EXHIBIT 1015

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

AN END-TO-END ARCHITECTURE FOR QUALITY ADAPTIVE STREAMING APPLICATIONS IN THE INTERNET

by

Reza Rejaie

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Science)

December 1999

Copyright 1999 Reza Rejaie

Webgroup CZ a.s. Ex. 1015, Page 4 of 204

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 9987598

UMI®

UMI Microform 9987598
Copyright 2001 by Bell & Howell Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90007

This dissertation, written by

REZA REJAIE

under the direction of his Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of

DOCTOR OF PHILOSOPHY

Dean of Graduate Studies

Date 9-14-99

DISSERTATION COMMITTEE

Chairperson

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Dedication

To my parents *Mohammad & Zohra*, who believed in me and gave me the courage.

To my wife *Maryam*, who gives me meaning and wholeness.

To my brothers *Ali, Amir & Iman*, who have always supported me.

Webgroup CZ a.s. Ex. 1015, Page 7 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgments

I would like to thank first and foremost my advisor, Deborah Estrin. Deborah has been far beyond an advisor, she has been a great friend. Deborah's continuous support, constant encouragement, consistent enthusiasm and generosity with her time all combine to make her an exceptional advisor. I specially thank Deborah for showing me the right way to do research.

I am grateful to Mark Handley for his supervision, insightful discussions, comments and criticism over the past three years. Close interaction with Mark has certainly been a key factor in my progress.

I had the privilege of working with exceptional group of people at Information Sciences Institute(ISI) and benefited from many fruitful conversations, and insightful comments on my work. My special thanks go to Joseph Bannister, Bob Braden, Ted Faber, Greg Finn, Ramesh Govindan, John Heidemann, Steven Hotz, Allison Mankin, Rodney Van Meter, Katia Obraczka and Joe Touch. I also need to thank Becky Jordan, Jeanine Yamazaki, Melissa Smith and Sungita Patel for their consistent helpfulness at ISI.

I am also grateful to many colleagues and friends who provided feedback on various parts of this work: Prof. Jon Crowcroft, Prof. Peter Danzig, Jeremy Elson, Sally Floyd, Roger Kermode, Vishwesh Kulkarni, Art Mena, Matt Podolsky, Ahmed Helmy, Pavlin Radoslavov, Scott Shenker, Lorenzo Vicisano, Subramaniam Vincent, David J. Wetherall, Haobo Yu and Prof. Lixia Zhang.

I gratefully acknowledge Prof. Antonio Ortega for participating in my defense committee and many insightful discussions on various aspects of layered encoding and media compression.

I am grateful to all members of the Computer Network and Distributed Systems Lab (dgroup) and Database Lab at USC for their encouragement, feedbacks, and above all

Webgroup CZ a.s. Ex. 1015, Page 8 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for their friendship. My Special thank goes to Ahmed Helmy who has always been a wonderful friend. I would also like to acknowledge Haobo Yu for his major collaboration on design and evaluation of the Multimedia Proxy Caching mechanisms.

I would like to acknowledge Prof. Shahram Ghandeharizadeh for supervising me during my M.S. at USC. I am thankful to Prof. Meshakti and Dr. Shahabi for their consistent encouragement and attention for the past five years.

Lastly but most important, I need to thank my family–my parents, my brothers and specially my wife Maryam for providing such a caring and supportive environment that helped me to reach here. I also need to thank Zahra Javadi for everything she has done for me and my family. I am specially grateful to my brother and best friend Ali for his endless attention and support during the past twenty five years. Above all, I am thankful for Maryam's confidence, support, patience and unconditional love, which in the end, is all that matters.

Webgroup CZ a.s. Ex. 1015, Page 9 of 204

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

Dedication ii

Acknowledgments iii

List Of Tables ix

List Of Figures x

Abstract xii

1 Introduction 1
- 1.1 The Research Problem . . . 3
- 1.2 A Solution . . . 3
- 1.3 Contributions . . . 3
- 1.4 Dissertation Overview . . . 5

2 Related Work 7
- 2.1 Congestion Control - An Overview . . . 7
  - 2.1.1 Two Paradigms . . . 8
  - 2.1.2 Design Principles . . . 10
  - 2.1.3 A Taxonomy for Closed-loop Congestion Control Schemes . . . 10
  - 2.1.4 Previous Works on Congestion Control and Avoidance . . . 14
- 2.2 Congestion Control in the Internet . . . 17
  - 2.2.1 Additive Increase, Multiplicative Decrease . . . 17
  - 2.2.2 TCP Congestion Control . . . 18
  - 2.2.3 TCP-friendly Congestion Control . . . 19
  - 2.2.4 End-to-End Congestion Control: Remaining Challenges . . . 21
- 2.3 Streaming Applications in the Internet: An Overview . . . 22
  - 2.3.1 A Taxonomy for Streaming Applications . . . 23
  - 2.3.2 Internet Video/Audio Streaming: Issues . . . 24
  - 2.3.3 Congestion Control for Streaming Applications . . . 26
  - 2.3.4 Error Control for Streaming Applications . . . 30
  - 2.3.5 Internet Streaming Tools . . . 33

Webgroup CZ a.s. Ex. 1015, Page 10 of 204

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3.6 Integrated & Differentiated Services 34
2.3.7 Smoothing and Selective Dropping 36
2.3.8 Summary of Streaming Applications in the Internet 37
2.4 Proxy Caching for Multimedia Streams 37
2.4.1 Summary of Proxy Caching for Multimedia Streams 39

3 The End-to-end Architecture 41
3.1 Design Principles 42
3.1.1 Social Behavior 42
3.1.2 Being Adaptive 43
3.1.3 Recovery From Loss 43
3.2 Design Space 43
3.2.1 Congestion Control 44
3.2.2 Quality Adaptation 44
3.2.3 Error Control 46
3.3 The Architecture 47
3.4 Generalizing The Architecture 49
3.5 Summary 50

4 The Rate Adaptation Protocol 51
4.1 The RAP Protocol 51
4.1.1 Decision Function 52
4.1.2 Increase/Decrease Algorithm 54
4.1.3 Decision Frequency 55
4.2 Auxiliary Mechanisms 57
4.2.1 Clustered Losses 57
4.2.2 Fine-Grain Rate Adaptation 58
4.3 Self-limiting Issues in RAP 61
4.4 Random Early Drop Gateways 62
4.5 Startup Phase 63
4.6 Simulations 63
4.6.1 Phase Effect 65
4.6.2 Evaluation Methodology 66
4.7 Experiments and Results 67
4.7.1 TCP-friendliness 67
4.7.2 Fine-grain Rate Adaptation 75
4.7.3 Burstiness 75
4.7.4 RED Gateways 78
4.7.5 Intra-protocol Fairness 80
4.7.6 Smoothness of transmission rate 82
4.8 Summary 84

Webgroup CZ a.s. Ex. 1015, Page 11 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5 The Quality Adaptation **85**
5.1 Quality Adaptation Mechanisms 86
5.2 Layered Quality Adaptation 87
5.2.1 Problem Overview 89
5.2.2 Adding a Layer 93
5.2.3 Dropping a Layer 95
5.2.4 Inter-layer Buffer Allocation 96
5.2.5 Optimal Inter-layer Buffer Allocation 97
5.3 Smoothness Constraints 101
5.3.1 Smoothing 101
5.3.2 Buffering Revisited 102
5.4 Buffer Allocation with Smoothing 103
5.4.1 Filling Phase with Smoothing 105
5.4.2 Draining Phase with Smoothing 110
5.5 Simulations 111
5.5.1 Smoothing Factor 112
5.5.2 Responsiveness 112
5.5.3 Efficiency 113
5.6 Summary 118

6 Multimedia Proxy Caching **120**
6.1 The Proxy-based Architecture 123
6.2 Delivery Procedure 124
6.2.1 Relaying on a Cache Miss 124
6.2.2 Pre-fetching on a Cache Hit 125
6.2.3 Challenges and Trade-offs 127
6.3 Replacement Algorithm 132
6.3.1 Replacement Pattern 133
6.3.2 Popularity Function 135
6.3.3 Locking Mechanism 137
6.3.4 Supporting Other Caching Functions 137
6.4 Simulations 139
6.4.1 Evaluation Metrics 139
6.4.2 Simulation Setup 140
6.5 Experiments and Results 142
6.5.1 Prefetching 143
6.5.2 Replacement Algorithm 144
6.5.3 Replacement Algorithm with Realistic Background Traffic 151
6.6 Summary 158

7 Conclusions and Future Work **159**
7.1 Conclusion 159
7.2 Future work 162

Webgroup CZ a.s. Ex. 1015, Page 12 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A
A Simple Model for RAP 166
A.1 Summary 171

Reference List **171**

Webgroup CZ a.s. Ex. 1015, Page 13 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List Of Tables

4.1 Simulation setup for RAP evaluation 65
4.2 RED configuration for RAP evaluation 79

5.1 Parameter list for Quality adaptation 91
5.2 Efficiency of buffer allocation 118
5.3 % drops due to poor buffer distribution 118

6.1 Sample of a popularity table 136

A.1 Simulation setup for RAP evaluation 167

Webgroup CZ a.s. Ex. 1015, Page 14 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List Of Figures

2.1 A typical scenario for deployment of streaming applications over the Internet 25
2.2 Network integrated video encoding 27
2.3 Memory Caching versus Proxy Caching of Multimedia Streams 38

