Media and Protocols for Multimedia Communications over Disparate Networks

Stephen Jacobs

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Graduate School of Arts and Sciences

Columbia University

1998

ABSTRACT

Media and Protocols for Multimedia Communications over Disparate Networks

Stephen Jacobs

It is widely recognized that multimedia applications will be a "killer-app." These applications depend so heavily on the type of network used that no real time application can claim to be truly network-independent. We present two disparate networks and address the challenges in building real-time applications for each. First we examine a network which can provide guarantees on bandwidth and delay. Despite these network guarantees, the interaction between the server and the client can severely affect the performance of the system, including scalability. We present a model for providing an end-to-end guarantee on Quality of Service (QoS) by mapping QoS parameters at the video server to subjective parameters at the video client. We then present a network which cannot guarantee QoS. This network requires special protocols and algorithms for maintaining quality for the end user. Applications operating in this environment must continually adapt to the state of the network and must do so in a way that does not damage the performance of other traffic on the network. This involves extracting implicit or explicit information about the state of the network. It also requires adapting the data (audio, video or images) so that they do not demand more resources than the network can provide. We present a variety of codecs for different media and show ways in which they can be made adaptable. We then present a new semi-reliable transport protocol for streaming real time media on today's Internet. The protocol uses the TCP congestion window to pace the delivery of data into the network, without using other TCP algorithms that are poorly suited to real time protocols. For this reason, our protocol competes fairly with all other Internet data by optimally sharing the available bandwidth. We then examine the performance of TCP, and hence the performance of our protocol, over wireless networks. A model for TCP utilization in the presence of random wireless errors is presented which is much more tractable than previous work. This model is used to draw conclusions about the utilization of TCP over wireless Local Area Networks.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Relationship of Network to Media 3

1.3 Challenges of Streaming Multimedia 4

1.3.1 Guaranteed Quality 4

1.3.2 Server Scalability 5

1.3.3 Client Heterogeneity 5

1.3.4 Network Heterogeneity 6

1.4 Thesis Contributions 6

1.4.1 QoS Streaming 6

1.4.2 Non-QoS Streaming 7

I QoS Streaming **11**

2 Video on Demand over ATM **13**

2.1 Introduction 13

2.2 VoD Model 15

2.2.1 Server Model 15

2.2.2 Network Model 17

2.2.3 Client Model 18

2.3 Columbia's Video on Demand Testbed 19

2.3.1 Testbed Server 20

2.3.1.1 Video Pump Environment 23

2.3.1.2 Architecture 25

2.3.1.3 Resource Reservation 27

2.3.1.4 High Resolution Timers 29

2.3.2 Testbed Network 37

2.3.3 Testbed Clients 38

2.4 Concluding Remarks 38

3 Video QoS Parameter Mapping: Server to Client 40

3.1 Introduction 40

3.2 Overview of Video Server QoS Parameters 42

3.2.1 Delay Distribution Moments 42

3.2.2 Traffic Control Metrics 43

3.3 Experimental Choice of QoS Parameters 45

3.3.1 Delay Distribution Moments 45

3.3.2 Traffic Control Metrics 46

3.3.3 Cross Traffic 47

3.4 Mapping Qos Parameters to the Client 50

3.5 Schedulable Region 54

3.5.1 Definition of Schedulable Region 54

3.5.2 Methodology and Example of Schedulable Region 56

3.6 Concluding Remarks 58

II Non-QoS Streaming **63**

4 Adaptable Media for Non-QoS Streaming **65**

4.1 Introduction 65

4.2 Video 66

4.2.1 MPEG 66

4.2.1.1 Feedback to the Encoder 68

4.2.1.2 Frame Dropping 68

4.2.1.3 Dynamic Rate Shaping 69

4.2.2 H.263+ 70

4.2.3 Highly Scalable Codecs 70

4.3 Audio 70

4.3.1 Silence Detection 70

4.3.2 Previous Work 76

4.3.3 Non-Linear Silence Detection in the μ-law Domain 78

4.3.4 Results 85

4.4 Image 89

4.4.1 JPEG 89

4.4.2 Flashpix 90

4.5 Text 90

4.6 Concluding Remarks 91

5 Protocol for Non-QoS Streaming **92**

5.1 Introduction 92

5.2 Wired Internet 94

5.2.1 Real-time Transport Protocol 94

5.2.2 The Proposed Protocol 95

5.2.2.1 Internet Friendly Bandwidth Estimation 97
5.2.2.2 Rate Control 100
5.2.2.3 Retransmissions 104
5.2.3 The Application 106
5.2.4 Internet Results 107
5.2.4.1 Bandwidth Fairness 107
5.2.4.2 Controlled Bottleneck 110
5.2.4.3 Wide Area Network 113
5.3 Wireless Internet 116
5.3.1 Previous Work 117
5.3.2 Analytical Formulation: TCP 118
5.3.2.1 Wired Network 119
5.3.2.2 Wireless Network 120
5.3.3 Performance of Model 123
5.3.3.1 Comparison with Previous Models 123
5.3.3.2 Comparison with Simulations 125
5.4 Wireless LAN 127
5.4.1 Previous Work 127
5.4.2 Analytical Formulation 129
5.4.3 Performance over Wireless LAN 134
5.5 Concluding Remarks 138

6 Conclusions **140**

References **143**

List of Figures

2-1 Video Server Model. 16

2-2 Video Client Model. 18

2-3 Columbia's Multimedia Testbed. 20

2-4 Columbia's VoD Testbed Architecture. 21

2-5 CV^2 of PDU interarrival times vs. PDU size for a 2.5 Mbps video stream. 24

2-6 Video Pump Object. 26

2-7 PDU interarrival times for 2.5 Mbps video stream measured at the output of the video server. 27

2-8 Characterization of the timers. Actual time waited by the timer over 1000 trials. 31

2-9 Experimental configuration for HP Broadband Analyzer. 32

2-10 Deviation from the average PDU interarrival time for a 5 Mbps stream measured at the output of the video pump through one ATM switch. 33

2-11 Probability Mass Function for the interarrival times. 34

2-12 Interarrival times for a video pump operating at 5 Mbps while another video pump is running on the same processor. 35

2-13 Example R_f curve. R_{f_1} vs. ρ for $R_o = 5$ Mbps and $R_{f_0} = 5$ Mbps. . . 36

3-1 Probability Mass Function for PDU interarrival times of a 2.5 Mbps video stream. 47

3-2 PDU Delay Variation for PDU interarrival times of a 2.5 Mbps video stream. 48

3-3 GCRA for varying CBR rates. 49

3-4 GCRA for a CBR 2.5 Mbps with varying rate exponential cross-traffic. 50

3-5 Jitter effect on the subjective QoS. (a) Robin Hood frame with a PDU loss, $P_n < 10^{-3}$. (b) Robin Hood frame with a PDU, $P_n > 10^{-3}$. 60

3-6 Relationship of GCRA(T, τ) with video client network interface buffer occupancy. 61

3-7 Procedure to obtain the schedulable region for two type of traffic classes at the video server. Example for the video stream combination $\{v_{1,1}, v_{1,2}, v_{1,3}; v_{2,1}, v_{2,2}\} \equiv (3, 2)$. 61

3-8 Schedulable region for Type 1 and Type 2 traffic vs. video client network interface buffer size for $P_n < 10^{-3}$ in the video client. 62

4-1 Block diagram of Savoji algorithm. 78

4-2 μ-law function. 80

4-3 2's complement function. 80

4-4 Composite function. 81

4-5 AMF for a voice/silence/voice segment. 82

4-6 Percentages of positive AMF samples for varying frequency and magnitude. 83

4-7 Block diagram of proposed algorithm. 84

4-8 Definition of starting and ending hang time. 85

4-9 Silence with low background noise. 87

4-10 Savoji algorithm operating on low noise signal. 88

4-11 Proposed algorithm operating on low noise signal. 88

5-1 The Internet-friendly protocol framework. 96

5-2 Screen dump from MPEG video streaming application. 106

5-3 Two TCP streams sent from the same server to the same client across 19 hops in the Internet, showing that they do in fact share the available bandwidth. 107

5-4 A TCP stream (solid) and a congestion-unaware pure UDP stream (dotted) are sent from the same server to the same client across 19 hops in the Internet. The available bandwidth is not shared as the TCP stream reacts to the congestion caused by the UDP stream and the UDP stream ignores any indication of congestion. 109

5-5 A TCP stream (solid) and a stream using our protocol (dotted) are sent from the same client and server across 19 hops in the Internet, showing that they also share the available bandwidth. 110

5-6 Two streams using our protocol sent from the same server to the same client across 19 hops in the Internet, showing that they share the available bandwidth. 111

5-7 The rate (solid) and buffer occupancy (dotted) for a stream originally encoded at 250 kbps going through a 200 kbps bottleneck. The buffer occupancy has been normalized to fit on the same graph as the rate. The horizontal lines show the desired rate and buffer occupancy. . . . 112

5-8 The actual throughput (solid) and bottleneck rate (dotted) for a stream originally encoded at 250 kbps going through a varying bottleneck. 113

5-9 The rate (solid) and buffer occupancy (dashed) for a stream originally encoded at 250 kbps going through the Internet, traversing 38 hops. The buffer occupancy has been normalized to fit on the same graph as the rate. 114

5-10 The buffer occupancy from the previous graph using our protocol (solid) and the buffer occupancy of a stream which did not adapt to the available bandwidth (dotted). Buffer overflow indicates an interruption in playback. The buffer size is shown as a horizontal line. 115

5-11 Utilization as calculated by our model, Lakshman's model, and simulation for 100 packets/sec, 80 packet buffering, a one second round trip time and increasing loss rate. 123

5-12 Deviation of utilization model from simulation for our model and Lakshman's model at 100 packets/sec, 20 packet buffering, a one second round trip time and increasing loss rate. 124

5-13 Utilization as calculated by our model, Lakshman's model, and simulation for 100 packets/sec, 20 packet buffering, a one second round trip time and increasing loss rate. 125

5-14 Utilization as calculated by our model, and simulation for 10 packets/sec, 10 packet buffering, a one second round trip time and increasing loss rate. 126

5-15 Utilization as calculated by our model, and simulation for 1000 packets/sec, 1000 packet buffering, a one second round trip time and increasing loss rate. 127

5-16 Utilization as calculated by our model, and simulation for 10 packets/sec, 100 packet buffering, a one second round trip time and increasing loss rate. 128

5-17 Deviation of utilization model from simulation for our model over varying rates and bufferings, a one second round trip time and increasing loss rate. 129
5-18 Link throughput (solid) and goodput (dashed) for increasing maximum number of transmissions. The bit error probability is 0.0002, r=1 Mbps, and s=576 bytes. 134
5-19 Link layer delay for increasing maximum number of transmissions. The bit error probability is 0.0005, r=1 Mbps, and s=576 bytes. . . . 135
5-20 Link goodput (solid) and TCP goodput (dashed) for increasing maximum number of transmissions. The bit error probability is 0.00001, r=1 Mbps, and s=576 bytes. 136
5-21 Link goodput (solid) and TCP goodput (dashed) for increasing maximum number of transmissions. The bit error probability is 0.0001, r=1 Mbps, and s=576 bytes. 137
5-22 Link loss rate (solid) and congestion loss rate (dashed) for increasing maximum number of transmissions. The bit error probability is 0.0001, r=1 Mbps, and s=576 bytes. 138
5-23 The product of the loss probability and the square of the bandwidth-delay product for increasing maximum number of transmissions. The bit error probability is 0.0001 (solid) and 0.00001 (dashed), r=1 Mbps, and s=576 bytes. 139

List of Tables

3.1 First, second and third moments for a single CBR video source. . . . 46

3.2 QoS parameters for an 800 kbps stream under different types of cross-traffic. 48

3.3 CV^2 for a CBR target video source under exponential cross-traffic. . 51

4.1 Comparison of Savoji and proposed algorithms in kilobytes. 86

5.1 Summarized results for Bandwidth Fairness. Difference is defined to be the difference between the two throughputs divided by the reference throughput, Stream 1. 110

5.2 Summary of results. The last column indicates the random loss rate that decreases the utilization most dramatically. In all simulations the round trip time is 1 second. 130

List of Abbreviations

AAL	ATM Adaptation Layer
ABR	Available Bit Rate
AMF	Average Magnitude Factor
ATM	Asynchronous Transfer Mode
CDV	Cell Delay Variation
DAVIC	Digital Audio-Visual Council
DCF	Distributed Coordination Function
DCT	Discrete Cosine Transform
DIFS	DCF Interfame Space
DRS	Dynamic Rate Shaping
DSM-CC	Digital Storage Media Command and Control
DSSS	Direct Sequence Spread Spectrum
FTP	File Transfer Protocol
GCRA	Generic Cell Rate Algorithm
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IPC	Interprocess Communication
MF	Magnitude Factor
MPEG	Moving Pictures Experts Group
NTSC	National Television Standards Committee
PDU	Protocol Data Unit
PLL	Phase-Locked Loop
PMF	Probability Mass Function
QoS	Quality of Service
RTP	Real-time Transport Protocol
RTCP	Real-time Transport Control Protocol
RTT	Round Trip Time
SIFS	Short Interframe Space
STU	Set Top Unit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoD	Video on Demand

Acknowledgements

A doctoral dissertation is a massive undertaking. Like all such works, it is virtually impossible to complete in seclusion and I am certainly no exception. I have received guidance and advice from many people over the past several years.

Perhaps most influential has been Prof. Alexandros Eleftheriadis. He has been a constant source of support and encouragement in helping me find and maintain direction in my research. He is one of the few people who can speak with equal fervor about rate control algorithms as about the latest release of Internet Explorer. His enthusiasm has been nothing less than infectious.

My time at Columbia has also allowed me to interact with a variety of exemplary faculty, students and people from industry. I would like to thank Prof. Dimitris Anastassiou who gave me my first research project as an undergraduate. It was a pleasure working with the Video on Demand team of Prof. Shih-Fu Chang, Hari Kalva and Javier Zamora. I also worked on the video server project with Javier, who is both an excellent engineer and a good friend. I would like to thank Prof. Henning Schulzrinne who has always been available to give me helpful guidance and feedback. Special thanks go to Dr. Charlie Judice for giving me the opportunity to work at Bell Atlantic, as well as his recommendation to Kodak to finance this research. Finally, I would like to thank my office-mates, Amy, Nemo and Raymond, who were always available for fun and discussions.

Lastly, I would like to thank Ira for his never-ending patience and encouragement without which this work could never have been completed.

To my family.

Chapter 1

Introduction

1.1 Background and Motivation

One of the main goals of information technologies has been to reduce the effect of a user's physical location on his/her business and personal productivity. Applications such as telecommuting, video conferencing, and video streaming all have one thing in common: rendering the location of the user irrelevant to the user's ability to access information or communicate. Science fiction movies often depict people living a tranquil life far out in the countryside, but using technology to be just as productive as if they were living in center of a city. These applications require two fundamental resources. The first is computing power which could be in the form of a workstation or highly specialized computing appliance. The second resource is access to a common network over which the user can access other users of that application as well as information.

Computers have been accessible to the mainstream consumer since the 1980's. Nonetheless, usable multimedia communications applications have only begun to emerge over the past several years. This is due in large part to the recent and enormous expansion of the Internet. By unifying so many computers, the Internet provides a single network where users can find other users and sources of information.

Since the Internet is packet-based it is relatively easy to deploy new services. Adding this to the recent advances in multimedia compression yields a very fertile area for the development of networked multimedia communications. The importance of a universal network in multimedia communications can be seen by the recent commercial efforts in IP telephony [64, 66]. Here, the goal is to use the Internet to transmit calls and thereby avoid traditional long distance charges. Since more people have telephones than Internet access, the telephone network is still more universal. Nonetheless, many people are betting that the Internet is a more cost effective way of carrying long distance phone calls.

The union of a universal network with increased computing power has created an explosion of new applications which exploit both network and computer. Many of these applications use multimedia as a means of communicating. Audio and video often provide a much more efficient and simple means of communication than simple text. For this reason, there has been an enormous effort for providing video services to end users, from streaming to conferencing.

The applications for such technologies are many. Video streaming can be used over the Internet or ATM networks. Video on Demand (VoD) is primarily concerned with entertainment or corporate based training applications. Streaming on the Internet has a large variety of applications. Again, this is because the Internet is becoming more ubiquitous. Although streaming can support entertainment applications, the quality is lower than typical entertainment video, such as VCR quality. For this reason, streaming has been used primarily for access to news and information. There are also internal uses for streaming in corporate based training or access to video databases by salespeople.

An even more dramatic revolution in the use of video will be seen once users are given access to video wirelessly and on small handheld or wearable computing devices. The applications for such technology are virtually unknown primarily because people have never had access to video outside of their homes. Only time will tell what applications will result when people have access to video anywhere, while commuting on the train or climbing a mountain.

1.2 Relationship of Network to Media

An important parameter of such multimedia communication environments is the Quality of Service (QoS) support they can provide. In contrast with data-oriented communication, video and audio impose strict requirements on available bandwidth and end-to-end delay. These requirements affect all system components, from the sender to the receiver. High-quality communication can only be provided if the total system behavior satisfies those requirements. At the lowest end, a purely data-oriented system provides no QoS guarantees: throughput and delay depend on the current state of the hosts and the network they reside on. At the highest end, both bandwidth and delay are assured within prespecified and agreed-upon bounds.

Asynchronous Transfer Mode (ATM) is an example of a network which can provide QoS guarantees. Nonetheless a variety of services in ATM networks are provided for applications which don't need any guarantees or need very loose bounds. The current Internet cannot make any guarantees on throughput or delay.

By using a network which can provide QoS guarantees, the application is freed from having to be concerned with these issues. Instead, it relies on the intelligence within the network to ensure that all of the data arrives in time. In the case of ATM, the approach to building the network was to identify potential applications, such as multimedia streaming or conferencing, and build a network which could support these and future services.

Multimedia on the Internet evolved in exactly the opposite way. Suddenly, the public became aware of the network and began discovering ways in which the Internet could be used and exploited financially. Many of the desired applications did not fit the original role of the Internet. The Internet cannot guarantees QoS and this is exactly what real time applications need.

In the Internet, the applications were built to fit the network, rather than the network being built to fit the applications. This means that applications must be able to continually adapt to the state of the network. The media either must adapt to the limited network resources or else the user will experience an interruption in playback. Therefore, a major consideration for Internet applications is the type of codec used and more specifically whether or not that codec is scalable. By scalable we mean a codec which produces compressed streams whose bandwidth requirements are easily changed. For the longevity of the Internet, it is also important that these applications coexist well with existing Internet traffic.

In a network with QoS guarantees, a scalable codec is no longer needed. The application tells the network about the requirements of the non-scalable media and the network provides a bound on those guarantees.

1.3 Challenges of Streaming Multimedia

1.3.1 Guaranteed Quality

The Internet was never designed to handle multimedia real time traffic. This was a major motivating factor for building networks which could provide some guarantees, such as ATM. However, guaranteed quality is not only dependent on the network. It also depends on the computing resources at the source and destination of the connection. These resources include CPU, memory and disk bandwidth.

Recently, handheld computing devices have become very popular. Some market research estimates that within the next two years there will be over 12 million handheld computing devices sold worldwide. Nonetheless, applications which were developed for a desktop computer may not be able to run well (or not at all) on a handheld computer. These devices often have very limited processing power, memory and displays. Therefore, guaranteeing quality on these devices may be even harder than on powerful desktop computers.

1.3.2 Server Scalability

Another challenge in developing a multimedia application is building a cost-effective system. This means that the costs of purchasing and administering the system are commensurate with the price that the end users are willing to pay for the service. In the context of a video service, a pivotal factor is the scalability of the server. Here, scalability refers to the number of simultaneous users a server can support.

The number of users a server can support is related to the processing power, memory requirements and storage required. A client with very strict timing requirements places an undue burden on the server to fulfill these requirements. This burden may translate into a very simple client, but a very unscalable server, which increases the cost of the entire system.

1.3.3 Client Heterogeneity

Another impedement to the successful deployment of multimedia applications is the fact that a single streaming server may attempt to serve a range of different clients. For instance, a desktop computer with hardware decompression capabilities is much different than a handheld computer with a very limited display. Not only can the handheld not deal with the same incoming bandwidth that a desktop can, but it will also be slower in decompressing and displaying the media. Although it is a waste of resources to send 30 frames per second video to a client which can only decode and display 5 frames per second, it may be a better solution than maintaining a different server for each different type of client.

1.3.4 Network Heterogeneity

In addition, a user accessing a server via a Local Area Network (LAN) will be much different than a mobile user accessing content via a Cellular Digital Packet Data network, where the maximum throughput is 14.4 kbps but is variable based on the users mobility. In this case, it is not possible to send the same content to both users. Again the need for a scalable codec arises.

Current streaming technologies require the user to know the type of connection being used to access the Internet. The user then selects the appropriate stream for playback. One solution is to allow the application to determine how much bandwidth is available and automatically choose the stream with the appropriate rate for that user. An even more robust solution is to use a highly scalable codec so that the server maintains only one stream, which can be sent selectively to support a continuum of user requirements.

1.4 Thesis Contributions

1.4.1 QoS Streaming

We address many of the challenges presented. Part I of this thesis deals with streaming video over networks which do have QoS guarantees. By assuming that the network can guarantee QoS, we are freed to examine other factors which may affect the final QoS perceived by the end user. This consists of the video server and video pump architecture, as well as the specific design of the video client.

We present a model of a generic VoD service which is comprised of video server, network and video client. We then focus on the video server and examine the effect of a very stringent video client requirement on the design of the video server. We find that if the video client is too demanding in its desired QoS profile, the video server may not be scalable. We also present a brief overview of Columbia's VoD testbed as a proof-of-concept.

Beginning in Chapter 3, the model presented in the previous chapter is used to determine appropriate QoS parameters which are easily accessible at the video server. A novel and simple approach for defining end-to-end QoS in VoD services is presented. Using this approach we derive a schedulable region for a video server which guarantees end-to-end QoS. We translate a specific QoS required in the video client into a QoS specification for the video server. Our methodology is based on a generic model for VoD services which is extensible to any VoD system.

In this type of system, both the network and the video server are potential sources of QoS degradation. Specifically, we examine the effect that impairments in the video server and video client have on the video quality perceived by the end user. The Columbia VoD testbed is presented as an example to validate the model through experimental results. Our model can be connected to network QoS admission control models to create a unified approach for admission control of incoming video requests in the video server and network.

1.4.2 Non-QoS Streaming

The underlying technologies of today's Internet are not sufficient to support QoS guarantees. The evolution of the base technologies of the Internet could result in any number of possibilities, including ATM backbones, IP switching, or fully deployed ATM networks. Regardless of the specifics, the consensus is that the network infrastructure of the future will have QoS and that users will be able to request connections with or without QoS.

Certainly, connections which reserve resources would demand a higher cost and users may not always want to pay this extra cost for a particular real time service. If a user is watching a movie, perhaps it would be worthwhile. However, just browsing a video database may not warrant the extra cost since the user may browse many video streams before deciding on the one that is being sought. Thus, some video services may not warrant the extra cost of QoS.

Another motivation for being able to provide good quality real time services without QoS is that some networks may never be able to provide QoS guarantees. Wireless networks fit into this category because there may be no way to guarantee a certain bandwidth when so many variables exist that affect the throughput on the wireless link.

In Part II, we turn to these non-QoS networks. This is a much different environment for developing real time applications. The application must be fully aware of the type of network and its characteristics. It must extract information about the state of the network such as available bandwidth and delay. These parameters are continually changing over time. It is also the responsibility of the application to adapt to these variations in the network.

Adapting to the network requires two fundamental tools. The first is a way of extracting this information from the network. In a network such as ATM, the Available Bit Rate (ABR) mode of operation makes it easy to obtain this information. Periodically, the network sends notification to the application about the bandwidth available. In a network such as the Internet, no such explicit feedback is provided. Instead, implicit information is gathered through the use of variation in the delay, loss rates or simply a packet loss.

The second tool required for adapting is that of a media source, or codec, which is scalable. Once the application has determined the state of the network, it must change the bandwidth or delay requirements of the media to fit the network. This can be done by adapting playout mechanisms or changing the bitrate and therefore the quality.

Chapter 4 discusses a variety of codecs for video, audio and still images. Most of the codecs were not intended to be scalable. For instance, the originally intended application for MPEG-1 was for CD-ROMs, where adaptivity and scalability were unnecessary. For the traditionally non-scalable codecs, methods are presented which circumvent their lack of adaptivity. In addition, newer codecs which are inherently scalable are also presented. Regardless of the method, codecs which are scalable are an essential part of a non-QoS network adaptive system.

One technique for adapting an audio codec is to use silence detection. Silence detection and removal is an essential building block of any multimedia video conferencing system. It reduces the bandwidth requirements of the underlying network transport service and helps to maintain an acceptable end-to-end delay for audio. We analyze the requirements for a silence detection algorithm hosted on a multimedia communication system, and propose a novel low complexity algorithm operating in the non-linear μ-law domain.

In Chapter 5 we present a new semi-reliable client/server protocol for streaming real time media on today's Internet. The protocol uses a subset of the TCP congestion control algorithms to create a protocol which is completely Internet friendly. This means that it coexists with other Internet traffic optimally by sharing the available bandwidth.

We then extend this work to the case of a wireless Internet, where at least one of the hops of the connection is a wireless link. In order to evaluate the performance of our protocol in this environment we develop a highly tractable model for the utilization of TCP in the presence of wireless random errors. This work then directly extends to our protocol since it uses very similar algorithms as TCP. We also examine the specific case of TCP utilization over the IEEE 802.11 Wireless LAN standard.

Part I

QoS Streaming

Chapter 2

Video on Demand over ATM

2.1 Introduction

Video-on-Demand (VoD) and Interactive Television may become one of the most important services in high speed networks [65]. Evidence of this can be seen in the recent standardization efforts [26, 29] and the many recent publications [20, 22, 30, 54]. An important goal for any video service is to provide an acceptable QoS to the end-user. This QoS translates into various aspects, such as the frame loss frequency, blocky effect, audio and video synchronization (lip synchronization), and chroma stability (i.e. in NTSC display systems). Some of these parameters are not easily quantifiable since they depend on the subjective perception of the viewer.

We model a VoD service by breaking it into each of its individual components: video server, network, and video client. The video server model is based on the video pump architecture, network interface and operating system. The network model uses the concept of the schedulable region [42] which guarantees QoS with an efficient use of resources. The video client model includes the network interface, memory, and error concealment components.

One of the main concerns in developing video services is interoperability. Many institutions are working on one or more pieces of the video service puzzle [18, 19, 30, 71, 79], i.e. Set Top Units (STUs), video servers and network protocols. However, it is of the utmost importance that pieces of the puzzle from different vendors fit together in such a way that maintains quality of service for the end user.

As with the design of any system, it is possible to minimize the cost of the system as a whole by distributing the intelligence to the entities that are least prevalent in the system. In the video services scenario, this means giving more intelligence to the video pumps than to the STUs, since it is anticipated that there will be many more STUs than video pumps. However, if too much responsibility is given to the video pump, it could prevent general purpose high-end workstations from being used as video pumps. This would result in expensive special purpose hardware, which would most likely be much less programmable than a workstation. The added expense might preclude individuals or small companies from becoming video information providers while the lack of programmablility might mean that new services will be more difficult to disseminate quickly to the end user.

The interaction between the STU and the video pump is quite significant [44]. One technique for both reducing STU cost and increasing video pump complexity is to minimize the buffer space in the network interface. However, this makes the video pump design much more complicated. The pump must send smaller packets more often. As the buffer space in the STU becomes smaller, it becomes more difficult for general purpose workstations to meet these requirements.

The ATM Forum also has been working on STU and video pump interaction. They have suggested that the unit of transmission, which is called a Protocol Data Unit (PDU), be N MPEG-2 Transport Stream (TS) packets. This suggestion is based on the fact that two MPEG-2 TS packets and a PDU trailer fit perfectly within 8 ATM cells. Our prototype STU requires $N = 2$, or 376 bytes.

The organization of this chapter is as follows. The generic VoD model is presented in Section 2.2, where we describe the three basic components: video server, network and video client. In Section 2.3 the Columbia VoD Testbed is presented as an example of how a real system fits into the model of the previous section. Here we place particular emphasis on the Columbia video server to analyze the tradeoffs and pitfalls between interoperability and scalability of the server.

2.2 VoD Model

In this section we introduce the three basic components of a generic VoD system model: video server, network and video client. Our model provides a generic abstraction and captures the common elements that can be found in a any VoD system, avoiding elements that are dependent upon a specific implementation or platform.

2.2.1 Server Model

Fig. 2-1 shows the three main components of the video server model: video pump, network interface and operating system. A video stream is stored on a disk as a consecutive number of video packets (i.e. MPEG-2 transport packets). The function of the video pump is to read this video stream and write the video packets to the network interface. The video pump attempts to preserve the timing information of the stored video. For instance, in the case of a Constant Bit Rate (CBR) video stream the video pump must deliver the video packets to the network according to the rate of the stream.

An ideal video pump generates the same video traffic as that generated from a real-time encoder. Therefore the critical issue in a video pump is its ability to reproduce the original traffic pattern, which implies a coordination between disk accesses and network writings. One video pump engine is associated with each video stream. The output of each video pump contends for the network resource through the network interface. The mission of the network interface is to resolve contention among video packets from different video pumps using a scheduler, to construct the PDUs with the video packets pumped out from a particular video stream and to transmit those PDUs over the network.

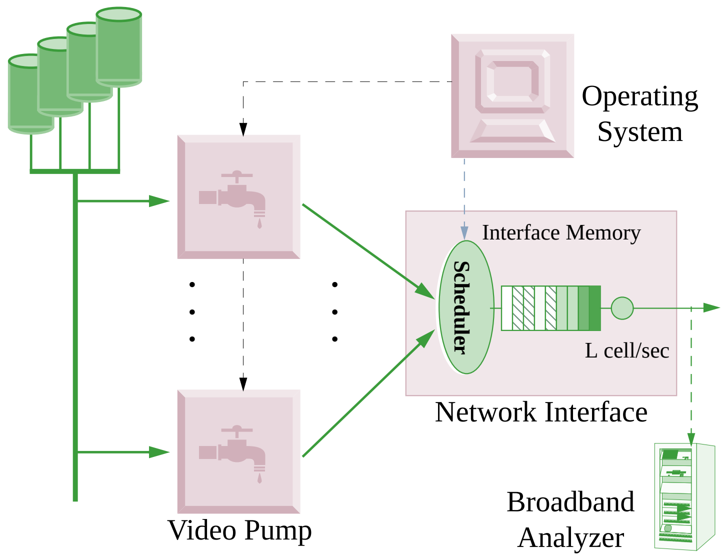


Figure 2-1: Video Server Model.

A video server could introduce some jitter into the traffic pattern of the stream due to contention for common resources, i.e. network interface and disk access. However, it is more likely to happen in the case of a software implementation where there are additional tasks in the operating system. For this reason, we model the video server as a virtual multiplexer, where several sources are accessing a multiplexing stage (network interface). In a software-based implementation, the interaction with other processes that use common resources such as the bus or the disk causes additional jitter to the video sources. This effect can be modeled as cross-traffic accessing the multiplexer with a priority (scheduler) dictated by the operating system.

A software implementation provides a high degree of interoperability and ease of deployment because it can be easily adapted to different platforms and networks, dramatically reducing the cost of such systems. For this reason, it is expected that many video servers will be based on software implementations. However, general purpose workstations do not have real-time operating systems and this has a detrimental effect on the traffic of the transmitted video stream.

The video server can be considered as the first switch of the connection. This has the advantage of simplifying admission control since the behavior of the video server can be characterized as an additional switch of the connection with a specified QoS performance. The QoS degradation not only occurs in the network switches but also in the video server itself. The video server QoS degradation can be dominant in scenarios where the network utilization is low, as is in many private local ATM networks.

2.2.2 Network Model

With regard to the network model we use the approach described in [61, 62] which provides QoS guarantees with efficient use of resources. For this purpose the concept of schedulable region is introduced in [42], which defines the possible combination of loads for each traffic class under a QoS constraint (i.e. loss and delay). Although this admission control policy is different than that which is specified by the ATM Forum [25], it is possible to map the schedulable region into ATM Forum admission policy. This implies mapping the different service classes into network traffic classes. We then extend this concept of schedulable region to provide an admissible region for the video server based on experimental results of QoS measurements.

2.2.3 Client Model

Fig. 2-2 shows a diagram of our Video Client Model. It consists of four sections, Reception, Demultiplexing, Decoding and Presentation. Reception comprises the network interface that receives the PDUs from the network. The buffer size of the network interface is on the order of several PDUs. In some digital set-top-boxes this size is only 376 bytes (8 ATM cells using AAL5). This size prevents the server from sending larger PDUs. The output of the network interface is connected to the smoothing buffer to eliminate the jitter between the PDUs.

Generally a Phase-Locked Loop (PLL) driven by the smoothing buffer occupancy is used to smooth the accumulated jitter in the server and network, and the jitter caused by independent clocks used in the server and client [102]. The output of this buffer is an MPEG-2 transport stream which is demultiplexed into the corresponding video and audio streams. In the Decoding stage, the video and audio streams feed the Video Decoder and the Audio Decoder, respectively. Finally in the Presentation Stage the video and audio are played.

