at http://www.mit.edu:8001/people/mkgray/net/.

[48] Rick Grehan. NT in real time. *Byte*, October 1996.

[49] Mike Guidry and Mike Strayer. *Computational Science for the Physical and Life Sciences.* January 1997. The URL for this book is http://csep1.phy.ornl.gov/guidry/phys594/phys594-root.html. This course is completely online, with lectures delivered electronically from the network, and student exercises turned in as Web (html) documents.

[50] J.L. Gustafson and Q.O. Snell. HINT: A new way to measure computer performance. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, volume 2, pages 392 – 401, IEEE Computer Society Press, January 1995. Technical report available via ftp from ftp.scl.ameslab.gov.

[51] Joe Habermann and John Riedl. Using real-time priorities to eliminate jitter in a multimedia stream. Technical report, University of Minnesota Department of Computer Science, January 1996.

[52] Matti Hamalainen, Andrew B. Whinston, and Svetlana Vishik. Money in electronic commerce: Digital cash, electronic fund transfer and ecash. *Communications of the ACM*, 39(6), June 1996.

[53] Charles Hansen and Stephen Tenbrink. Impact of gigabit network research on scientific visualization. *IEEE Computer*, May 1993.

[54] John H. Hartman and John K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, August 1995.

[55] Edward P. Harvey and Richard L. Shaffer. Capability of the distributed interactive simulation networking standard to support high fidelity aircraft simulation. In *Proceedings of the 13th Interservice/Industry Training Systems Conference*, 1993.

[56] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 1990.

[57] William L. Hibbard, Brian E. Paul, David A. Santek, Cahrles R. Dyer, Andre L. Battaiola, and Marie-Francoise Voidrot-Martinez. Interactive visualization of earth and space science computations. *IEEE Computer*, 27(7):65 – 72, July 1994.

[58] A. Hopper. Pandora – an experimental system for multimedia applications. *ACM Operating Systems Review*, 24(2), April 1990.

[59] JJ Garcialunaaceves HP Dommel. Floor control for multimedia conferencing and collaboration. 5(1):23 – 38, January 1997.

[60] Satoru Iai, Takaaki Kurita, and Nobuhiko Kitawaki. Quality requirements for multimedia communcation services and terminals – interaction of speech and video delays. In *Proceedings of Globecom '93 – IEEE Telecommunications Conference*, pages 394 – 398, Houston, TX, November 1993.

[61] Magid Igbaria and Snehamay Banerjee. Computer capacity planning management: Definitions and methodology. *Journal of Information Technology*, (9):213 – 221, 1994.

[62] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[63] K. Jeffay, D. Stone, and D. Poirier. Yartos – kernel support for efficient, predictable real-time systems. In *Joint IEEE Workshop on Real-Time Operating System and Software and IFAC/IFIP Workship on Real-Time Programming*, pages 8 – 13, May 1991.

[64] K. Jeffay, D. Stone, and F. Smith. Kernel support for live digital audio and video. *Computer Communications*, 15(6), 1992.

[65] K. Jeffay, D. Stone, and F. Smith. Transport and display mechanisms for multimedia conferencing across packet-switched networks. *Computer Networks and ISDN Systems*, 26(10), July 1994.

[66] K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith. Adaptive, best-effort, delivery of audio and video data across packet-switched networks. In *3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.

[67] Sanjay K. Jha and Bruce R. Howarth. Capacity planning of LAN using network management. In *Proceedings of Conference on Locacl Computer Networks*, pages 425 – 430, Washington D.C., 1994.

[68] Saimin Jin, Dhadesugoor R. Vaman, and Divyendu Sina. A performance mangement framework to provide bounded packet delay and variance in packet switched networks. *Computer networks and ISDN Sytems*, pages 249 – 264, September 1991.

[69] Rick Jones. *Netperf*. Hewlett-Packard, 1995. The netperf home page can be found at http://www.cup.hp.com/netperf/NetperfPage.html.

[70] Henry C. Lucas Jr. Performance evaluation and monitoring. *Computing Surveys*, 3(3):78 – 91, September 1971.

[71] Kalkbrenner. Quality of servie (qos) in distributed hypermedia systems. In *Proceedings of the 2nd International Workshop on Principles of Ducment Processing*, 1994.

[72] E.R. Kandel, J.H. Schwartz, and T.M. Jessel. *Principles of Neural Science, 3rd Ed.* Appleton & Lange, Norwalk, CT, 1991.

[73] Jonathan Kay and Joseph Pasquale. Measurement, analysis, and improvement of UDP/IP throughput for the DECstation 5000. Technical report, University of California at San Diego, 1993.

[74] S. Khanna, M. Serbree, and J. Zolnowsky. Realtime scheduling in SunOS 5.0. In *Proceedings of the Winter '92 Usenix Conference*, 1992.