3.1 End-to-end architecture for playback streaming applications in the Internet 47

4.1 ACK-based loss detection in RAP 53
4.2 Effect of per-packet fine-grain rate adaptation on transmission rate of a single RAP flow 60
4.3 Simulation Topology 64
4.4 Comparison of RAP with TCP(Tahoe and Reno) 68
4.5 Comparison of RAP with TCP(NewReno and Sack) 70
4.6 Exploring the inter-protocol fairness across the parameter space 72
4.7 Variation of the Fairness ratio with TCP's congestion window 73
4.8 Exploring the inter-protocol fairness across the parameter space 74
4.9 Bursty RAP with per-ack fine grain adaptation against Sack 76
4.10 Impact of burstiness on RAP's behavior 77
4.11 Mean number of outstanding packets for one RAP and one TCP 78
4.12 Impact of RED on the fairness 79
4.13 Effect of RED configuration of fairness 81
4.14 Intra-protocol fairness among RAP flows 82
4.15 Transmission rate of a sample TCP flow 83
4.16 Transmission rate of a sample RAP flow 83

5.1 Layered encoding with receiver buffering 88
5.2 Filling and draining phase 89
5.3 End-to-end components of quality adaptation mechanism 90
5.4 Critical situation due to a back-off or overestimated slope 98
5.5 The optimal inter-layer buffer distribution 98
5.6 Optimal buffer sharing 99
5.7 Revised draining phase algorithm 102
5.8 Possible double-backoff scenarios 104
5.9 Buffer distributions for k backoffs 105
5.10 Distributions in increasing order of buffering 106

Webgroup CZ a.s. Ex. 1015, Page 15 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.11 Step-by-step buffer filling 107
5.12 First 40 seconds of K_{max}=2 trace 114
5.13 First 5 seconds of K_{max}=2 trace 115
5.14 Effect of K_{max} on buffering and quality 116
5.15 Effect of long-term changes in bandwidth 117

6.1 The end-to-end server/client/proxy architecture 124
6.2 A sample quality adaptive stream in the cache 125
6.3 Delivery of lower bandwidth stream from the cache 126
6.4 Delivery of higher bandwidth stream from the cache 127
6.5 Pre-fetching and Playback Streams 127
6.6 Pre-fetching along time axis 128
6.7 Pre-fetching along quality axis 129
6.8 Pre-fetching pattern in a Win-based approach for window size of 2 segments 130
6.9 pre-fetching mechanism 130
6.10 Replacement priority within a cached stream 134
6.11 Vertical pattern of replacement 138
6.12 Snake-like pattern for replacement 138
6.13 Average quality and layer breaks for a cached stream 140
6.14 Simulation topology 141
6.15 Quality improvement due to prefetching. 144
6.16 Effect of popularity on cache replacement, *The most popular stream.* . . 147
6.17 Effect of popularity on cache replacement, *The least popular stream.* . . 147
6.18 Effect of client bandwidth on cache replacement, *The most popular stream.* 149
6.19 Effect of client bandwidth on cache replacement, *The least popular stream.* 149
6.20 General case of cache replacement, *The most popular stream.* 151
6.21 General case of cache replacement, *The least popular stream.* 151
6.22 Effect of popularity on cache replacement with realistic background traffic, *The most popular stream.* 153
6.23 Effect of popularity on cache replacement with realistic background traffic, *The least popular stream.* 153
6.24 Effect of client bandwidth on cache replacement with realistic background traffic, *The most popular stream.* 155
6.25 Effect of client bandwidth on cache replacement with realistic background traffic, *The least popular stream.* 155
6.26 General case of cache replacement with realistic background traffic, *The most popular stream.* 157
6.27 General case of cache replacement with realistic background traffic, *The least popular stream.* 157

A.1 Transmission rate of a single RAP flow 166

Webgroup CZ a.s. Ex. 1015, Page 16 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

Lack of QoS support in the Internet has not prevented rapid growth of streaming applications(audio and video). However many of these applications do not perform congestion control effectively. Thus there is significant concern about the effects on co-existing well-behaved traffic and the potential for congestion collapse. In addition, many such applications are unable to perform quality adaptation on-the-fly as available bandwidth changes during a session. The problem is one of adapting the compression without requiring video-servers to re-encode the data, and fitting the resulting stream into the rapidly varying available bandwidth. At the same time, rapid fluctuations in quality will be disturbing to the users and should be avoided.

In this dissertation, we will design and evaluate an end-to-end architecture suited for unicast playback of layered-encoded stored multimedia streams over the Internet. Our architecture reconciles congestion control and quality adaptation which occur on different timescales. It exhibits a TCP-friendly behavior by adopting the Rate Adaptation Protocol(RAP) for end-to-end congestion control. Additionally, it employs a layered framework for quality adaptation to maximize perceptual quality while minimizing rapid, disturbing changes in the quality of the delivered stream as available bandwidth changes. Furthermore, the quality adaptation mechanism provides a tuning parameter that allows the server to trade short-term improvement for long-term smoothing of delivered quality.

The quality of delivered streams in the end-to-end architecture is limited by the bottleneck bandwidth along the path between the server and the client. To overcome this limitation, we extend our end-to-end architecture by adding multimedia proxy caches. Proxy caches perfectly complement our end-to-end architecture. We describe a fine-grain replacement algorithm for proxy caching mechanism of layered-encoded multimedia streams as well as a pre-fetching scheme to smooth out the variations in quality of a

Webgroup CZ a.s. Ex. 1015, Page 17 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cached stream. Interaction between the pre-fetching and replacement algorithms results in the state of the cache converging to the optimal state such that the quality of a cached stream is proportional to its popularity, and the variations in quality of a cached stream are inversely proportional to its popularity. Thus the proxy can maximize the delivered quality of popular streams to interested clients.

Webgroup CZ a.s. Ex. 1015, Page 18 of 204

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

The Internet has been experiencing explosive growth of audio and video streaming. Such applications are *delay-sensitive*, *semi-reliable* and *rate-based*. Thus they require isochronous processing and quality-of-service (QoS) from the end-to-end point of view. However, today's Internet does not attempt to guarantee an upper bound on end-to-end delay or a lower bound on available bandwidth. As a result, the quality of delivered service to realtime applications is neither controllable nor predictable. Lack of support for QoS has not prevented rapid growth of realtime streaming applications. This growth is expected to continue, and multimedia traffic will form a higher portion of the Internet load. Thus the overall behavior of these applications will have a significant impact on the Internet traffic.

Most current applications involve web-based audio and video playback[58, 93] where stored video is streamed from the server to a client upon request. Examples include continuous media servers, digital libraries, distance learning, shopping and entertainment services. These playback clients can afford to slightly delay the playback point and buffer some data to partially absorb variation of the network bandwidth and end-to-end delay.

Since the Internet is a shared environment and does not currently micro-manage utilization of its resources, end systems are expected to be cooperative by reacting to congestion and adapting their transmission rates properly and promptly[41]. Another important issue is inter-protocol fairness: the rate adjustment should result in a fair share of bandwidth for all the flows that coexist along the same path. Applications that adapt their transmission rates properly and promptly are known as "good network citizens". Deploying end-to-end congestion control also results in higher overall utilization of the network

Webgroup CZ a.s. Ex. 1015, Page 19 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and improves inter-protocol fairness. A congestion control mechanism determines the available bandwidth based on the state of the network, and the application should then use this bandwidth efficiently to maximize the quality of the delivered service to the user. Since a dominant portion of today's Internet traffic is TCP-based, it is crucial that real-time streams perform TCP-friendly congestion control. By this, we mean that a realtime flow should obtain approximately the same average bandwidth over the timescale of a session as a TCP flow along the same path under the same conditions of delay and packet loss [80].

Currently, many of the commercial streaming applications lack end-to-end congestion control or are not TCP-friendly. This is mainly because stored video has an intrinsic transmission rate. These rate-based applications either transmit data with a near-constant rate or loosely adjust their transmission rates on long timescales since the required rate adaptation for effective congestion control is not compatible with their nature. Large scale deployment of these applications could result in severe inter-protocol unfairness against TCP-based traffic and possibly even congestion collapse. One solution would be to make realtime flows use reservations or differentiated service. However, even if such services become widely available, there will remain a significant group of users who are interested in using realtime applications at low cost. Even in a network that supports reservation, different users that fall into the same class of service or share a reservation still interact as in best effort networks. Thus we believe that congestion control for these applications is critical for the health of the Internet.

In a nutshell, to support streaming application over the Internet, one needs to address the following two conflicting requirements:

- *Network requirement*: All end-systems should perform congestion control. This implies transmission rate of end-systems vary randomly and potentially over a wide range.

- *Application requirement*: Streaming applications require sustained transmission/consumption rate to deliver acceptable and stable quality.

Webgroup CZ a.s. Ex. 1015, Page 20 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.1 The Research Problem

The precise statement of our thesis research problem is the following: *"How can we support streaming applications in large scale over the Internet such that":*

- *These applications exhibit a "network-friendly" and in particular "TCP-friendly" behavior and at the same time,*
- *These applications can deliver streams with high and stable quality.*

1.2 A Solution

We propose a new end-to-end architecture to support streaming applications in best-effort networks and in particular in the Internet. We separate network-dependent congestion control from application-dependent quality adaptation and error control. Towards this end, we break the problem down into two sub-problems:

- 1. Design and evaluation of a TCP-friendly congestion control mechanism suited for streaming applications.
- 2. Developing a mechanism for delivery of acceptable and stable quality streams while performing congestion control.

The quality of delivered streams in the end-to-end architecture is limited by the bottleneck bandwidth along the path between the server and the client. To improve the delivered quality, we extend our end-to-end architecture by adding multimedia proxy caches. Thus the third sub-problem that we address is:

- 3. Design and evaluation of multimedia proxy caching mechanism for streaming applications over the Internet.