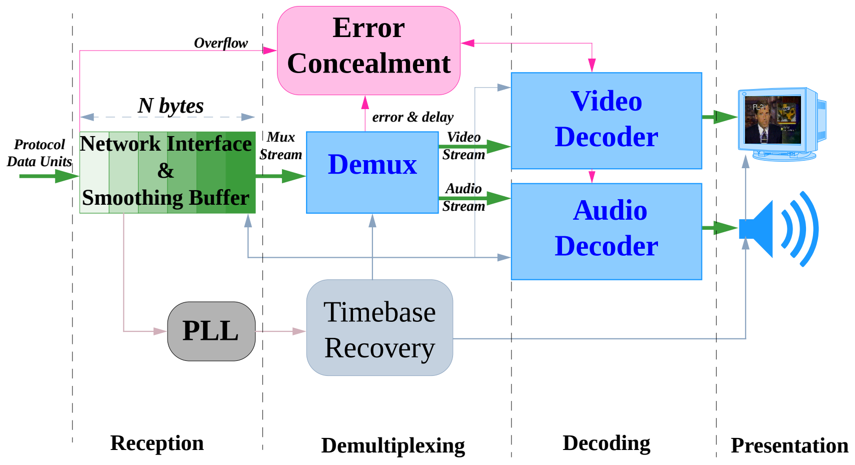


Figure 2-2: Video Client Model.

The whole system is controlled by two subsystems: Time Recovery and Error Concealment. The Time Recovery subsystem generates different levels of synchronization (i.e. demux, decoding and presentation) from the PLL signal and the time stamps inside the incoming stream. The Error Concealment subsystem takes the appropriate actions (i.e. repeating previous frame) in the case of some error condition.

Such error conditions can be caused by an overflow in the network interface buffer (loss of PDUs), underflow of the reception buffer causing loss of synchronization, and error in the received streams [103]. Some parameters of this model can be scaled depending on the type of video client used. Memory constraints in digital set-top-boxes can be eliminated in software-based implementations. For instance, a PC network interface card has substantially more memory.

In the next section, the Columbia VoD testbed is described as an example implementation of a VoD system, and is used to show how the model can be applied to a real system as well as how design constraints can affect the overall QoS performance.

2.3 Columbia's Video on Demand Testbed

Columbia's VoD testbed is designed with advanced features of video storage, coding, manipulation, transmission and retrieval [18]. The main objective is to use this testbed as a platform for multimedia research and application development such as Columbia's Electronic News System, Digital Libraries and Interactive Video Courses on Demand (see Fig. 2-3). A broadband analyzer measures performance parameters such as the delay jitter at different points in the network. Moreover, the broadband analyzer can emulate network impairment conditions (i.e. cell losses) on real video services in the testbed. In Fig. 2-4, the Columbia VoD testbed architecture is shown. We identify the three components of the VoD model.

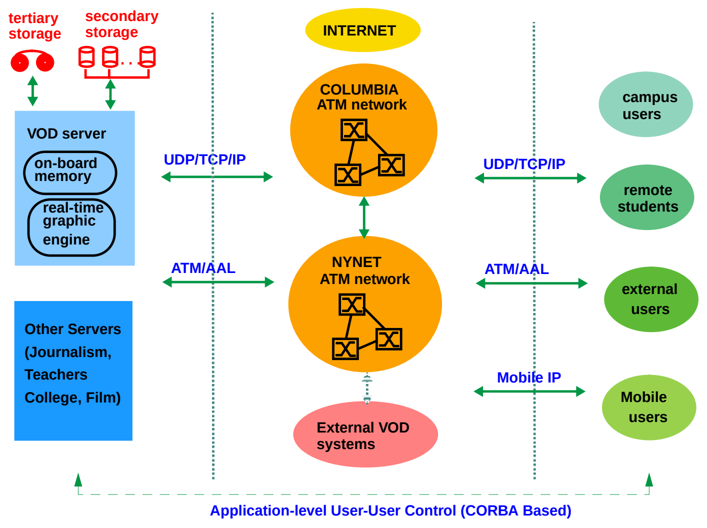


Figure 2-3: Columbia's Multimedia Testbed.

2.3.1 Testbed Server

In general there will be many more video clients than video servers. Therefore, it is beneficial to give more intelligence to the video server than to the video client. However, if too much responsibility is given to the video server, it could prevent the use of general purpose workstations for this purpose. This would result in expensive special purpose hardware, which is usually much more difficult to program. The added expense precludes individuals or small companies from becoming video information providers. The lack of programmability means that new services will be more difficult to implement and deploy.

The server is implemented in software using a general purpose UNIX workstation. Specifically it includes a Silicon Graphics ONYX 6-processor system, running IRIX 5.3. The video material is stored on an array of disks in compressed form. An MPEG-2 software encoder or real-time hardware encoder compresses the video sequences and prepares the video elementary streams as well as MPEG-2 transport streams [55]. The video server has the capability to pump both CBR and VBR

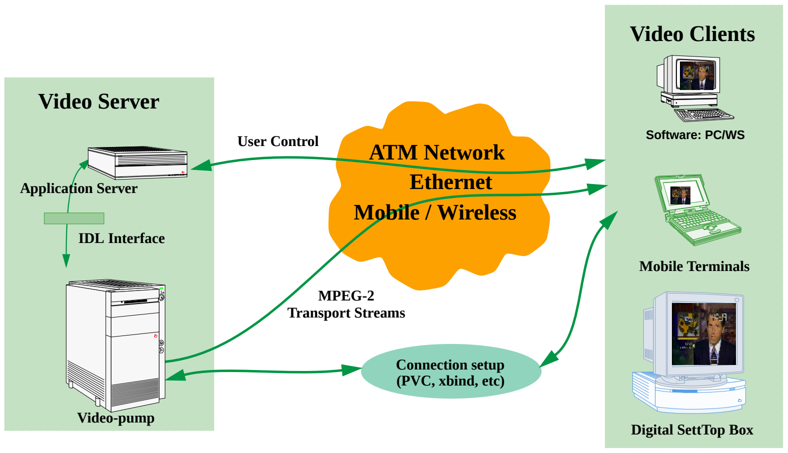


Figure 2-4: Columbia's VoD Testbed Architecture.

video streams.

The video server uses an absolute timer scheme to send out the PDUs according to a specific interarrival pattern. At time t_0 the interarrival time is set to a period, T, between two consecutive PDUs. The first PDU is sent and the pump waits for the timer to expire. The timer, t_e, will only expire when it is equal to $t_0 + T + \Delta t$, where Δt is the time by which the timer overshoots the expiration time. At this point one PDU is sent out and the timer is set to $t_e - \Delta t + T$, to negate the effect of the timer overshoot.

From the perspective of the video client, absolute timers enforce a constant average bit rate. This reduces the potential for underflow in the decoder in the absence of a PLL mechanism. If one PDU is sent out late with a delay of Δt, the absolute timer will compensate for this delay by trying to send out the next PDU with a period $T - \Delta t$. An advantage of using absolute timers is their ability to mask the inconsistencies of the resources that are logically below them. For instance, if the time it actually takes to send out a PDU has a large variance, an absolute timer attempts to compensate for this. If the network interface delays the sending of a PDU, the absolute timer will make the period for the next PDU smaller. The variance of the underlying resource is therefore reduced since it is being corrected by the changing period in the timer. However, this also introduces jitter and PDU clumping.

For the network interface, we use a *ForeRunner* SBA-200 ATM VMEBus Adapter which resolves the contention among the outputs of the different active video pumps. The output of the adapter uses a TAXI line as the physical interface. Since our implementation is software-based, the operating system affects the performance of the video pump objects and the network interface.

MPEG-2 transport streams are broken into transport packets which are 188 bytes long. Taking into account the previous considerations, the ATM Forum [26] has specified that, for MPEG-2 transport stream transmission over ATM using AAL-5, the PDU size will be $188N$ bytes ($N = 1, 2, \ldots$). An AAL-5 PDU is allowed 48 bytes of payload per ATM cell. However, the PDU also has a trailer of 8 bytes which also must fit into this payload. If the PDU payload does not use all of the ATM cell payload, the unused bytes are wasted.

For optimal throughput and so that no bytes are wasted we solve the following equation: $188N + 8 = 48M$, where N is the number of transport packets and M is the number of ATM cells. Integer solutions to this equation occur at $N = 2 + 12i$, for $i \geq 0$. Therefore the minimum PDU size is for the case of $N = 2$ or 376 bytes per PDU. Some digital set-top-boxes only admit this small PDU size for memory cost constraints. This is the case for our digital set-top-box.

In the next section, we give an overview of the environment in which our video pump is operating. This leads to a discussion of specific problems of interoperability between video pumps and STUs. In Section 2.3.1.2 we present the design of our video pump. The methods used for maintaining quality of service guarantees are presented in Section 2.3.1.3. Section 2.3.1.4 provides a solution to the interoperability problem by using a non-conventional timing mechanism. This timing mechanism is the only one available to general purpose computers which can provide the resolution required by the STU. It will be shown that although this timing mechanism works quite well for a few streams, it creates problems for scalability. A solution to the scalability problem is proposed and experimental results are provided.

2.3.1.1 Video Pump Environment

User level timing mechanisms on general purpose workstations are all based on signals. Signals are sent periodically to interrupt the CPU. Applications can use them to perform periodic actions. Normally, signals occur on the order of once every millisecond. The time between arriving 376 byte PDUs is 600 microseconds for a 5 Mbps high quality stream. The difference between what is available on general purpose computers and what is needed to support high bandwidth video streams using small PDUs, is the core of the problem that we address here.

In the case of our workstation, interrupts can come more quickly, as fast as one every 500 microseconds. This number is a system tunable parameter which is set at boot-time and affects all processors. The signal resolution has a limit though. Each interrupt has substantial overhead which will degrade the performance of the processor if the interrupt frequency is too high.

Even though a 500 microsecond timer may seem like enough, it is not. For example, at 500 microseconds a rate of 6 Mbps can be supported, however a rate of 5 Mbps cannot because this rate requires interrupts every 600 microseconds and the interrupts are only coming at integer multiples of 500 microseconds. This is clearly unacceptable since a video pump must be able to support clients with differing bandwidth capabilities. A client may have a software decoder that can only decode up to a maximum bit rate or may only have access to a certain amount of bandwidth. In either case the client wants to specify and receive the maximum bandwidth it can handle [33].

The use of a small PDU has an adverse effect on the performance of the video server. There is a certain amount of processing overhead each time a PDU is sent to the network. Each send command is a system call and each system call initiates a request for services from the device driver. If the send command is called for larger PDUs, then the overhead is amortized over a larger number of bits and the resultant overhead per bit is lower. Fig. 2-5 is a graph of PDU size vs. coefficient of variation for the interarrival time for a rate of 2.5 Mbps. As can be seen, the smaller PDUs have a noticeably larger variance. This is due to the increased number of requests sent to the device driver.

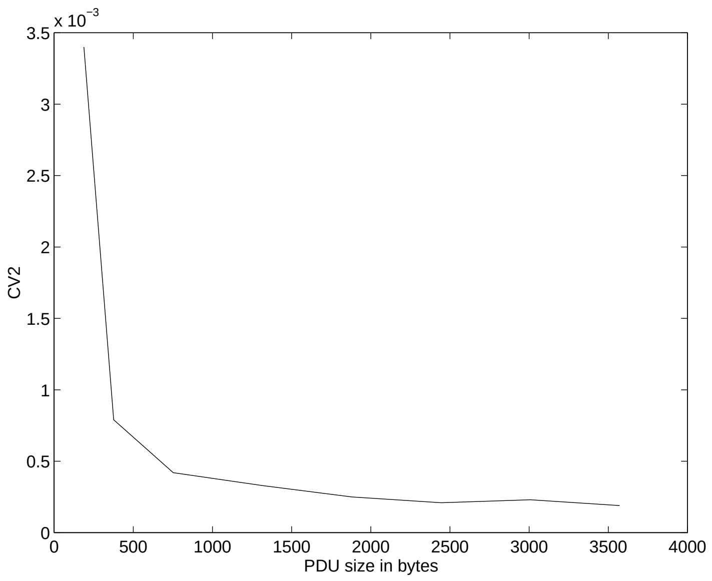


Figure 2-5: CV^2 of PDU interarrival times vs. PDU size for a 2.5 Mbps video stream.

For this reason, the use of a small PDU size as mandated by our hardware video clients, in our video server is translated into a limit in scalability (i.e. maximum number of simultaneous video streams). This limitation is overcome, when the video client does not have such memory constraints. This is the case for our software video clients and any client that resides on a general purpose computer.

Nonetheless there are also drawbacks to large PDUs. When a PDU is received in error, the entire PDU is discarded. Although ATM optical networks tend to have low bit error rates on the order of 10^{-9}, they may lose cells due to congestion at switches along the way. In the case of large PDUs, more data is lost each time there is an error. This will have an increasingly detrimental effect on the video.

2.3.1.2 Architecture

A video pump is implemented as a software object. Each time a client requests a stream from the video server, a new video pump object is created for that client. The management and control of these objects is performed via the Client Object Request Broker Architecture (CORBA Rev. 1.3) [38]. CORBA is an advanced object-oriented Remote Procedure Call (RPC) facility. Each video pump object consists of 3 separate threads, as shown in Fig. 2-6. The first thread is the Control thread. Its job is to interpret commands from the client, such as play, pause, resume and stop. The second and third threads are responsible for moving the data from the disk to the network interface. The Reader thread reads data from the disks and fills the shared buffers according to a round robin schedule. Meanwhile, the Writer thread reads the buffers and sends data out to the network interface, one PDU at a time.

The video pump object is separated into threads for both logical and performance reasons. From a logical perspective, each thread performs a task independent of the others and is therefore a separate entity. The more significant reason is performance.

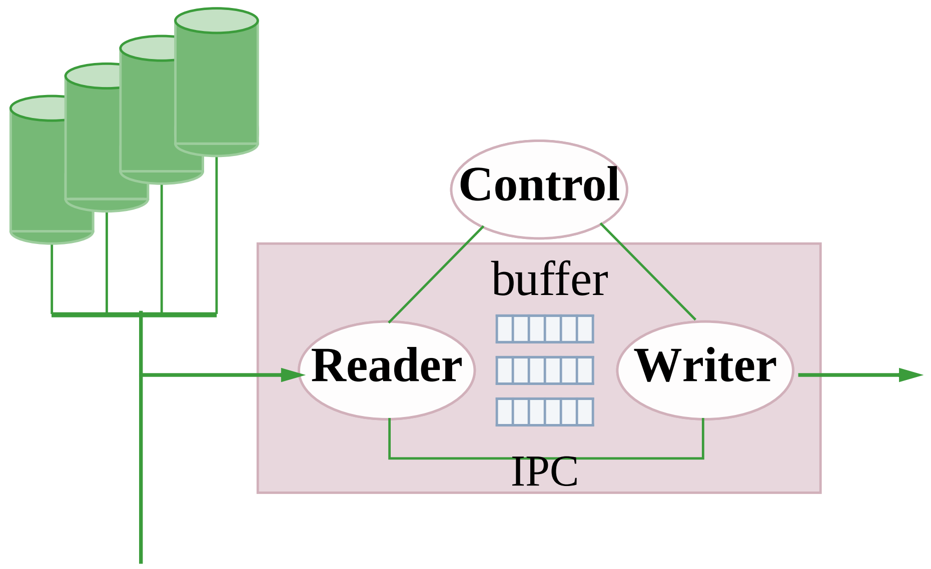


Figure 2-6: Video Pump Object.

The small PDUs result in correspondingly small periods between PDUs. These indicate that the Writer thread is working the hardest. For this reason, it cannot suffer the added delays required for reading large blocks of data from the disk or for waiting for a client request.

The interaction between the threads is coordinated via interprocess communication (IPC) and more specifically semaphores. When the Reader is getting data from the disk and filling one of the common buffers, it must verify that the Writer is not currently sending data from that buffer by evaluating the semaphore. The IPC also occurs when the Writer finishes data in one buffer and must get new data from the next one. This implies that each time the Writer needs a new buffer, it will be delayed by the amount of time it takes to evaluate the semaphore; this takes about 200 microseconds in our system.

This delay is periodic and is a trade-off between the size of the buffer and the rate of the video pump. For example, for 5 Mbps and a buffer size of 156 kB, the frequency of the IPCs which is due to switching buffers is 4 per second. Increasing the buffer size decreases the frequency of the IPCs, but increases memory costs. This phenomenon can be observed in Fig. 2-7, where we can see 200 microsecond periodic peaks. This Figure is a plot of PDU interarrival times for a 2.5 Mbps CBR stream.

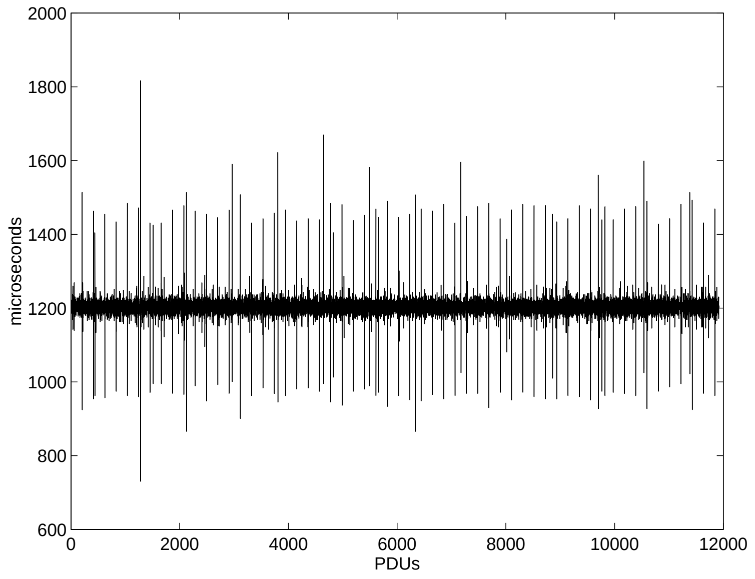


Figure 2-7: PDU interarrival times for 2.5 Mbps video stream measured at the output of the video server.

2.3.1.3 Resource Reservation

The development of economically feasible video services will be essential to the evolution of the industry. A system which provides a video service must, by definition, provide guarantees on end to end quality of service. These guarantees can only be met if the underlying resources provide guarantees on their performance. Therefore, one purpose of a video server is to coordinate the usage of these resources.

IRIX Release 5.3 has several features for providing quality of service guarantees to applications. These guarantees are provided by isolating the resource for a specified period of time. When there is no operating system level provision for isolating the resource, other user-level best-effort arrangements must be made. The resources used by the video pump are CPU, memory, system bus bandwidth, disk bandwidth, and network bandwidth.

1. CPU

   A process is allowed to isolate an entire processor such that no other regular process can run on it. This is essential to guarantee that the video pump will not be competing with other processes for time on the CPU. Multiple video pumps could also share a single processor.

2. Memory

   Each video pump uses a small portion of memory. To minimize random delays, the program text and data segments may be locked into physical memory. Locking prevents any information from being paged out to disk. Paging happens when other user processes or system processes need to be in physical memory and there is not enough physical memory available to hold all processes.

3. System Bus Bandwidth

   This is actually the most critical area because there is no control over it. If other users or system processes are running they may require significant access to the system bus bandwidth. This will deplete video pump resources. Also, no guarantees or reservations can be made.

4. Disk Bandwidth

   There are no operating system level guarantees for allocating disk bandwidth in IRIX. Nonetheless, there are ways of minimizing the likelihood that a video pump object is unable to provide a specified quality of service due to a lack of disk bandwidth. The file system supports striped disks. This means a portion of the data in a file is placed on each disk so that accessing the file requires accessing all of the disks simultaneously. Disk bandwidth is thus increased in proportion to the number of striped disks.

However, there may also be jitter at the output of the disks since they are being accessed by other video pump objects, too. This jitter is smoothed by having each video pump fill two buffers with 250 milliseconds of data each before delivery to the network can begin [28]. These buffers will smooth up to 500 milliseconds of disk jitter. If the striped disks' throughput is 200 Mbps and the stream being served is 5 Mbps, it will take approximately 10 milliseconds to fill the buffer once the request is acted upon by the disk, depending on seek and rotation times. If we assume round robin scheduling then even if there are 50 other streams operating simultaneously, each one will have access to the data before their buffers empty. This user level guarantee is obtained by assuming worst case delays for seek, rotation and transfer time.

5. Network Bandwidth

   Currently our ATM LAN does not support resource reservation and does no admission control. Nonetheless, it can be considered an isolated resource since it is used for experimental purposes.

2.3.1.4 High Resolution Timers

As discussed earlier, traditional timing mechanisms based on signals cannot be used for a video pump which must use small PDUs because of the high resolution required. For this reason, new timer objects had to be created. The only mechanism which could provide the resolution required by the STU is simple busy-wait timers. Busy-wait timers are a last resort since they consume extra CPU cycles. At the beginning of a period the timer is set to the appropriate expiration time. After the data has been sent out for that period, the pump waits for the timer to expire. During this waiting the timer object is continually looping and monitoring the time. It will only stop looping when it discovers that the time is greater than or equal to the expiration time.

IRIX provides a way of mapping the hardware clock into memory, thereby obtaining a clock which has an accuracy of 21 nanoseconds. Of course this level of accuracy cannot be fully exploited because of the time it takes to actually read the value. Nonetheless, the timers can measure times from 60 microseconds and up.

The obvious problem with busy-wait timers is that they will use the processor at times when ordinarily it could be released to do other things, such as running a different video pump. One possible solution is to have only one processor suffer the penalty of using a busy-wait timer by running a timer daemon on that processor. Video pumps would register themselves with this daemon and would be put to sleep and revived by it when their period ended and began, respectively. However, the overhead due to interproccess communication is too high at about 200 microseconds, which would be incurred each time a PDU is sent.

An additional short-coming of having to use these timers is that they are accurate only to several microseconds. A 5 Mbps stream needs a period of 601 microseconds. If the closest we can get to this is 603 microseconds, then we will be sending too slowly. The STU will be expecting 5 Mbps as indicated within the MPEG stream, but we will be sending 4.98 Mbps. Eventually this drift will accumulate and cause blanks to appear on the display due to buffer underflow in the decoder.

The above scenario assumes that there is no Phase Locked Loop (PLL) in the STU. The PLL modulates the system clock of the decoder based on the buffer occupancy in the network interface. Slight differences in rate can be accommodated by the STU if it implements a PLL. Our prototype runs in open loop mode so that it cannot tolerate deviations from the actual rate. This is another example of how oversimplifying the decoder leads to a much more complex video server.

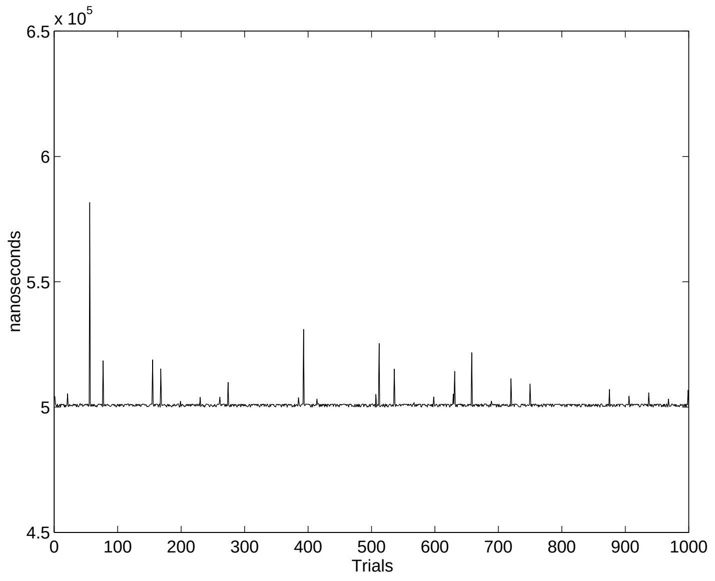


Figure 2-8: Characterization of the timers. Actual time waited by the timer over 1000 trials.

A characterization of the timing mechanisms independent of the video pump is given in Figure 2-8. The timer was repeatedly set for a 500 microsecond period. The points in the graph show the actual time waited by the timer for 1000 such trials. Clearly there is some deviation from the average. The average time waited is 501 microseconds while the standard deviation is 3 microseconds.

Since the timers are not accurate, Long Term Rate Adaptation (LTRA) must be performed. The average throughput is calculated periodically by dividing the number of bytes sent by the amount of time it has taken to send them. If the throughput is too low, then the period is decreased; if the throughput is too high, the period is increased. This adaptation creates small fluctuations around the desired throughput which ensure that the accumulated jitter does not reach an intolerable level.

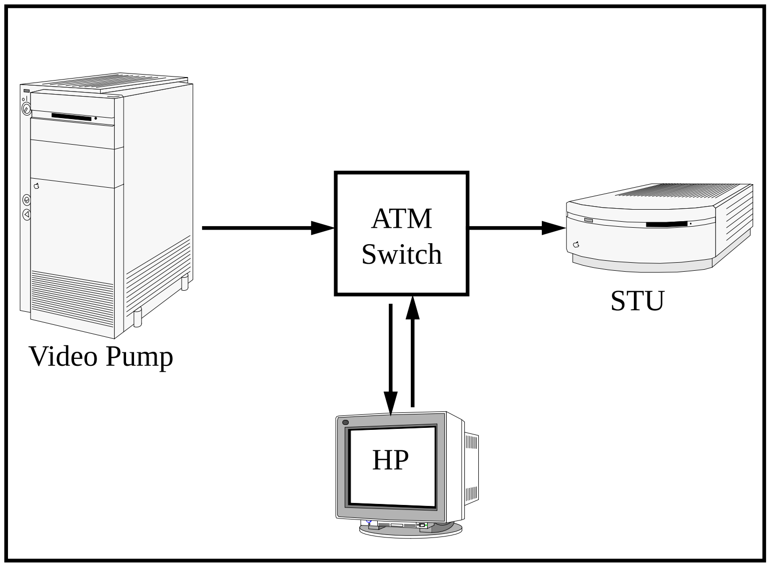


Figure 2-9: Experimental configuration for HP Broadband Analyzer.

Figure 2-9 is a diagram of the set-up for generating experimental results used in this chapter. The HP is a stand alone Broadband Analyzer workstation with special purpose hardware for network analysis. It provides timing data which has a 10 nanosecond resolution. It also reassembles ATM cells into AAL5 PDUs and even into MPEG-2 Transport and Packetized Elementary Streams.

The data was sent from the video pump to the Broadband Analyzer through a single ATM switch. The Broadband Analyzer stores the data in memory as it receives it. Once data collection is done, it produces a list of arrival times of PDUs. We then process this list to obtain the interarrival times of the PDUs and various statistics, such as the probability mass function, the mean, the variance, and the coefficient of variation.

Figure 2-10 is a graph of interarrival times of consecutive PDUs with the actual average interarrival time subtracted. The stream was sent at 5 Mbps. The standard deviation of the interarrival times is 21 microseconds. This is larger than the 3

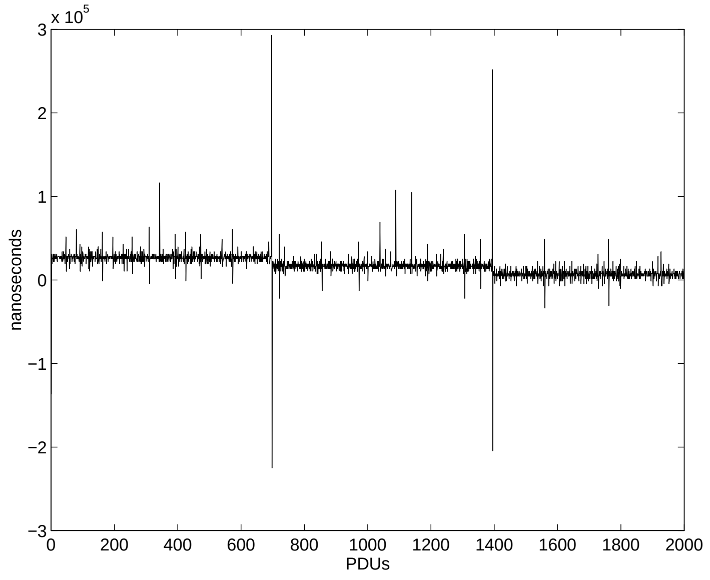


Figure 2-10: Deviation from the average PDU interarrival time for a 5 Mbps stream measured at the output of the video pump through one ATM switch.

microsecond standard deviation for the timers operating alone, as shown back in Figure 2-8. This difference is due in part to the sharp peaks at 700 and 1400 PDUs. The peaks are caused by the calculation of the LTRA as well as the interprocess communication overhead for switching buffers. This delay is much better than if the Reader thread and Writer thread were merged into one because then the delay would be that of reading from disk, which would be several orders of magnitude higher.

Immediately after the peak comes a trough. The pump attempts to make up for lost time since it missed the deadline on the previous PDU. The jitter introduced by these peaks and troughs is acceptable since the average bit rate remains constant over a small period of time.

The subtle staircase nature of the graph is due to the LTRA determining that the current rate is too slow and that it must decrease the interarrival time to increase the throughput. Figure 3-1 shows the Probability Mass Function (PMF) for the

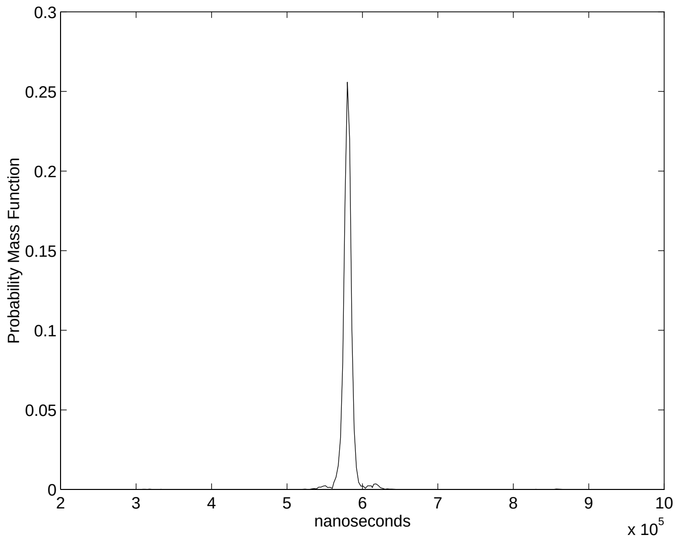


Figure 2-11: Probability Mass Function for the interarrival times.

interarrival times of Figure 2-10. The ideal PMF would be an impulse of height one, indicating that the interarrival times were always exactly the same. In our case, we have one main peak and two side lobes which correspond to the peaks and troughs mentioned previously.

Scalability is another problem for busy-wait timers and small PDUs. Each video pump object will use 100% of a processor since the pump is not asleep while waiting. This is clearly not an efficient use of that resource. When two pumps are run on the same processor the operating system's scheduler gives the processor to each pump for a 30 millisecond quantum [101]. This is not a constant for the scheduler, but an experimentally obtained value. This means that a pump will not be able to send out any data for 30 milliseconds. Since the average rate must remain constant, the pump must deliver data more quickly during the time it occupies the processor.

The problem is solved by using short term rate adaptation (STRA). If a video pump senses a sudden and severe drop in average throughput, it automatically changes the rate at which it is trying to pump. If the throughput drops significantly

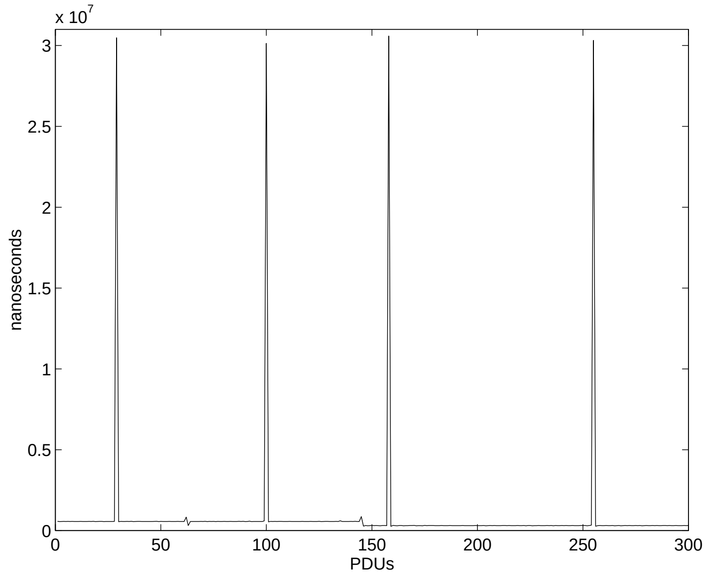


Figure 2-12: Interarrival times for a video pump operating at 5 Mbps while another video pump is running on the same processor.

enough, it is possible that STRA will not respond until after the decoder's buffer empties, causing a momentary blank screen. In Figure 2-12, STRA occurs just before the 150th PDU where there is a sudden drop off. The dropoff in interarrival times corresponds to sending the data at a higher rate. Before the dropoff the average interarrival time is about 570 microseconds; after the dropoff the average time is 280 microseconds.

The sharp peaks are the times when the video pump has been descheduled by the operating system to run the other video pump. The peaks show a 30 millisecond interarrival time, which is of course the time the pump is asleep. The average throughput from one peak to the next is still the desired throughput. However, the rate at which the pump is operating is much higher than this average to compensate for the 30 milliseconds of delay.