[75] Kleinrock and Naylor. Stream traffic communication in packet switched networks: Destination buffering considerations. *IEEE Transactions on Communications*, COM-30(12):2527 – 2534, December 1982.

[76] Peter Kroon and Kumar Swaminathan. A high-quality multirate real-time CELP coder. *IEEE JSAC Selected Areas in Communications*, 10(5), June 1992.

[77] Edward D. Lazowska, John Zahorjan, David R. Cheriton, and Willy Zwaenepoel. File access performance of diskless workstations. *Transactions on Software Engineering*, 4(3):238–268, August 1986.

[78] S. Leffler, M. McKusick, M. Karels, and J. Quarterman. *The Design and Implementation of the 4.3BSD Unix Operating System.* Addison-Wesley Publishing Company, 1989.

[79] Mengjou Lin, Jenwei Hsieh, David Du, and James MacDonald. Performance of high-speed network I/O subsystems: Case study of a Fibre Channel network. In *Proceedings of Supercomputing '94*, November 1994.

[80] Mengjou Lin, Jenwei Hsieh, David H.C. Du, Joseph P. Thomas, and James A. MacDonald. Distributed network computing over local ATM networks. *ATM LANs: Implementation and Experience with An Emerging Technology*, 1995.

[81] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz. NPSNET: A network software architecture for large scale virtual environments. *Presence*, 3(4):265 – 287, October 1994.

[82] Vahid Mashayekhi, Janet Drake, Wei-Tek Tsai, and John Riedl. Distributed, collaborative software inspection. *IEEE Software*, 10(5), September 1993.

[83] Vahid Mashayekhi, Chris Feulner, and John Riedl. CAIS: Collaborative Asynchronous Inspection of Software. In *The Second ACM SIGSOFT Symposium on the Foundations of Software Engineering.* Association of Computing Machinery, December 1994.

[84] Michael J. Massimino and Thomas B. Sheridan. Teleoperator performance with varying force and visual feedback. In *Human Factors*, pages 145 – 157, March 1994.

[85] W. Dean McCarty, Steven Sheasby, Philip Amburn, Martin R. Stytz, and Chip Switzer. A virtual cockpit for a distributed interactive simulation. *IEEE Computer Graphics and Applications*, January 1994.

[86] F.M McMahon. The livermore fortran kernels: A computer test of numerical performance range. Technical Report UCRL-55745, Lawrence Livermore National Laboratory, University of California, December 1986.

[87] Evi Nemeth, Garth Snyder, and Scott Seebass. *Unix System Adminstration Handbook*. Prentice Hall, 1989.

[88] Judith S. Olson, Gary M. Olson, and David K. Meader. What mix of video and audio is useful for remote real-time work? In *Proceedings of CHI'95 – Proceedings of the Conference in Human Factors in Computing Systems*, pages 362 – 368, 1995.

[89] Davis Yen Pan. Digital audio compression. *Digital Technical Journal*, 5(2):28 – 40, Spring 1993.

[90] Craig Partridge. *Gigabit Networking*. Addison-Wesley, 1994.

[91] Pastmaster. *Smilies Unlimited*. June 1997. The URL for this document can be found at http://www.czweb.com/smilies.htm.

[92] Ketan Patel, Brian C. Smith, and Lawrence A. Rowe. Performance of a software mpeg video decoder. In *Proceedings of ACM Multimedia*, Anaheim, CA, 1993.

[93] Stephen Travis Pope. A taxonomy of computer music. *Computer Music Journal*, 18(1), 1994. This article can be found at: http://www-mitpress.mit.edu/Computer-Music-Journal/EdNotes/Taxonomy.

[94] L. R. Rabiner and M. R. Sambur. An algorithm for determining the endpoints of isolated utterances. *The Bell System Technical Journal*, February 1975.

[95] Lynda Radosevich. Retailer swings its partners to edi. *Computerworld*, 27, September 1993.

[96] Ramachandran Ramjee, Jim Kurose, Don Towsley and Henning Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies on Networking for Global Communciation. Volume 2*, pages 680–688, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[97] Andy Reinhardt. Building the data highway. *Byte*, March 1994.

[98] John Riedl, Vahid Mashayekhi, Jim Schnepf, Mark Claypool, and Dan Frankowski. SuiteSound: A system for distributed collaborative multimedia. *IEEE Transactions on Knowledge and Data Engineering*, August 1993.

[99] Lawrence A. Rowe and Brian C. Smith. A continuous media player. In *3rd International Workshop on Network and OS Support for Digital Audio and Video*, 1992.

[100] Radhika R. Roy. Networking contraints in multimedia conferencing and the role of ATM networks. *AT&T Technical Journal*, July/August 1994.