1.3 Contributions

In our attempt to design and evaluate an end-to-end architecture to support streaming applications in best-effort networks:

Webgroup CZ a.s. Ex. 1015, Page 21 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- **End-to-end Architecture:** We have designed an end-to-end architecture that reconciles congestion control, quality adaptation and error control as three key components for any streaming applications in the Internet. We argue that this architecture can be viewed as a generic architecture for streaming applications.

- **End-to-end Congestion Control:** We have designed, developed and extensively evaluated the rate adaptation protocol(RAP) to be well-behaved and achieve inter-protocol fairness in general and exhibit TCP-friendly behavior in particular. We have only emulated those mechanisms from TCP's congestion control that are known as strengths of TCP and avoid those issues that might cause performance problems.

  We have presented a methodology for simulations to limit the inter-dependency among different variables. This methodology allows us to recognize an effect that is caused by TCP's performance problem from those phenomena that are due to coexisting with RAP flows.

- **Layered Quality Adaptation:** We have designed, developed and evaluated a quality adaptation mechanism using layered-encoded streams in the context of unicast congestion control. This quality adaptation mechanism adds and drops layers of the video stream to perform long-term coarse-grain adaptation, while using a TCP-friendly congestion control mechanism to react to congestion on very short timescales. The mismatches between the two timescales are absorbed using buffering at the receiver. We present an efficient scheme for the distribution of buffering among the active layers. Our scheme allows the server to trade short-term improvement for long-term smoothing of quality. We discuss the issues involved in implementing and tuning such a mechanism.

- **Extending the End-to-end Architecture:** We have added multimedia proxy caches to the end-to-end architecture. The proxy-based architecture is able to improve the quality of a popular stream despite the presence of a bandwidth bottleneck between the server and a client. Furthermore, multimedia proxy caches significantly reduce startup delay, facilitate more interactive VCR-functionalities, and reduce load on the server and network.

Webgroup CZ a.s. Ex. 1015, Page 22 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- **Multimedia Proxy Caching:** We have designed, developed and evaluated a novel proxy caching mechanism for layered encoded multimedia streams. We presented a pre-fetching mechanism to support higher quality cached streams during subsequent playbacks and improve the quality of the cached stream with its popularity. We also devised a fine-grain replacement algorithm suited for layered-encoded streams. Our simulation results show that the interaction between the replacement algorithm and pre-fetching mechanism causes the state of the cache to converge to an efficient state. Thus the proxy can effectively hide low bandwidth paths to the original server from interested clients after serving several requests for a stream.

1.4 Dissertation Overview

This dissertation is organized as follows:

Chapter 2 reviews related work and addresses some of the differences between previous work and the work presented in this dissertation.

Chapter 3 provides a high level architectural view for the design of streaming applications in the Internet. Addressing design principles for Internet applications leads us to identify congestion control, quality adaptation and error control as three key components for any streaming application in the Internet. We briefly explore the design for each one of these components and select an appropriate mechanism from its corresponding space. Then we compose the three key components into a coherent architecture and describe the interaction among these components. We also argue that the architecture can be viewed as a generic architecture for streaming applications as long as the different modules are properly integrated.

Chapter 4 presents the Rate Adaptation Protocol(RAP) as an end-to-end TCP-friendly congestion control mechanism suited for unicast delivery of multimedia streams as well as other semi-reliable rate-based applications. We evaluate RAP through extensive simulation, and conclude that bandwidth is usually evenly shared between TCP and RAP traffic. Unfairness to TCP traffic is directly determined by how TCP diverges from the Additive Increase, Multiplicative Decrease(AIMD) algorithm. Basic RAP behaves in a

Webgroup CZ a.s. Ex. 1015, Page 23 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TCP-friendly fashion in a wide range of likely conditions, but we also devised a fine-grain rate adaptation mechanism to extend this range further. Finally, we show that deploying Random Early Drop(RED) queue management can result in a fairness between TCP and RAP traffic.

Chapter 5 presents a quality adaptation mechanism for using layered encoded streams in the context of unicast congestion control. This quality adaptation mechanism adds and drops layers of the video stream to perform long-term coarse-grain adaptation, while using a TCP-friendly congestion control mechanism(i.e. RAP) to react to congestion on very short timescales. The mismatches between the two timescales are absorbed using buffering at the receiver. We present an efficient scheme for the distribution of buffering among the active layers. Our scheme allows the server to control the level of smoothing, i.e. the server can trade short-term improvement for long-term smoothing of quality. We evaluate the quality adaptation mechanism using simulations.

Chapter 6 addresses the inherent limitation on delivered quality of the end-to-end architecture. Then we extend the architecture by adding proxy caches and describe a multimedia proxy caching mechanism suited for layered encoded multimedia streams in the Internet to maximize the delivered quality of popular streams to interested clients.. We present a pre-fetching mechanism to support higher quality cached streams during subsequent playbacks and improve the quality of the cached stream with its popularity. We exploit inherent properties of multimedia streams to extend the semantics of popularity and capture both level of interest among clients and usefulness of a layer in the cache. We devise a fine-grain replacement algorithm suited for layered-encoded streams. Our simulation results show that the interaction between the replacement algorithm and pre-fetching mechanism causes the state of the cache to converge to an efficient state such that the quality of a cached stream is proportional to its popularity, and the variations in quality of a cached stream are inversely proportional to its popularity. This implies that after serving several requests for a stream, the proxy can effectively hide low bandwidth paths to the original server from interested clients.

Chapter 7 concludes the dissertation and addresses some of our future plans.

Webgroup CZ a.s. Ex. 1015, Page 24 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Related Work

Different aspects of streaming applications have been extensively studied during the last decade. Since we can not cover such a wide spectrum of work, this chapter only reviews related work that are more relevant to delivery of multimedia streams over best-effort networks. In particular the following sections survey related work in each of the following areas:

- Congestion Control
- Congestion Control in the Internet
- Streaming Applications in the Internet
- Proxy Caching for Multimedia Streams

2.1 Congestion Control - An Overview

When a source starts transmitting data, the available bandwidth between the source and a destination in the network is not always known a priori. If the transmission rate is too high, it results in congestion and subsequently packet loss, whereas low transmission rate leaves the connection underutilized.

Congestion control refers to a mechanism that enables the source to match its transmission rate to the current available bandwidth. The mechanism should scale well with the number of sources without creating substantial overhead for the network. It should

Webgroup CZ a.s. Ex. 1015, Page 25 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

also limit usage of resources(i.e. bandwidth and buffer) such that none of the existing flow starves. Furthermore, the mechanism should be stable-that means once the situation is static and there is no change in available bandwidth, the transmission rate of active sources should converge to an equilibrium.

Congestion control usually refers to a mechanism that enables the network to recover after a congestion event, whereas *congestion avoidance* attempts to proactively prevent congestion. There is also a clear distinction between *flow control* and congestion control. Flow control is a mechanism that prevents a source from over running a receiver's resources whereas congestion control prevents the source from over running network resources. Although both mechanisms usually operate simultaneously, only one of them limits source's transmission rate.

The required functionality to control congestion can be implemented at end-systems, in the network, or a combination of the two. Moreover, there are various ways for detecting, signaling and reacting to congestion. This design space introduces several interesting trade-offs in design of a congestion control mechanism.

2.1.1 Two Paradigms

There are two basic resource management paradigms to regulate the offered load to a network: Open-loop and Closed-loop.

2.1.1.1 Open-loop Congestion Control

In this paradigm, a source describes its traffic to the network with a few parameters. The network reserves some resources along the path during call establishment. If resources are not available, the request for a new connection is rejected. The source should abide the flow specification and shape its traffic to stay within that profile. The main challenge is to present the traffic pattern to the network adequately so that it provides sufficient information for the network to perform resource management effectively.

This paradigm requires that resource management and admission control mechanisms be implemented in the network. In essence, in this paradigm the burden of traffic management is on the network. The congestion control mechanism becomes rate shaping at

Webgroup CZ a.s. Ex. 1015, Page 26 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the end-system based on a pre-established profile. This approach is well suited to *circuit-switch* or *reservation-based* networks. The main disadvantage of this approach is that resources allocated to a connection might remain unutilized by the requested source while the network rejects requests from other clients. This reduces the overall utilization of the network. Furthermore, this approach raises scaling issues because each intermediate switch requires per-flow state.

Given a reasonable understanding of the offered traffic on the network, a best-effort network can be properly provisioned to meet the demand most of the time. However, it is easier to study the behavior of traffic in a network with homogeneous flows(e.g. telephone networks) and provision sufficient resources. In contrast, it is challenging to characterize the behavior of heterogeneous traffic well enough to achieve efficient and sufficient provisioning. Furthermore, the ratio of peak to average load on any particular link is quite high, thus provisioning for peak load is not economical. Therefore, even in the presence of a well-provisioned network, congestion will still occur and congestion recovery mechanisms are needed.

2.1.1.2 Closed-loop Congestion Control

In this paradigm, the source receives *feedback* periodically about the state of the network and should react to that feedback by adjusting its transmission rate accordingly using a rate adaptation *strategy*. The network might provide *explicit* feedback that carries some information about the state of the network back to the source. Alternatively, the source can exploit *implicit* feedback by inferring the state of the network from a change in behavior of the channel such as change in RTT or loss rate. The rate adaptation strategy adjusts the transmission rate based on the accuracy of the feedback signal. The more information the feedback signal provides, the more effectively the congestion control mechanism behaves. This paradigm is well suited to *packet-switch* networks such as the Internet.