Let R_{f_i} be the rate at which the video pump is sending data during interval i. The video pump is attempting to maintain a rate of R_o so when the pump first starts,

$$R_{f_0} = R_o. \tag{2.1}$$

Now when another video pump starts on the same processor, the actual throughput for the first one will drop to $\rho < R_o$. The effort, E, is defined as the desired rate divided by the actual throughput,

$$E = \frac{R_o}{\rho}. \tag{2.2}$$

E indicates how much harder the video pump must work to obtain the desired rate. To obtain an actual throughput of R_o, the video pump must pump at the faster rate,

$$R_{f_{i+1}} = R_{f_i} E = \frac{R_{f_i} R_o}{\rho}. \tag{2.3}$$

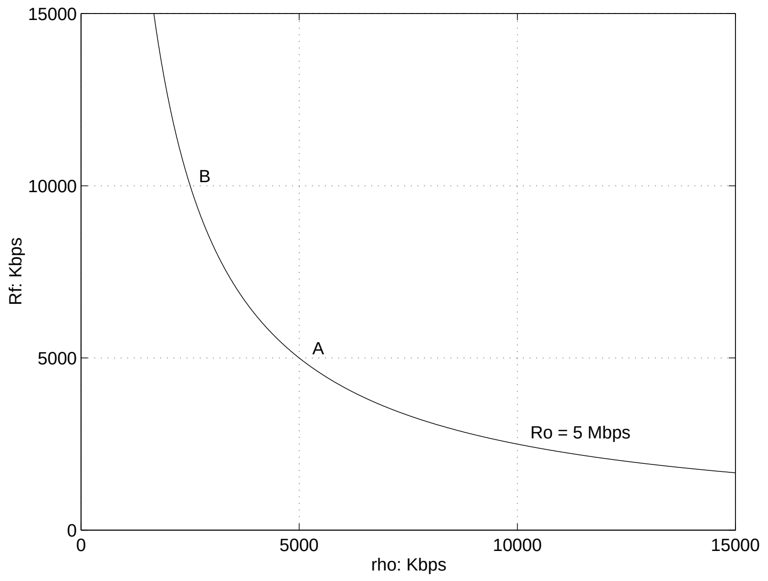


Figure 2-13: Example R_f curve. R_{f_1} vs. ρ for $R_o = 5$ Mbps and $R_{f_0} = 5$ Mbps.

Figure 2-13 is a graph of Equation 2.3. Point A is where the pump is attempting to deliver at 5 Mbps and the throughput is also 5 Mbps. In other words, only one pump is running on the processor. When another pump starts on the same processor, the scheduler runs each pump for 30 milliseconds. This has the effect of cutting the first pump's CPU time in half. The operating point then shifts to B on the graph. The throughput has dropped to 2.5 Mbps and the effort, E, is 2. This signifies that the pump must deliver data twice as fast. Now when the pump is active it must send data at $R_{f_1} = 10$ Mbps. If the pump knew in advance that it would get only 50% of the processor, it could simply send the data twice as fast and these calculations would be unnecessary. However, the only information available to the pump is the rate at which it is attempting to send and the actual throughput.

When one of the video pumps running on the single processor exits, the throughput of the remaining video pump will increase substantially. R_o is still 5 Mbps. However, now the throughput, ρ, jumps to 10 Mbps and $E = 1/2$. R_{f_2} is calculated to be 5 Mbps, indicating a return to our original rate. If the size of the PDU were much larger, such that the period was greater than the operating system scheduler quantum, STRA would not have to be performed at all. This is because the idle time between PDUs could be freed and used to schedule other video pumps.

2.3.2 Testbed Network

The transport network infrastructure of the Columbia VoD testbed includes ATM and Internet as the core technologies. The ATM network consists of a ATM LAN, which is interconnected to the campus-wide ATM network. This campus ATM network is also connected to an ATM Wide Area Network (WAN), which provides high-speed connections to other VoD testbeds on the East Coast.

For the ATM network the **xbind** Open Signalling Architecture is used to control and manage the connections with QoS guarantees and efficient use of resources [60]. **xbind** is based on the schedulable region concept. Users on the campus access the video servers directly through the native ATM network (using AAL5) or via Internet (i.e. over Ethernet or wireless).

An important feature is how the user can search, select and control a specific video stream. For this purpose we have an application server that communicates with the client through a CORBA-based interface using DSM-CC [38, 56] User-to-User primitives.

2.3.3 Testbed Clients

The testbed has three types of video clients. The first type uses digital set-top-boxes with direct ATM connections (DS3 lines). These clients can decode up to 15 Mbps MPEG-2 TS streams. The output of the decoders are connected to TV monitors, providing better quality than regular VCRs and displaying video at 30 frames per second. The second type uses personal computers and workstations with our in-house software decoder. The software implementation provides a high degree of design flexibility but with some processing constraints, such as frame rate and resolution, since it has to demultiplex and decode video and audio components received through the network. Finally, we have mobile terminals, using a hardware decoder and mobile IP protocol.

2.4 Concluding Remarks

We have described each of the components which make up our VoD model: video server, network and video client. The model is an abstraction and captures the common elements that can be found in any VoD system, avoiding specific elements that are particular to a specific implementation or platform. The video server model allows us to consider the video server as the first switch of the connection. This has the advantage of simplifying admission control, since the behavior of the video server can be characterized as an additional switch of the connection with a specified QoS performance. This approach can be applied to both software-based and hardware-based video servers. The Columbia VoD testbed is used as a practical implementation example of how our VoD model can be applied as well as how design constraints can affect the overall QoS performance.

We have also addressed the tradeoff between scalability and interoperability. STUs which require the use of small PDUs make video pumps based on general purpose computers more difficult to design. The problems with using small PDU sizes are many. Efficient timing mechanisms for small PDUs do not exist. Therefore, the only solution is to use busy-wait timers. These timers are problematic too, because of their lack of accuracy, their inability to scale well and their inefficiency.

One solution to this might be a programmable network interface card which could receive an entire frame of data from the video pump. The card could then be programmed to deliver that frame at a given rate. This relieves the video pump of the intensive timing responsibilities. However, until there are such network cards available, some compromise will have to be made between video servers and STUs.

Small PDUs require substantially more overhead per bit to deliver. Larger PDUs would mean less overhead and therefore a higher utilization of the CPU resource. Currently, each processor can only support an aggregate throughput of 12 Mbps due to the intense workload in having to deliver it in 376 byte PDUs. This means a maximum of 2 MPEG-2 streams per processor or 8 MPEG-1 streams.

If STUs continue to have such strict requirements, video pumps will have to be built using special purpose hardware to be scalable. This works against the basic premise of interoperability where different platforms and architectures can work together to provide a useful service to the end user.

Chapter 3

Video QoS Parameter Mapping: Server to Client

3.1 Introduction

The QoS at the video client reflects how the original video stream has been delivered from the remote video server, where concepts of semantic transparency and time transparency [78] characterize the performance of video services over packet networks. Such services require specific constraints regarding the delay (time transparency), specifically the delay variation or jitter, experienced across the connection, as well as constraints regarding the rate of errors (semantic transparency), from video server to video client. Therefore, a VoD service must be able to map a specific QoS required in the video client into a QoS specification for the video server and network.

In this chapter, a novel and simple approach for mapping QoS from video server to video client is presented by using a generic model which can accommodate any VoD system [105]. An example of this methodology is provided by applying it to the Columbia VoD testbed [18]. The goal of our model is to provide an end-to-end QoS. In other words, we examine how impairments in the video server and network affect the quality of the video perceived by the end user. Since VoD is a point-to-point service, we provide QoS guarantees per individual stream rather than over an aggregate video traffic in the video server or network switch.

From the QoS point of view, each of the components of the VoD model (video server, network and video client) has its own characteristic parameters. For instance, the video server's parameters are the bit rate, burstiness, and autocorrelation of the video streams, as well as the conditions in the system, such as number of video streams currently active and the impact of other processes running on the system. The performance in the network can be described by several parameters such as probability of loss and error, end-to-end delay, jitter, and burst tolerance. In the video client, frame loss, lip synchronization, blocky effect and picture distortion are the parameters that define the QoS perceived by the final viewer. It is important to map the parameters from each of these domains into those of the video client, i.e. a certain jitter distribution will be translated to a specific frame loss rate.

The scales in the three domains are different, too. The video server handles PDUs whose size can range from 376 bytes, per the ATM Forum [26] specification, to several kilobytes; the ATM network operates with 48-byte-payload cells; and the video client works in the frame domain with an average size of 200 kilobits for a 6 Mbps video stream. These differences in scale mean that the loss of one cell will cause the loss of an entire PDU. This same PDU loss could cause the loss of an entire frame or Group of Pictures, depending on the position of that PDU in the transport stream. Also, losing an I frame would be more detrimental than losing a B frame. Therefore, not all losses have the same impact on the final QoS.

Starting from several real-time video traffic measurements, Section 3.3 identifies which of several QoS parameters provide a comprehensive description of the performance of a video server. Section 3.4 presents a novel mapping from the video server QoS parameters identified in Section 3.3 to the video client QoS parameters. In Section 3.5, this mapping is used to define a video server schedulable region and the specific example of the Columbia VoD testbed is again used.

3.2 Overview of Video Server QoS Parameters

In this section, we first define the QoS metrics of the video server. We then apply these to several experimental results obtained from our testbed in order to determine which metrics most accurately characterize a video server.

The most important feature in evaluating the performance of a video server is to study the output traffic pattern. The deviation from the theoretical video traffic pattern will indicate the QoS degradation. For this reason, we are interested in studying the delay distribution of the PDU interarrival process associated with a video stream. There are many measures for evaluating performance and jitter. The performance metrics used in this chapter refer to the transport of PDUs, since it is the information unit that the video client handles. The PDU metrics can be directly translated to cell metrics when the video server's network interface sends equispaced cells over a PDU period. Unfortunately this is not always the case in commercial ATM network adapters. We divide the PDU metrics into the ones associated with delay distribution moments and others associated with traffic control.

3.2.1 Delay Distribution Moments

From the PDU interarrival distribution, we can calculate three basic parameters associated with the first three moments of the distribution: mean, variance and skewness. The simplest measures are the mean, μ, and the variance, σ^2, which together form the coefficient of variation, $\mathrm{CV}^2 = \sigma^2/\mu^2$. The mean is the most basic measure for a CBR video server as it indicates whether it is pumping at the correct rate. The variance measures how consistently the server is pumping this rate, while the CV^2 provides similar information about the burstiness of the traffic.

These measures are based on the first and second moments; for the third moment we use the skewness defined as $\nu = E[(X-\mu)^3]/\sigma^3$, which is a measure of symmetry of the delay distribution and indicates the uniformity of the delay variation.

3.2.2 Traffic Control Metrics

A second group of metrics is related to traffic control. In this case, we are more interested in studying the compliance of the video server output with a network contract (i.e. peak rate and burstiness). Examples of useful measures are the 1-Point PDU Delay Variation (PDV) and the Generic Cell Rate Algorithm (GCRA) which is equivalent to the continuous state leaky bucket algorithm [25]. The PDV for PDU k, d_k, is defined as

$$d_k = c_k - a_k \tag{3.1}$$

where c_k is the PDU's reference arrival time and a_k is the actual arrival time. The reference arrival time is defined as follows

$$c_{k+1} = \begin{cases} c_k + T & \text{if } c_k \geq a_k \quad \text{(early arrival)} \\ a_k + T & \text{otherwise} \quad \text{(late arrival)} \end{cases} \tag{3.2}$$

where T is the period which corresponds to the desired rate. The above method tries to eliminate the effects of PDU gaps and provides a measurement solely of PDU clumping, since a clumping in the PDU traffic pattern implies a violation of the peak rate specified in the network contract.

As a measure for video services, the PDV is problematic. It is very sensitive to the average rate. For instance, if the video pump is delivering a 5 Mbps video stream, using a PDU size of 376 bytes, with a period that is slightly smaller than the nominal period T by $\Delta T = 1$ microsecond in the period, the PDV could accumulate to one second over a period of less than 10 minutes:

$$\begin{aligned} a_0 &= c_0 \\ a_i &= a_{i-1} + (T - \Delta T); \quad && \forall i \geq 1 \\ c_i &= c_{i-1} + T && \forall i \geq 1 \\ d_i &= d_{i-1} + \Delta T = i\Delta T && \forall i \geq 1 \end{aligned} \tag{3.3}$$

However, this is not the case if the rate is slightly lower. In that case $d_i = -\Delta T$, for all values of $i \geq 1$. In practical systems, a slightly different rate could be compensated for by the video client PLL and therefore should not cause such a dramatic effect in performance metrics.

The GCRA(T,τ) defines a leaky bucket running at a rate of $1/T$ with a tolerance of $100\tau/T$ and is a second-order statistic that measures the burst tolerance. If the tolerance is 0%, we only admit the PDUs that are not violating the network contract, i.e. PDUs whose interarrival time is greater than or equal to the nominal period T. By increasing the value of τ we can admit more bursty PDU traffic patterns. A less bursty source will have more PDUs admitted since it requires less tolerance in the leaky bucket to accommodate all the PDUs. It is important to note that two video streams with the same average bit rate can have totally different GCRA performance depending on how their PDUs are distributed over time.

The performance of a software-based video server can also be affected by other processes that are running on the workstation. Most of the resources such as the CPU, memory, disk bandwidth, and network bandwidth can be isolated using operating system level or user level mechanisms. Processes which require a substantial amount of bandwidth from the bus can cause erratic video pump performance.

3.3 Experimental Choice of QoS Parameters

In this section, we apply the metrics defined in Section 3.2 to experimental results obtained from the Columbia VoD testbed in order to identify which metrics are more effective in characterizing a video server. A video server must be able to support clients with largely varying bandwidth requirements. For this reason, we test our video server over a broad range of bit rates, 800 kbps (for software decoders), 2.5 Mbps (VCR quality), 5 Mbps (digital TV quality), and 10 Mbps (HQ-TV). In all cases, our video streams are MPEG-2 transport streams encoded from the movie *Robin Hood*. Each stream has a minimum duration of 15 seconds, or equivalently 50,000 PDUs for 10 Mbps (376 byte PDUs).

All measurements of the video server performance are made using an HP Broadband Analyzer, which is connected to the ATM switch. We begin testing the video server for a single CBR video stream. Afterwards we test the video server under different cross-traffic scenarios in order to model the effect of contention for resources. Finally we identify metrics defined in Section 3.2 that best characterize the video server performance.

3.3.1 Delay Distribution Moments

For each of the rates (800 kbps, 2.5 Mbps, 5 Mbps and 10 Mbps) the video server maintains the average rate of a single CBR video stream to within 0.006% of the desired rate, with 10 Mbps being this worst case. The standard deviation is relatively constant over the different rates too, ranging between 30 and 50 microseconds (Table 3.1).

Fig. 3-1 shows the probability mass function for a 2.5 Mbps video source, where the side lobes correspond to the effect of the delay introduced by IPC. The right lobe is due to the PDU which is delayed by IPC and the left lobe to an early-

Table 3.1: First, second and third moments for a single CBR video source.

CBR video (kbps)	μ (kbps)	σ (μs)	CV^2	ν
800	800.00	51.77	0.0001	0.3780
2500	2499.99	31.99	0.0007	1.1499
5000	5000.08	30.18	0.0025	1.5235
10000	9999.37	30.84	0.0105	5.2072

arriving PDU, which is an attempt by the absolute timer to compensate for the previous delay. These simple measures do not provide enough information in this case, because as will been seen, the video pump actually performs more consistently at lower rates and this cannot be seen from the mean, standard deviation or CV^2. The timer has a larger impact on streams with higher rates than on streams with lower rate. This is made obvious by the third moment statistic, skewness, which increases with the rate (Table 3.1). The skewness indicates that the probability mass function becomes less symmetric and heavier on the side of long delay.

3.3.2 Traffic Control Metrics

Fig. 3-2 shows the PDV evolution for the times between arriving PDUs for a 2.5 Mbps stream. The staircase shape is caused by the way the reference and arrival time is defined in equation 3.2. Due to the use of absolute timers, when one PDU is delayed by Δt, the next one will be sent with a period which is decreased by Δt. This causes a PDU delay, followed by a PDU clumping. When the large delay is incurred, the PDV goes negative and the subsequent reference arrival time, c_{k+1}, is updated using the actual arrival time, a_k. If d_{k-1} is the PDV just before the delay and d_k is negative, due to the delay, then the PDV for the clumping is, $d_{k+1} = d_{k-1} + |d_k|$.

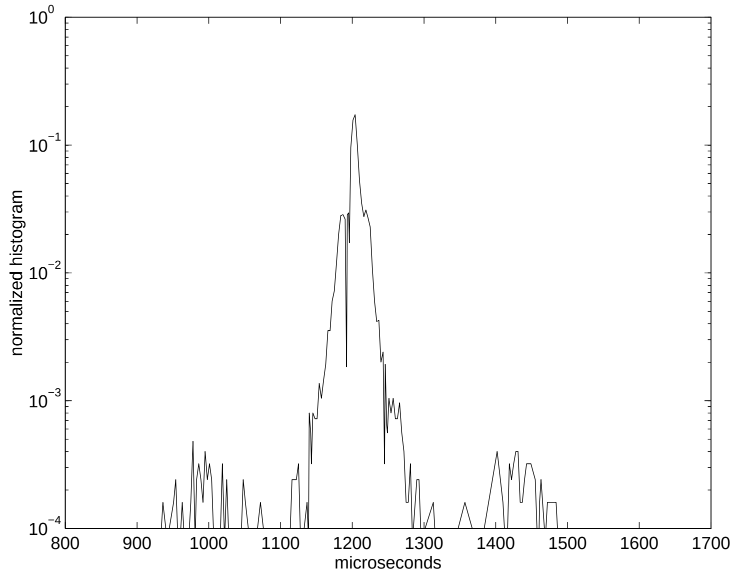


Figure 3-1: Probability Mass Function for PDU interarrival times of a 2.5 Mbps video stream.

As the rate increases, burstiness increases too. This can be seen in Fig. 3-3 which is a graph of the GCRA curves for each of the rates. Clearly, the lower bit rates have a much higher percentage of admitted cells than the higher rates. This is because the higher rates are burstier and the PDUs are often more clumped together.

Since the video pump buffers are of fixed size, regardless of rate, the number of IPCs between PDUs increases with decreasing rate. For the 800 kbps source, we observe a plateau that is caused by the increased frequency of the IPC at lower rates. In this case, more tolerance is needed to absorb this jitter.

3.3.3 Cross Traffic

As mentioned earlier, we model the video server as a virtual multiplexer. The competition for common resources among video pumps can be modeled as cross-traffic and the origin of this cross-traffic can be considered as the aggregate traffic of other video sources. Table 3.2 gives the values of the CV^2 and the average PDV

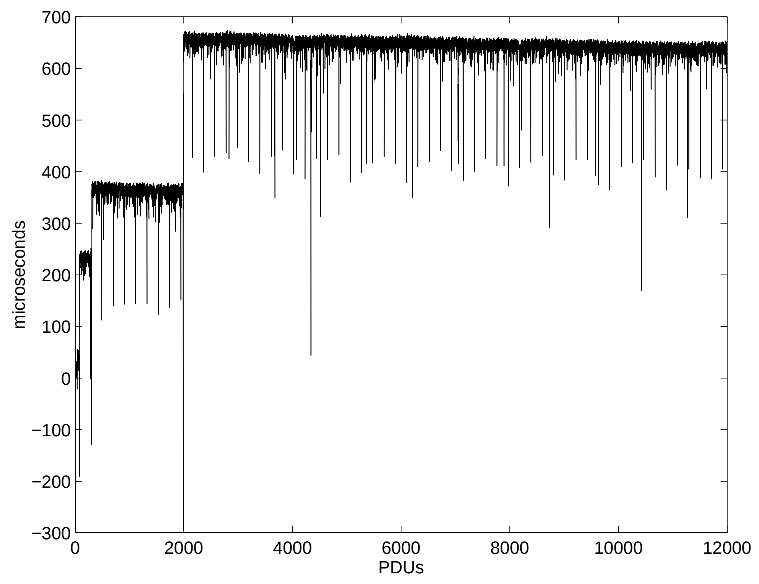


Figure 3-2: PDU Delay Variation for PDU interarrival times of a 2.5 Mbps video stream.

for a CBR 800 kbps video source and for three types of cross-traffic: deterministic, exponential and hyperexponential.

Table 3.2: QoS parameters for an 800 kbps stream under different types of cross-traffic.

Type of Cross-traffic	CV^2	1-point PDV μs
Deterministic	0.003189	1018
Poisson	0.033041	3790
Hyperexponential	0.108488	8358

The average bit rate of this cross-traffic is also 800 kbps and the latter two were generated using a gamma distribution with CV^2_{cross} values of 1 and 3, respectively. We observe that the jitter introduced in the video source is greater when the background traffic has a higher degree of burstiness, with a severe degradation when the background traffic is hyperexponential.

The average rate of the cross-traffic also has an impact on the QoS of the video

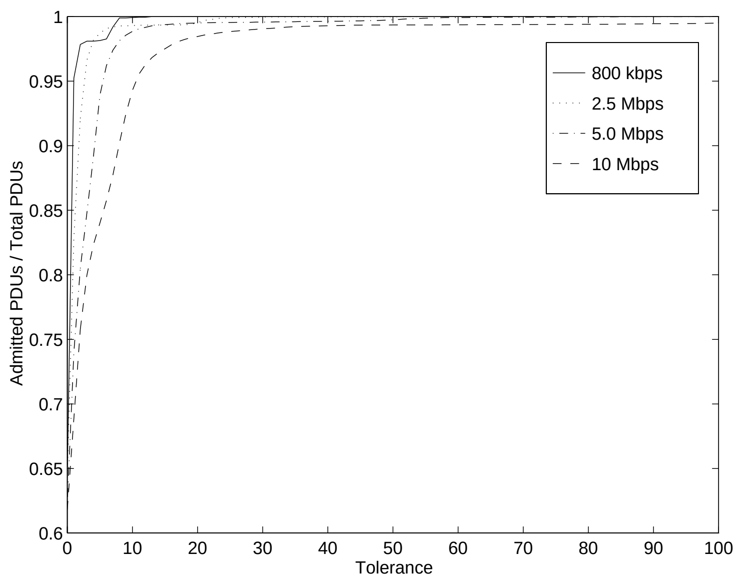


Figure 3-3: GCRA for varying CBR rates.

stream. Fig. 3-4 shows the GCRA for a target CBR 800 kbps video source when it is multiplexed with cross-traffic of average rates of 800, 2500 and 5000 kbps and exponential distribution. For a higher cross-traffic average bit rate, we see a better performance for the jitter. This result is consistent with the analysis of M+D/D/1 [39] and simulation of video sources in a multiplexer [104] where the higher bit sources always experience more jitter, since the probability of contention with other sources is higher. On the contrary, for a fixed average rate cross-traffic, the higher the target CBR video source is, the higher the jitter is. Table 3.3 shows the CV^2 for these two possible scenarios.

From the results shown in the previous sections we can conclude that the delay distribution moments measures such as the mean, variance, coefficient of variation and skewness, do not give enough information about the performance of the video server. On the other hand, the 1-point PDV is too sensitive to the momentary fluctuations of the video streams as well as the average rate. It also does not provide a stationary measure of the performance of the video server over a long period of

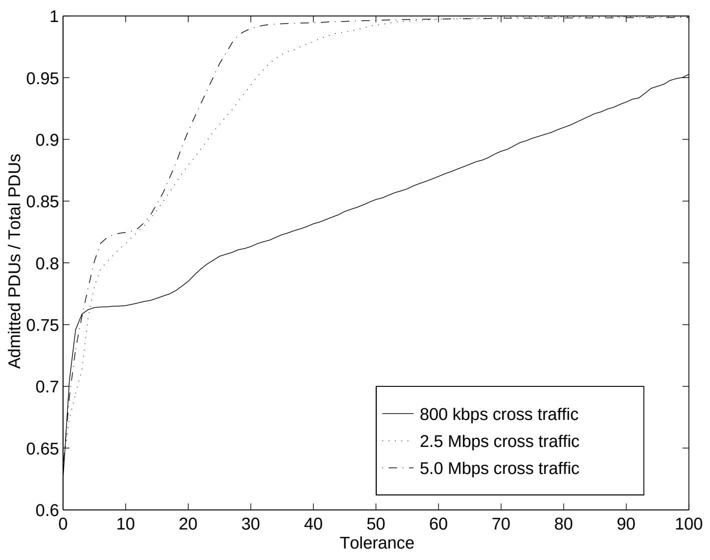


Figure 3-4: GCRA for a CBR 2.5 Mbps with varying rate exponential cross-traffic.

time and therefore does not accurately characterize the performance of a video client. We can see that a single clumping around PDU 2000 is translated into a jump in the PDV measure for the following PDUs (see Fig. 3-2). The GCRA captures the second-order statistics of the video stream or how the PDUs are distributed in bursts over time. It provides a measure of the behavior from an ideal video server over the connection time. For these reasons, the GCRA is the most informative measurement for end-to-end QoS at the video client and a good indicator of the video server performance.

3.4 Mapping Qos Parameters to the Client

Now that we have identified the GCRA as the most comprehensive QoS parameter for characterizing the video server performance, we map the GCRA at the video server into an equivalent QoS parameter at the video client. As mentioned earlier, this task is not trivial because of the different scales handled, the varying effect of errors on the information in the corrupted PDU, and the subjective criteria of the

Table 3.3: CV^2 for a CBR target video source under exponential cross-traffic.

Exponential Cross-Traffic (kbps)	CBR-Video (kbps)	CV^2
800	800	0.033041
800	2500	0.242368
800	5000	0.576097
800	2500	0.242368
2500	2500	0.025269
5000	2500	0.018787

end user. One of the most significant parameters for the video client is PDU loss. For this reason we use the network interface buffer overflow as an indication of PDU loss and therefore QoS degradation. Although this parameter alone cannot fully characterize the QoS at the video client, it is certainly a significant indicator of QoS degradation.

Subjective tests have been performed which indicate that a probability of PDU loss for MPEG-2 decoders on the order of 10^{-3} is acceptable [91]. These tests have been performed using digital set-top-boxes with ATM connections, where the set-top-box has no error concealment capabilities. We have reproduced those tests by injecting jitter, using a Network Impairment Emulator in the Broadband Analyzer, into the traffic pattern and observing the results on the digital set-top-box video clients. The frame loss and blocky effect is observed to be tolerant to an expert user at the PDUs loss rate of 10^{-3}.

Fig. 3-5 shows two frames from the MPEG-2 transport stream *Robin Hood* encoded at 2.5 Mbps. The left frame corresponds to the case where the PDU loss is $P_n < 10^{-3}$ ($P_n \simeq 10^{-4}$), whereas the right frame corresponds to the case where the PDU loss is $P_n > 10^{-3}$ ($P_n \simeq 10^{-3}$). Clearly, the right frame shows a significant decrease in the subjective QoS. It is important to note that the effect of PDU loss depends on the the encoding technique (MPEG-2) as well as the error concealment capabilities of the decoder.

For a given PDU loss rate, a higher rate video stream translates into a higher absolute number of PDUs lost per second. However the most critical case is when losses occur in the header information (i.e. sequence, group of pictures). The occurrence of these PDU losses is almost independent of the rate of the video stream since the number of PDUs containing this header information is the same regardless of the rate. We have empirically verified this fact by confirming that the subjective QoS is similar for different rates and a constant PDU loss.

Now that we have identified a relationship between the subjective QoS and a PDU loss rate at the video client, we map the GCRA performance at the video server with the PDU loss probability (i.e. overflow in the network interface) at the video client to make the connection from video server performance to video client subjective performance.

The GCRA(T, τ) algorithm is equivalent to the continuous-state leaky bucket algorithm [25]. It can be modeled as a finite capacity buffer of size $T + \tau$. Every time a PDU conforms, the buffer is incremented by T units of content and the buffer is continuously drained at a rate 1 unit of content per unit of time. A PDU is admitted on arrival if and only if there is enough room in the buffer to accommodate it.

The primary difference between the network interface and GCRA(T, τ) is the granularity, because the network interface is dimensioned in units of PDUs. Every time a PDU arrives it is served with a service time equal to the ideal interarrival time. We are assuming that the service time of the client's network interface is driven by the rate of the video stream. Under this condition, if P_n is the probability of PDU overflow in the network interface with a memory of n PDUs, we can establish the following relationship:

$$\text{GCRA}(T, nT) = 1 - P_n, \;\; n = \text{ 0,1,2...}, \tag{3.4}$$

This relationship can be justified by Fig. 3-6, where a source with a slightly higher bit rate than the nominal is analyzed with both the GCRA(T,τ) and the network interface state. We notice that the GCRA(T,τ) is always less than the network interface occupancy and is equal for each $\tau = nT$. Therefore, for the intermediate values of τ, we have the following relationship that is based on the monotonicity of the function GCRA(T,τ):

$$\begin{aligned} 1 - P_n \quad &\leq \; \text{GCRA}(T, T(n+\alpha)) \quad \leq 1 - P_{n+1}, \\ &n = 0, 1, 2 \ldots ; \, 0 \leq \alpha \leq 1. \end{aligned} \tag{3.5}$$

From the previous analysis, a network interface with a capacity of 1 PDU will have a tolerance of 100%. This tolerance is related to the maximum burst size in any time interval of size T. For a $\tau = nT$, the maximum burst the system can tolerate in any interval T will be equal to $n + 1$ PDUs. However this burst size is a maximum; two consecutive smaller bursts could lead to overflow as well, since the GCRA(T,τ) is a second order statistic with memory.

In the next section, the relationship established in equation 3.5 will be used to define a metric that jointly captures the QoS constraints both of video server and video client.

3.5 Schedulable Region

We now make use of our previous results in mapping the video server QoS into the video client QoS by defining a schedulable region. This region indicates the set of admissible combinations of traffic types which will guarantee a certain level of QoS for each individual stream. It is important to make the distinction between guaranteeing QoS for the aggregate traffic and guaranteeing QoS for each stream. A guarantee for the aggregate traffic only provides an average QoS for the set of video streams. The QoS for a particular stream could likely degrade below the guarantee for the aggregate traffic. Our definition of QoS is on a per-stream basis and specifies a minimum QoS that a stream will encounter.

3.5.1 Definition of Schedulable Region

Let us assume I is the number of possible types of video streams at the video server. Each type of video stream is characterized by the pair (R_i, ϵ_i). The first component, R_i, is the average bit rate for video streams of Type i, with $i = 1, \ldots, I$. The second component, ϵ_i, is the acceptable PDU loss probability at the video client for video streams of Type i. Let us also define $v_{i,j}$ as the jth video stream of Type i that is simultaneously pumped. The schedulable region, $S \in \mathbf{N}^I$, is defined as the set of all possible compliant combinations of simultaneously pumped video streams. We say that a combination of video streams, $\{v_{1,1}, v_{1,2}, \ldots, v_{1,\Phi_1}; v_{2,1}, \ldots, v_{2,\Phi_2}; \ldots; v_{I,1}, \ldots, v_{I,\Phi_I}\}$, expressed as the $\mathbf{N}^I$ vector $\mathbf{\Phi} = (\Phi_1, \Phi_2, \ldots, \Phi_I)$, where Φ_i is the number of pumped video streams of Type i, is compliant and belongs to S if it conforms to the following conditions:

$$\mathbf{\Phi} \in S \Longleftrightarrow \begin{cases} \sum_{i=1}^{I} \Phi_i \leq C \\ r_{i,j} = R_i \\ P_{n,i,j} < \epsilon_i \\ \forall i, j \qquad 1 \leq i \leq I, 1 \leq j \leq \Phi_i \end{cases} \tag{3.6}$$

where C is the number of processors allocated to the video server, $P_{n,i,j}$ is the probability of PDU loss at the video client for the jth video stream of Type i with buffer space for n PDUs in the network interface.

The first constraint in equation 3.6 implies that the video server can only serve a maximum number of video streams according to its processing capacity. In our software architecture it is possible to run several video pump objects on a single processor. In this case, the operating system scheduler acts as the multiplexer. The scheduler task-switches the video pumps into and out of the CPU so that only one pump is active at a time and each pump receives a slotted period of time to operate. This results in the scheduler performing as a TDMA multiplexer. Our interest here is to demonstrate the multiplexing performance of the video pump and its interaction with the network interface. Therefore, we restrict our attention to video pump objects running on separate processors.

The second constraint in equation 3.6 enforces that video streams are being pumped at the right rate to avoid severe underflow situations at the video client (i.e. $r_{i,j} < R_i$). Finally the third condition ensures a compliance with a network contract, such as GCRA, which is mapped into a video client QoS parameter, $P_{n,i,j} < \epsilon_i$, as was seen in equation 3.5.

3.5.2 Methodology and Example of Schedulable Region

In order to calculate the schedulable region, S, for a specific video server, we use the following methodology. First, we define how many types of video streams to consider. There is a trade-off between the unlimited number of service classes and the reduced number of traffic classes that the network can handle. For a given processing capacity of the video server, C, and a given number of types of video streams, I, there are C^I possible vectors or video streams combinations. Only a subset of these vectors are compliant with conditions on S in equation 3.6. Each video stream from a particular vector is individually traced using a Line Interface (LIF) from the broadband analyzer. The captured trace is analyzed using a Traffic Monitor, which checks the average bit rate, $r_{i,j}$, and the GCRA. Only if all the video streams in a vector are compliant with the conditions in equation 3.6, can that vector be contained in S. We have to note, that it is not necessary to check all the C^I combinations. For instance if $(\Phi_1, \Phi_2, \ldots, \Phi_I) \notin S$, it will imply that $(\Phi'_1, \Phi'_2, \ldots, \Phi'_I) \notin S$ for all $\Phi_i \leq \Phi'_i \leq C$. For this reason, choosing the initial vectors carefully can save many calculations for the schedulable region, S.