[101] Thomas M. Ruwart and Matthew T. O'Keefe. Storage and interfaces '94. In *Proceedings of the Storage and Interfaces '94*, Santa Clara, CA, January 1994.

[102] James Schnepf, Vahid Mashayekhi, John Riedl, and David Du. Computer supported, participative, media-rich education. *Educational Technology Review*, 1994.

[103] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. *RFC 1889*, January 1996.

[104] Henning Schulzrinne. Voice communications across the internet: A network voice terminal. Technical report, University of Massachussetts Department of Electrical Engineering, August 1992.

[105] Shashi Shekhar, Sivakumar Ravada, Greg Turner, Douglas Chubb, and Vipin Kumar. Load balancing in high performance GIS: Partitioning polygonal maps. In *Proceedings of the Internationall Symposium on Large Spatial Databases*, 1995.

[106] John F. Shoch and Jon A. Hupp. Measured performance of an Ethernet local network. *Communications of the ACM*, 23(12):711–720, December 1980.

[107] Jaswinder Pal Singh, Anoop Gupta, and Marc Levoy. Parallel visualization algorithms: Performance and architecural implications. *IEEE Computer*, 27(7):45 – 55, July 1994.

[108] B. M. Slotnick and C. M. Leonard. *A Stereotaxic Atlas of the Albino Mouse Forebrain.* Superintendent of Documents, US Government Printing Office, Washington DC, 1975.

[109] Ben Smith. The Byte unix benchmarks. *Byte*, pages 273 – 277, March 1990.

[110] Brian C. Smith and Lawrence A. Rowe. A new family of algorithms for manipulating compressed images. *IEEE Computer Graphics and Applications*, September 1993.

[111] Monica Snell. Internet-savvy capacity-planning software helps guide network growth. *LAN Times*, September 1996. The URL for this journal can be found at http://www.lantimes.com/.

[112] Michael V. Stein and John Riedl. The effects of transport method on the quality of audioconferences with silence deletion. Technical report, University of Minnesota Department of Computer Science, June 1995.

[113] Donald Stone and Kevin Jeffay. An empirical study of delay jitter management policies. *Multimedia Systems*, 1995.

[114] Merryanna Swartz and Daniel Wallace. Effects of frame rate and resolution reduction on human performance. In *Proceedings of IS&T's 46th Annual Conference*, Munich, Germany, 1993.

[115] T. Talley and K. Jeffay. Two-dimensional scaling techniques for adaptive, rate-based transmission control of live audio and video streams. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 247 – 254, October 1994.

[116] Tanenbaum. *Networks*. Prentice Hall, 1989.

[117] R. Terek and J. Pasquale. Experiences with audio conferencing using the X window system, UNIX, and TCP/IP. In *Proceedings of the Summer '91 Usenix Conference*, pages 405 – 417, 1991.

[118] Robert H. Thomas, Harry C. Forsdick, Terrence R. Crowley, Richard W. Schaaf, Raymond S. Tomlinson, Virginia M. Travers, and George G. Robertson. Diamond: A multimedia message system built on a distributed architecture. *IEEE Computer*, 18(3), December 1985.

[119] Don Tolmie and Don Flanagan. *HIPPI: It's Not Just for Supercomputers Anymore*. Los Alamos National Laboratory, 1997. The URL for this document can be found at http://nocone.lanl.gov/lanp/HIPPI_Data_Comm.html.

[120] Claudio Topolcic. Experimental internet stream protocol, version 2 (ST-II). *RFC 1190*, October 1990.

[121] Dinesh C. Verma, Hui Zhang, and Domenico Ferrari. Delay jitter control for real-time communication in a packet switching network. *IEEE Computer*, pages 35 – 43, 1991.

[122] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, April 1991.

[123] Peter Wayner. Inside the NC. *Byte*, November 1996.

[124] R.P. Weicker. Dhrystone: A synthetic systems programming benchmark. *Communications of the ACM*, 27(10):1013 – 1030, October 1984.

[125] Duminda Wijesekera and Jaideep Srivastava. Quality of service metrics for continuous media. *Protocols for Multimedia Systems*, pages 269 – 289, 1995.

[126] Brian L. Wong. *Capacity Planning for Solaris Servers*. Prentice Hall, 1996.

[127] David K.Y. Yau and Simon S. Lam. Adaptive rate-controlled scheduling for multimedia applications. In *Proceedings of the Fourth ACM International Multimedia Conference*, Boston, MA, 1996.

[128] J.A. Zebarth. Let me be me. In *Proceedings of Globecom '93 – IEEE Telecommunications Conference*, pages 389 – 393, Houston, TX, November 1993.

[129] Hongjiang Zhang, Chien Yong Low, and Stephen W. Smoliar. Video parsing and browsing using compressed data. *Multimedia Tools and Applications*, 1(1), March 1995.