Webgroup CZ a.s. Ex. 1015, Page 27 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A hybrid paradigm is also feasible where a group of end-systems reserve a channel with a specific amount of resources and share the resources among themselves in a best-effort fashion. Today's Internet does not widely provide reservation or resource management mechanism. Thus we focus mainly on the closed-loop paradigms that are applicable to the Internet and other shared channels.

2.1.2 Design Principles

Ideally, a congestion control mechanism for a heterogeneous public network should follow several design principles[77]:

1. The network should provide isolation for individual flows. In other words, behavior of a single flow should not affect the quality of service given to other flows.

2. The network should provide sufficient feedback to enable users to effectively utilize their share of resources. If the feedback signal does not have sufficient accuracy, It may take a long time for the network to reach equilibrium or it may not stabilize at all.

3. The network should not solely rely on the behavior of the end-systems to be well-behaved. Instead, the network should provide sufficient isolation for individual flows to protect them against potential aggressive flows.

2.1.3 A Taxonomy for Closed-loop Congestion Control Schemes

Closed-loop congestion control has three orthogonal components[77]:

1. Network Resource Management

   (a) Bandwidth Management, i.e. Scheduling

   (b) Buffer Management

2. Feedback

3. End-system adjustment, i.e. rate adaptation strategy

Webgroup CZ a.s. Ex. 1015, Page 28 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The network monitors utilization of its resources (i.e. buffer and bandwidth) and sends a feedback signal to end-systems when congestion is forming. The end-system should react to the feedback signal appropriately to reduce load on the network and relieve the congestion. The deployed mechanism for each component has implications for other components and their interactions. We briefly address the role of each component in the following subsections:

2.1.3.1 Network Resource Management

The network should actively manage utilization of its resources. Thus there needs to be both a packet scheduling and a buffer management mechanism for effective control of these resources..

Scheduling: Scheduling is the most direct control over allocated bandwidth to individual flows since it controls the order in which individual packets are served. Two of the most popular scheduling algorithms are: 1. First-in-first-out(FIFO) and 2. Weighted Fair Queuing (WFQ). FIFO is widely used due to its simplicity. With the FIFO algorithm, all packets experience the same queuing delay on average. However, it does not provide isolation for individual flows, thus a mis-behaved flow can affect delivered service to other flows. It could also result in unfairness among well-behaved flows[42, 34] and packet clumping[77].

WFQ algorithms try to allocate the available bandwidth evenly or based on the specified weight among all active flows[34]. WFQ provides a high level of isolation against mis-behaved flows and reduces the effect of packet clumping. Since implementation of WFQ requires per-flow state at each switch along the path, there are some scalability concerns for deploying it over wide area networks. Thus approximations have been proposed which trade the amount of required state with obtained fairness [122, 123].

Scheduling alone can not prevent substantial packet loss. Because of sudden change in network load and the bursty nature of traffic, the switch is forced to drop any packets that can not be sent. Adequate buffer space enables the switch to absorb short-term extra-load

Webgroup CZ a.s. Ex. 1015, Page 29 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

without dropping large numbers of packets. Thus the switch requires a buffer management scheme to control usage of the buffer space as we describe next.

Buffer Management: Buffer management is complementary to packet scheduling. The buffer management scheme should provide sufficient buffering to store the excess offered load of a "well-behaved" flow while the congestion signal is being delivered to the end-system. This feedback loop usually takes one RTT for closed loop schemes. The amount of buffer space should be large enough to keep the pipe full and results in high link utilization. However, if the only congestion signal is packet loss due to buffer overflow, very large buffers will result in unnecessarily long queuing delays.

Two of the most popular buffer management mechanisms are 1. Shared buffer pool and 2. Per-flow allocation; These are compatible with FIFO and WFQ scheduling, respectively. Note that buffer management and scheduling mechanisms must be compatible. For example, using shared buffer pool with WFQ is meaning less because shared buffer space does not provide isolation for individual flows.

2.1.3.2 Feedback

Feedback signals the end-system about the state of the network and should provide sufficient information for the end-system to adjust its transmission rate properly. The more information is provided in the feedback, the more accurate and effective the rate adaptation can be.

Implicit versus Explicit Feedback: As we mention earlier, the source might use performance measurement to the infer state of the connection and use this measurement as an implicit feedback. For example the Slow-Start mechanism in TCP[65] and Tri-S[133] consider packet loss as an implicit congestion signal; the Packet-Pair protocol[72] exploits inter-ack-gap to estimate bottleneck bandwidth; NETBLT[27] compares observed throughput with expected transmission rate to determine appropriate transmission rate; and Delay-based Congestion Control[66] and TCP Vegas [2] adjust measure variation of RTT as implicit congestion feedback.

Webgroup CZ a.s. Ex. 1015, Page 30 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

With explicit feedback, the network sends an explicit control message to the source to inform the end system about the state of the network. Some well known examples are: ICMP[102], DEC-bit[103, 104] and Explicit Congestion Notification(ECN)[105]. The implicit scheme has less overhead on the network but the explicit scheme is more accurate.

2.1.3.3 End-system Adjustment

End-systems are expected to be cooperative and react to the network feedback properly and promptly by reducing their transmission rate. The rate adaptation strategy closely depends on the form and accuracy of the feedback signal. For example, using packet loss as a binary feedback for congestion does not specify the level of congestion, thus end-systems should be conservative and decrease their rate exponentially[65]. Furthermore, packet loss may occur due to packet corruption instead of congestion, but end-system can not differentiate between these two scenarios.

It is useful for end-system to know about the deployed scheduling algorithm (e.g. WFQ or FIFO) in network switches in order to interpret and rely on the feedback appropriately. However, in a heterogeneous network such as the Internet, end systems can not make any specific assumptions about underlying scheduling algorithm in various intermediate switches. Since the core of the network(i.e. Scheduling, Buffer management and Feedback) is relatively stable, end-system adjustment is the most accessible component of congestion control control loop. Thus end-system adjustment is the component where significant changes are possible and desirable.

As we mentioned earlier, the network should not assume that all end-systems employ the same rate adaptation strategy. However, it should provide sufficient isolation against unresponsive flows, or at least limit their negative effect on other co-existing flows.

End-system adjustment can be classified based on:

1. The manner in which transmission rate is adjusted and

2. The point where rate adaptation functionality is implemented as follows:

Webgroup CZ a.s. Ex. 1015, Page 31 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Window versus Rate-based Congestion Control: There are two ways to control the transmission rate of an end-system[47]: 1. Window-based or 2. Rate-based. In a window-based scheme, a source directly controls number of packets in transit by adjusting a window that is an upper bound for number of packets in flight. Thus, the gap between consecutive packets may vary. In a rate-based scheme, the source directly adjusts its transmission rate by controlling the gap between every two consecutive packets, called the inter-packet-gap.

Window-based schemes are more popular because they are easier to implement. Rate-based schemes require a fine-grain timer that is often expensive on typical end-systems. Window-based scheme are also known to be self-limiting since they effectively control the number of packets in the pipe.

End-to-End versus Hop-by-Hop: The functionality of congestion control can be placed at different points. One may deploy a congestion control mechanism between every two elements(or hops) of the network. This is called hop-by-hop congestion control[92]. Alternatively, we can deploy congestion control only between two end points. While the hop-by-hop approach has a smaller delay and is typically more effective, it adds to the complexity of intermediate nodes in the network and assume more homogeneity.

It is worth clarifying that end-to-end congestion control is sometimes referred to as flow control. Although they operate in parallel, they serve different purposes.

2.1.4 Previous Works on Congestion Control and Avoidance

As we described in the previous section, any congestion control mechanism must address all three components or make specific assumptions about them. Most of the previous work has focused on feedback and end-system adjustment because they are more accessible for changes whereas Scheduling and Buffer management are likely to be slow to change.

In this section, we briefly review some of the previous works on different components of congestion control:

Webgroup CZ a.s. Ex. 1015, Page 32 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1.4.1 Source Quench

Source quench [102] is one of the earliest router-based mechanism for congestion control. An overwhelmed gateway or receiver sends a source quench Internet Control Message Protocol (ICMP) message to the source. Upon arrival of such a message, the source cuts back its transmission rate and then gradually increases the rate.

2.1.4.2 DEC-bit

DEC-bit[103, 104] employed a window-based rate adaptation strategy and used a bit in the packet header as explicit feedback. Intermediate routers set a bit in the header of all packets when congestion occured (i.e. when the sustained queue length exceeded a threshold). Receivers copy the bit in the ACK packet to report it back to the source. Based on the number of received ACK packets with a set bit, the source decided how to adjust its transmission rate.

2.1.4.3 Random Early Drop(RED)

Random Drop[81, 53] is another congestion avoidance and control mechanism for FIFO routers. The idea is that a congested router can randomly drop packets from its queue instead of loss due to buffer overflow. The intuition is that a flow with a larger number of packets and consequently more bandwidth share is more likely to observe packet loss. The packet loss is intended to serve as feedback to those end-systems whose traffic has a higher share in congestion. Random drop is also helpful for congestion avoidance because it can signal end-systems before queue length exceeds a certain threshold (i.e. congestion becomes too bad).

Previous work on router-based mechanism resulted in Random Early Detection(RED)[43] for congestion detection. The idea is inspired by the Random Drop mechanism. RED is an active queue management that can replace traditional DropTail queuing scheme. A well-configured RED switch with sufficient buffer space is able to effectively absorb short-lived bursts while avoiding buffer over flow(i.e. without substantial packet loss). RED can either drop or mark packets. In the latter case, RED can be combined with Explicit

Webgroup CZ a.s. Ex. 1015, Page 33 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Congestion Notification(ECN)[105] where a bit is set in the header of packets to signal congestion to the source (similar to DEC-bit scheme). RED also decreases the occurrence of phase effects resulting from from synchronization of different flows[42].