Following this methodology, we calculate S for the Columbia video server. In this example, we have chosen two traffic classes, Type 1, with $R_1 = 800$ kbps and Type 2, with $R_2 = 2.5$ Mbps. The QoS constraint is $\epsilon = \epsilon_1 = \epsilon_2 = 10^{-3}$. The processing capacity in our server is $C = 5$ (ONYX 6-processor system, restricting one video pump per processor). Fig. 3-7 shows how the video stream combination $\{v_{1,1}, v_{1,2}, v_{1,3}; v_{2,1}, v_{2,2}\}$ or vector $(3, 2)$ is checked for its inclusion in S. We can trace all the video streams simultaneously or in subsequent measurements depending on the number LIF available in the broadband analyzer, since it is possible to reproduce the same scenario in the video server. Fig. 3-8 shows several schedulable regions for these two types of video streams. Each region corresponds to different PDU buffer sizes, n, in the network interface. In this case we use a PDU size of 376 bytes. The higher the buffer is, the higher GCRA tolerance is and more video streams combinations are compliant with equation 3.6. We also observe that increasing the PDU size above a certain value does not enlarge S. At that value, all the potential jitter has been absorbed by the network interface and the limitation is imposed by the video server with the second condition in equation 3.6.

Note that the calculation of S is an off-line procedure and must only be performed once for each video server-video client combination. We know beforehand the characteristics of the video streams stored in the video server. Therefore we can have as much accuracy (i.e. different types of video streams) as desired. It is convenient however to have a similar number of service classes for the video server as traffic classes for the network. In this way, the admission control can be performed in a more straightforward manner, since video server and network are using similar metrics. Moreover, a commercial VoD system will likely have just a few traffic classes, actually maybe only one (i.e. all the streams encoded at a nominal rate).

Although admission control procedures are outside the scope of this work, we describe two different scenarios where S can be applied. The first scenario is when the video server and the video client are connected to the same ATM LAN. In this case, the main source of QoS degradation is the video server since the low utilization of such networks prevents the switches from building up significant congestion. Therefore, a direct video server to video client mapping can be applied considering a transparent network which is not introducing QoS degradation. We have empirically verified this fact in the Columbia VoD testbed. We have simultaneously traced the traffic pattern at the output of the video server and at the input of the video client, and no significant differences (i.e. GCRA performance) have been detected, for even a moderately loaded ATM switch.

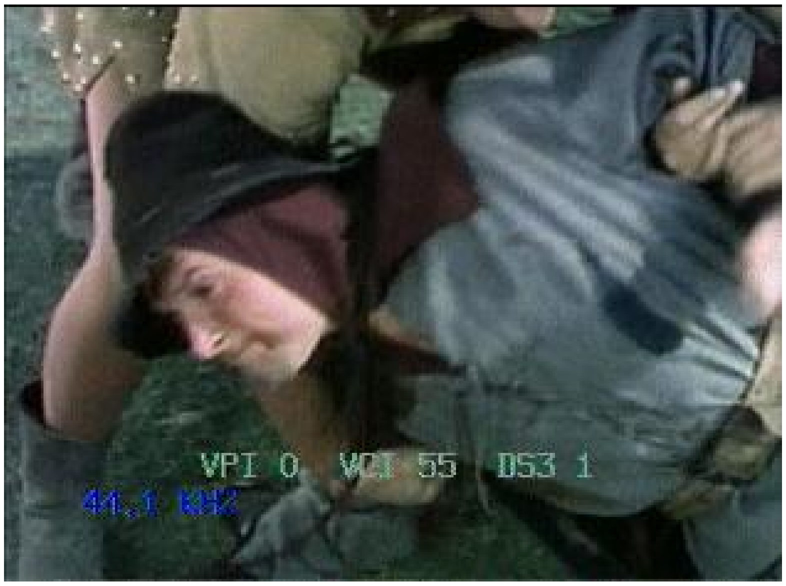
The second scenario is when the video server and the video client are interconnected through a virtual path in a ATM WAN. In this case, the switches along the path introduce QoS degradation, since the utilization of the network is higher than in the ATM LAN case. Let us assume that the virtual path consists of an interconnection of L switches. We can associate with each switch a schedulable region, S^l, with $1 \leq l \leq L$. As long as the traffic classes and QoS constraints considered in each switch were the same ones as in the schedulable region of the video server, S^0, we can assume an equivalent schedulable region, $S^{eq} = \min_{0 \leq l \leq L} \{S^l\}$ for admission control purposes. The equivalent QoS constraint, which is the one to map in the video client, will be $\epsilon^{eq} = \sum_{l=0}^{L} \epsilon^l$, which is a conservative bound. More sophisticated admission control procedures can be applied to have a more efficient use of resources (i.e. schedulable region with more than one QoS constraint per type of video stream). The use of one technique or another is a trade-off between the cost of bandwidth in the network and the cost of processing in the network nodes.

3.6 Concluding Remarks

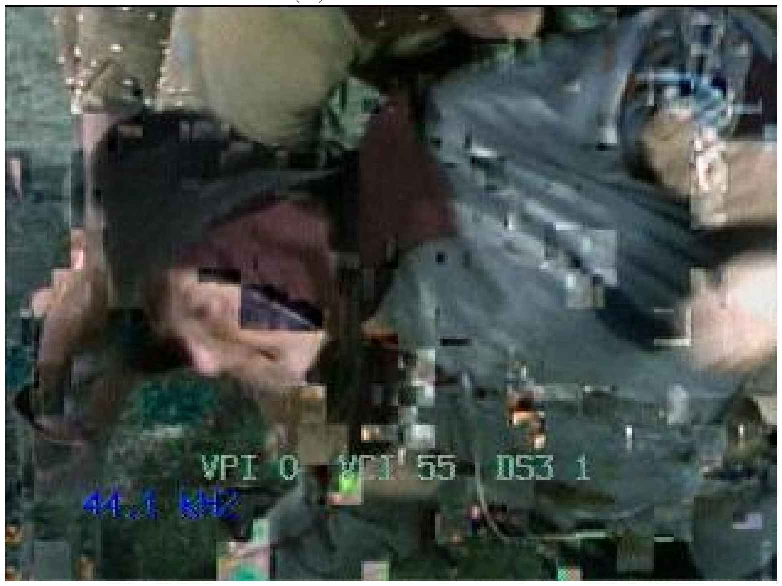
We have presented a novel and simple approach for mapping end-to-end QoS in VoD services. Using this approach, we have derived a schedulable region for a video server which guarantees end-to-end QoS, where a specific QoS required in the video client translates into a QoS specification for the video server.

We have defined the QoS metrics of the video server, and have applied them to several experimental results obtained from the Columbia video server. From this analysis, we have concluded that the moments from the delay distribution do not give enough information about the performance of the video server. On the other hand, a traffic control measure like the PDV does not provide a stationary measure of the performance of the video server over a long period of time. The GCRA, as a measure of how PDUs are distributed in bursts over time, provides the most complete characterization of the performance of a video server.

Once we identified a comprehensive QoS parameter for the video server, we then mapped the GCRA at the video server into an equivalent QoS parameter at the video client. We established a relationship between the subjective QoS and the PDU loss rate at the video client. Then, we linked the GCRA performance at the video server with the PDU loss probability (i.e. overflow in the network interface) at the video client. From this relationship we defined the concept of a schedulable region for a video server which models the capacity of the video server under video client QoS constraints. Our model can be connected to network QoS admission control models to create a unified approach for admission control.



(a)



(b)

Figure 3-5: Jitter effect on the subjective QoS. (a) Robin Hood frame with a PDU loss, $P_n < 10^{-3}$. (b) Robin Hood frame with a PDU, $P_n > 10^{-3}$.

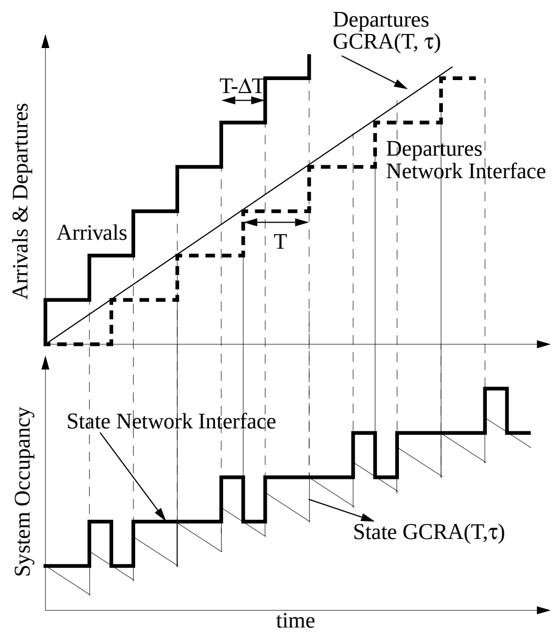


Figure 3-6: Relationship of GCRA(T, τ) with video client network interface buffer occupancy.

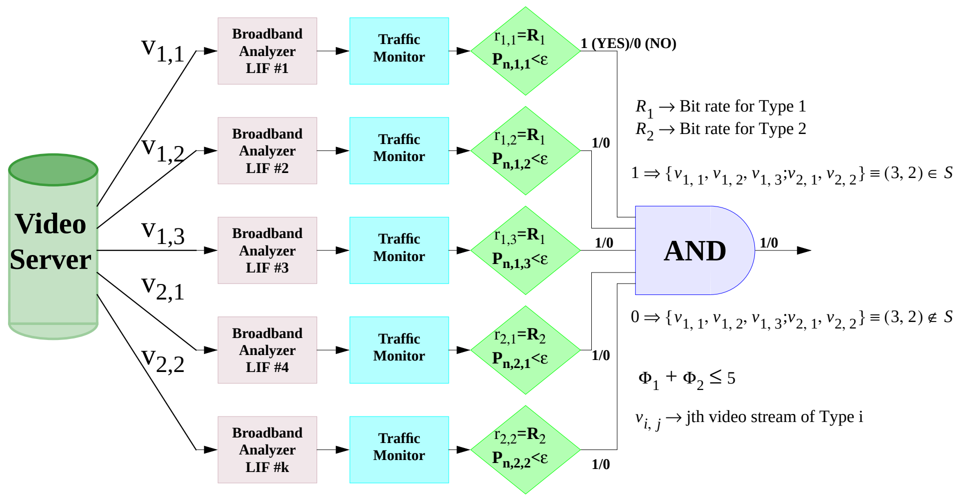

Video Server
v1,1
v1,2
v1,3
v2,1
v2,2
Broadband Analyzer LIF #1
Broadband Analyzer LIF #2
Broadband Analyzer LIF #3
Broadband Analyzer LIF #4
Broadband Analyzer LIF #k
Traffic Monitor
r1,1=R1 Pn,1,1<ε
r1,2=R1 Pn,1,2<ε
r1,3=R1 Pn,1,3<ε
r2,1=R2 Pn,2,1<ε
r2,2=R2 Pn,2,2<ε
1 (YES)/0 (NO)
1/0
AND
R1 → Bit rate for Type 1
R2 → Bit rate for Type 2
1 ⇒ {v1,1, v1,2, v1,3; v2,1, v2,2} ≡ (3, 2) ∈ S
0 ⇒ {v1,1, v1,2, v1,3; v2,1, v2,2} ≡ (3, 2) ∉ S
Φ1 + Φ2 ≤ 5
vi,j → jth video stream of Type i

Figure 3-7: Procedure to obtain the schedulable region for two type of traffic classes at the video server. Example for the video stream combination $\{v_{1,1}, v_{1,2}, v_{1,3}; v_{2,1}, v_{2,2}\} \equiv (3, 2)$.

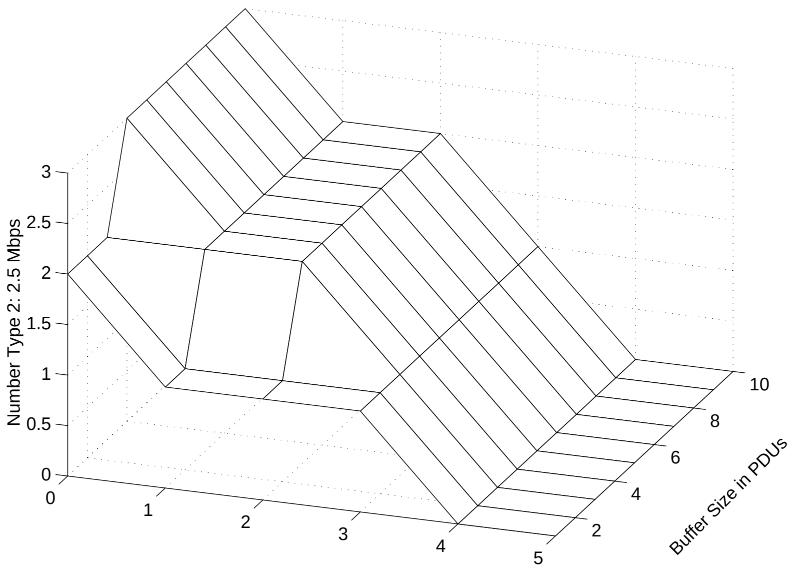


Figure 3-8: Schedulable region for Type 1 and Type 2 traffic vs. video client network interface buffer size for $P_n < 10^{-3}$ in the video client.

Part II

Non-QoS Streaming

Chapter 4

Adaptable Media for Non-QoS Streaming

4.1 Introduction

In the previous chapters we examined the case where the network can guarantee a certain level of QoS. Although it is more difficult to build a network with QoS guarantees due to issues of scalability and efficiency, the existence of QoS networks make designing applications much easier.

In the case where the network does not provide QoS guarantees, the application must be fully aware of the environment in which it is operating. The Internet is the most prevalent example of a network without QoS today. The use of the Internet has grown tremendously and much of what has fueled this growth are applications for which the Internet was not originally designed to handle. Building application for the Internet requires innovative approaches that would be unnecessary if the Internet had been designed with QoS.

Real time multimedia applications are the most obvious example of applications that require innovative approaches. In a network with QoS, the application informs the network about its resource requirements and the network fulfills this request if the resources are available. The resources are then guaranteed for the life of the connection.

In non-QoS networks the reverse is true. The application must extract information from the network about the available resources and determine a way of dealing with the limited resources gracefully. Oftentimes, and especially in the case of the Internet, the information from the network is provided implicitly. This type of application requires techniques for gathering this implicit network information as well as techniques for using the information effectively.

In QoS networks, applications often specify such parameters as average bandwidth and maximum delay. In non-QoS networks, applications must determine the available bandwidth in the network and then send only that amount of data. Sending more only adds to congestion and does not necessarily guarantee better quality. This requires the application to be able to change the bandwidth requirements of the data in real time. The means by which applications can perform this real time adaptation is the subject of this chapter. Techniques exist for all types of media including video, audio, images and text.

In all cases, the server of the real time media must know which type of media it is dealing with and must know the techniques specific to adapting that media. The adaptation does not have to be performed only at the source. By adding intelligent gateways in the network it is possible to defer adaptation to points within the network. An argument against this strategy is that it adds much complexity to the network.

4.2 Video

4.2.1 MPEG

MPEG is one of the most commonly used codecs [37]. This is primarily due to the fact that it is an international standard and provides very good compression for bitrates greater than 1 Mbps [57].

MPEG uses many different techniques to increase compression efficiency. The highest level method exploits redundancy between each frame in the video sequence. Since video is a sequence of consecutive images, there is almost always a relationship between one frame and the next, except in the case of a scene change. For this reason, MPEG uses motion compensation by sending only the difference between the two frames.

There are three different frame types. I frames do not use motion compensation to improve compression. P frames use previous I frames as a reference for their motion compensation. Finally, B frames use previous and future I and P frames for motion compensation. For this reason, losing an I frame will cause the loss of all frames which have been predicted from it; however, losing a B frame will only cause the loss of that frame.

Each frame is broken into blocks 8 pixels high by 8 pixels wide. The block is then transformed using the two-dimensional Discrete Cosine Transform (DCT). The resultant coefficients are quantized using weights according to their importance to subjective quality. The coefficients are then run length encoded to exploit the fact that many of the higher frequency coefficients are now zero. The final step is to Huffman code the resultant run lengths. This information is buried deep within the bitstream and many layers of headers must be traversed to arrive at this block data.

MPEG-2 introduced a variety of scalability options, but unfortunately these were too limited to be of much use for non-QoS networks. Each scalability mode (Signal to Noise Ratio (SNR), spatial and temporal) requires a base layer and an optional enhancement layer. The base layer is mandatory for decoding; the enhancement layer provides added quality. This gives a choice of only two possible rates (base or base plus enhancement), which does not provide enough variability to be useful in a situation like the Internet.

From this description it is clear that MPEG is not inherently scalable. Not all frames types can be dropped, rendering temporal scalability complex. In addition, it is non-trivial (but quite possible) to drop coefficients to obtain SNR scalability. Just as the Internet was not designed to transfer non adaptive real time media since it has no QoS guarantees, MPEG was not originally designed to be adaptive. Nonetheless there are ways of changing the bandwidth requirements of MPEG in real time.

4.2.1.1 Feedback to the Encoder

When the video source is a live video feed and the internals of the encoder are accessible to the application it is possible to use the information from the network to adapt the quality and bandwidth requirements of the video. These techniques have been examined in [9, 41, 58, 73, 96], the basic limitation is that this will not work for precompressed video.

4.2.1.2 Frame Dropping

Frame dropping, or temporal scalability, is the most common way of adapting MPEG to non-QoS network conditions [23]. If the application determines that the network is being overloaded by too much data, it will begin to drop frames from the video stream. In MPEG, this is done by identifying and dropping the B frames from the bitstream. Dropping P frames or I frames is only used as a last resort because other frames are dependent on the P and I frames. If all of the other dependents are dropped, then dropping a P or I frame is acceptable. Dropping frames alters the original timing information and this must be communicated to the client, often through the use of a transport protocol such as the Real-time Transport Protocol (RTP). However, frame dropping provides a very coarse means of adjusting the rate since the smallest unit of adjustment is an entire frame.

4.2.1.3 Dynamic Rate Shaping

Dynamic Rate Shaping (DRS) is a process by which the bandwidth requirements of MPEG video (or any block-based transform coding scheme) can be adapted in real time [33, 49]. DRS provides the ability to dynamically change the bit rate of a precompressed stream. Each MPEG frame consists of a quantity of header information which cannot be removed. The rest of the frame consists of DCT coefficients. In its simplest form, DRS selectively drops these coefficients from the MPEG-1 or MPEG-2 bit stream that are least important in terms of image quality.

DRS operates on the compressed video bit stream, eliminating DCT coefficient run-lengths. The coefficients to be eliminated are determined using Lagrangian optimization, resulting from an operational rate-distortion formulation. The problem is complicated by the fact that MPEG coding utilizes predictive and interpolative modes for motion compensation, and thus any modification of the bit stream will result in error propagation. It has been experimentally shown in [33] that, if decisions within each frame are optimal, then ignoring the accumulated error does not impact the quality in any significant way. Thus, the algorithm can operate in a memoryless mode that has significantly less complexity, while achieving essentially optimal (within 0.3 dB) performance. In addition, DRS is much less complex than a complete decoder, making it feasible to run on a video server.

There are several advantages of this technique. DRS can meet any reasonable bandwidth estimate exactly. This means that the bandwidth estimate is more fully utilized. It also maintains the original frame rate of the video. Lastly, DRS decouples the adaptable media from the encoder so that it can be used for both live and stored video streams. A combination of DRS and frame dropping will probably yield better perceptual results than either technique working alone, especially for large rate reductions.

4.2.2 H.263+

H.263+ is a codec very similar to MPEG except that it was designed for very high compression and for transmission over low bandwidth links [24, 36, 100]. In the evolution from H.263 to H.263+ the codec was been given several scalability features [89]. H.263+ suffers from the same lack of variability in the scalable output as MPEG because it also employs a very coarse level of adaptation using a base layer and an enhancement layer for SNR and spatial scalability. For temporal scalability, B frames are used which are similar to B frames in MPEG. DRS can also operate on H.263+.

4.2.3 Highly Scalable Codecs

Just as the rise of the Internet has demanded adaptive applications, so has it fueled the need for scalable codecs for a variety of different environments [40]. In [95] the authors present a codec for very low bitrates which is still scalable from 10 kbps to 30 kbps. The authors in [92, 93, 94] present a codec which is highly scalable. From a single encoded stream, users from 50 kbps up to 3 Mbps can be supported. This is done through the use of both spatial and temporal scalability modes.

4.3 Audio

4.3.1 Silence Detection

Speech is a specific type of audio content which has a very important characteristic. In audio conferencing or a typical one-on-one conversation, each participant speaks for only about half of the time. When the speaker is silent there is no need to use valuable bandwidth sending silence. In addition, by removing silence it is possible to adapt the playout rate to reduce the end to end delay [81]. Therefore, silence detection and removal is an essential building block of any multimedia conferencing system [50]. Silence detection is a technique which allows for the graceful adaptation to both the available bandwidth and delay. Another technique for adapting audio is to store several streams at different rates. Then the application can switch between various streams to accommodate the fluctuations in the Internet [13].

The purpose of silence detection is to identify and remove long runs of silence from the audio signal stream. Its importance as a building block stems from two basic problems. First, in video conferencing the bandwidth requirements of video are significant, and any reduction in terms of the total bandwidth required by the system is an important benefit. We should note that the bandwidth requirements for high-quality audio in a multimedia system can be quite close to that of low-quality video, and hence this may represent a significant fraction of the total bandwidth. For example, in the MBONE (the multicast enabled part of the internet) video is transmitted using H.261 at a rate of 128 kbps to 256 kbps, which is on the same order of magnitude as 64 kbps audio. Of course, silence detection and removal cannot solve all of the problems in multimedia communications; no amount of silence detection can help if there is simply not enough bandwidth in the network to support audio transmission.

Second, and more importantly, the removal of silence helps to reduce the end-to-end delay experienced by the users [34]. This is because it transforms the constant bit rate source into a bursty one which can be accommodated more easily in networks with limited QoS guarantees.

The generality of the concept of a multimedia communication system makes it difficult to capture it in a universally acceptable definition. For our purposes, such a system is assumed to be composed of a general purpose computer and a computer network. The most important characteristic of such an environment is that all data transfer and communication is handled in packets. This includes acquisition (delivery of data from the audio device to the operating system), application handling (transfer of data by the application between the audio device and the transport layer), and network delivery (actual transfer of data over the network) at all protocol layers. A typical example would be a set of workstations connected via an Ethernet or ATM local area network.

There is a direct tradeoff between bandwidth availability and end-to-end delay. To illustrate this, consider the example of a constant bit rate audio signal being transported over a multi-access network such as Ethernet. The signal is buffered at the receiver, using a buffer size of B samples, and playback starts after a prespecified buffer occupancy B_{w} is reached. The end-to-end delay D is determined by the time needed to obtain the audio data (acquisition delay) τ_{a}, the transmission delay τ_{t}, and the waiting time in the receiver's output buffer τ_{w}. The acquisition delay is a result of the frame-based structure of the sampling process; the audio samples are placed into a buffer by the audio device driver and delivered to the application by the operating system when the buffer has filled. We then have:

$$D = \tau_{\mathrm{a}} + \tau_{\mathrm{t}} + \tau_{\mathrm{w}} \ . \tag{4.1}$$

If the end-to-end delay of the network were constant, then the buffer occupancy would remain constant on the average since both the source and receiver produce and consume audio samples at the same rate, ignoring any possible clock mismatch. Denoting the sampling rate of the audio signal as r, in steady state we would then have:

$$D = \frac{B_{\mathrm{a}}}{r} + \tau_{\mathrm{t}} + \frac{B_{\mathrm{w}}}{r} \ , \tag{4.2}$$

where B_{a} is the acquisition buffer size.

In case significant congestion occurs in the network, audio frames at the source will start accumulating, awaiting transmission, which increases τ_{t}. After congestion subsides, these frames will be transmitted to the receiver, and upon arrival will encounter an empty buffer if the duration of the congestion is long enough. Denoting by τ_{c} the congestion duration, receiver buffer starvation will occur if:

$$\tau_{\mathrm{c}} \geq B_{\mathrm{w}}/r \ . \tag{4.3}$$

Since audio samples cannot be removed from the receiver buffer faster than their playback rate r, the end-to-end delay increases by the amount of time the receiver buffer remains empty, $\Delta\tau = \tau_{\mathrm{c}} - B_{\mathrm{w}}/r$. The new end-to-end delay, in the case of receiver buffer starvation is:

$$D' = \frac{B_{\mathrm{a}}}{r} + \tau_{\mathrm{t}} + \frac{B_{\mathrm{w}}}{r} + \Delta\tau = \frac{B_{\mathrm{a}}}{r} + \tau_{\mathrm{t}} + \tau_{\mathrm{c}} \ . \tag{4.4}$$

Note that this increased end-to-end delay will be present for the remainder of the session, or even possibly increased if congestion periods longer than τ_{c} occur in the future. Each time silence is removed, τ_{c} is decreased by B_{s}/r, where B_{s} is the amount of silent data removed.

The output buffer at the receiver can be thought of as an M/D/1 queue, which has random arrivals and fixed length packets. The audio data arrives at the queue at a rate of r samples/sec and is played back at the same rate. Therefore the offered load to the queue is 1. This means that the queue will eventually fill. In practice, the buffer is of finite size and the probability that it will overflow due to jitter can be reduced by making it large enough. Such an approach is typically unacceptable, however, since it would incur a very high end-to-end delay.

In multi-access networks jitter can be extremely variable; experiments [35] have shown that the end-to-end delay on a moderately loaded Ethernet can reach 1 sec in less than 1 min. Long distance telephony standards prescribe end-to-end delays as being acceptable if they are below 200 msec. Even in applications not involving person-to-person communication, such as video-on-demand, end-to-end delay is still important because it has a direct impact on interactive response time.

Since it is not possible to control the transmission delay in this type of network, all other controllable delays must be minimized; this includes the acquisition/processing time and the buffer waiting time. One of the purposes of silence detection and removal is to reduce the arrival rate so that the offered load is less than 1. This is accomplished by exploiting the fact that during an average conversation, each subscriber speaks only 40%-50% of the time [32]. If no data were sent the other 50%-60% of the time, the arrival rate would be reduced to about $0.5r$ samples/sec and the offered load would be reduced to about 0.5. The expected number of packets waiting in an M/D/1 queue is [88]:

$$E(n) = \frac{\rho}{(1-\rho)}(1 - \frac{\rho}{2}) \ . \tag{4.5}$$

With $\rho = \frac{1}{2}$, $E(n) = \frac{3}{4}$ packets. Thus, the output buffers would not fill and on average there would only be about $\frac{3}{4}$ packet in queue.

Silence detection and removal for a video conferencing (or general multimedia communication) system is not concerned with inter-syllabic or inter-word silences. In fact, removal of these silences would degrade the quality of speech due to the temporal non-linearity. It would create an effect of talking very quickly since the syllables and words would run very closely to each other. Studies have shown that 99.56% of "continuous speech" segments have periods of silence smaller than 150 msec [63]. For this reason, and using a 64 kbps (8 kBps) signal as an example, removal of a block of silence smaller than 1200 bytes would reduce the quality of speech. Also, the performance improvement gained by removing inter-word and inter-syllabic silence would be quite small in comparison to the substantial removal of 50%-60% of the signal, which will not affect the perceived quality of the audio.

Summarizing the above discussion, a silence detection algorithm for this particular environment should then meet the following five requirements:

1. It should remove as much silence as possible (in lengths of 150 msec and more) and should not affect the quality of speech;

2. it should have fairly low complexity since software implementations are desirable, and also because the processing delay will be added to the end-to-end delay;

3. it should require a small buffer since it is operating in an environment with limited resources;

4. it should be able to make a speech/silence decision within a frame, since storage of a frame would increase the end-to-end delay by the frame's duration; and

5. it should detect long runs of silence rather than short ones.

An additional reason in support of the last requirement is that when silence is removed from a frame, the following samples in the frame must be moved back to the point where silence occurred in order to create a continuous—yet shorter—packet. If this happens several times within a frame, the processing time for the memory-to-memory copies can become significant. The argument still holds even if scatter-gather I/O is supported by the transport layer routines.

In Section 4.3.2, several recently proposed silence detection schemes are compared with the desired characteristics ennumerated above, and it is shown that none meets all of our design criteria. In Section 4.3.3 we describe a novel, low-complexity silence detection algorithm that operates directly in the non-linear μ-law domain. The algorithm capitalizes on the small- and large-signal behavior of the μ-law companded speech waveform; due to its low complexity, it can be easily implemented in software and can be directly incorporated into existing multimedia communication system designs. In Section 4.3.4 we present experimental results that verify the effectiveness of the proposed algorithm. We also perform a comparative analysis with a recently proposed silence detection scheme that meets several of our design criteria, and show that our approach outperforms it in several key areas.

4.3.2 Previous Work

Since the issue of silence detection has been explored time and again since the Time Assigned Speech Interpolation (TASI) plan in 1959 [15], a large variety of research is available which approaches the problem from a broad spectrum of viewpoints. In its simplest form it can be a magnitude based decision. The algorithm compares the magnitude of the signal against a preset threshold. If a percentage of the data is smaller than the threshold, silence is declared. This algorithm has fairly mediocre performance in the presence of any background noise. It does however, meet the other requirements for low complexity, low buffer size, large run length, and intra-frame decision. More sophisticated approaches are based in differential decision logic [34]. Here the speech/silence decision is based on the difference between successive samples. The heuristic behind this is that speech has larger magnitude variations than silence or even moderate background noise.

Recent research has increased performance but at the added cost of complexity. Rangoussi, et al., approached the problem from the mobile communications perspective where very low signal to noise ratios exist. Their work required a robust, even if computationally more demanding, method. They used third-order statistics by exploiting the non-linearity of speech to accurately end-point the silence/speech changeovers [82]. Un and Lee developed a technique using linear delta modulation bit streams [97]. DonVito and Schoenherr used the advantages of subband coding to aid in their attempts to perform silence detection [31]. Other currently used methods are zero-crossing rates, signal energy, one sample delay correlation coefficient and prediction error energy [86]. The difficulty with all of these is that they were not designed to tackle the specifics of the presented problem and are also too complex, particularly for real-time software-based implementations on general purpose computers.

An algorithm proposed by Savoji [86], enhancing an algorithm originally proposed by Lynch [63], satisfies several of our criteria. The modified algorithm has the lowest complexity of the available research and still performs quite well. It creates and maintains adaptive metrics for speech and noise and then uses them, coupled with the short term energy, to make silence/speech decisions. The algorithm first transforms the data by converting from μ-law and then high-pass filters it, as shown in Fig. 4-1. Then the metrics are established by filtering the transformed data. The resulting metrics, coupled with a simple short term energy calculation, form the basis for a speech/silence decision.

Although its complexity is low compared with the other schemes, it uses 4 filters that each require 1 previous sample. It is shown in Sect. 5.2.4 that this approach meets most, but not all, of the five requirements. This technique is used for comparison purposes with the algorithm proposed in this chapter.

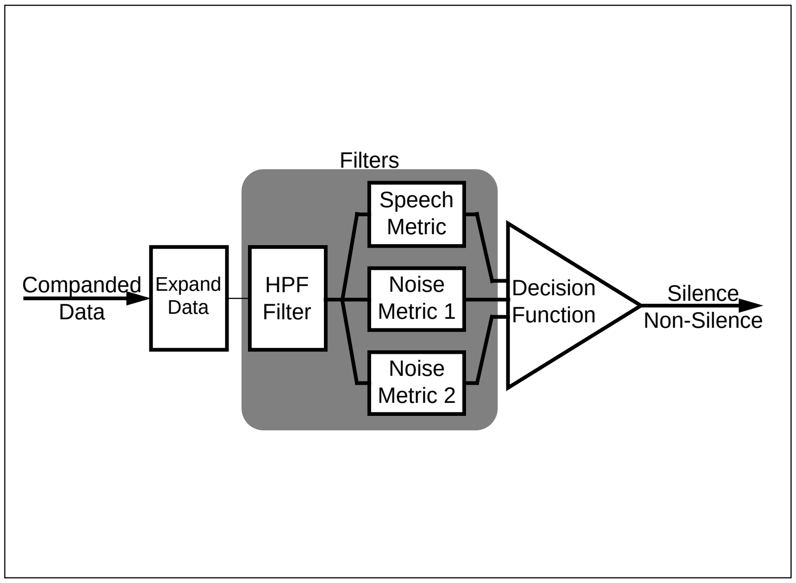


Figure 4-1: Block diagram of Savoji algorithm.

4.3.3 Non-Linear Silence Detection in the μ-law Domain

When obtaining an audio frame, the input device collects the data and then μ-law compands it. Companding a signal *com*presses it at the source and then ex*pands* it at the receiver. In all of the previously mentioned work, data are decoded to their linear form, with the exception of Drago [32]. The authors there use an approximation by assuming linearity at low signal levels. Since this conversion consumes valuable CPU time, our proposed algorithm exploits the non-linearity of the μ-law rather than trying to sidestep it.

The μ-law compander is used to give finer quantization to low signal levels than would ordinarily be obtained through linear quantization. If the input signal is x and the output y, then the formula for the μ-law transformation is

$$y = \frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \quad \text{for} \ -1 \le x \le 1 \ , \tag{4.6}$$

where $\mu = 255$ for the United States digital carrier system and $0 \le y \le 1$. This formula is shown in Fig. 4-2 for the case where x is an 8-bit signed byte and y is interpreted as an 8-bit unsigned byte.

After the μ-law transformation, the result is interpreted in 2's complement. If y is as above and is input to the 2's complement transform, then the output z is given by:

$$z = \begin{cases} y & \text{for } 0 \leq y \leq 127 \\ y - 256 & \text{for } 128 \leq y \leq 255 \end{cases}, \tag{4.7}$$

(see Fig. 4-3). The composite of these two functions is shown in Fig. 4-4 as a function mapping x to z. The value z will be referred to as the *Magnitude Factor* (MF).

An assumption that can be seen quite clearly from empirical data is that voice has no DC component, i.e. it is a zero-mean signal. To demonstrate the use of the MF, a simple example is needed. Referring to Fig. 4-4, if a small magnitude, zero mean signal with frequency π is the input to the MF function, the positive peak of the voice signal will create a small negative MF, while the negative peak of the signal will create a large positive MF. If the output of the MF function is then averaged over time, or low-pass filtered, the result will be positive. The low-pass filtered MF signal will be called the *Average Magnitude Factor* (AMF).

The same analysis can be performed on a large signal by again referring to Fig. 4-4. The positive peak of the voice signal will generate a large negative MF while the negative peak will create a small positive MF. The AMF will therefore be negative. An example of the AMF is shown in Fig. 4-5, where the input is a segment which contains voice and silence. As can be seen the AMF is much greater than zero only between 2500 and 3500 samples. This is exactly the segment corresponding to silence. Superimposed on this signal is a square wave demonstrating where the algorithm actually detected the silence based on the AMF.

The behavior of this non-linear transformation for sine waves of various frequen-

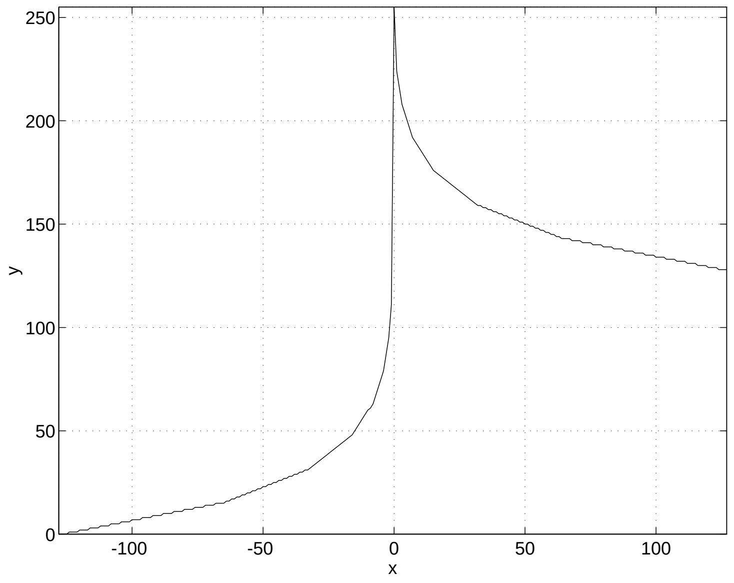


Figure 4-2: μ-law function.

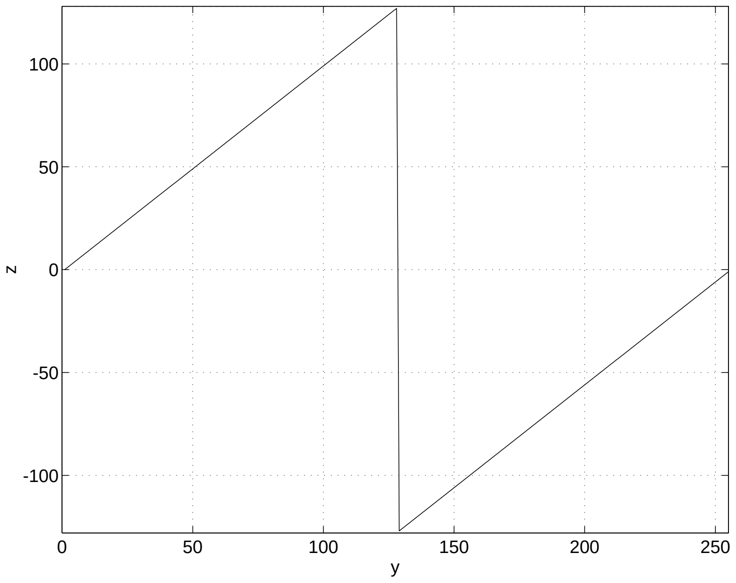


Figure 4-3: 2's complement function.

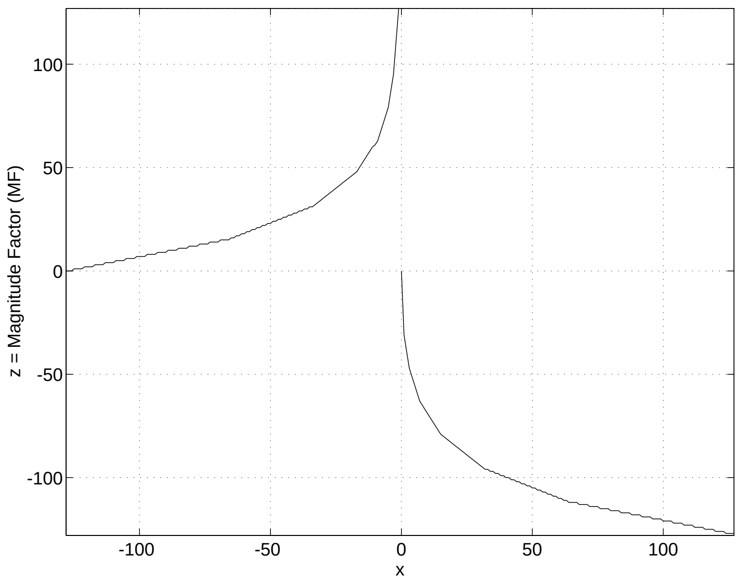


Figure 4-4: Composite function.

cies and magnitudes is depicted in Fig. 4-6. We see that at small magnitudes we obtain a larger portion of positive samples than negative ones. At larger magnitudes, the transformation yields primarily negative samples. This corresponds to the low valley in the graph. The small and large signal analysis in the previous paragraphs assumed signals of frequency π, which in this case, corresponds to 4000 Hz. This assumption was made for ease of analysis. A cosine wave at this frequency would have values of ± 1 which would correspond to the positive and negative peaks referred to above. These peaks pick off the various MF's, which are then averaged to obtain the AMF. A decision is then made based on the percentage of AMF samples that are positive.

Since the transformation is non-linear, superposition does not hold. For this reason, we cannot generalize this discussion to that of arbitrary signals. We present this line of reasoning here merely to foster the intuitive notion of why it works, and defer actual experimental proof to the results presented in Sect. 5.2.4.

So far the averaging required to create the AMF has been referred to as a filter.

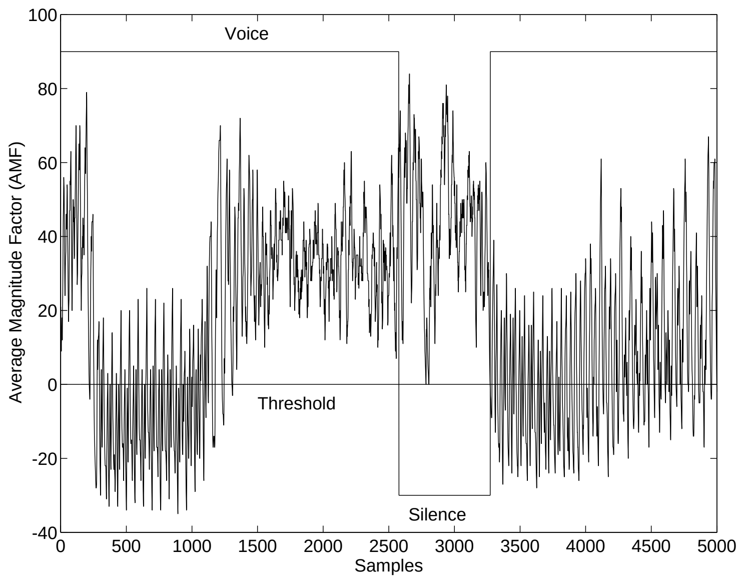


Figure 4-5: AMF for a voice/silence/voice segment.

Very simple filters do not provide the smoothing necessary to provide an AMF that is stable over small blocks of time. After extensive experimentation, we found that a Chebyshev first order filter with cutoff frequency at 50 Hz yielded very good results [72]. The Chebyshev filter attenuates the higher frequencies enough to obtain a more slowly varying AMF so that the appropriate decision can later be made.

A block diagram of the algorithm is shown in Fig. 4-7. In our experiments, audio is acquired at 64 kbps, using 8-bit μ-law companded samples (8 KHz sampling frequency). The A/D converter provides a 1024 sample frame through the operating system's device driver. Although, smaller frames would yield a lower end-to-end delay, the particular audio device that was being used had a lower limit of 1024 bytes. The frame is then examined by breaking it into smaller blocks of 256 bytes. Each block is determined to be silence if fewer than 25% of the AMF samples in the block are negative; otherwise the block is deemed speech.

A commonly employed technique for silence detection and removal algorithms is to use a starting and ending hang time. The starting hang time is the time between

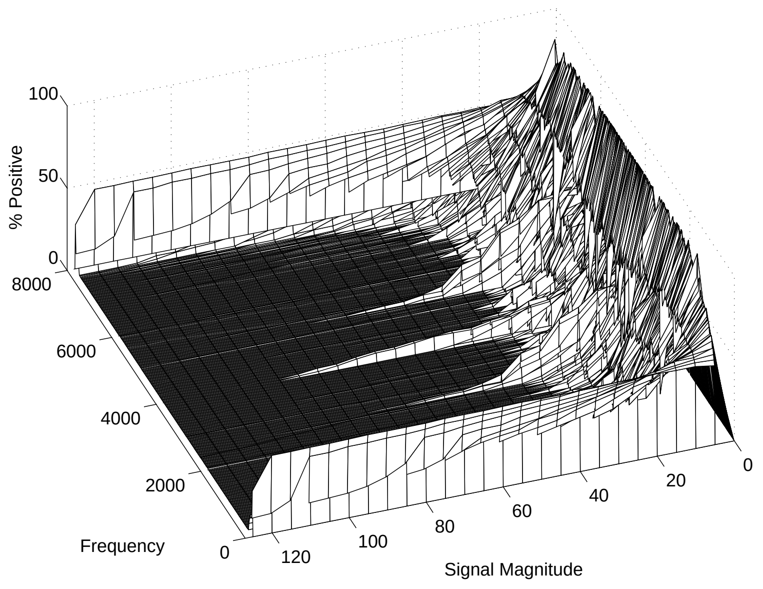


Figure 4-6: Percentages of positive AMF samples for varying frequency and magnitude.

when silence is detected and when data is actually removed. Figure 4-8 shows where silence and speech were detected using a solid line. The dashed line represents where silence was actually removed based on the hang times. The purpose of the starting hang time is to make sure that the removal of silence doesn't inadvertently remove the end of a talk spurt.

The ending hang time is the time between when the algorithm stops removing data and the beginning of a talk spurt. Its purpose is to ensure that the beginning of a talk spurt is not removed. It is important to note that the hang times do not contribute in any way to the end-to-end delay.

The starting hang time was set to seven consecutive blocks or 224 msec. This was determined experimentally because shorter hang times resulted in truncation of the end of talk spurts. Once the starting hang time is reached, all subsequent and consecutive silent blocks are removed.

If a non-silent block is detected within a frame, no data will be removed from

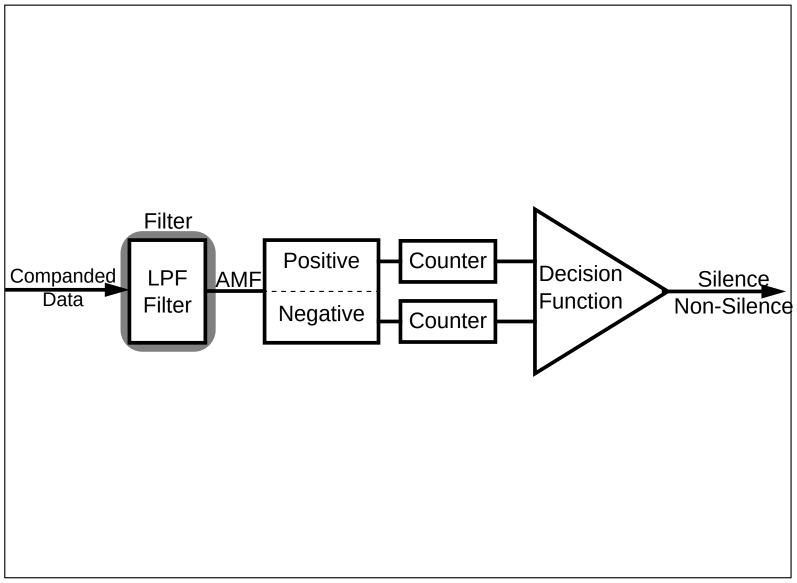


Figure 4-7: Block diagram of proposed algorithm.

that frame. This corresponds to a variable length ending hang time of between 96 msec and 0 msec. For example, if silence has been detected and removed in previous frames, but the last block of the current frame is non-silent, no silence will be removed from the current frame. This corresponds to an ending hang time of 96 msec because the algorithm has detected the other three blocks of silence and is not removing them.

Since the ending hang time is variable, it can also be 0 msec. If silence has been detected and removed in previous frames and the current frame has all silent blocks, then this entire frame will be removed. If the first block of the subsequent frame is non-silent, then none of that frame will be removed and the ending hang time will be 0 msec. In theory, this large range is not ideal but is the best that can be done in a system where frames cannot be delayed by buffering. In practice, it only slightly degrades removal performance while protecting, above all else, quality of speech.

The determination to remove silence happens only at the end of a frame. This means that the removal is accomplished by sending only the part of the frame that

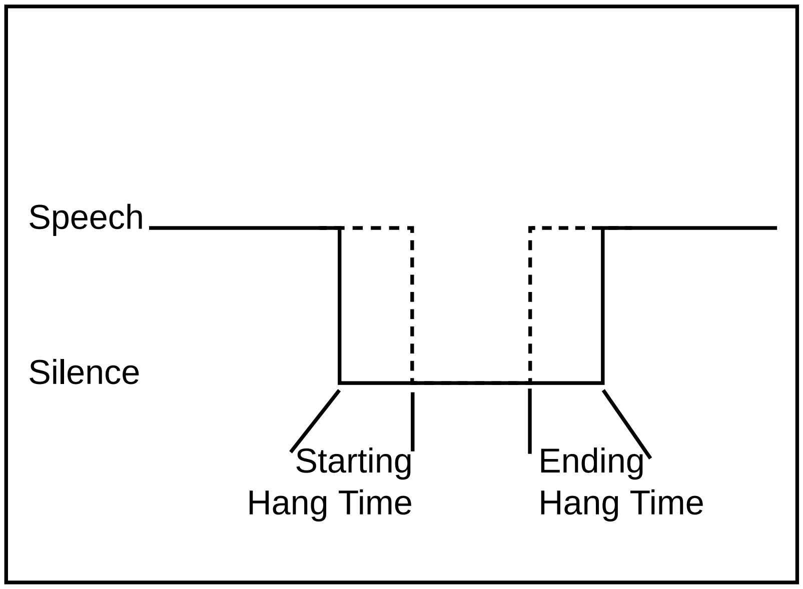


Figure 4-8: Definition of starting and ending hang time.

corresponds to voice. Since the silence will always be at the end of the frame, data is never moved; the frame is merely truncated. This significantly decreases the complexity of the algorithm in comparison to others. It is also less complex because the conversion from μ-law to linear is eliminated and only one filter is used.

The algorithm meets the other requirements, too. Only one previous sample is needed across frames, hence satisfying the low buffer size specification. Since the silence/speech decision is made at the end of each frame, previous frames are not stored. Finally, the average period of silence detected is quite large and the quality of speech is completely unaffected. In short, all of the requirements are met and the last analysis to be made is how much silence is actually removed.

4.3.4 Results

In analyzing the performance of the proposed algorithm, we compared it to the technique demonstrated by Savoji [86] since it satisfies several of our requirements for a scheme appropriate for multimedia systems. In both techniques, there was no

Table 4.1: Comparison of Savoji and proposed algorithms in kilobytes.

Type of silence	Percent removed proposed	Percent removed Savoji	Percent more removed by proposed	Average length proposed	Average length Savoji
Almost no noise	91	73	18	15.7	1.6
Low noise	85	64	21	14.8	1.5
Moderate noise	69	0	69	6.5	0.0
Loud noise	0	0	0	0.0	0.0
Inter-sentence silence	46	0	46	3.5	0.0

degradation in the quality of speech. The Savoji technique initially removed much more silence without cutting off words. However, the silence it was removing was inter-word and inter-syllabic silence. To alleviate this, the constraint was imposed that only periods of silence longer than 125 ms were to be removed. The result of modifying the algorithm to fit this situation was that the amount of silence removed was dramatically decreased, and in general was smaller than the silence removed by the proposed algorithm. Experimental results are shown in Table 5.2.

The 64 kbps, μ-law companded audio signals for these experiments were obtained under the typical conditions for a video conferencing session. They were recorded at a workstation in a laboratory with a variety of different background noise sources such as computer fans and cooling systems. Silence was recorded with a large range of noise intensities, as indicated by the first column in Table 5.2.

The most important test cases are those that have extended periods of silence with and without background noise, i.e. the first four in the table. It is evident that at this level of complexity, very loud background noise could not be detected and removed. However, the results for moderate background noise, corresponding to nearby air conditioning system and computer fans, proved favorable. They showed that the proposed algorithm removed over 2/3 of the signal while the modified Savoji technique could not detect silent periods that were large enough to merit any removal at all. Low background noise had even better removal percentages.

Also demonstrated in Table 5.2 is the fact that, in general, the average length of silence removed is much higher for the proposed algorithm. Even though the proposed algorithm has no constraint for removing a minimum of 125 ms or 1000 bytes, the average length removed was much higher in most cases. It was highest in the case of almost no noise. For this signal a continuous segment of 91% of the signal was found to be silence. The average length is an important parameter because it demonstrates what type of silence is being removed. Savoji's algorithm performs best when used to detect small periods of silence for the purpose of isolating individual words. The proposed algorithm finds the extended consecutive silences that occur while one subscriber is idle during a conversation. This is a critical difference in the context of a video conferencing system.

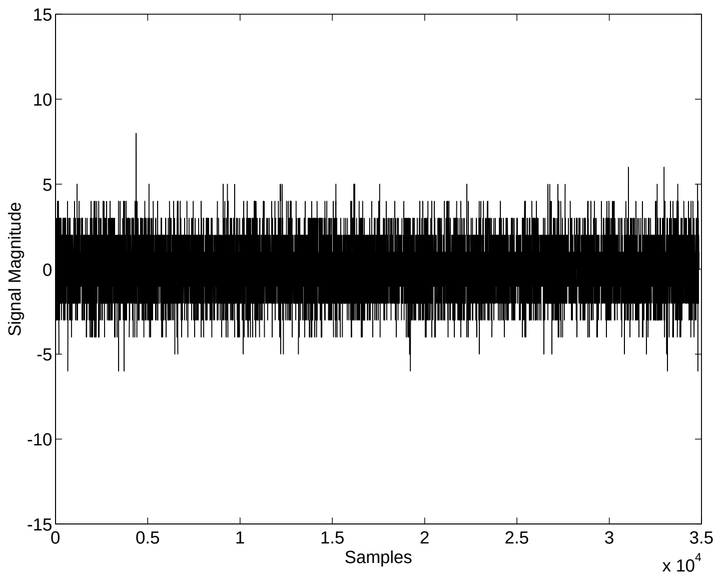


Figure 4-9: Silence with low background noise.

Figure 4-9 shows silence with low background noise. Figure 4-10 shows where Savoji's modified algorithm could not detect and remove silence. The peaks are where speech was detected, while the troughs are where silence was detected. Figure 4-11 shows the more favorable results for the proposed algorithm. As can be seen, the proposed algorithm is not confused by the difference between background noise and speech. As desired, it finds most of the signal to be silence.

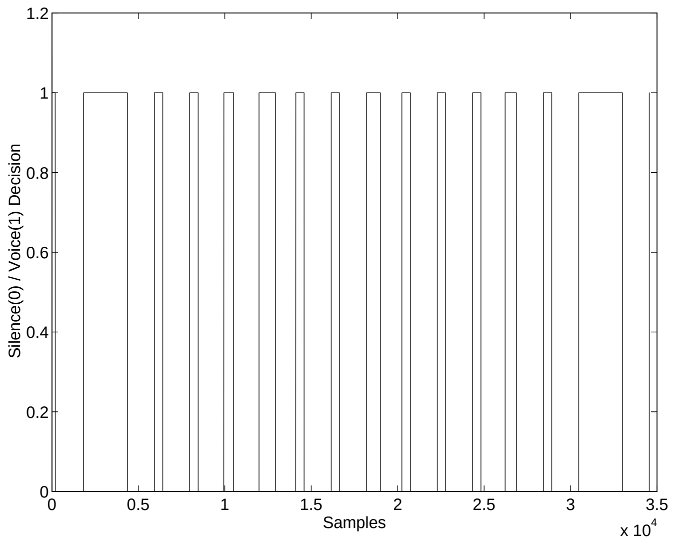


Figure 4-10: Savoji algorithm operating on low noise signal.

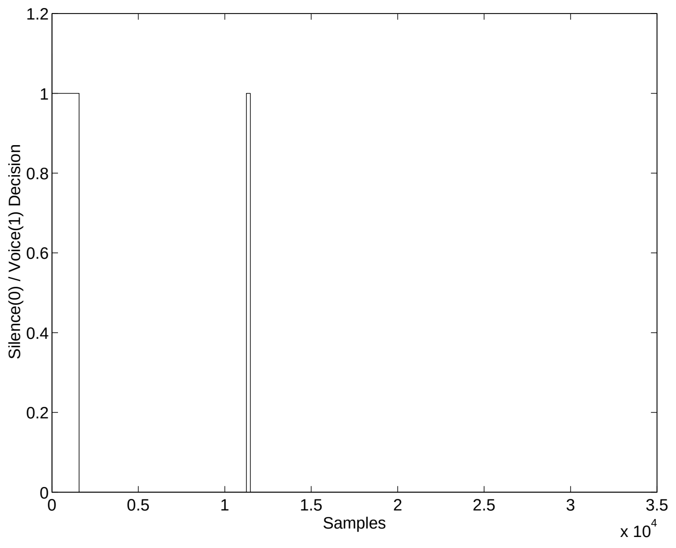


Figure 4-11: Proposed algorithm operating on low noise signal.

Experiments were also conducted using an actual video conferencing system, Xphone [35], based on workstations interconnected over an Ethernet network. The simple difference-based algorithm used in Xphone was substituted with the proposed one, with very good results. Most noticeable was the elimination of the frequent false positives in which the algorithm removed segments of speech, mistaking them for silence.

We have posed the silence detection problem in a multimedia communications environment. These environments are characterized by their packet-based data handling and communication facilities, as well as a varying degree of QoS support (i.e. bandwidth and delay). We have identified several requirements for efficient use of silence detection, namely elimination of only inter-sentence (or longer) silence, low complexity and low decision delay. Due to the particular structure of multimedia communication systems, existing algorithms cannot be easily accommodated.

We have presented a novel algorithm for silence detection, based on the small and large signal behavior of the speech waveform in the μ-law domain. The algorithm exploits the μ-law non-linearity instead of trying to sidestep it, hence allowing a particularly fast and robust design. It outperforms others in its class of complexity. Aside from the simple magnitude detector, it is the simplest in its class: it has only one filter, performs no μ-law transformations, requires very little memory, and only removes silence through frame truncation. Implementation and experimentation in an actual desktop video conferencing system proved to have diminishing effect on the degradation of the quality of speech, while facilitating a low end-to-end delay.

4.4 Image

4.4.1 JPEG

As mentioned in Section 4.2, DRS can be used to alter the bandwidth requirements of any block-based transform coding scheme. JPEG is such a coding scheme. A possible application would be for Web browsing where the user wants an entire page to be downloaded within a certain maximum period of time. In this case, the server could then use a technique such as DRS to reduce the bandwidth requirements of the image, as well as the quality, in order to meet the timing requirements of the user.

4.4.2 Flashpix

Just as the rise of the Internet precipitated the need for scalable video codecs, so has it driven the need for scalable image codecs. Flashpix is an image codec built by Kodak, Hewlett-Packard, Live Picture and Microsoft. One of its goals is to be a scalable codec. Within a single stream, multiple spatial resolutions are stored. Flashpix, coupled with the Internet Imaging Protocol, allows the user to control the view of the image by specifying a particular area on which to zoom, without having to wait for the arrival of the entire high resolution image.

4.5 Text

The way we use text makes it a naturally scalable media. Students are often taught to arrange their papers hierarchically. The introduction and conclusion is usually a summary of the entire paper. Each paragraph begins with a topic sentence which highlights the main point of the paragraph and the rest of the paragraph proves that point. A well-written paper is easily summarized by exploiting these stylistic rules. There are even software packages which condense entire documents into a user defined number of sentences. Although this technology is far from perfect because it must deal with such a wide spectrum of writing styles, it does demonstrate the feasibility of such a system

4.6 Concluding Remarks

In this chapter we have shown that the major media types (video, audio, images and even text) all have means for scalability. Although none started out inherently scalable, the rise of networked multimedia and of a non-QoS network has driven the demand for making non-scalable codecs scalable as well as reinventing the codec so that it is inherently scalable.

The existence of these scalable codecs is imperative to the next chapter, where we attack the problem of adaptive applications from the perspective of the network. We identify ways of gathering implicit information about the state of the network. Then we present a method for translating this information from network parameters into codec parameters. We also extend this work to the case of a wireless LAN.

Chapter 5

Protocol for Non-QoS Streaming

5.1 Introduction

We present a new semi-reliable client/server protocol for streaming real time media on today's Internet. The protocol uses the TCP congestion window to pace the delivery of data into the network, without using other TCP algorithms that are poorly suited to real time protocols. For this reason, our protocol competes fairly with all other TCP data, by optimally sharing the available bandwidth. We also describe a real application that uses this protocol to stream MPEG video on the Internet.

We then present a battery of experiments, performed in a controlled environment and in the wide area Internet, that were used to evaluate the effectiveness and fairness of the protocol. The results show that the protocol achieves fairness by sharing bandwidth equally with other non-real-time connections, while avoiding the latency problems commonly associated with TCP. The results also show that when coupled with a rate controllable real time data source, the protocol can significantly improve the user's experience by reducing the likelihood of buffer starvation at the client.

Next, we extend this work to the case of a wireless Internet, where at least one of the hops in the connection is wireless. Since the streaming protocol presented is very similar to TCP, it is possible to leverage research on the utilization of TCP over wireless to obtain results for the proposed protocol.

TCP performance over wireless networks can be poor because TCP assumes losses occur in the network only when there is congestion. However, wireless networks often drop packets due to random errors on the wireless link. TCP performance can be severely degraded when packets with errors are mistaken for congestion. This is because TCP uses packet loss as an implicit indication that the network is congested. This assumption does not necessarily hold for wireless networks.

In Section 5.3, we present a novel, tractable, and closed-form analytical model for determining the performance of TCP over hybrid wired/wireless networks. We compare our model with previous work in this area from an analytical perspective. We also compare our model with several simulations. The results show that our model matches both the previous work and the simulations very well. We also provide a simple heuristic for determining when wireless errors will severely degrade TCP performance.

This analysis can then be readily used to evaluate not only the performance of TCP, but also the performance of the proposed streaming protocol in the presence of wireless errors. This analytical model is then married to a simplified model of the IEEE 802.11 wireless LAN standard. In Section 5.4, we show that a fixed number of link level retransmissions can improve TCP performance substantially, while still maintaining a bound on delay for real time applications.

Although it is beyond the scope of this work, it is important to recognize some of the problems of TCP (and also the proposed protocol) over satellite [1, 2, 53, 74, 75, 76, 85]. One factor which affects TCP performance is the round trip time. Satellite links often have very high round trip times because of the large distance the signal must travel to the satellite and back for the packet and for the acknowledgement of the packet.

Another issue for satellite communications is error rate. Many external factors such as meteorological activity can affect satellite error rates. Newer satellite systems use substantial error corrections schemes in order to provide a network which is in one of two states. Either it has a low error rate due to error correction or the network is out of service if the error rate is too high to be corrected.

In the next section, we present the proposed protocol and provide a brief overview of previous works. We then demonstrate a real application that uses this protocol to stream MPEG video, while sharing available bandwidth fairly. Next, we present experimental results from a controlled environment and also results from testing in the wide area Internet, across more than 30 hops.

Section 5.3 develops the analytical model for TCP utilization over a hybrid wired/wireless Internet. After building the model, it is compared with previous models as well as simulations. Finally, this model is applied to the IEEE 802.11 wireless LAN standard.

5.2 Wired Internet

5.2.1 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) and the Real-time Transport Control Protocol (RTCP) are used on top of UDP [87]. RTP allows for the packetization of a variety of different media. The packet header identifies the media and also allows for sequence numbering and time stamps. RTCP works with RTP and provides for the communication of information about the delivery of data between the senders and receivers. For instance, a receiver can use RTCP to provide information to a sender about the loss rate being experienced. The sender can then choose to reduce its rate. In this way RTP is an excellent protocol for adaptive applications and is especially well suited for multicast.

5.2.2 The Proposed Protocol

The main goal of our protocol is to facilitate unicast real time services that do not degrade the performance of other data transfers on the Internet, such as FTP or HTTP. Since the Internet is a shared medium, there is always competition among users for its resources. The most important data transfer protocol is TCP because it promotes fairness between different data transfers by sharing the available bandwidth evenly among users. Many real-time client/server software packages that exist today make little or no effort at maintaining this fairness. We will show that our system preserves the fairness that the Internet and TCP originally intended.

Although our protocol was not intended to scale to multicast, we believe that unicast is a very important class of applications. There is a large body of work in the area of Internet streaming with rate controllable sources, though much of it has focused on multicast [9, 11, 12, 16, 23, 58, 70, 80].

Since unicast can be viewed as a special case of multicast, these protocols are often used for unicast. However, this is not the most efficient solution. Multicast protocols attempt to solve a superset of the problems posed by unicast. In this way, they contain tradeoffs which may solve multicast problems, but at the expense of unicast performance.

For example, although multicast protocols are often congestion aware, they are not necessarily fair, in terms of sharing available bandwidth equally. But unicast applications should not be denied this feature solely because it has not been possible to obtain this fairness with multicast protocols. We will demonstrate that our unicast protocol does something quite novel - it competes fairly with data traffic

(TCP) and still remains a real time protocol, by maintaining relatively low delay and minimizing the likelihood of interruptions in playback.

The technique for developing this type of non-QoS protocol involves an explicit attempt at avoiding network congestion. Clearly, network congestion hurts the performance of all data transfer on the network. The goal is to send only the data that can fit into the network at a particular time. This requires both a networking and an image processing solution. From the networking perspective, an estimate of the available bandwidth in the network must be found. Then, from the image processing perspective, it is necessary to change the bandwidth requirements of the compressed real time media into the available bandwidth through the use of a scalable codec. This adaptation to the available bandwidth in the network continues over the life of the connection.

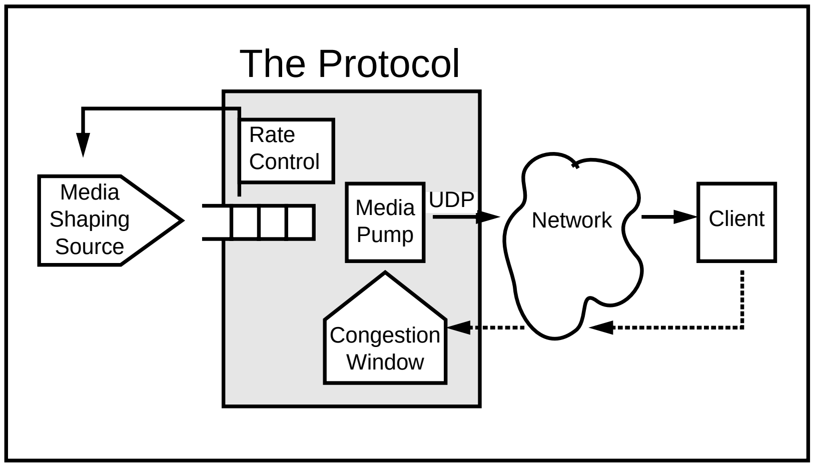


Figure 5-1: The Internet-friendly protocol framework.

The protocol as shown in Figure 5-1 receives data from a media shaping source. This could be a precompressed media stream or a live source. This source is responsible for matching the bit rate of the stream to the estimated available bandwidth in the network; the protocol provides this estimate. The source is continually filling the buffer with the media. Meanwhile, the media pump reads the data from the buffer and sends it into the network using UDP/IP. The congestion window is the third part of the server. The media pump only sends out data when the congestion window indicates that more data can enter the network. The congestion window uses feedback from the client, such as acknowledgements of packets received and explicit retransmission requests for packets assumed lost.

If the media shaping source is filling the buffer at a rate, R, and the congestion window over time is allowing the media pump to send data out only at a rate $S < R$, then the buffer will begin to fill. The media shaping source should then decrease the rate entering the buffer. A rate control algorithm is responsible for translating the buffer occupancy dynamics into an adjustment to the media shaping source rate to ensure that the buffer does not underflow or overflow.