The main challenge is to configure a RED switch properly since appropriate thresholds depend on the behavior of background traffic that is not usually known a priori.

2.1.4.4 NETBLT

NETBLT[27] was the first widely known end-to-end rate-based congestion control mechanism. NETBLT was intended for bulk data transfer. The main contribution of NETBLT to congestion control is separation of congestion control from error control. NETBLT periodically compares the observed throughput with the transmission rate to determine maximum achievable throughput and exploit the result as implicit feedback.

2.1.4.5 Packet Pair

Packet Pair[72, 73] is a rate-based congestion control mechanism that relies on implicit feedback. However it assumes fair-queue scheduling in the network. The idea is the following: if two back-to-back packets are sent and acknowledged, the gap between ACK packets shows the bottleneck bandwidth. Although the Internet does not support fair-queuing, schemes similar to packet-pair can be used to probe a path and obtain an estimate of available bandwidth from the end-system[3].

2.1.4.6 CARD

CARD is an end-to-end window-based congestion avoidance mechanism that uses variations in RTT as implicit feedback[66]. Normalized gradient of RTT determines the direction of window adjustment using an Additive Increase, Multiplicative Decrease algorithm. The idea is to keep the operating point at the peak of the Power-Load curve. The problem with this approach is that in a shared network with FIFO scheduling, the RTT signal is too noisy because of the random behavior of background traffic(e.g. many short-lived bursty

Webgroup CZ a.s. Ex. 1015, Page 34 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TCP connection). Thus its normalized gradient does not necessarily imply the right direction for window adjustment. CARD has inspired other works(e.g.[30]) on congestion avoidance.

2.2 Congestion Control in the Internet

Many of the proposed schemes for congestion control can not be deployed over today's Internet since the required scheduling or feedback component are not implemented. Most of the current routers in the Internet implement FIFO scheduling and DropTail queuing. Thus end-to-end congestion control is the only feasible approach and end-system can only rely on packet loss as an implicit signal for congestion. Thus proper end-system adjustment is crucial for stability of the network[41]. However, there is no clear incentive for end-systems to be well-behaved and obey a congestion control rate limit. As a result, there needs to be a mechanism to identify and punish aggressive flows that are not responding to packet loss [41]. To avoid packet loss for well-behaved flows, the Explicit Congestion Notification (ECN) mechanism for the Internet has been proposed[105]. The main idea is to signal the end-system by marking packets instead of dropping them.

2.2.1 Additive Increase, Multiplicative Decrease

In the absence of any coordination among end-systems and lack of isolation for individual flows, it is extremely challenging to find an adaptation strategy that converges to a *fair* share of resource for each flow. Furthermore, there is no well accepted criteria for fairness, i.e. fair share is not clearly defined. For example assuming two flows with different RTT share a bottleneck, if both flows obtain 50% of bottleneck bandwidth, the flow with longer RTT uses more resources in the network since its RTT-Bandwidth product is larger. Alternatively, if the flow with shorter RTT obtains proportionally higher share of bandwidth, both flows use the same amount of network resources but bottleneck bandwidth is not shared in a fair manner. The only promising end-system adjustment strategy that efficiently converges to a fair state is Additive Increase, Multiplicative Decrease(AIMD) [25].

Webgroup CZ a.s. Ex. 1015, Page 35 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

AIMD has been used in many congestion control protocol. in particular TCP employs an AIMD window adjustment strategy as we describe next.

2.2.2 TCP Congestion Control

TCP is certainly the most popular transport protocol and different aspects of it have been studied extensively during last decade. TCP performs end-to-end window-based congestion control using an Additive Increase, Multiplicative Decrease rate adaptation strategy. It only relies on packet loss as congestion signal. Since TCP treats any packet loss as a signal for congestion, performance of TCP could be substantially degraded over lousy links or wireless channels where random losses occur due to reasons other than congestion(e.g. corruption). TCP's performance have been extensively studied [34, 119, 140, 40, 74]

TCP has been evolved during last decade and various modifications were proposed to mainly improve its congestion control mechanism. However, the core of congestion control mechanism(i.e. AIMD) remains intact and most of the refinements improved efficiency of the error control mechanism. Van Jacobson re-engineered TCP and introduced the slow-start and congestion control avoidance[65]. This version of TCP is known as *Tahoe*. TCP tightly couples Congestion and Error control mechanisms, i.e. Packet loss signals congestion, triggers a backoff in transmission rate and retransmission of the lost packet. The main challenge is to detect losses as soon as possible. TCP Tahoe deploys a timeout mechanism for loss detection. This may result in a long delay and decreases the throughput. TCP *Reno* incorporates a fast-retransmit mechanism to use duplicate acks as a signal for congestion and improve the efficiency of error control.

Selective Acknowledgment(*SACK*) is a recent revision of TCP that further improves performance of TCP's error control mechanism. In that

1. it prevents unnecessary retransmission,

2. it prevents TCP from losing its ACK-clocking.

The most recent enhancement in TCP's performance is FACK TCP[84].

Webgroup CZ a.s. Ex. 1015, Page 36 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.2.1 TCP Vegas

TCP-Vegas[18] is another revision of TCP. The main contribution of Vegas is improvement of the congestion avoidance mechanism in TCP. TCP-Vegas compares observed throughput with expected throughput and adjusts its transmission rate accordingly before congestion occurs. As a result, it reduces the number of losses due to congestion as well[18, 2]. The main concern is weather TCP-Vegas can co-exist with TCP Reno and Tahoe. Congestion avoidance mechanism in Vegas slows down its transmission rate before congestion occurs while Reno and Tahoe increase the rate until they experience loss. Thus Vegas is likely to achieve less share of resources when it co-exist with Reno or Tahoe.

One of the main concerns for any modifications in TCP is the need for backward compatibility. Various implementations of TCP are widely deployed over today's Internet. It is crucial to ensure any modification in TCP does not affect the correctness and performance of connections between the modified TCP and other existing implementations in either direction.

2.2.3 TCP-friendly Congestion Control

Since the Internet does not provide sufficient isolation for individual flows, it is crucial for new transport protocol to incorporate proper end-adjustment strategy for the sake of congestion control. Furthermore, a dominant portion of today's Internet traffic consists of a variety of well-behaved TCP-based flows(e.g. Web, FTP, email) [26]. Thus *TCP-friendly* congestion control is prefered otherwise in the absence of network isolation non-congestion-controlled flows will "shout out" TCP traffic.

It is not clear how closely new end-adjustment strategies should mimic TCP's behavior to be considered TCP-friendly. Intuitively, if a new end-adjustment strategy obtains in average the same bandwidth as a TCP flow along the same path, it can be considered TCP-friendly although short-term transmission rate might be different from TCP. Thus the main question is "over what time-scale the transmission rate is averaged and compared?".

Webgroup CZ a.s. Ex. 1015, Page 37 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Clearly, the longer the time-scale is, the less similar to TCP the short-term behavior could be.

Note that TCP is a moving target and its behavior may substantially change with network parameters(e.g. RTT, loss rate, available bandwidth). For example, under heavy load TCP's congestion control mechanism diverges from AIMD and starts a timer-driven mode when only one packet sent per RTT is sent. TCP might also become bursty in the absence of sufficient statistical multiplexing over long-delay paths. Thus designing TCP-friendly congestion control over a wide range of network condition is challenging.

Works on TCP-friendly congestion control can be divided into two main categories:

1. *Modeling TCP*: Mahdavi and Floyd proposed the idea of modeling TCP's and presented a simple equation to estimate TCP's transmission rate as a function of loss rate and RTT[80]. The idea has been further elaborated by other researchers and more accurate models have been proposed by Mathis et al.[85] and Padhye et al.[97]. Evaluations have revealed that these models closely emulate TCP's behavior over a limited range of loss rate and RTT.

   To design a congestion control mechanism using these equations, the source should periodically(e.g. once per RTT) estimates TCP's transmission rate based on observed loss and RTT. Then it compares its current transmission rate with the calculated rate of a TCP flow under these condition. The remaining question is "what is the appropriate increase/decrease strategy to match the current transmission rate with estimated TCP rate?". For example the source can perform exponential back off or simply drop the rate to the estimated TCP rate, if its current rate is higher than the estimated TCP rate. Work in [96] presents a complete congestion control mechanism using TCP-equations. Initial evaluation showed that such an end-adjustment strategy could be unstable and result in an oscillatory behavior[49].

2. *Additive Increase, Multiplicative Decrease*: Given the properties of AIMD algorithm, deploying an AIMD strategy should result in a TCP-friendly behavior as long as TCP remains in its AIMD regime, e.g. [121]. Most of these scheme do not mimic

Webgroup CZ a.s. Ex. 1015, Page 38 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TCP's behavior in timer-driven mode. If the timer-driven mode in TCP play a major role in stability of the network under heavy load, then all other end-adjustment strategies should incorporate such a conservative mechanism as well.

Performing TCP-friendly congestion control could result in wide and seemingly-random variations in bandwidth. Thus it is extremely challenging for rate-based applications to sustain a particular average rate(e.g. streaming applications) to perform TCP-friendly congestion control. This is in fact the main question that we try to address in this dissertation.

2.2.4 End-to-End Congestion Control: Remaining Challenges

Despite extensive work on end-to-end congestion control, there are still several challenging problems that require further investigation:

1. **Achieving Fairness:** In order to achieve appropriate fairness, the required machinery has to be implemented in the network[117], i.e. Deploying end-to-end congestion control is not sufficient to achieve fairness.