5.2.2.1 Internet Friendly Bandwidth Estimation

Network bandwidth estimation has been explored extensively for the Internet. Many algorithms provide a somewhat different view of the state of the network. In [16], the authors use packet loss as an indication of congestion. Other metrics such as delay or delay jitter are also used as in [14, 83, 84]. In [58], the authors use explicit feedback from the routers to inform senders of congestion.

Despite the different techniques for finding the available bandwidth, probably the most important factor in designing one is fairness. When a single network maintains multiple transport protocols, each using a different congestion control algorithm (or none at all), it tends to lead to unfairness [68, 99]. This is because different transport protocols use different definitions of congestion and react differently on detection of congestion. For this reason, it is important that the bandwidth estimation technique does not result in a degradation in the quality of other data transfers on the network.

As real-time services have become more prevalent over the last few years, there has been a growing concern that the current Internet infrastructure may not be able to support them, leading perhaps, to a "congestion collapse." This is a valid concern in general since many real-time services send their bits through the network without concern for congestion control or avoidance.

For this reason, we have designed a technique which is optimally Internet-friendly, in the sense that it shares the available end-to-end bandwidth equally with any other data transfer using TCP [46, 47]. For example, if between any two network endpoints there is a TCP based connection and another connection using our protocol, the average bandwidth that both connections will use will be equal. This eliminates concerns about "congestion collapse" since the protocol detects and reacts to congestion in exactly the same way as all other TCP-based non-real-time Internet traffic.

To determine the available bandwidth in the network at a given time we use the TCP congestion window [52, 77] coupled with a rate control algorithm as shown in Figure 5-1. The congestion window indicates the number of allowable outstanding unacknowledged packets. This is an indirect indication of the available bandwidth in the network. If the number of outstanding packets is greater than or equal to the congestion window size, then the media pump can send no further data into the network until the situation changes. The congestion window is what paces the delivery of data from the media pump into the network [48].

Pacing according to a TCP congestion window is exactly what makes the system Internet-friendly. TCP streams work well together and today's Internet is proof of that. They operate according to a greedy but "socially-minded" and cooperative algorithm which attempts to get as much bandwidth as possible, but backs off substantially during congestion. Forcing all data to go through the TCP congestion window can make real-time traffic look as harmless as a file transfer to the network and still maintain relatively low delay [43].

The TCP congestion window is continually changing. It increases quickly at first during the poorly named Slow Start Phase and then continues to increase more slowly during the Congestion Avoidance Phase. When a packet is lost in the network, TCP assumes this is due to congestion. A packet is lost when the server does not receive an acknowledgement from the client within a specified timeout period. The timeout period is related to the round trip time, the time it takes to send a packet and receive an acknowledgement. The reader is referred to the RFC 793 for more details on the TCP algorithms [77].

The maintenance of the congestion window and the associated variables are usually kept within the operating system. To implement our protocol we have rebuilt the congestion window and its necessary components in user space. It is important to note that we have not moved all of TCP into user space, but only the algorithms needed to determine the congestion window. For example, we have not included the mandatory retransmissions that TCP normally implements, since we do not want a completely reliable protocol.

Recently, work has been done indicating that Internet-friendly bandwidth estimation can be achieved without using congestion windows, but by using a rate-based algorithm based on packet loss rates [67]. However, simplifying assumptions about the dynamics of TCP are made and the algorithm functions well only during low loss rates. At high loss rates, it overestimates the available bandwidth, defeating the goal of being Internet-friendly.

Another problem with rate-based pacing is the different time scales that rate-based and window-based pacing operate on. A rate-based scheme must calculate the average loss rate over time. Thus, the actual output rate may be adjusted much more slowly than a window-based scheme, which adjusts its rate at least every round trip time. If the rate-based scheme reacts too slowly, the window-based scheme could react to congestion and reduce its transmission rate more quickly. By the time the rate-based scheme is ready to adjust its output rate, it would be seeing a lower average loss rate since the competing window-based scheme had already reduced. The lower loss rate will make the rate-based scheme use more bandwidth than the window-based scheme, leading to unfairness.

Our protocol is shown in Section 5.2.4 to share available bandwidth equally with other TCP connections, which corresponds with our stated goal of being Internet-friendly.

5.2.2.2 Rate Control

As mentioned before, the rate control algorithm is an essential part of bandwidth estimation. During periods of congestion, the media pump cannot send much data into the network due to the TCP congestion window. When this happens, the buffer in Figure 5-1 will start to fill. In this case, a rate control algorithm should force the media shaping source to decrease the rate entering the buffer. If the buffer subsequently begins to empty, the rate control should request an increase in the rate entering the buffer from the source. Clearly, the dynamics of the buffer modulate the rate. Therefore, the estimate of the available bandwidth comes from the rate control algorithm, but is an indirect result of the dynamics of the TCP congestion window.

Other work has been done in the design of rate control algorithms. It has been studied extensively in the context of entire networks, where the buffer occupancy from a bottleneck router is used to adjust the rate at which data is injected into the network [3, 6, 10]. Here, the occupancy information traverses the network and is delayed, adding complexity to the algorithms. In our case, the buffer occupancy information is available immediately and simpler models can therefore be used.

Simpler controller models are presented in [6, 58]. In [58], the authors propose the use of a slightly modified proportional control system, where the rate change is based on the distance the current buffer occupancy is away from a desired occupancy. It is called proportional because the change in the rate is proportional to this distance. In [6], the authors show that the pure proportional controller is less than ideal, yielding nondecaying oscillations in the buffer occupancy and rate over time.

A more complex proportional plus derivative (PD) controller can be used which will not exhibit such oscillations [98]. Although these controllers are more effective in time invariant systems, we have found them to function quite well in the time varying environment based on the selection of key tuning parameters. Performance of this algorithm is detailed in Section 5.2.4.

The buffer occupancy sampling period for the rate control algorithm depends on the medium. Since MPEG video consists of several different frame types whose sizes vary greatly, estimates of the buffer occupancy must be taken as averages over no less than a one second interval to avoid momentary fluctuations in the buffer occupancy. A small sampling period means that the rate control algorithm can adapt more quickly to sudden changes in the network. However there is a tradeoff since a rapid change in quality is subjectively unpleasant to the user. Larger sampling periods require larger buffers to absorb changes, since the algorithm would respond more slowly.

The buffer in Figure 5-1 is being emptied based on the TCP congestion control window. During interval i, the output rate of the buffer, μ_i, is unknown and the input rate which is controlled entirely by the rate control algorithm is λ_i. At the end of interval i, the buffer occupancy, b_i, can be described by the following:

$$b_i - b_{i-1} = \lambda_i - \mu_i. \tag{5.1}$$

A PD control law would yield a change in the input rate as given by

$$\lambda_{i+1} - \lambda_i = -\alpha_0(b_i - b_d) - \alpha_1(b_i - b_{i-1}), \tag{5.2}$$

where b_d is the desired buffer occupancy. The weighting factors α_0 and α_1 indicate the relative importance of the proportional and derivative factors, respectively. Quick convergence is attained by making α_0 large, while minimizing overshoot can be accomplished by making α_1 large.

The goals for a rate control algorithm in this environment, in order of importance, are to:

1. prevent buffer overflow, which will cause delays,
2. converge quickly to a new output rate,
3. minimize the size of the oscillations around the new output rate.
4. prevent buffer underflow, unless the adaptable media source is at the maximum rate, and
5. keep the buffer at a desired occupancy, b_d.

In order to minimize the likelihood of buffer overflow, we chose $b_d = b_{max}/4$, where b_{max} is the total buffer size. This encourages the occupancy to stay at about a quarter full. The choice of b_{max} is based on the original rate of the compressed media, R bps. To further minimize the likelihood of buffer overflow, we chose a

b_{max} in bits corresponding to 8 seconds ($b_{max} = 8R$), though smaller values can be chosen, with a correspondingly higher probability of buffer overflow.

The parameter α_0 was chosen to be 0.005. The reason has both analytical and empirical roots. If the buffer is at the maximum and we ignore the derivative term ($b_i - b_{i-1} = 0$), then equation 5.2 becomes

$$\lambda_{i+1} - \lambda_i = -0.005(b_{max} - b_{max}/4). \tag{5.3}$$

Substituting for b_{max}, we find that the change in the rate is $-3R/100$ bps. For example, if $R = 100$ kbps, then the change in the rate would be -3 kbps. This may not seem like much, but if α_0 is too large, there will be too much overshoot.

For α_1 we use 0.1. If in one interval, the buffer increases by 10% of b_{max} and we ignore the proportional term ($b_i - b_d = 0$), then equation 5.2 becomes

$$\lambda_{i+1} - \lambda_i = -0.1(8R/10). \tag{5.4}$$

This corresponds to a decrease in the rate by $8R/100$. In other words, an increase in the occupancy of 10% yield a decrease in the rate of 8%. For the same 100 kbps example, the decrease in the rate would by 8 kbps. As shown by our choice of parameters, we have placed more emphasis on the derivative term since our main goal is to converge quickly and without much overshoot.

Although using the congestion window with the rate control algorithm is an indirect way of calculating the available bandwidth, the only drawback is the need for buffering. But even this is not a major deficit of the protocol, as most systems operating in non-QoS environments employ some buffering to absorb fluctuations in the available bandwidth in the network anyway. In the case of MPEG, the total buffering must be at least a few seconds to absorb the variation in frame sizes as mentioned earlier. However, for other codecs that do not use inter-frame prediction, much smaller buffering can be used. Applications such as video conferencing are then feasible, since three seconds of latency would be highly unacceptable for conferencing.

5.2.2.3 Retransmissions

Mandatory retransmissions increase delay, as the client must wait for the retransmitted packet. In general, this is unacceptable for real-time applications. It is certainly the case for video, where we would like to trade quality by losing some data and possibly frames for a lower probability of buffer overflow, which would force the video to stop and then restart playback.

We have used UDP coupled with the TCP congestion window in order to build an Internet-friendly, semi-reliable protocol; TCP is not used for transport. It is semi-reliable because we have added selective retransmissions. The server in TCP maintains an estimate of the round trip time, the time it takes to send a packet and receive an acknowledgement for that packet. In our system, the server sends this information to the client in every packet so that the client has a (somewhat delayed) estimate of the round trip time.

The client then knows best whether or not to request a retransmission. If the client detects that a packet has been lost in the network, it can then determine if there is enough time to request a retransmission and receive the packet before it is needed. For example, if the client has 100 msec of data buffered and knows that the round trip time is 50 msec, then it is likely that the retransmitted packet will arrive on time. If there is not enough buffering to cover the round trip time, then the client will accept the loss and not request a retransmission.

Requesting a retransmission in this case would waste precious bandwidth and would likely not improve the user's experience since the packet would likely not arrive in time. Even if the client requests a retransmission, it is not forced to wait for it to arrive. Error concealment techniques for packet loss are beyond the scope of this work, but an example for video when part of a frame is lost would be to drop the entire frame.

A retransmission request affects the congestion window similarly to the Fast Recovery Algorithm [90]. In TCP, when a server realizes that it needs to retransmit a packet but that other packets are still arriving at the receiver, the server implements a less severe congestion avoidance algorithm. The idea is that since packets are still arriving at the receiver the network is only moderately loaded and the congestion back-off should reflect this. For this reason, TCP enters the Congestion Avoidance Phase rather than the Slow Start Phase.

Our protocol does the same thing. When a retransmission request is received and the server is still receiving acknowledgements, the protocol enters the Congestion Avoidance Phase since this indicates that the network is moderately loaded.

The protocol packet header consists of a 2 byte sequence number, a 4 byte presentation timestamp, and a 4 byte round trip time, measured in milliseconds. Since the underlying protocol is UDP, there is no need for a size field. The sequence number is for packet reordering at the receiver. The timestamp is media dependent, but is an indication of the presentation time of that packet. The actual size of the payload is also media dependent.

For example, in MPEG video the payload is a frame of MPEG data. If a frame is too large to fit in the payload, it will be fragmented over more than one packet. This creates a stream of packets that have variable lengths. Since TCP normally operates on fixed length packets, alterations were needed. In the case of variable length packets, the congestion window is the number of allowable outstanding unacknowledged bytes. The initial size of the congestion window is defined as 512 bytes, the typical size of a TCP segment. The maximum payload size is also 512 bytes to maintain fairness with TCP.

Figure 5-2: Screen dump from MPEG video streaming application.

5.2.3 The Application

We have built an application on top of this protocol for streaming MPEG-1 or MPEG-2 video across the Internet. This application is quite similar to Figure 5-1 except that the media shaping source is now a video rate shaping source. The server runs on a UNIX workstation, while the client runs on a PC running Windows 95/NT. The client uses Microsoft's ActiveMovie in order to take advantage of the high performance software MPEG decoder. Figure 5-2 is a screen shot of the client window, receiving a streamed MPEG video.

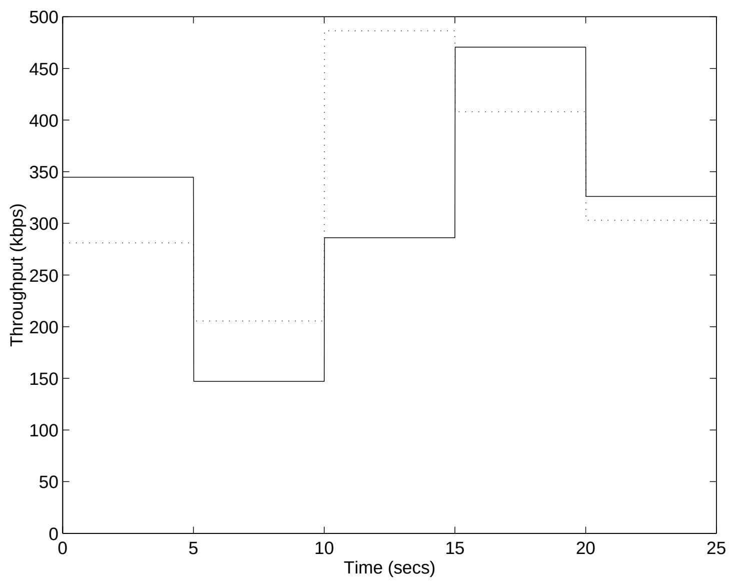


Figure 5-3: Two TCP streams sent from the same server to the same client across 19 hops in the Internet, showing that they do in fact share the available bandwidth.

5.2.4 Internet Results

To evaluate our system, we performed three sets of experiments using the application described in Section 5.2.3. The first set was to determine if our system was fair in its use of bandwidth, which is the main stated contribution of this chapter. The next set of experiments consisted of a client and server with a controllable bottleneck in between, simulating a bottleneck router. The goal here was to determine how quickly the rate control converged to a desired rate and how well it maintained that rate. Finally, since we have a real system, we examined the behavior in the wide area Internet.

5.2.4.1 Bandwidth Fairness

The definition of fairness in this environment is that our protocol can share equally the available bandwidth end-to-end with a TCP connection. We used the actual throughput rather than goodput to compare the protocols, since this is where fairness matters most. Also, since TCP requires mandatory retransmissions and our protocol retransmits selectively, our protocol would have a higher goodput. This is because our protocol uses less throughput on retransmissions than TCP.

We used `tcpdump` to capture how and when the data was sent out. We then averaged the throughput over an interval to calculate the average throughput for each interval. Each test was performed several times, of which a representative graph is presented here to point out common trends. The loss rates were in the range of 0.8% to 2.3%.

We performed four tests here. First, using the same client and server, we sent two TCP streams into the Internet, a total of 19 hops. The goal was to verify experimentally how well the TCP streams shared the available bandwidth, as reference. Figure 5-3 shows the bandwidth used by the two streams, averaged over 5 second intervals. The average over the entire connection was 342 kbps and 292 kbps. As expected, the graph shows that the two streams share the available bandwidth, except for the perturbation between 10 and 15 seconds.

Next we wanted to examine the effect that a congestion unaware transport protocol, such as pure unregulated UDP, would have on a TCP stream. This is shown in Figure 5-4. Clearly the UDP stream uses all of the available bandwidth. The TCP stream is then continually reacting to the congestion caused by the UDP stream. The TCP stream receives the bandwidth not used by UDP. The average over the entire connection was 205 kbps for TCP and 810 kbps for the pure UDP. The rate for UDP represents the rate at which the computer could send data into the network, regardless of the state of the network.

Now we verify that our protocol acts like TCP in sharing the bandwidth. Again using the same client and server, we set up a TCP connection and a connection using our protocol. The results are shown in Figure 5-5, where the throughput

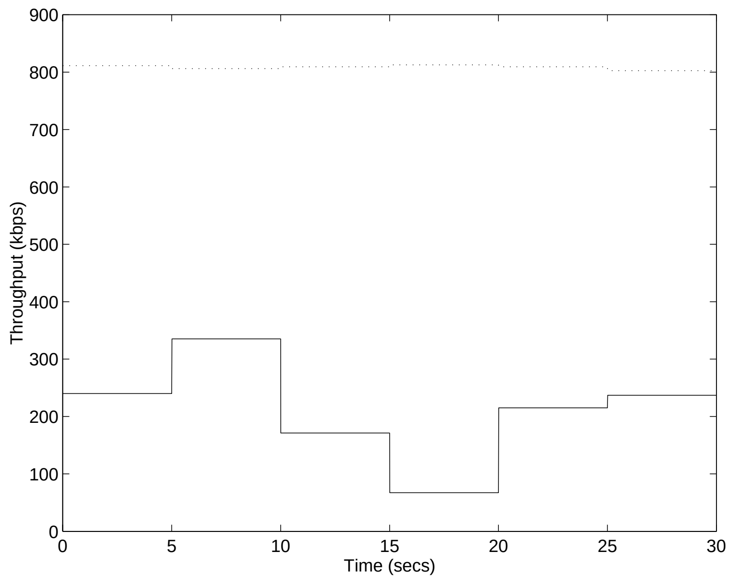


Figure 5-4: A TCP stream (solid) and a congestion-unaware pure UDP stream (dotted) are sent from the same server to the same client across 19 hops in the Internet. The available bandwidth is not shared as the TCP stream reacts to the congestion caused by the UDP stream and the UDP stream ignores any indication of congestion.

is averaged over 20 second intervals. The average bandwidth over the life of the connection is 112 kbps for TCP and 129 kbps for our protocol. The slightly higher throughput for our protocol is not indicative of unfairness. The test was performed many times; sometimes the TCP throughput was higher, but always in the same range as indicated by this representative graph.

The final test is to examine the performance of two streams using our protocol to verify that they too share the available bandwidth. This is shown in Figure 5-6. The average bandwidth of each stream over the life of the connection is 258 kbps and 243 kbps. Table 5.2 is a summary of the results presented in this section.

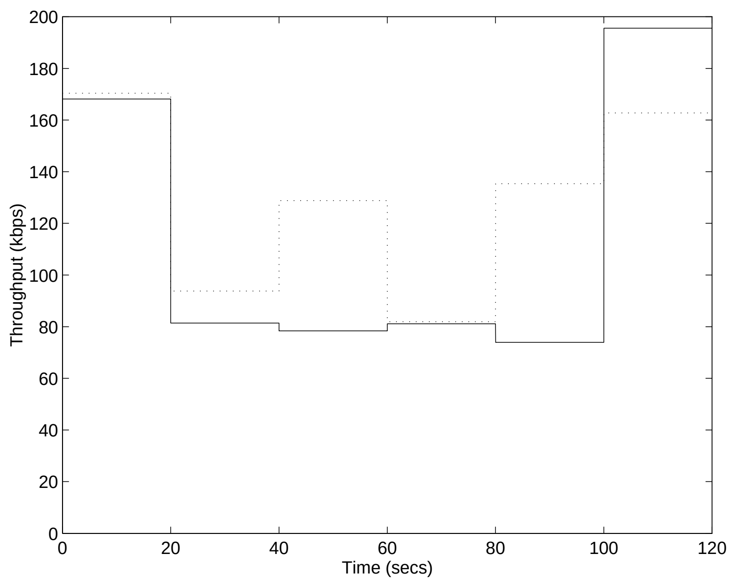


Figure 5-5: A TCP stream (solid) and a stream using our protocol (dotted) are sent from the same client and server across 19 hops in the Internet, showing that they also share the available bandwidth.

5.2.4.2 Controlled Bottleneck

Now that we have shown that our protocol competes fairly with TCP and with itself, the next experiments demonstrate the performance of the rate control algorithm. They were performed with a controlled bottleneck between the client and server, used to simulate a bottleneck router. The bottleneck was set up with a certain bottleneck rate and a maximum buffer size. It reads in packets destined for the

Stream 1 Type	Stream 2 Type	Throughput Stream 1	Throughput Stream 2	Difference
TCP	TCP	292 kbps	342 kbps	17%
TCP	Pure UDP	205 kbps	810 kbps	295%
TCP	Our Protocol	112 kbps	129 kbps	15%
Our Protocol	Our Protocol	243 kbps	258 kbps	6%

Table 5.1: Summarized results for Bandwidth Fairness. Difference is defined to be the difference between the two throughputs divided by the reference throughput, Stream 1.

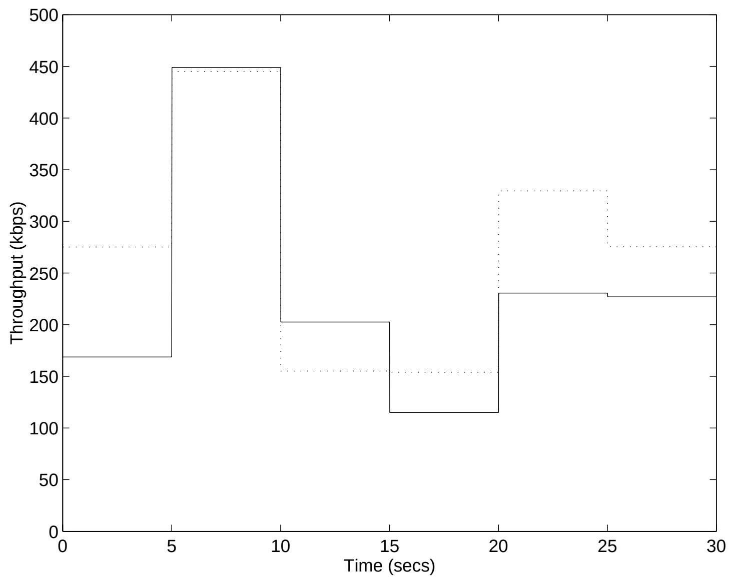


Figure 5-6: Two streams using our protocol sent from the same server to the same client across 19 hops in the Internet, showing that they share the available bandwidth.

client and adds them to the queue. At the same time, it sends out packets onto the network as if the network were operating at the bottleneck rate. It does this by delaying subsequent packets until the time that it would take to send a packet at the bottleneck rate. If the incoming rate is faster than the bottleneck rate, the queue will start to build. If the buffer size is then exceeded, packets are dropped.

Figure 5-7 shows the effects of an MPEG stream originally encoded at 250 kbps going through a 200 kbps bottleneck with a buffer depth of 10 packets. The duration of the connection is 9 minutes. The horizontal lines indicate the desired rate and occupancy. The rate moves to a slightly lower value (180 kbps) within the first 10 seconds, reflecting the bottleneck and the corresponding increase in the buffer occupancy. This is the overshoot period. After the overshoot, the rate settles on the bottleneck rate and stays quite close to it for the remaining 8 minutes, fluctuating no more than 15 kbps below. The buffer too stays within a very narrow range around the desired occupancy. This graph, which is representative of many such tests we

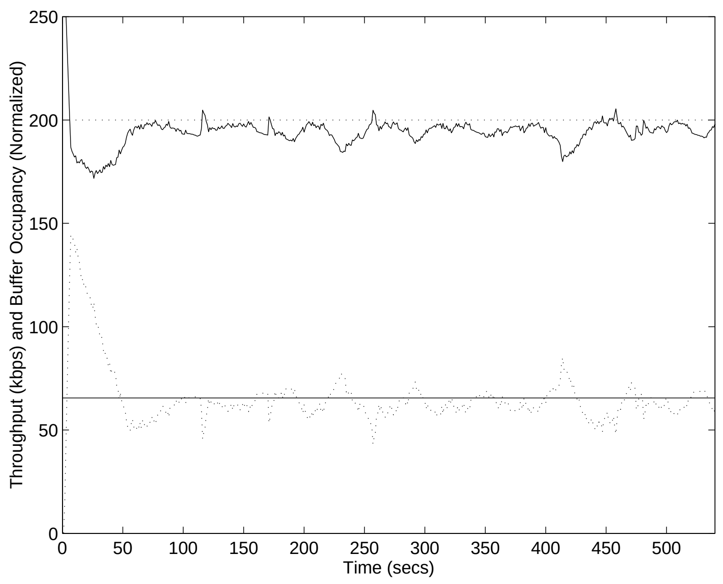


Figure 5-7: The rate (solid) and buffer occupancy (dotted) for a stream originally encoded at 250 kbps going through a 200 kbps bottleneck. The buffer occupancy has been normalized to fit on the same graph as the rate. The horizontal lines show the desired rate and buffer occupancy.

performed, demonstrates that the rate control does converge and varies very little around the converged rate. This is not surprising because the PD rate control law is convergent with minimal oscillations in the time invariant case (constant bottleneck rate).

Figure 5-8 examines the time varying case because now the bottleneck changes every 10 seconds. The intent here is not to simulate the Internet, but to see how well the rate control can follow a continually changing random pattern. The values for the bottleneck were chosen using a uniformly distributed random variable between 100 kbps and 250 kbps.

Despite the fact that the output is time varying, the throughput curve does track the bottleneck curve consistently. This is due to our selection of parameters α_0 and α_1. They were chosen to aggressively attack change, which explains the overshoot in Figure 5-7, but also explain how well it tracks the bottleneck in Figure 5-8. When

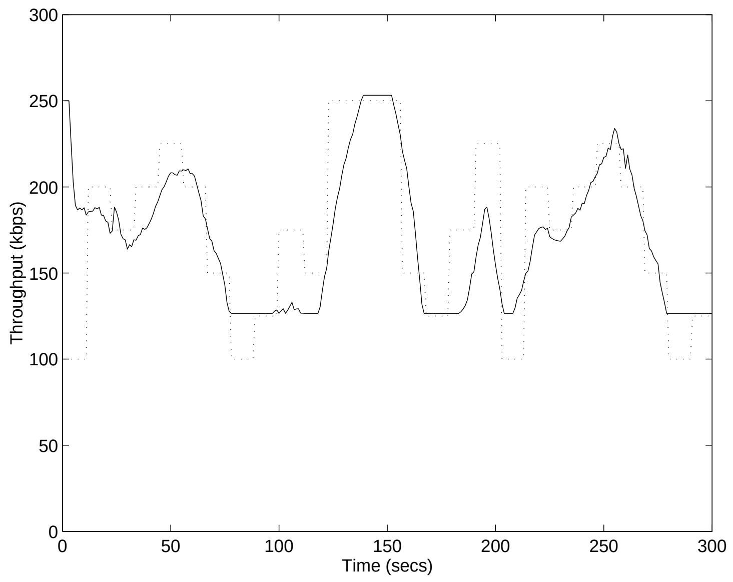


Figure 5-8: The actual throughput (solid) and bottleneck rate (dotted) for a stream originally encoded at 250 kbps going through a varying bottleneck.

the output rate reaches down to 125 kbps, DRS is at its lowest limit and can no longer reduce the bandwidth. This can be seen as the flat bottoms to the throughput curve and is why the throughput curve does not follow the bottleneck curve below 125 kbps.

5.2.4.3 Wide Area Network

In the previous section we showed the convergence of the rate control algorithm. We also showed that it could track a slowly changing output rate quite well. However, the changing output rate was on a very specific timescale, every 10 seconds. Rather than try to simulate the dynamics of the Internet, which is clearly not on a 10 second interval timescale, the next experiment is performed using the Internet.

The stream was sent from a computer in our laboratory in New York to a computer located 19 hops away in New Jersey, where the packets were read in and sent back immediately to a client in our laboratory again, traversing another 19 hops on

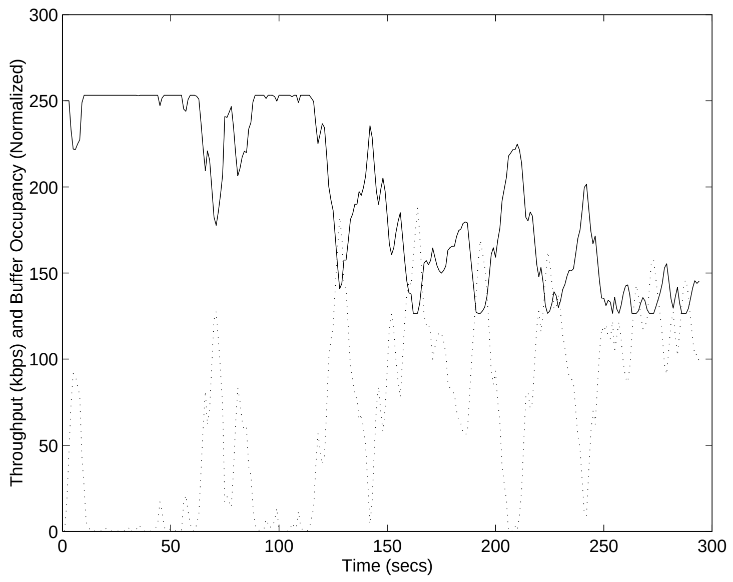


Figure 5-9: The rate (solid) and buffer occupancy (dashed) for a stream originally encoded at 250 kbps going through the Internet, traversing 38 hops. The buffer occupancy has been normalized to fit on the same graph as the rate.

the way back. This experiment is shown in Figure 5-9. Again the original stream is 250 kbps MPEG.

The experiment was run over a 20 minute period, of which this is a 5 minute excerpt. The rate and the buffer occupancy vary substantially, as one would expect given the rate fluctuations in the Internet.

Although the rest of the 20 minute period (not shown here) had some buffer overflows, Figure 5-9 shows none for the given 5 minute period. The goal is to show the advantage of adapting to network conditions. If the original stream had been sent without decreasing its rate based on the available bandwidth, there would be many more instances of buffer overflow. Figure 5-10 shows the buffer occupancy from Figure 5-9 and the buffer occupancy of a stream which could not adapt to the available bandwidth, sending at the originally encoded rate of 250 kbps. The horizontal line indicates the size of the buffer at the server. When the curve crosses it, the user experiences an interruption in playback. The length of the interruption is the time it takes to refill the buffering at the client.

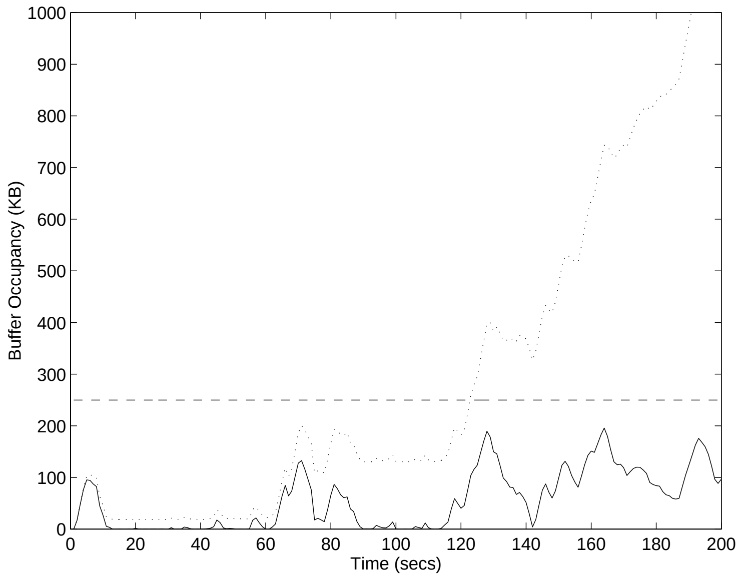


Figure 5-10: The buffer occupancy from the previous graph using our protocol (solid) and the buffer occupancy of a stream which did not adapt to the available bandwidth (dotted). Buffer overflow indicates an interruption in playback. The buffer size is shown as a horizontal line.

From 120 seconds on, the user would have experienced a series of buffer overflows at the server, which would translate into an empty buffer at the client. This is because although there was enough bandwidth in the first 120 seconds, the average available bandwidth over the remainder appears to be only 180 kbps. By continuing to adapt to the network, we significantly reduced the number of interruptions that the user would have experienced with non-adaptable video. The protocol does not provide a guaranteed service, but does provide a service which is much less prone to interruptions in playback. Also any sustained file transfer or web surfing between the same computers would receive the same bandwidth as the video connection.

5.3 Wireless Internet

TCP is arguably the most important transport protocol today. This is primarily due to the huge growth in the use of the Internet, which uses TCP for most data applications, as well as variants of TCP for some real-time applications [45]. The expansion of the Internet has fueled the growth of all kinds of Internet access, including wireless Internet access. As wireless becomes a more common way of accessing the Internet, researchers have sought to provide the same level of service and performance that wireless users have come to expect from traditional wired networks.

TCP operates by continually probing the network for available bandwidth [77]. When TCP detects that the network has lost a packet, it reacts by reducing its transmission rate so as not to add to the congestion. The reasoning is that if a packet has been lost in the network due to congestion, an Internet-friendly protocol should react to this information by reducing its transmission rate. These techniques, called congestion detection and congestion avoidance, were originally designed for wired networks.

For transport protocols, the primary difference between wired and wireless networks is that wireless networks lose packets due to random errors on the wireless link. These errors could be caused by path loss, slow fading or fast fading. The result is that TCP cannot differentiate between a packet lost due to an error or due to congestion. Even the IP layer cannot determine the cause of a packet loss. TCP reduces its transmission rate by invoking its congestion avoidance algorithms in response to a packet loss that may not indicate congestion. In some cases, this leads to TCP under-utilizing the link.

In this chapter, we develop a tractable model for TCP performance in the presence of wireless errors. The advantages of this model are that it is very easy to use and yet still maintains a high degree of accuracy and all of the heuristic content of current more complex models.

The next section provides an overview of previous work in analyzing TCP performance over wireless. Section 5.3.2 presents our analytical model for TCP performance in the presence of random wireless link errors. In Section 5.3.3, we compare the results of the model with both previous work and simulations, showing that our simple model yields quite favorable results.

5.3.1 Previous Work

There has been much research in the area of evaluating TCP performance over wireless networks. Much of the work has been aimed at techniques for improving performance [4, 5, 7, 8]. However, the analytical tools available to these researchers in evaluating the improvements are limited strictly to simulations. Analytical models of TCP performance over wireless that are easy to use should prove invaluable to the furtherance of this work.

Some models have been presented in the past. In [67, 69] the authors present an analytical model for measuring TCP performance on wired networks. Although the model is quite simple to use, it is only used for wired networks where loss is due to congestion alone.

In [59], Lakshman and Madhow present a more accurate model of TCP performance in wired networks for both TCP Tahoe and TCP Reno. The authors then extend this closed form solution to TCP performance in the presence of wireless errors. However, the wireless solution is not closed form and requires a numerical solution of an integral. In addition, many simplifying assumptions are made. Finally, in [21] the authors extend their previous work to incorporate a Markovian model of the channel.

In [17], the authors present a model of TCP dynamics in the presence of wireless errors by using a fluid flow approximation. Of course, the resultant equation for TCP utilization is quite complex and non-trivial to solve.

This work differs from the previous work in that the end result is a simple closed form solution of TCP utilization in the presence of random errors [51]. It unifies the simplicity of the work in [67] with a wireless model. We do make simplifying assumptions, but we clearly point out the limited regions over which these assumptions reduce the accuracy of the model.

5.3.2 Analytical Formulation: TCP

Our model consists of a TCP server located in the wired Internet and sending data across the Internet to a wireless client. The last link of the connection consists of a base station receiving data from the wired network and forwarding it to the client over the wireless link.

We assume that the bottleneck link over the entire TCP connection is the wireless network. In other words, all of the other links along the connection have sufficient bandwidth and buffering so that no packets are dropped due to congestion on those links. Packets are regularly dropped at the base station as TCP continually probes for available bandwidth in the network.

The analysis assumes that there is a given probability of error that the packet will be dropped on the wireless link due to random losses, P_r. We do not attempt to model the channel as a Markov chain switching between good and bad states with a correspondingly good and bad P_r. Our goal here is to evaluate the performance of TCP over wireless. Therefore, as long as the average time spent in a good state or bad state is large compared with the TCP round-trip time, we can assume that each time the channel changes state, TCP will reach a steady state before the channel changes state again. There will certainly be cases where this assumption is not valid; the user of this model could then pick a worst case P_r and use the model to generate a lower bound on performance.

5.3.2.1 Wired Network

In this section we first characterize the TCP stream as if it were operating in a wired network with loss only due to congestion [67, 69]. Then, we add the effects of loss due to errors on the wireless link to examine the performance degradation.

In a wired network, the bandwidth that a TCP connection uses is proportional to the size of the packets being sent, s, the round-trip time (RTT) and the frequency of lost packets due to congestion, P_c. The bandwidth used by TCP, given P_c, is:

$$\lambda_c = \frac{\alpha s}{\mathrm{RTT}\sqrt{P_c}}, \tag{5.5}$$

where α is dependent on the specific TCP implementation as well as the type of loss [69] and $P_c \ll 1$. A suggested value of $\alpha = 1.31$ is used. It should be noted that this equation is only valid for loss rates of less than 5% [67] and that it will produce inflated results beyond this range. We would like to determine the loss rate that TCP experiences due to congestion alone. To do this, we solve equation 5.5 for P_c.

$$P_c = \left(\frac{\alpha s}{\mathrm{RTT}\lambda_c}\right)^2 \tag{5.6}$$

In a wired network where there is very little random loss, TCP will fully utilize the bottleneck link if the buffering at the bottleneck node is larger than the bandwidth-delay product [59]. If the available bandwidth is 1 Mbps and the RTT is 50 milliseconds, then TCP will fully utilize the bandwidth if the buffering at the base station is more than 50 kilobits.

However, if the buffering is less than the bandwidth-delay product in a wired network, then the utilization must be calculated explicitly. In [59], the authors derive the TCP utilization for TCP Reno given the packet rate, μ, and buffering at the bottleneck node, b. In its most simplified form, the equation is as follows:

$$\lambda_c = \begin{cases} \frac{3\mu(b+t\mu)^2}{4(b^2+bt\mu+(t\mu)^2)}, & \text{if } b < \mu t \\ \mu, & \text{otherwise} \end{cases} \tag{5.7}$$

where $t = \tau + 1/\mu$ and τ is the propagation delay. The utilization, ρ_c, is then λ_c/μ. Although we focus on TCP Reno, results can also be derived for TCP Tahoe simply by replacing this equation with a formula for Tahoe utilization and continuing with the subsequent derivation. We continue the derivation assuming that $b < \mu t$ because this is the more complex case, as seen from equation 5.7.

Now that we know the throughput a TCP connection would receive in a wired network, we can substitute λ_c from equation 5.7 into equation 5.6 to calculate the loss rate experienced by this connection.

$$P_c = \left(\frac{4\alpha s(b^2 + bt\mu + (t\mu)^2)}{3t\mu(b + t\mu)^2}\right)^2, \tag{5.8}$$

where the RTT $= t$.

5.3.2.2 Wireless Network

P_c is the loss that the TCP connection experiences due to drops at the base station caused by congestion only, since the base station is the bottleneck node. Now we add the wireless random losses into the model. The losses occur at a rate P_r. The total loss seen by the TCP connection, P_t, could be considered simply as the sum,

$P_c + P_r$.[1] However, this might yield an incorrect value of P_t for two reasons.

Increasing P_r decreases P_c. For instance, a particular TCP connection has a loss rate due to congestion alone of P_c and there are virtually no errors due to the wireless link ($P_r = 0$). Suddenly, the wireless link becomes very noisy and P_r increases dramatically. Many more packets will be dropped due to these errors. TCP will detect these drops and reduce the rate at which it injects traffic into the network. This, in turn, will reduce the number of losses due to congestion (P_c) at the base station since the sender is transmitting at a lower rate and congesting the network less. Therefore, P_c is actually inversely related to P_r so that the sum of P_c and P_r is larger than the actual total loss rate seen by the TCP connection. The inflated value of P_t will create a lower utilization and will be particularly noticeable when P_r is on the same order of magnitude or larger than P_c.

P_r affects throughput more severely. Another problem with summing P_r and P_c is that random losses affect throughput more severely than congestion losses. The losses due to congestion happen periodically. TCP probes the network for available bandwidth by increasing its rate. In the beginning, TCP will be sending at a lower rate than the bottleneck rate so that the bottleneck buffer will be almost empty on average. Eventually, TCP will be sending at a rate which is faster than the bottleneck link and the buffer will begin to fill. After the buffer is completely full, a packet will be lost and TCP will start the cycle over again. In this way, the losses due to congestion happen periodically and only occur after the congestion window has reached the bandwidth delay product plus buffering.

[1] Of course, P_c and P_r must be small so that the sum will always be less than one.

However, random losses can occur at any time. If a random loss occurs when TCP is growing its window and increasing its transmission rate, then the transmission rate may not reach the bottleneck link rate during that cycle. This will severely impair the throughput. Equation 5.5 assumes that losses are periodic. By using random errors in this equation, we are underestimating their effect on throughput. This will yield an artificially high throughput when P_r is large.

Nonetheless, we will use $P_t = P_c + P_r$. In Section 5.3.3 we will show that under most circumstances the effect of these inaccuracies is minor.

$$\lambda_t = \frac{\alpha s}{RTT\sqrt{P_t}} = \frac{\alpha s}{RTT\sqrt{P_c + P_r}} = \frac{\alpha s}{t\sqrt{\left(\frac{4\alpha s(b^2+bt\mu+(t\mu)^2)}{3t\mu(b+t\mu)^2}\right)^2 + P_r}} \tag{5.9}$$

The utilization is $\rho_t = \lambda_t/\mu$.

An important point to be observed from this equation is that it is the relationship between P_c and P_r which truly affects the throughput. If P_r is large, one might expect the performance to be degraded. This is not necessarily true because if P_c is even larger than P_r, the throughput will not be severely degraded. This is because P_c still dominates the square root in the denominator. Therefore, a simple heuristic in evaluating performance is simply to compare P_c with P_r. If P_r is larger than P_c then performance will be significantly degraded. This heuristic will be verified in the next section via simulations. We will also compare ρ_t with previous work in this area as well as simulations.

5.3.3 Performance of Model

5.3.3.1 Comparison with Previous Models

In this section, we compare equation 5.9 with the model presented by Lakshman and Madhow in [59]. We then compare both our model and Lakshman's against our simulations. Our goal in choosing simulation scenarios was to explore a range of values for the bandwidth-delay product, packet rate, buffering, and P_c.

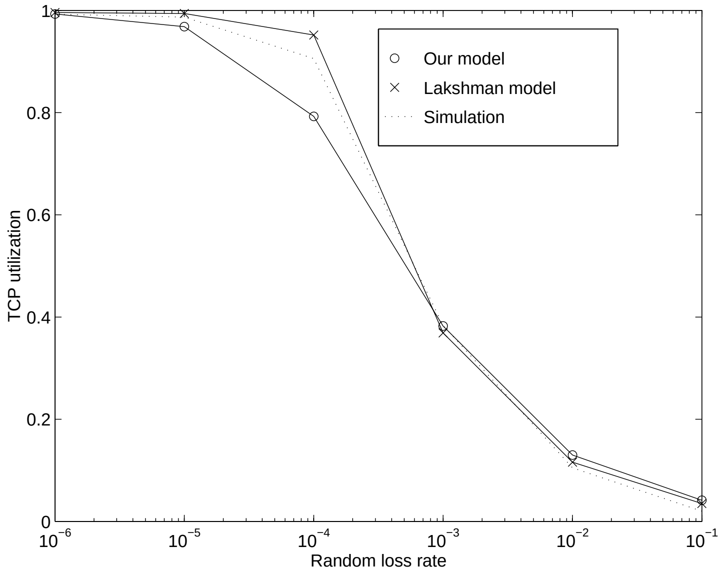


Figure 5-11: Utilization as calculated by our model, Lakshman's model, and simulation for 100 packets/sec, 80 packet buffering, a one second round trip time and increasing loss rate.

Figure 5-11 shows the utilization as calculated by our model, Lakshman's model and obtained through simulations. The packet rate is 100 per second, the buffering is 80 packets and the round trip time is one second. The utilization in a purely wired network, ρ_c, would be 0.996 and $P_c = 0.00017$. The general shape of the curves are the same, though a significant deviation can be seen at $P_r = 10^{-4}$.

This deviation was explained in Section 5.3.2. As explained there, an increase in P_r will cause P_c to decrease. However, we do not take this effect into account in our model. For this reason, our model has an artificially high loss rate, resulting in an artificially lower utilization, when P_r (10^{-4}) is similar to or larger than P_c ($1.7 \cdot 10^{-4}$).

The other effect that our model ignores is the fact that wireless errors reduce utilization more than drops due to congestion. Because of this, our model will overestimate utilization at higher loss rates. These two inaccuracies serve to cancel each other out since the model simultaneously overestimates and underestimates the utilization at higher loss rates. At $P_r = 0.1$, there is also the effect that equation 5.5 overestimates utilization with loss rates higher than 5%.

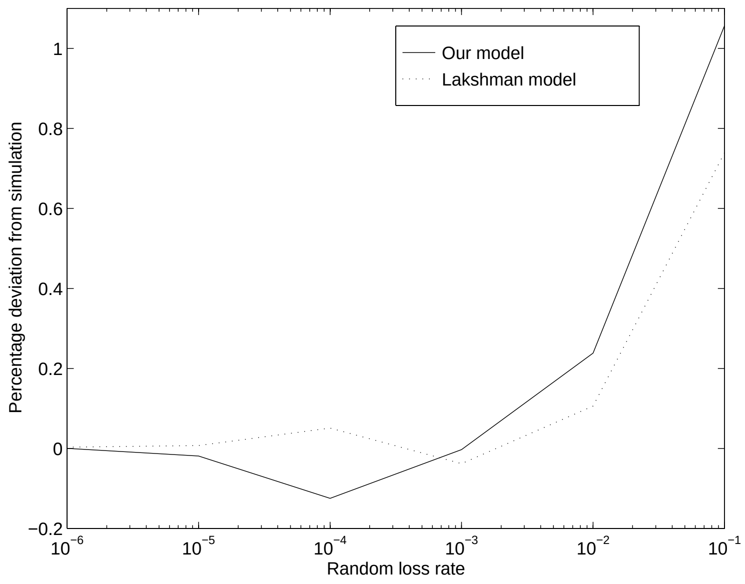


Figure 5-12: Deviation of utilization model from simulation for our model and Lakshman's model at 100 packets/sec, 20 packet buffering, a one second round trip time and increasing loss rate.

In Figure 5-12 we have plotted the deviation of the model efficiencies from the simulated efficiencies,

$$\text{deviation} = \frac{\rho_{model} - \rho_{simulation}}{\rho_{simulation}} \tag{5.10}$$

It is clear from this figure that both models deviate from the simulations at higher loss rates and that the performance is quite comparable, even though the complexity of our model is much lower.

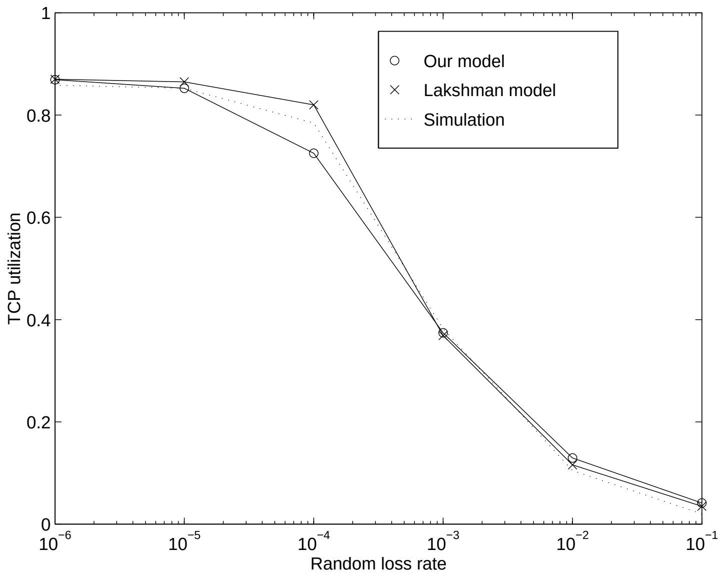


Figure 5-13: Utilization as calculated by our model, Lakshman's model, and simulation for 100 packets/sec, 20 packet buffering, a one second round trip time and increasing loss rate.

Figure 5-13 shows the same scenario except with less buffering, only 20 packets. The utilization in a purely wired network would be 0.871 and $P_c = 0.00023$. The same comments with respect to the deviation at $P_r = 10^{-4}$ can be made. In addition, the deviation of the models from the simulation is quite similar and is not shown here for that reason.

5.3.3.2 Comparison with Simulations

Figure 5-14 is a graph of our model of utilization and simulated utilization for a rate of 10 packets/sec, buffering of 10 packets and a round trip time of 1 second. This would yield a utilization of 1.0 on a purely wired network and a P_c of 0.01716. Again, the deviation from the simulation is worse at higher loss rates.

Figures 5-15 and 5-16 show the final two simulations. The first one is for a

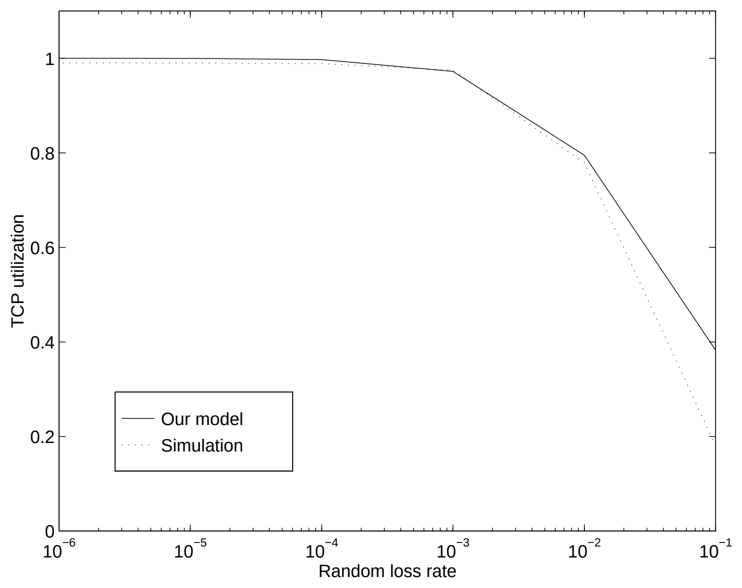


Figure 5-14: Utilization as calculated by our model, and simulation for 10 packets/sec, 10 packet buffering, a one second round trip time and increasing loss rate.

packet rate of 1000 per second and buffering of 1000 packets. The second has a packet rate of 10 per second and buffering of 100 packets. In this case, since the buffering is larger than the bandwidth delay product, equation 5.7 no longer holds. For this reason, we can assume that there is enough buffering for TCP to completely utilize the available bandwidth and we can use $\lambda_c = \mu$ from equation 5.7. Finally, Figure 5-17 shows the deviation from simulation for Figures 5-14 through 5-16.

Table 5.2 summarizes the simulations. The values for the last column (P_r) were chosen by identifying the point at which the slopes of the utilization curve (both simulated and modeled) were most steep. The table demonstrates the validity of the heuristic presented: the most significant degradation in utilization occurs when P_r and P_c are of the same order of magnitude. In every case, the most significant degradation in utilization occurs when P_r is of the same order as P_c. As mentioned earlier, this can be seen very clearly in our model since it is the sum of P_c and P_r which affects the utilization.

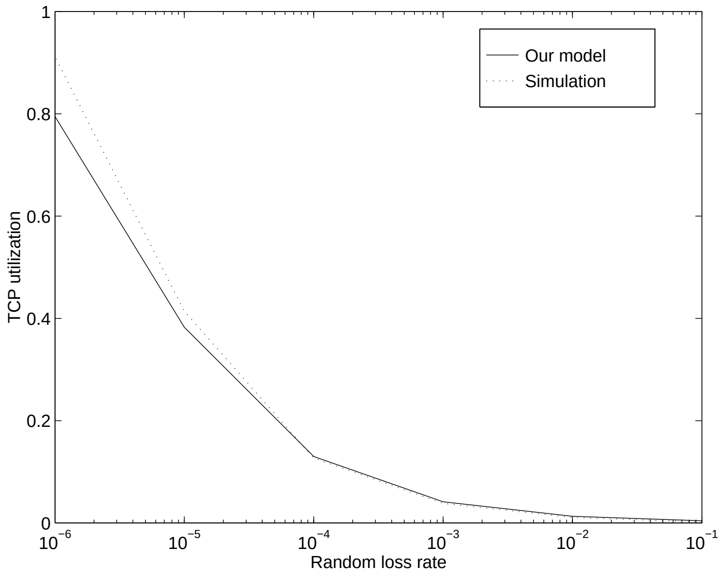


Figure 5-15: Utilization as calculated by our model, and simulation for 1000 packets/sec, 1000 packet buffering, a one second round trip time and increasing loss rate.

5.4 Wireless LAN

5.4.1 Previous Work

There has been much research in the area of improving TCP performance over wireless networks. These efforts can be broken down into three main groups: end-to-end protocols, split connection protocols, and link layer protocols [5]. We focus on the link layer enhancements in this chapter.

Making the link layer reliable is an obvious solution to the problem of TCP performance. Previous work [4, 21] has used simulations to demonstrate that link layer retransmissions can improve performance. In this chapter we generate similar results, but through a more formal analysis. Infinite retransmissions have the disadvantage of introducing substantial delay on links with high bit error rates, since the link layer protocol keeps attempting to retransmit the frame. This delay may be inappropriate for some real-time applications. An example would be video

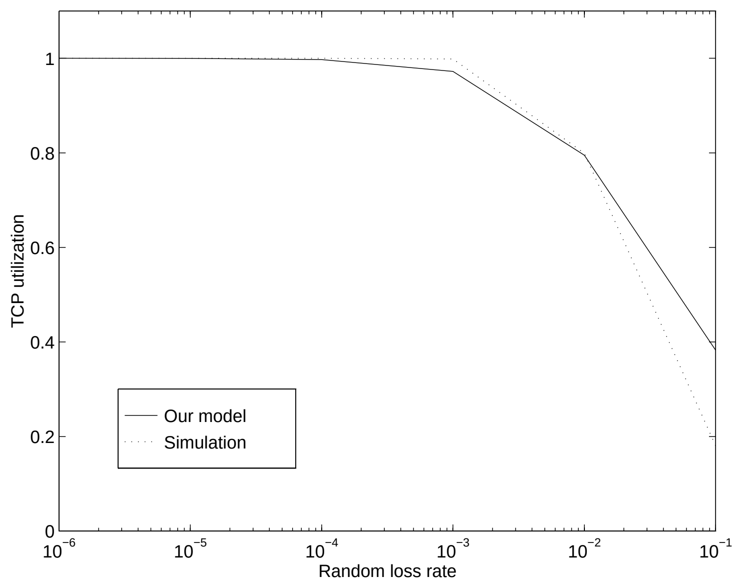


Figure 5-16: Utilization as calculated by our model, and simulation for 10 packets/sec, 100 packet buffering, a one second round trip time and increasing loss rate.

conferencing, where maintaining low delay is sometimes more important than getting all of the data through perfectly. We propose the use of a finite number of retransmissions.

Link level retransmissions may not solve all problems though. It is possible that retransmissions will cause frames to arrive out of order and this will cause TCP to invoke a mild form of congestion avoidance. The frames could arrive out of order in link layer that use a Go-Back-N protocol. This problem does not exist in IEEE 802.11 because it uses a variant of a stop-and-wait protocol. However, the added time it takes to do retransmissions could cause the TCP timers to expire, resulting in a false indication of congestion. This is not a problem if the round-trip time is much larger than the time to do the link layer retransmissions.

Other link level techniques exist too, such as intelligent packet scheduling. The base station gives priority to frames traversing better quality links. If a frame destined for a poor quality link is at the head of a FIFO queue, the retransmission

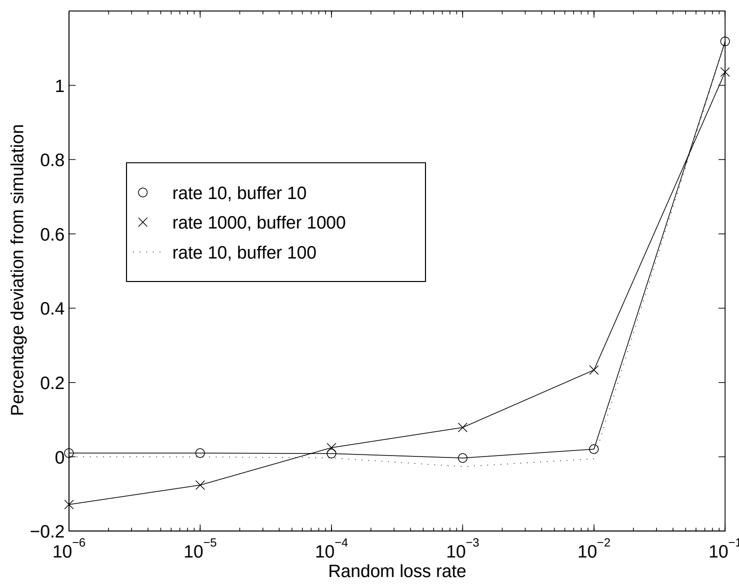


Figure 5-17: Deviation of utilization model from simulation for our model over varying rates and bufferings, a one second round trip time and increasing loss rate.

of that frame could severely delay the transmission of frames which could quickly be delivered over a good quality link [7].

We analyze the simplest link layer case to evaluate TCP performance. A TCP server is located in the Internet and is sending data across the Internet to a wireless client. The last link of the connection consists of a base station receiving data from the wired network and forwarding it to the client over the wireless link. Here we assume the use of the IEEE 802.11 standard for the wireless link. We use the Distributed Coordination Function (DCF) without RTS/CTS. We further assume that there is only one wireless client on the wireless network to avoid obscuring the data with the issue of media access contention.

5.4.2 Analytical Formulation

IEEE 802.11 uses a scheme similar to a stop-and-wait protocol [27]. After sensing the channel is idle, the server waits for a specified period, called the DCF Interframe

Packet Rate (packets/sec)	Buffering (packets)	Bandwidth Delay Product, μt	Wired Utilization	P_c	P_r with Steepest Slope of Utilization
100	80	100	0.996	$1.73 \cdot 10^{-4}$	10^{-4}
100	20	100	0.871	$2.26 \cdot 10^{-4}$	10^{-4}
10	10	10	1.000	$1.72 \cdot 10^{-2}$	10^{-2}
1000	1000	1000	1.000	$1.72 \cdot 10^{-6}$	10^{-6}
10	100	10	1.000	$1.72 \cdot 10^{-2}$	10^{-2}

Table 5.2: Summary of results. The last column indicates the random loss rate that decreases the utilization most dramatically. In all simulations the round trip time is 1 second.

Space (DIFS). The server then sends a frame to the client. The client waits a small period of time (Short Interframe Space, or SIFS) and sends an ACK back to the server. Upon receipt of an ACK, the server then must recontend for the medium. If the sender does not receive an ACK, it waits for a timeout. After the timeout, it must recontend for the medium and then wait a random backoff time, called the contention window.

After the first timeout, the contention window is a uniformly distributed random variable between 0 and 31 time slots. A time slot for the Direct Sequence Spread Spectrum physical layer (DSSS) is 20 microseconds. Each successive timeout increases the range of the contention window exponentially. The second timeout will yield a contention window in the range of 0 to 63 time slots; the third would be from 0 to 127 time slots. Once the range reaches 0 to 255 time slots, it no longer increases.

In this section, the analysis will ignore the random backoff and the frame will be resent immediately after timing out and waiting a DIFS. This simplification will affect our characterization of the throughput on the link by slightly inflating it. Adding this to our model is a subject for future work. The sender attempts to resend the frame only up to a maximum number of retransmissions. After this maximum is reached, it gives up and lets the higher level protocols deal with the loss.

Using this simple model, we attempt to derive the expected time that a frame will occupy the network resources. For instance, under high probability of errors on the channel, the frame will probably be sent, received in error, and resent some number of times. This will increase the expected time to send the frame. Since there are only a finite number of retransmissions, it is possible that a frame will be sent the maximum number of times and then it will be dropped by the link layer.

The time to send the frame and receive an ACK is the sum of the DIFS, the time it takes to actually send the frame, the SIFS, and the time it takes to actually send the ACK. We assume that the timeout is equal to the time it would take to receive an ACK. Each data and ACK frame has 34 bytes of protocol overhead, including frame control information, addresses, and CRC. In addition, the physical layer (DSSS, in this case) has 192 bits of protocol overhead (preamble and header). The rate of the channel can be 1 Mbps or 2 Mbps, but these physical layer overhead bits are always sent at 1 Mbps. From these details and assumptions, the time it takes to send a frame and receive an ACK in microseconds is

$$t = 444 + \frac{354 + s}{r}, \tag{5.11}$$

where s is the size of the frame in bits and r is the physical channel rate in Mbps.

The protocol efficiency is then simply the ideal time it would take to send the frame, s/r, divided by the actual time it takes to send the frame, or

$$\rho = \frac{s/r}{444 + \frac{354+s}{r}}. \tag{5.12}$$

For typical values of $r = 1$ Mbps and $s = 1000$ byte frames, $\rho = 0.91$. Again, this efficiency ignores media access contention.

The analysis assumes that there is a given probability of bit error, P_b, for the wireless link. We do not attempt to model the channel as a Markov chain switching between good and bad states with a correspondingly good and bad P_b. Our goal here is to evaluate the performance of TCP over wireless. Therefore, as long as the average time spent in a good state or bad state is large compared with the TCP round-trip time, we can assume that each time the channel changes state, TCP will reach a steady state before the channel changes state again.

Given P_b, we want to find the probability that a frame is received in error, P_f. This is equivalent to the probability that any of the bits in the frame have an error.

$$P_f = 1 - (1 - P_b)^s \tag{5.13}$$

As the size of the frame increases, the probability that the frame is hit with an error also increases.

To calculate the expected time to send the frame we use P_f [88]. If N is the maximum number of times the frame is sent before the link layer drops the frame, then $N - 1$ is the maximum number of retransmissions. The probability that a frame makes it to the receiver on the first try is $1 - P_f$. The time it would take to do this is simply t, from equation 5.11. The probability that a frame is sent once, received in error, and then resent successfully is $P_f(1 - P_f)$. The time for this to transpire is then $2t$. Finally, the probability that the frame is sent N times is the probability that the frames is sent and fails $N - 1$ times and succeeds the last time or fails the last time,

$$(1 - P_f){P_f}^{N-1} + {P_f}^N. \tag{5.14}$$

The time for this to happen is Nt. Hence, the average time it would take to send the frame, given a probability of bit error is

$$E[T] = NtP_f{}^N + \sum_{i=1}^{N} it(1-P_f)P_f{}^{i-1}, \tag{5.15}$$

where N is the maximum number of times the frame is sent ($N-1$ is the maximum number of retransmissions).

This can be written as

$$E[T] = NtP_f{}^N + \frac{1-P_f}{P_f} t \sum_{i=1}^{N} iP_f{}^i. \tag{5.16}$$

If $N \to \infty$, this equation simplifies to

$$E[T] = \frac{t}{1-P_f} = \frac{t}{(1-P_b)^s}. \tag{5.17}$$

However, $N \to \infty$ means infinite retransmissions. As mentioned earlier, this creates substantial delay and could be intolerable for some applications.

We define the link throughput as the size of the frame divided by the time it takes to send that frame, $\Lambda_l = s/E[T]$. Again, the time to send the frame does not necessarily mean that the frame actually makes it to its destination. It is possible that after the N-th transmission, the frame is dropped. For this reason, the throughput is really a measure of the amount of data that is put on the link and not a measure of the data that actually arrives correctly at the receiver. For the case of a large P_b, the throughput, Λ_l, will decrease with increasing number of transmissions. This is because the link spends more time retransmitting frames and less time transmitting new data.

We then define the link goodput as the amount of usable data. To do this, we define the probability that after transmitting the frame N times, it still is received in error and the link drops it, $P_d = {P_f}^N$. The link goodput is then, $\lambda_l = \Lambda_l(1 - P_d)$, or

$$\lambda_l = \Lambda_l(1 - (1 - (1 - P_b)^s)^N). \tag{5.18}$$

5.4.3 Performance over Wireless LAN

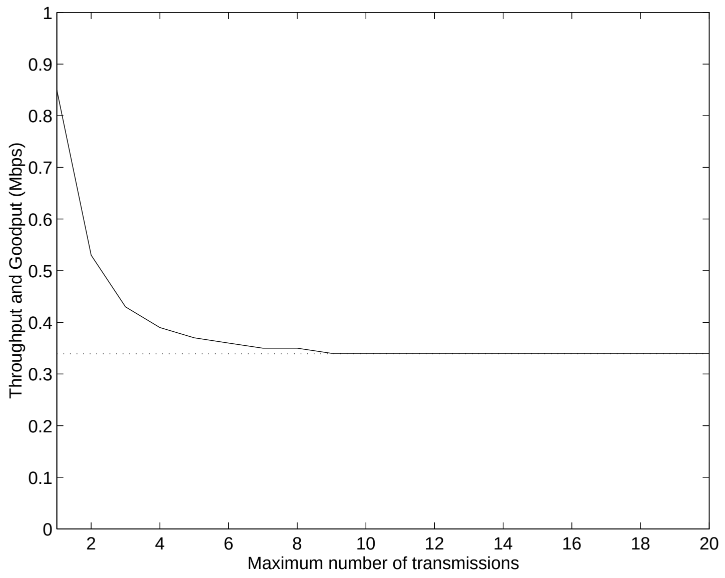


Figure 5-18: Link throughput (solid) and goodput (dashed) for increasing maximum number of transmissions. The bit error probability is 0.0002, r=1 Mbps, and s=576 bytes.

Figure 5-18 shows the throughput, Λ_l, and the goodput, λ_l, for increasing maximum number of transmissions. With no retransmissions, new data is continually put on to the wireless link, regardless of whether or not it makes it to its destination. This is why the throughput is high. However, since there are no retransmissions the goodput is low because much of the data does not arrive at the destination. As the number of retransmissions increases the throughput approaches the goodput as the link becomes more reliable.