2. **Startup Behavior:** End-to-end protocols aggressively increase their transmission rate during the startup phase to probe availability of bandwidth(e.g. Slow-start mechanism in TCP). This aggressive increase in bandwidth could result in potentially large overshoot over available bandwidth. While more conservative increase of the rate can reduce this problem, it increases duration of the startup phase. It does not seem feasible to resolve this problem without assistance from the network.

3. **Congestion Control for Short-lived Flows:** Many of the flows in today's Internet are short-lived TCP flows (e.g. Web transactions), known as mice, that end before they finish their slow-start phase. As a result, they do not perform congestion control effectively. Because of the aggressive behavior of these flows, they manage to obtain a higher share of bandwidth over their life time scale than co-existing long-lived flows along the same path. This problem can be addressed from two orthogonal perspectives:

Webgroup CZ a.s. Ex. 1015, Page 39 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(a) Intermediate switches can treat long and short-lived stream differently. The obvious question is how to identify these two groups.

(b) End-systems can multiplex all the short-lived flow between two end points [9].

4. **Smoother Increase/Decrease Strategy:** As we mentioned earlier AIMD seems to be the most promising rate adaptation strategy for end-to-end congestion control. However, AIMD can result in potentially wide and seemingly random variations in bandwidth. It would be more desirable to design a strategy that exhibits smoother variations in transmission rate but still has most of the good properties of AIMD.

2.3 Streaming Applications in the Internet: An Overview

Many streaming applications require real-time performance guarantees such as bounded delay and minimum bandwidth. These performance requirements are known as "end-to-end quality-of-service". The current Internet does not guarantee any upper bound on delay and any lower bound on available bandwidth[118]. Thus the quality of a delivered stream could change with network load.

Experiments with audio in the early days of the Internet demonstrated that interactive audio is feasible[29]. However video was less practical due to higher bandwidth requirement. Advances in high speed networking and media compression along with high performance workstation all together made video applications both feasible and economical. Growing interest to the Internet and proliferation of the World Wide Web with relatively rich multimedia content further increased the demand for streaming applications. This in turn motivates further research on streaming applications over the Internet. This has inspired interesting areas of research in different aspect of Multimedia networking. While some researchers are working to integrate QoS requirements to the existing architecture, others are focusing on improvement of delivered streams over current Internet. Instead of covering the entire spectrum, we address those efforts that are more relevant to this dissertation as follows:

- A taxonomy for streaming applications

Webgroup CZ a.s. Ex. 1015, Page 40 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Internet video/audio streaming: Issues
- Congestion control for streaming applications
- Error control for streaming applications
- Internet streaming tools
- Integrated & Differentiated services
- Smoothing and Selective Dropping

2.3.1 A Taxonomy for Streaming Applications

Streaming applications can be categorized based on various criteria such as type of *distribution*, *liveliness* and level of *Interactivity* as follows:

2.3.1.1 Unicast versus Multicast

Multimedia streams can be distributed in a "Unicast" or "Multicast" fashion. In the former case, a source transmits a stream to a single receiver. In the latter scenario, the streams is sent to a multicast address that might have a large number of members with different network characteristics. Performing congestion and error control is more challenging in a multicast session with a large number of receivers due to the level of heterogeneity among clients.

2.3.1.2 Live versus Playback

A source may transmit a stream as it is generated without any extra delay (except processing and delivery). This is usually known as a *live* session. The other extreme is the case when a pre-recorded, stored stream is played back from secondary storage for a requested client. This is known as a *playback* session.

The main distinction between live and playback session is that in a live session the future data is not available whereas in a playback all the future data is available at the beginning of the session. Thus the stream can be transmitted faster than its consumption

Webgroup CZ a.s. Ex. 1015, Page 41 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

rate and buffered at the receiver side in a playback session. Clearly there is a middle ground where data is slightly delayed and buffered at the source before transmission. This implies that the receiver is always behind the source by a short period delay.

2.3.1.3 Lecture Mode versus Interactive

Streaming sessions can be classified based on the level of interactions between the client and the server. At one extreme case, the server and the client do not interact at all; the server sends the stream (either playback or live) and the client passively receives, and displays the stream. At the other end of the spectrum, the client may frequently interact with the server to change the order of playback, e.g. performing VCR-functions. Note that the level of interaction(i.e. frequency of interaction) is limited by the end-to-end delay between the server and the client. The end-to-end delay includes buffering and processing delay at both ends along with RTT. For example if the end-to-end delay is 100 ms, the level of interaction can not be more than 10 operations per second.

Based on our classification various combinations of above classifications are feasible. A session could be interactive playback, lecture-mode playback, interactive live, lecture-mode live, etc.

2.3.2 Internet Video/Audio Streaming: Issues

Despite lack of quality of service in the Internet, streaming applications have been increasingly developed and deployed over the Internet during the last decade. Figure 2.1 depicts different components of a typical deployment scenario.

Media streams are encoded at the server and can be stored for future playback or directly transmitted after adaptation. Transmission rate(r_t) is controlled by an adaptation module that implements a congestion control mechanism to regulate the transmission rate based on the state of the network. A client may slightly delay the starting playback time to buffer some data. The buffered data is used to cope with the network jitter but it results in a startup latency.

Webgroup CZ a.s. Ex. 1015, Page 42 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

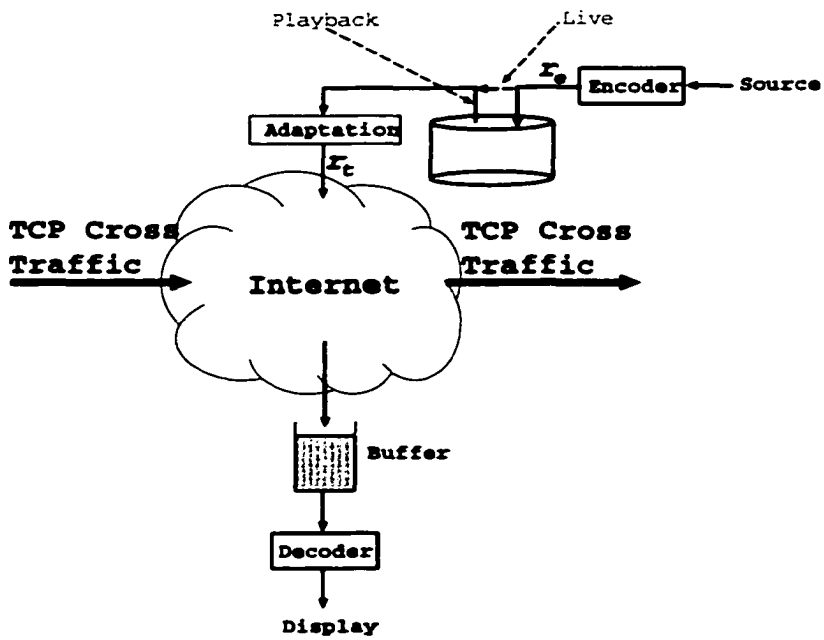


Figure 2.1: A typical scenario for deployment of streaming applications over the Internet

The average output rate of the encoder(r_e) directly depends on the quality (i.e. resolution) of encoding. The higher the quality of the stream, the higher the average bandwidth requirement of the encoded stream. However, to generate a stream with a certain quality, the output rate of the encoder exhibits a bursty behavior which is a function of stream's content and details of the encoding scheme. The main challenge for delivery of congestion controlled multimedia streams over the Internet is to shape the bursty, content-driven output rate of the encoder(r_e) into a congestion-controlled rate limit communication channel(r_t); because variations of encoding rate and transmission rate are completely uncorrelated. The key point is that multimedia streams could be quality adaptive. The output rate of the encoder could be adjusted by changing its quantization parameters. This in turn affects the quality of the encoded stream. Thus the resolution of the encoded stream changes with network load. However frequent changes in encoded quality is disturbing, i.e. low but stable quality is prefered to variable quality that is on average higher.

Webgroup CZ a.s. Ex. 1015, Page 43 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Another parameter that affects the delivered quality is packet loss. Packets loss occurs mainly due to congestion and losses seem to have a seemingly-random pattern from end-points. If a dropped packet is not recovered before its playout time, it could degrade the quality of decoded stream depending on its content and details on the encoding.

In summary, the quality of delivered stream is determined by the following mechanisms:

1. Congestion control mechanism that limits the resolution of encoding.

2. Error control mechanism that tries to minimize the number of packet losses and their effect on the delivered quality.

The goal is to maximize the delivered quality while obeying the congestion control rate limit. The following subsections review related work on congestion control and error control for streaming applications over the Internet:

2.3.3 Congestion Control for Streaming Applications

Gilge et al. [46] proposed an early end-to-end congestion control scheme, called *network-integrated video encoding*, for video transmission over best-effort networks. They used explicit signaling feedback from the receiver to regulate the transmission rate of the source as shown in figure 2.2. In this approach, congestion control is implemented by the encoder based on the feedback from the receiver.

Kanakia et al. [70] built on Gilge's model and proposed an architecture where the feedback is generated by a congested switch along the path. The bottleneck switch communicates its queuing delay back to the source. A controller at the source exploits this information to control the output rate of an MPEG[78] encoder before packet loss occurs at the bottleneck due to queue overflow.

Turletti, Bolot and Huitema[129, 14] proposed a multiplicative increase, multiplicative decrease algorithm to periodically adjust the output rate of a H.261[108] codec based on the reported loss rate from the receiver.