Nonetheless, there is a tradeoff when increasing the number of retransmissions

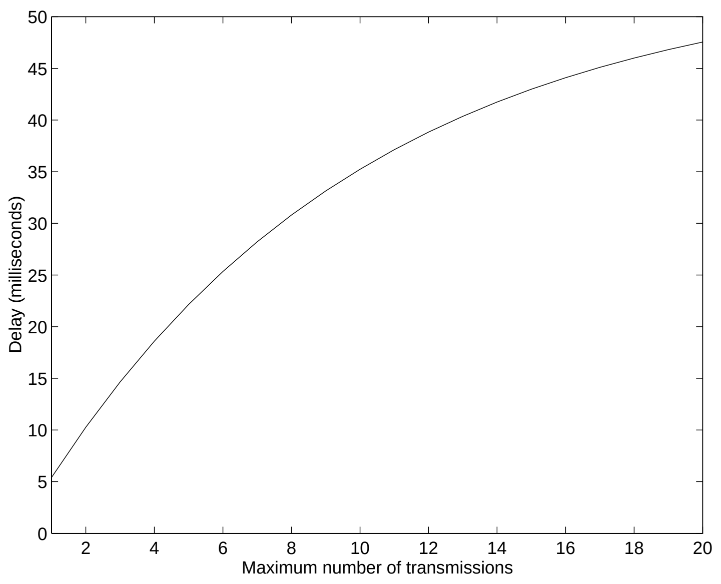


Figure 5-19: Link layer delay for increasing maximum number of transmissions. The bit error probability is 0.0005, r=1 Mbps, and s=576 bytes.

since the delay increases as well. This is seen in Figure 5-19, where we graph equation 5.16 for $P_b = 0.005$. A 45 millisecond increase in delay could be quite intolerable for many applications.

Using the derivation in the previous sections, we now examine the link goodput, λ_l, and TCP goodput, λ_{TCP} for different maximum number of retransmissions and different probabilities of error. With $P_b = 10^{-7}$, the link is in a good state and can almost be considered a wired link in terms of reliability. As expected, the link goodput and TCP goodput are the same, regardless of the number of retransmissions. This is because a retransmission is rarely needed.

However, as P_b starts to increase to 10^{-5}, the link goodput and TCP goodput begin to diverge as shown in Figure 5-20. Figure 5-21 is the same, but with $P_b = 10^{-4}$. However, with just a few retransmissions at the link layer, TCP can use almost all of the available link layer bandwidth. Specifically, at about seven transmissions and for $P_b = 10^{-4}$, the TCP goodput is equivalent to the link goodput. Infinite

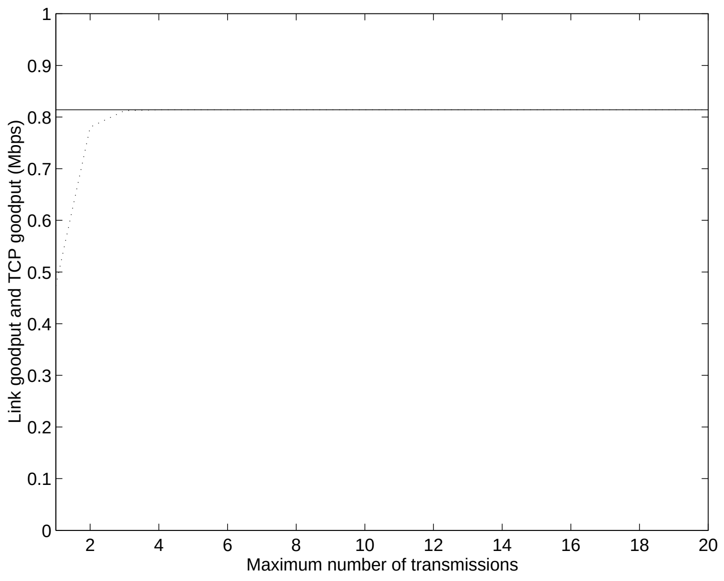


Figure 5-20: Link goodput (solid) and TCP goodput (dashed) for increasing maximum number of transmissions. The bit error probability is 0.00001, r=1 Mbps, and s=576 bytes.

transmissions would yield a substantial increase in delay as seen in Figure 5-19 while providing no performance improvements for TCP utilization. When P_b reaches 10^{-3}, the goodput on the link drops to less than 1%. At this point, the link is so lossy that almost no data is being transferred, regardless of the transport protocol used.

The reason that the TCP goodput reaches the link goodput is that the loss rate due to the wireless link, P_d, is much lower than the loss rate due to congestion, P_c. Therefore, the purpose of the retransmissions is to decrease the link error rate to the point where loss due to congestion is more significant than loss due to the link. Figure 5-22 shows P_d and P_c for $P_b = 10^{-4}$. With no retransmissions, the probability that a packet is dropped by the link, P_d, is high. This value decreases as the link becomes more reliable by adding retransmissions. The loss rate due to congestion, P_c, increases with retransmissions because it is inversely proportional to the throughput, Λ_l, as seen in equation 5.6. As the retransmissions increase, the link throughput decreases which increases P_c. The point at which TCP fully utilizes

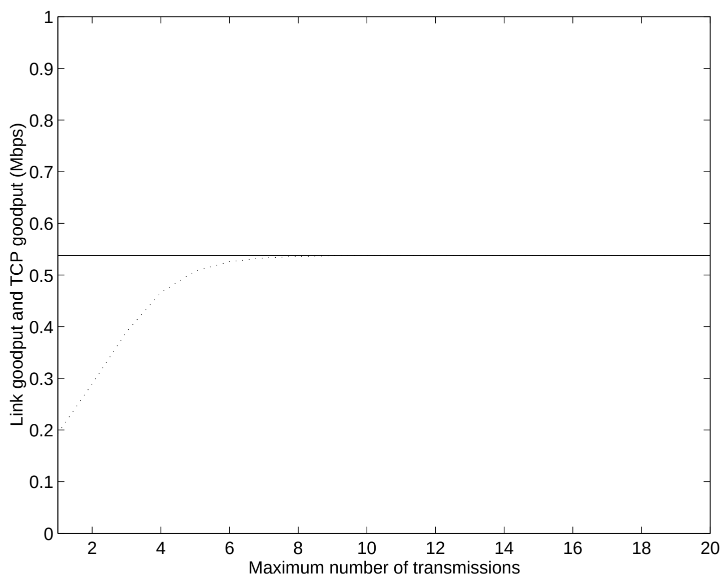


Figure 5-21: Link goodput (solid) and TCP goodput (dashed) for increasing maximum number of transmissions. The bit error probability is 0.0001, r=1 Mbps, and s=576 bytes.

the available link bandwidth is when $P_d \ll P_c$, and again this is at 7 transmissions.

In [59], the authors show that random loss can severely affect TCP performance when the product of the loss probability and the square of the bandwidth-delay product is larger than one.

$$P_d(\Lambda_l \cdot RTT)^2 < 1, \text{ yields good TCP performance.} \tag{5.19}$$

Figure 5-23 shows equation 5.19 for increasing maximum number of transmissions and a P_b of 10^{-4} and 10^{-5}. We can see that for P_b of 10^{-5}, the number of transmissions needed to obtain good TCP performance is 3; for P_b of 10^{-4} the link layer needs to do 7 transmissions.

From Figure 5-20, where $P_b = 10^{-5}$, we saw that TCP fully used the available bandwidth if the link layer transmitted the packet up to 3 times. In Figure 5-21, where $P_b = 10^{-4}$, we saw that the link needed to transmit the packet 7 times. Our

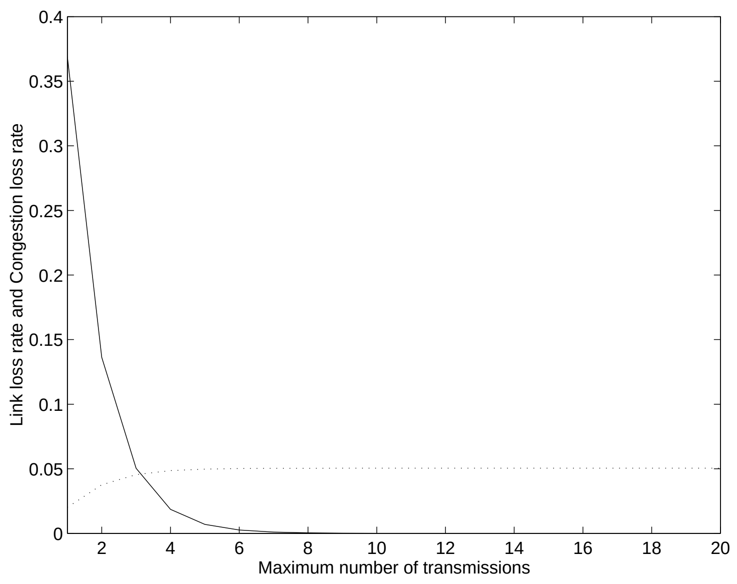


Figure 5-22: Link loss rate (solid) and congestion loss rate (dashed) for increasing maximum number of transmissions. The bit error probability is 0.0001, r=1 Mbps, and s=576 bytes.

analytical results correspond very well with the results presented in [59].

5.5 Concluding Remarks

We have presented a novel protocol for streaming real time media on the Internet. It is novel in that it is as friendly to the Internet as TCP, while still managing to be semi-reliable, providing relatively low delay and significantly decreasing the likelihood of interruptions in playback. It uses the TCP congestion window coupled with a rate control algorithm to obtain an estimate of the available bandwidth in the network. It also uses an intelligent selective retransmission scheme, so as not to waste bandwidth.

We have demonstrated an application that was built using this protocol. Finally, we performed a variety of tests to experimentally evaluate our protocol design goals of fairness, quick convergence, and stability. We showed analytically and experi-

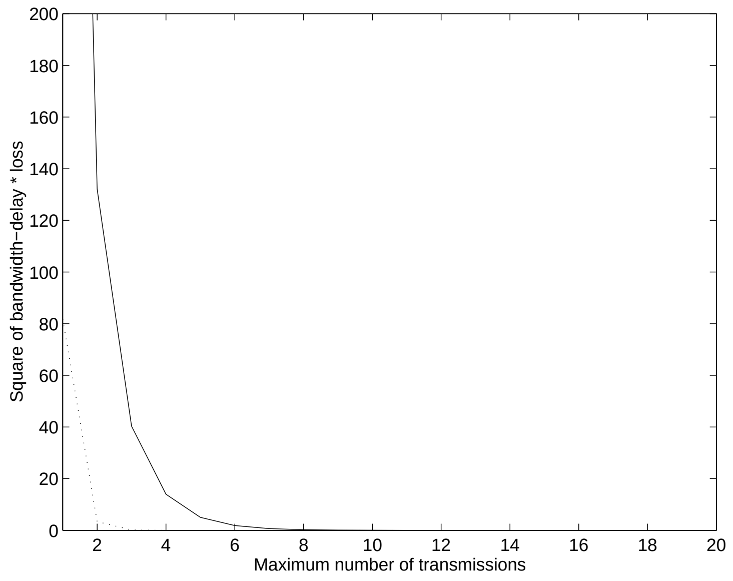


Figure 5-23: The product of the loss probability and the square of the bandwidth-delay product for increasing maximum number of transmissions. The bit error probability is 0.0001 (solid) and 0.00001 (dashed), r=1 Mbps, and s=576 bytes.

mentally that these goals were met in both a controlled environment and also in tests in the wide area Internet.

Since the protocol is so similar to TCP, we extended the analysis to the case of streaming over wireless networks. We developed a tractable model for TCP performance over wireless. This model has excellent agreement with both simulations and the less tractable models presented previously.

We then used a simplified version of the IEEE 802.11 standard for wireless LANs to evaluate the performance of TCP over these networks. The results showed that TCP utilization will remain high if there are enough link layer retransmissions to maintain a certain level of reliability on the link.

Chapter 6

Conclusions

We have presented two very distinct frameworks for multimedia communications. The first assumes that the network was designed for the application and therefore can guarantee QoS. The second takes the reverse approach by assuming that the application is built for the network, and is therefore adaptive to the state of the network.

In the case of a network with QoS guarantees, we have shown that if the QoS requirements of the application are extremely strict, general purpose computing devices may not be able to guarantee QoS as a video server. The case presented was a video client which had very strict timing requirements. These requirements could not be met through the use of traditional timing mechanisms at the video server. The use of unorthodox timing mechanisms provided a solution to the timing requirements of the client. However, this came at the added cost of the server being less scalable.

Subsequently, we introduced a model for analyzing the subjective quality of a VoD system at the end user. Using this model, we developed a methodology for characterizing the performance of the video server, where the performance metric is directly related to the subjective quality perceived at the end user of the application. We found that the GCRA can be used quite effectively in this capacity because it can be easily translated into a metric that is applicable to the video client. We combined this metric with the schedulable region to determine the range of scalability provided by our server. We also used the Columbia VoD testbed as an example of the methodology and provided empirical results of the scalability of our server while still maintaing the strict QoS requirements of the video client.

We then moved on to the second environment for deploying real time applications. Applications are built around networks when the network cannot support QoS guarantees. This scenario requires two fundamental tools. The first is that such applications must be adaptive to the network. To be adaptive, they must use media which are themselves adaptable. Secondly, each network has a different way of explicitly or implicitly communicating its state to the application and it is the responsibility of the application to gather this information and translate it into information which can be used by a scalable codec.

Chapter 4 discussed a variety of different codecs for audio, video, and images. Some of the codecs were inherently scalable; some were not, but were manipulated in clever ways in order to render them scalable. Specifically, we presented an audio codec which employed a novel silence detection algorithm. We presented several requirements for the use of silence detection. These characteristics consisted of the elimination of inter-sentence silence or longer, minimal complexity and low delay. The main features of the algorithm were its simplicity, efficiency and performance for the particular environment presented. By implementing the algorithm in a real system, the silence detection was shown to reduce the end-to-end delay while maintaing the quality of the speech.

The last chapter presented a streaming protocol for use with non-QoS networks. The protocol was highly adaptive in that it continually sensed the state of the network and then passed this information on to the scalable codec. It had the added benefit of being completely Internet-friendly through the use of TCP-like algorithms. This will prove to be a critical issue as real time media become more prevalent on the Internet. Experimental results were provided which demonstrated the performance of the algorithm in both controlled and wide area experiments.

Finally, we extended this work to the case of wireless networks. A new tractable model for TCP utilization in the presence of wireless random errors was presented. This model was shown to be much simpler to use than previous work, while maintaining a high degree of accuracy. The accuracy was demonstrated through the comparison of this model with previous work as well as simulations. This accuracy was verified through comparisons with previous work as well as simulations.

References

[1] M. Allman and C. Hayes. An evaluation of TCP slow start modifications. Technical report, Ohio University, February 1997.

[2] M. Allman, C. Hayes, H. Kruse, and S. Ostermann. TCP performance over satellite links. In *Proc. of the 5th Int'l Conference on Telecommunication Systems*, 1997.

[3] E. Altman, F. Baccelli, and J.C-.Bolot. Discrete-time analysis of adaptive rate control mechanisms. In *Proc. 5th Int'l Conference on Data Communication Systems and their Performance*, volume C-21, pages 121–140, October 1993.

[4] B.S. Bakshi, P. Krishna, N.H. Vaidya, and D.K. Pradhan. Improving performance of TCP over wireless networks. In *Proc. of the 17th International Conference on Distributed Computing Systems*, pages 365–373, May 1997.

[5] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proc. ACM SIGCOMM Int. Conf. Communications Architecture, Protocols and Applications*, pages 256–269, Stanford, CA, October 1996.

[6] L. Benmohamed and S. Meerkov. Feedback control of congestion in packet switching networks: The case of a single congested node. *IEEE/ACM Trans. Networking*, 1(6), December 1993.

[7] P. Bhagwat, P. Bhattacharya, A. Krishna, and S.K. Tripathi. Enhancing throughput over wireless LANs using channel state dependent packet scheduling. In *Proc. IEEE INFOCOM*, volume 3, pages 1133–1140, San Francisco, CA, March 1996.

[8] S. Biaz and N Vaidya. Using end-to-end statistics to distinguish congestion and corruption losses: A negative result. Technical report, Texas A and M University, Department of Computer Science, August 1997.

[9] R. Bollow. Video transmission using the available bit rate service. Technical report, Berlin University of Technology, 1996.

[10] J.-C. Bolot and A. U. Shankar. Dynamical behavior of rate-based flow control mechanisms. *ACM SIGCOMM Computer Communication Review*, 20(2):35–49, April 1990.

[11] J.-C. Bolot and T. Turletti. A rate control mechanism for packet video in the Internet. In *Proc. IEEE INFOCOM*, pages 1216–1223, Toronto, Canada, June 1994. IEEE.

[12] J.-C. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the Internet. In *Proc. ACM SIGCOMM Int. Conf. Communications Architecture, Protocols and Applications*, pages 58–67, London, England, August 1994.

[13] J.-C. Bolot and A. Vega-Garcia. Control mechanisms for packet audio in the Internet. In *Proc. IEEE INFOCOM*, pages 232–239, San Francisco, CA, March 1996.

[14] L.S. Brakmo, S. O'Malley, and L.L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. *Computer Communication Review*, 24(4):24–35, October 1994.

[15] K. Bullington. Engineering aspects of TASI. *BSTJ*, 38:353–364, 1959.

[16] I. Busse, B. Deffner, and H. Schulzrinne. Dynamic QoS control of multimedia applications based on RTP. *Computer Communications*, 19(1):49–58, January 1996.

[17] A. Chan, D. Tsang, and S. Gupta. TCP (transmission control protocol) over wireless links. In *Proc. of IEEE 47th Vehicular Technology Conference*, pages 1326–1330, Phoenix, AZ, May 1997.

[18] S.-F. Chang, A. Eleftheriadis, and D. Anastassiou. Development of Columbia's video on demand testbed. *Signal Processing: Image Communication*, 8(3):191–207, April 1996. Special Issue on Video on Demand.

[19] S.-F. Chang, A. Eleftheriadis, D. Anastassiou, S. Jacobs, H. Kalva, and J. Zamora. Columbia's VoD and multimedia research testbed with heterogeneous network support. *Multimedia Tools and Applications*, 5(2):171–184, September 1997. Kluwer Academic Publishers. Special Issue on Video on Demand Systems: Technology, Interoperability, and Trials.

[20] Y.-H. Chang, D. Coggins, D. Pitt, D. Skellern, M. Thapar, and C. Venkatraman. An open-systems approach to video on demand. *IEEE Communications Mag.*, 32(5):68–80, May 1994.

[21] H. Chaskar, T.V. Lakshman, and U. Madhow. On the design of interfaces for TCP/IP over wireless. In *Proc. of MILCOM '96 IEEE Military Communications Conference*, volume 1, pages 199–203, McLean, VA, October 1996.

[22] W. Y. Chen and D. L. Waring. Applicability of ADSL to support video dial tone in the copper loop. *IEEE Communications Mag.*, 32(5):102–109, May 1994.

[23] Z. Chen, S.-M. Tan, R. Campbell, and Y. Li. Real time video and audio in the World Wide Web. In *Proc. of Fourth International World Wide Web Conference*, Boston, Massachusetts, December 1995.

[24] P. Cherriman and L. Hanzo. Robust H.263 video transmission over mobile channels in interference limited environments. In *Proc. First IEEE Wireless Video Communication Workshop*, Loughborough, UK, September 1996.

[25] The ATM Forum Technical Committee. ATM user-network interface (UNI) signalling specification. Technical Report atf95-1434R13/at-sig-0061.000, The ATM Forum, June 1996. Version 4.0.

[26] The ATM Forum Technical Committee. Audiovisual multimedia services: Video on demand. Specification 1.1. Technical Report af-saa-0049.001, The ATM Forum, March 1997.

[27] B. Crow, I. Widjaja, J. Kim, and P. Sakai. IEEE 802.11 wireless local area networks. *IEEE Communications Mag.*, 35(9):116–126, September 1997.

[28] A. Dan. Buffering and caching in large-scale video servers. In *IEEE COMPCON: Technologies for the Information Superhighway*, pages 217–224, March 1995.

[29] DAVIC. 1.2 specification. Technical report, Digital Audio-Visual Council, Geneva, Switzerland, 1997.

[30] D. Deloddere, W. Verbiest, and H. Verhille. Interactive video on demand. *IEEE Communications Mag.*, 32(5):82–88, May 1994.

[31] M. DonVito and B. Schoenherr. Subband coding with silence detection. In *Proc. International Conference on Acoustics, Speech, and Signal Processing*, pages 1433–1436, Tampa, FL, 1985.

[32] P. Drago, A. Molinari, and F. Vagliani. Digital dynamic speech detectors. *ieeetcom*, 26:140–145, 1978.

[33] A. Eleftheriadis and D. Anastassiou. Constrained and general dynamic rate shaping of compressed digital video. In *Proc. 2nd IEEE International Conference on Image Processing*, Arlington, VA, October 1995.

[34] A. Eleftheriadis, S. Pejhan, and D. Anastassiou. Algorithms and performance evaluation of the xphone multimedia communication system. In *Proc. ACM Multimedia*, pages 311–320, Anaheim, CA, 1993.

[35] A. Eleftheriadis, S. Pejhan, and D. Anastassiou. Architecture and algorithms of the xphone multimedia communication system. *ACM/Springer Verlag Multimedia Systems*, 2:89–100, 1994.

[36] T. Faber, L. H. Landweber, and A. Mukherjee. Dynamic time windows: Packet admission control with feedback. In *Proc. ACM SIGCOMM Int. Conf. Communications Architecture, Protocols and Applications*, pages 124–135, Baltimore, Maryland, 1992.

[37] D. Le Gall. MPEG: A video compression standard for multimedia applications. *Comm. ACM*, 34(4):46–58, April 1991.

[38] Object Management Group. The common object request broker: Architecture and specification. Technical Report PTC/97-09-01, OMG, July 1995. Revision 2.0.

[39] F. Guillemin and J. W. Roberts. Jitter and bandwidth enforcement. In *Proc. IEEE Global Telecommunications Conf. (GLOBECOM)*, pages 261–265, Phoenix, AZ, December 1991.

[40] U. Horn and B. Girod. Scalable video transmission for the Internet. In *Proc. JENC8*, pages 358–365, Edinburgh, UK, May 1997.

[41] C. Hsu and A. Ortega. Joint encoder and VBR channel optimization with buffer and leaky bucket constraints. In *Symposium on Multimedia Communications and Video Coding*, Brooklyn, NY, October 1995.

[42] J. M. Hyman, A. A. Lazar, and G. Pacifici. Modelling VC, VP and VN bandwidth assignment strategies in broadband networks. In *Proc. IEEE Int. Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 102–113, Lancaster, UK, November 1993.

[43] S. Jacobs and A. Eleftheriadis. Providing video services over networks without quality of service guarantees. In *Proc. World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, Sophia-Antipolis, France, October 1996.

[44] S. Jacobs and A. Eleftheriadis. Video pump design for interoperability with set top units: The case against small PDUs. In *Proc. IEEE Int. Workshop Network and Operating System Support for Digital Audio and Video (NOSS-DAV)*, pages 123–130, Tokyo, Japan, April 1996.

[45] S. Jacobs and A. Eleftheriadis. Adaptive video applications for non-QoS networks. In *Proc. IFIP Fifth International Workshop on Quality of Service*, New York, NY, May 1997.

[46] S. Jacobs and A. Eleftheriadis. Real-time dynamic rate shaping and control for Internet video applications. In *Proc. IEEE Workshop Multimedia Signal Processing*, Princeton, NJ, June 1997. IEEE.

[47] S. Jacobs and A. Eleftheriadis. A real time protocol that guarantees fairness with TCP. Technical report, Center for Telecommunications Research, Columbia University, New York, NY, October 1997.

[48] S. Jacobs and A. Eleftheriadis. Real-time video on the Web using dynamic rate shaping. In *Proc. IEEE Int. Conf. Image Processing (ICIP)*, pages 14–17, Santa Barbara, CA, October 1997.

[49] S. Jacobs and A. Eleftheriadis. Streaming video using dynamic rate shaping and TCP congestion control. Technical report, Center for Telecommunications Research, Columbia University, New York, NY, January 1998.

[50] S. Jacobs, A. Eleftheriadis, and D. Anastassiou. Silence detection in multimedia networks. *ACM/Springer Verlag Multimedia Systems*, 1997. To appear.

[51] S. Jacobs, H. Wang, A. Eleftheriadis, and M. Schwartz. A simple analytical model for wireless tcp utilization. Technical report, Center for Telecommunications Research, Columbia University, New York, NY, January 1998.

[52] V. Jacobson. Congestion avoidance and control. *Computer Communication Review*, 18(4):314–329, August 1988.

[53] M.A. Jolfaei, B. Heinrichs, and M.R. Nazeman. Improved TCP error control for heterogeneous WANs. In *Proc. of IEEE National Telesystems Conference*, pages 257–260, San Diego, CA, May 1994. IEEE.

[54] J. R. Jones. Baseband and passband transport systems for interactive video services. *IEEE Communications Mag.*, 32(5):90–101, May 1994.

[55] ISO/IEC JTC1/SC29/WG11. Information technology - generic coding of moving pictures and associated audio: Systems. Technical Report ISO/IEC

13818-1, International Organization for Standardization, Geneva, Switzerland, November 1994. Recommendation H.222.0.

[56] ISO/IEC JTC1/SC29/WG11. Information technology - generic coding of moving pictures and associated audio: MPEG-2 digital storage media - command and control (DSM–CC). Technical Report ISO/IEC 13818-6, International Organization for Standardization, Geneva, Switzerland, June 1994. Recommendation H.222.0.

[57] ISO/IEC JTC1/SC29/WG11. Information technology - generic coding of moving pictures and associated audio: Video. Technical Report ISO/IEC 13818-2, International Organization for Standardization, Geneva, Switzerland, June 1994.

[58] H. Kanakia, P.P. Mishra, and A. Reibman. An adaptive congestion control scheme for real-time packet video transport. *Computer Communication Review*, 23(4):20–31, September 1993.

[59] T.V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Networking*, 5(3):336–350, June 1997.

[60] A. A. Lazar, K.-S. Lim, and F. Marconcini. **xbind** version 1.0: A CORBA based binding architecture. Technical Report 412-95-18, Center for Telecommunications Research, Columbia University, New York, NY, June 1995.

[61] A. A. Lazar and G. Pacifici. Control of resources in broadband networks with quality of services guarantees. *IEEE Communications Mag.*, 29(10):66–73, October 1991.

[62] A. A. Lazar, A. Temple, and R. Gidron. An architecture for networks that guarantees quality of service. *International Journal of Digital and Analog Communications Systems*, 3(2):229–238, April–June 1990.

[63] J. Lynch, L. Josenhans, and R. Crochiere. Speech/silence segmentation for real-time coding via rule based adaptive endpoint detection. In *Proc. IEEE International Conference Acoustics, Speech, and Signal Processing*, pages 1348–1351, Dallas, TX, 1987.

[64] A. Madden. Data and voice come together on the net. *The Red Herring*, (49):48, December 1997.

[65] A. Madden. Eyes on the prize. *The Red Herring*, (43):40–42, June 1997.

[66] A. Madden. No such thing as a free call. *The Red Herring*, (48):96–100, November 1997.

[67] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. Technical report, Internet Engineering Task Force, 1997.

[68] M. Marsan, M. Baldi, A. Bianco, R. Lo Cigno, and M. Munafo. Simulation analysis of TCP and XTP file transfers in ATM networks. In *Proc. of Protocols for High Speed Networks*, pages 29–47, Sophia-Antipolis, France, October 1996.

[69] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behaviour of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3):67–82, July 1997.

[70] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM Int. Conf. Communications Architecture, Protocols and Applications*, pages 117–130, Palo Alto, California, August 1996.

[71] V. Nirkhe and M. Baugher. Quality of service support for networked media players. In *IEEE COMPCON: Technologies for the Information Superhighway*, pages 234–238, March 1995.

[72] A. Oppenheim and R. Schafer. *Discrete Time Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[73] A. Ortega and M. Vetterli. Multiple leaky buckets for increased statistical multiplexing of atm video. In *Proc. Packet Video Workshop*, Portland, OR, September 1994.

[74] V. Padmanabhan, H. Balakrishnan, K. Sklower, E. Amir, and R. Katz. Networking using direct broadcast satellite. In *Proc. Workshop on Satellite-Based Information Systems*, Rye, NY, November 1996.

[75] C. Partridge, S. Floyd, and M. Allman. Increasing TCP's initial window. Internet Draft, Internet Engineering Task Force, August 1997. Work in progress.

[76] C. Partridge and T.J. Shepard. TCP/IP performance over satellite links. *IEEE Network Mag.*, 11(5):44–49, September 1997.

[77] J. Postel. Transmission control protocol. Internet RFC 793, Internet Engineering Task Force, September 1981.

[78] M. De Prycker. *Asynchronous Transfer Mode: Solution for Broadband ISDN.* Ellis Horwood Series in Computer Communications and Networking. Ellis Horwood, New York, NY, second edition, 1993.

[79] K. Ramakrishnan. Operating system support for a video-on-demand file service. *ACM/Springer Verlag Multimedia Systems*, (3):53–65, 1995.

[80] K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *Proc. ACM SIGCOMM Int. Conf. Communications Architecture, Protocols and Applications*, pages 303–313, Stanford, California, August 1988. ACM.

[81] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. In *Proc. IEEE INFOCOM*, pages 680–688, Toronto, Canada, June 1994. IEEE Computer Society Press, Los Alamitos, California.

[82] M. Rangoussi, A. Delopoulos, and M. Tsatsanis. On the use of higher order statistics for robust endpoint detection of speech. In *Proc. IEEE Signal Processing Workshop on Higher-Order Statistics*, pages 56–60, South Lake Tahoe, CA, 1993. IEEE.

[83] O. Rose. The Q-bit scheme: Congestion avoidance using rate adaptation. *ACM SIGCOMM Computer Communication Review*, 22(2):29–42, April 1992.

[84] T. Sakatani. Congestion avoidance for video over IP networks. In *Proc. Multimedia Transport and Teleservices*, pages 256–273, November 1994.

[85] N. Samaraweera and G. Fairhurst. Explicit loss indication and accurate RTO estimation for TCP error recovery using satellite links. *IEE Proceedings-Communications*, 144(1):47–53, February 1997.

[86] M. Savoji. A robust algorithm for accurate endpointing of speech signals. *Speech Communication*, 8:45–60, 1989.

[87] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp a transport protocol for real-time applications. Internet RFC 1889, Internet Engineering Task Force, January 1996.

[88] M. Schwartz. *Telecommunication Networks*. Addison-Wesley Series in Electrical and Computer Engineering. Addison-Wesley, Reading, MA, 1987.

[89] ITU SG/16. Video coding for low bit rate communication. Technical Report Draft H.263, International Telecommunication Union, September 1997.

[90] W. Stevens. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Internet Draft, Internet Engineering Task Force, March 1996.

[91] H. Sun, J. Zdepski, and D. Raychaudhuri. Error concealment for MPEG video over ATM. Technical Report MPEG92 AVC-308, CCITT ISO-IEC/JTC1/SC2/WG1, July 1992.

[92] W. Tan, E. Change, and A. Zakhor. Real time software implementation of scalable video codec. In *Proc. IEEE Int. Conf. Image Processing (ICIP)*, pages 17–20, Lausanne, Switzerland, September 1996. IEEE.

[93] D. Taubman and A. Zakhor. Multi-rate 3-d subband coding of video. *Proc. IEEE Transactions on Image Processing*, 3(5):572–588, April 1993.

[94] D. Taubman and A. Zakhor. A common framework for rate and distortion based scaling of highly scalable compressed video. *IEEE Trans. Circuits and Syst. for Video Technol.*, 6(4):329–354, August 1996.

[95] J. Tham, S. Ranganath, and A. Kassim. Highly scalable wavelet-based video codec for very low bit-rate environment. *IEEE J. Select. Areas Commun.*, 16(1):12–27, January 1998.

[96] T. Turletti and C. Huitema. Videoconferencing in the Internet. *IEEE/ACM Trans. Networking*, 4(3):340–351, June 1996.

[97] C. Un and H. Lee. Voiced-unvoiced-silence discrimination of speech by delta modulation. *IEEE Trans. Acoustics, Speech and Signal*, 28:398–407, 1980.

[98] R. Vaccaro. *Digital Control, A State Space Approach.* Electrical and computer engineering. McGraw-Hill, New York, 1995.

[99] R. Wilder. Fairness issues for mixed TCP/OSI internets. In *Military Communications in a Changing World*, pages 177–181, McLean, VA, 1991.

[100] C. Yap, K. Ngan, and R. Liyanapathirana. Error protection scheme for the transmission of H.263 codec video over mobile radio channels. In *Proc. SPIE Conf. Visual Communication & Image Processing*, pages 1241–1249, San Jose, CA, February 1997.

[101] R. Yavatkar and K. Lakshman. A CPU scheduling algorithm for continuous media applications. In *Proc. IEEE Int. Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 223–226, April 1995.

[102] J. Zamora. Issues of video services over ATM. Technical Report 405-95-11, Center for Telecommunications Research, Columbia University, New York, NY, May 1995.

[103] J. Zamora, D. Anastassiou, S.-F. Chang, and K. Shibata. Subjective quality of service performance of video-on-demand under extreme ATM impairment conditions. In *Proc. AVSPN*, pages 5–10, Aberdeen, United Kingdom, September 1997.

[104] J. Zamora, D. Anastassiou, and K. Ly. Cell delay variation performance of CBR and VBR MPEG-2 sources in an ATM multiplexer. In *Proc. European Signal Processing Conference (EUSIPCO)*, Trieste, Italy, September 1996.

[105] J. Zamora, S. Jacobs, A. Eleftheriadis, S.-F. Chang, and D. Anastassiou. A practical methodology for guaranteeing quality of service for video-on-demand. Technical Report 447-96-13, Center for Telecommunications Research, Columbia University, New York, NY, April 1996. Submitted to *IEEE Trans. Circuits and Syst. for Video Technol.*