Jeffy et al. [68] designed an unreliable connection oriented transport protocol on top of UDP/IP called the Multimedia Transport Protocol(MTP). MTP monitors the local packet

Webgroup CZ a.s. Ex. 1015, Page 44 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

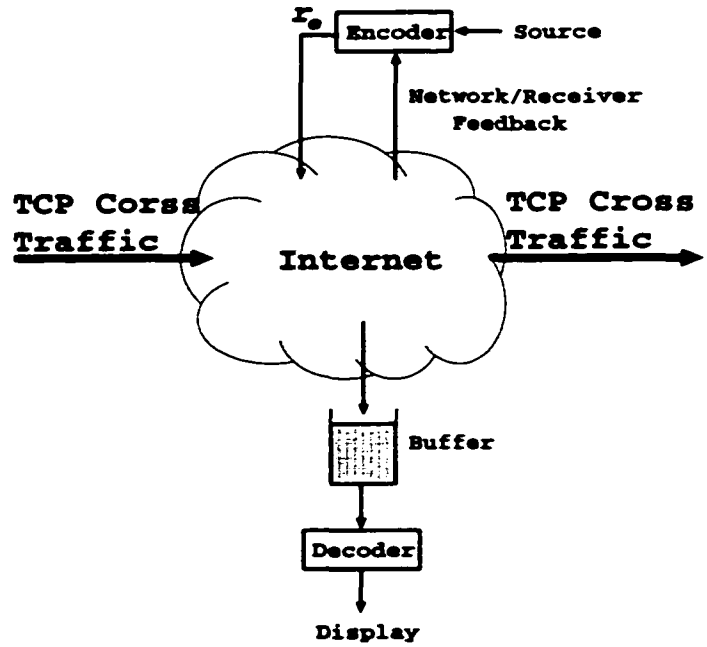


Figure 2.2: Network integrated video encoding

transmission buffer to detect congestion. Once a packet is discarded due to buffer overflow, the protocol signals the application to reduce its data rate. This scheme is only suited to local area networks where congestion could result in increased media access latencies at the local adaptor.

Chen et al. [24] proposed a datagram protocol, called VDP, to integrate video/audio streams with the Web. The adaptation mechanism of VDP degrades or improves the quality of the stream based on the client's feedback. The client reports frame-drop-rate due to CPU bottleneck or loss rate due to network congestion. However, they do not describe a specific strategy for rate adaptation.

Cen et al.[22] presented the SCP protocol for media streaming. SCP deploys a modified version of TCP's congestion control mechanism that performs Vegas-like rate adjustment in steady state.

Webgroup CZ a.s. Ex. 1015, Page 45 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The above approaches have either ignored congestion control for streaming application or did not extensively examined various aspects of their proposed congestion control mechanisms such as inter-protocol fairness, stability or responsiveness.

2.3.3.1 Addressing TCP-friendliness

Jacobs et al.[60] loosely isolates the congestion control mechanism from the rate adaptive encoding using a buffer. It leverages off the TCP's congestion control mechanism (AIMD window-based adaptation without any retransmission) to regulate the transmission rate of data that is drained from the buffer. A rate adaptive MPEG encoder fills the buffer and its output rate is controlled based on the level of occupancy of the buffer. The goal is to maximize the quality while preventing buffer overflow.

Mahdavi and Floyd [80] initially proposed a simple equation to model TCP's transmission rate as a function of loss rate and RTT. Mathis et al. [85] and then Padhye et al.[96] further improved the model. Given the value of loss rate and RTT, these equations can be used to estimate utilized bandwidth by a TCP flow under these conditions. Note that there still needs to be a complementary rate adjustment strategy to match the transmission rate to the TCP-equivalent rate. Work in [96] presents such a mechanism.

Tan et al.[124] also proposed an error-resilient adaptive encoding scheme using the TCP model presented in [80] to achieve TCP-friendly behavior. While TCP-friendly equation is a very promising direction, there are some concerns about its stability for large-scale deployment.

2.3.3.2 Encompassing Multicast

Multicast delivery of multimedia streams is fundamentally more complicated than unicast. This is mainly due to potentially high level of heterogeneity in bandwidth and processing capability among clients. If every receiver sends a feedback signal to the source, it could result in a problem known as *feedback implosion*. Such a mechanism does not scale to multicast group with a large number of members. Even if the source would be able to receive and process feedback from receivers in a scalable fashion(e.g. [15]), it is not clear how the source should react to conflicting feedback. Thus a source-based congestion

Webgroup CZ a.s. Ex. 1015, Page 46 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

control mechanism does not seem to be feasible for multicast and this is still an active area of research. For the sake of completeness, we briefly review some of the well-known related works on multicast streaming over the Internet.

Amir et al. proposed a video gateway(VGW) architecture[5] to accommodate bandwidth heterogeneity for multicast video streaming. Video gateway is a trans-coder that is placed at the point of bandwidth discontinuity to change the rate and consequently the quality of the video accordingly. Although the VGW does not implement congestion control, it allows a group of receivers in a low-bandwidth subtree to join a high bandwidth multicast session and receive the appropriate video quality without experiencing congestion. The main challenge is to dynamically identify the point of bandwidth discontinuity to place VGW. Work in [39] proposes a dynamic mechanism for VGW placement. Tennenhouse and Wetherall's proposed "Active Network Architecture"[125] that provides a generalized approach for deployment of rate-adaptive video gateways within the network.

Real-time Protocol(RTP) [115] has been standardized for real-time delivery of multimedia stream over multicast (or unicast) networks by the IETF's Audio/Video Transport(AVT) Working Group. RTCP is the control protocol that is embedded in RTP. RTCP requires receivers to periodically multicast reception reports that include loss rate, throughput, etc. to the entire group to provide better scaling properties. As the number of members in the group increases, each receiver transmit reception report less frequently such that the aggregate bandwidth for all reception report remains below a small percentage of the session bandwidth. RTP does not address congestion control but reception reports provide sufficient information to all members to identify receivers that are experiencing congestion. Thus the source might be able to exploit this information to adapt its transmission rate. Busse et al.[20] proposed such a source-based rate adaptation approach. As we mentioned earlier, this approach does not seem appropriate for a group with heterogeneous receivers because it results in either persistent congestion for some low-bandwidth receivers or low quality stream for some high-bandwidth receivers that where able to receiver higher quality stream. Furthermore, as the size of the multicast group increase, the resolution of reception report decreases. Thus the source becomes less responsive and congestion lasts for a longer period of time.

Webgroup CZ a.s. Ex. 1015, Page 47 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Receiver-driven Layered Multicast(RLM)[88] is the most promising approach that addresses a receiver-based mechanism to accommodate bandwidth heterogeneity for multicast delivery of layered video over the Internet. Layered encoded video is sent into multiple multicast group and receivers actively control the quality and subsequently the bandwidth of the received video by adjusting level of subscription (i.e. number of receiving layers). Observing increased packet loss signals congestion to the receiver and triggers the receiver to reduce level of subscription. In the absence of packet loss, receivers periodically probe the availability of bandwidth by increasing their level of subscription, called join experiment, using a mechanism similar to TCP. RLM incorporates several mechanisms to prevent synchronization of join experiments such that all receivers behind a bottleneck could share their probing results and converge to the same level of subscription. It is still not clear how effective such a "shared learning" mechanism would be in a real world scenario with multiple RLM sessions. Thin Streams[136] is a descendent of RLM that attempts to address some of the short coming of RLM. Thin Stream employs a Vegas-like mechanism to avoid congestion. It also uses a random clock edge to synchronize join experiments within a multicast session.

RLM does not exhibit TCP-friendly behavior. To address this problem, Bolot et al. [13] proposed a receiver-driven layered mechanism that is similar to RLM. However it replaces join experiment with an explicit estimation of TCP-friendly bandwidth to determine the right level of subscription for each receiver using TCP-equation. Vicisano et al.[130] describes yet another TCP-like congestion control mechanism for layered multicast.

2.3.4 Error Control for Streaming Applications

A source can not avoid losses even when network is in equilibrium. The visual degradation in perceived quality due to packet loss depends on level of resilience of the encoding, percentage of losses and loss pattern. The main encoding trade-off is between enhancement in error resilience and efficiency of compression. While efficiency of the compression can be enhanced by removing temporal dependency, this makes the stream less resilient to packet loss because an error caused by a single loss can propagate and

Webgroup CZ a.s. Ex. 1015, Page 48 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

affect the quality of delivered stream for a longer period of time. Most of the commonly used tools [44, 48, 63] use a technique called *Conditional replenishment* [90]. Since all the coded blocks are temporally independent, packet loss only affects those frames that are contained in the lost packet. This approach enhances error resilience at the cost of low compression efficiency.

In order to design effective error control mechanisms, it is useful to have some knowledge about characteristics of loss pattern that is likely to encounter. A number of studies have been conducted to examine characteristics of packet loss over the Internet[16, 17, 51, 79]. These studies revealed that loss rate is often low and the majority of losses consists of a single packet. As the Internet evolves, it is arguable how representative these results are. However they certainly provide some incentives for the design of error control mechanisms.

There have been many interesting works on error control mechanisms for streaming applications. However, almost all of the works of what we are aware have not incorporated an effective congestion control mechanism. In this subsection, first we briefly present two commonly used techniques for error control in the context of media streaming along with their advantages and disadvantages, and review some of the related work on each technique. Then we address the challenge of integrating error control with congestion control mechanisms.

Two of the most frequently used techniques[99] to repair packet loss for media streaming are:

1. Retransmission
2. Forward Error Correction(FEC)

Retransmission: Retransmission is a natural way to recover from loss. However it is not appropriate for interactive streaming applications due to the inherent latency associated to retransmission. Since retransmission-based repair is demand driven, these schemes usually incur low bandwidth overhead. A receiver should buffer sufficient data to accommodate at least three one-way trip delay for retransmitted packets to arrive in time for display [106, 98, 4, 137]. Rhee[112] showed that retransmission can still be effective to

Webgroup CZ a.s. Ex. 1015, Page 49 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

improve error resilience in interactive low-bit rate video. The idea is to use late packets to restore a frame and also to prevent error propagation to subsequent frames that rely on the late frame as reference.

Forward Error Correction(FEC): The basic idea of FEC is to add some amount of redundancy to the stream that enables receiver to recover from packet loss without referring to the source. Thus FEC is an attractive approach to error recovery for interactive streaming applications. There are two classes of FEC schemes[99]:

1. **Media Independent FEC**: In this class redundant data is transmitted in separate packets. The simplest example for independent FEC is a parity-based approach using XOR operation. Bandwidth requirement, latency and loss repair ability depend on details of parity calculation, the amount of redundancy and combination of media packets and parity packets.

   The main advantage of this class is that they are media independent and require very low processing requirement in compare to media-specific FEC. In Contrast, the coding has higher latency.

2. **Media-specific FEC**: This class exploits knowledge of media compression scheme to improve efficiency of recovery mechanism. This approach can result in significant bandwidth saving at the cost of additional processing overhead. Work in [17, 52, 101] present such a mechanism for streaming audio over the Internet.

2.3.4.1 Integrating Error and Congestion Control

A complete end-to-end framework should address both congestion and error control. However, the integration of these two components is not straight forward. The main issue is that as the loss rate increases, the congestion control mechanism decreases the transmission rate. But increase of loss rate results in higher bandwidth requirement for error control(using either more retransmission or higher amount of redundancy). This means that the amount of useful bandwidth to transmit new data further decreases and that in

Webgroup CZ a.s. Ex. 1015, Page 50 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

turn results in lower quality. Thus in a nut shell, it is not clear what portion of the congestion controlled bandwidth should be allocated for error control. Bolot et al.[12] describes an adaptive FEC mechanism for audio but the integration with congestion control is not addressed.

This problem can be also formulated in a slightly different manner. At each transmission time, the source has a choice to send either an old packet that was lost or a new packet. This essentially addresses the tradeoff of sending higher quality information versus increasing the reliability of lower quality but important information. This is the tradeoff between error control and quality adaptation. The goal is to maximize the quality of delivered stream. Podolsky et al. [100] adopts this approach for layered encoded streams and conduct a Markov-chain analysis to provide quantitative performance comparison of different transmission policies.

2.3.4.2 Join Source/Channel Coding

To minimize the effect of packet loss on delivered quality, work in [71, 45] proposed a joint source/channel coding approach. The source encodes its real time traffic into high and low level priority(e.g. layered video). The network uses a two-level priority drop scheme, i.e. a router serves high-priority traffic before low-priority traffic. Thus when congestion occurs, low-priority traffic is affected. They also proposed a mechanism that dynamically adjusts the bandwidth of low and hight priority levels at the source based on the feedback from the network. The goal in this approach is to maximize the quality and it does not incorporate any congestion control mechanism. This approach relies on the support of priority drop in the network, which is not available in today's Internet. Bajaj et al.[8] further investigated the relative merit of uniform vs priority dropping for layered video and concluded that performance benefit of priority dropping is less than they expected.

2.3.5 Internet Streaming Tools

Despite various difficulties in incorporating congestion control into streaming applications, many such applications have been developed and deployed over the Internet during

Webgroup CZ a.s. Ex. 1015, Page 51 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

recent years. Most of the works in streaming applications over the Internet have targeted non-interactive streaming application due to the need for buffering.

nv[44], IVS[128], *vat*, *vic*[63] and Nevot[120] are some of the well known tools that are frequently used for multicast delivery of multimedia streams.

Two of the most popular commercial tools for unicast delivery of video multimedia streams over today's Internet are Real audio/video [93], and NetShow[58]. Unfortunately, there is no technical information for evaluation of these applications.

2.3.6 Integrated & Differentiated Services

Rather than dealing with various issues for supporting streaming applications over a best-effort network without any QoS support, many researcher suggested to add a new service model to accommodate QoS requirements of streaming applications. This approach is usually referred to as *integrated services* because it incorporates QoS requirements for streaming applications and provides performance guarantees for this class of applications. The idea is similar to the open-loop congestion control paradigm that was presented in section 2.1.1.1. The network implements admission control mechanism during call setup phase and allocate part of its resources to each flow based on client requirement. During a session, the network should ensure that the allocated resources are available for the flow and the end-system should ensure that its offered traffic is within the presented profile during call establishment phase. In this paradigm, reactive congestion control by the end system changes to traffic shaping at the end-system. The ReSerVation Protocol(RSVP)[138] is one of the well-known reservation protocol for establishing Integrated services over the Internet.

Some researcher suggested *loose* instead of absolute guarantees for the quality of delivered service to individual flows. This *predictive Service*[28] monitors usage of the resources and based on the behavior of aggregate traffic in the past determines the availability of extra resources to admit or reject a new flow[67]. The basic assumption is that the observed traffic in the recent past is a good estimate for the offered load in a near future. Although this approach does not support absolute guarantees for quality of delivered service, it can achieve reasonably high performance.

Webgroup CZ a.s. Ex. 1015, Page 52 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The main challenge in supporting the integrated service model is the need for maintaining *per-flow* state at each intermediate routers in the network. Many researchers involved in this area have realized some of the resulting difficulties associated with providing Integrated services and per-flow reservation of resources. Some of these difficulties are as follows:

1. **Scalability:** Maintaining per-flow state results in high memory requirement and processing overhead for each intermediate router. Thus there are some scalability concerns on how to deploy such a mechanism for a router with large number of simultaneous flows.

2. **Flexibility of Service Model:** The Integrated service framework only provides a small number of pre-specified service classes. This set of classes does not allow more qualitative or relative definition of service model that is more appropriate and useful for end-systems. For example service class gold receives a better service than class silver.

3. **Need for Implementing RSVP Signaling:** Most of the host in today's Internet do not support RSVP signaling. Many applications may only require qualitative specification of their service requirement based on observed service in a class or cost associated to a class, e.g. Better service than class silver.

These issues motivated a new service model, called *Differentiated Services*)[11], that is currently being developed at the IETF. The Differentiated Services proposed different classes of services within the Internet in a scalable and flexible fashion. To accommodate scalability, per-flow state are kept at the edge routers where the number of flows are relatively small and the router is able to manage the complexity and resource requirement. Edge routers are mainly in charge of 1) assigning each packet to a particular class of service based on requested service by the host and 2) conditioning the offered traffic by that host. Core routers simply serve each packet based on the specified class of service in the packet header. The Differentiated Services is still at its early stages and is still evolving.

Webgroup CZ a.s. Ex. 1015, Page 53 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Currently, there are couple of evolving frameworks for Differentiated Service architecture that are under active discussion within the diffserv group at IETF: 1) An expedited Forwarding PHB [64] and 2) An Assured Forwarding PHB[54].

2.3.7 Smoothing and Selective Dropping

As we mentioned earlier, to encode a stream with a particular quality, the codec generates a variable bit rate output which exhibits bursty behavior. The variations in bandwidth depends on the content of the stream and details of encoding mechanism. To reduce the variability of bandwidth requirement of encoded stream, many researchers have proposed *smoothing* mechanisms [75, 114, 132, 116]. The main idea in many of these mechanisms is to take advantage of client buffer and available information about future changes in bandwidth requirements to smooth out these changes over time. The ultimate goal is to send constant-bit-rate streams. Thus these schemes are mostly applicable to stored media because the pattern of changes in bandwidth is known a priori. Furthermore they are only suited for reservation-based networks and do not address congestion control.

A slightly different approach is to fit output of an encoder into a CBR channel using *selective dropping*[38]. Here the basic idea is to use application-specific information to decrease the effect of a packet loss on delivered quality. Whenever transmission rate exceeds the available bandwidth or client's buffer experiences overflow, the source discards frames with least priority until the bandwidth or buffer requirement are met. This approach can be used to smooth out the transmission rate or match the transmission rate with congestion control rate limit. Clearly, the latter case would be more challenging. Notice that this approach results in relatively coarse-grain changes in transmission rate. Furthermore, the application can only express the preference of one frame to others instead of directly controlling the effect of bandwidth changes on delivered quality

Webgroup CZ a.s. Ex. 1015, Page 54 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3.8 Summary of Streaming Applications in the Internet

In the previous subsection, we presented a taxonomy for streaming applications and addressed related work in many areas of Internet Multimedia Streaming that are more relevant to this dissertation, in particular:

- Challenges in supporting Audio/Video streaming over the Internet
- Congestion control for Internet streaming applications
- Error Control for Internet streaming applications
- Integrated and Differentiated services
- Smoothing and Selective dropping

While these research efforts present a rich spectrum of work in this area, none of these contributions presented a comprehensive solution for delivery of multimedia streams in a congestion controlled fashion over the Internet while enabling the server to control level of smoothing(i.e. stability of quality) of delivered streams.

2.4 Proxy Caching for Multimedia Streams

Proxy caching has been a key factor in the scalability of the Web. Caching the popular objects at a proxy close to the interested clients substantially reduces load on the network, the server, and the startup latency. While performance improvement and evaluation of various web caching mechanisms have received great attention, proxy caching of multimedia streams has not been studied extensively by the caching community.

Work on multimedia caching has focused mainly on memory caching in the context of multimedia servers [33, 69, 31, 32]. The idea is to reduce disk and tertiary access by grouping requests that are relatively close and retrieving a single stream for the entire group. In other words, these approaches tried to minimize the object migration among different levels of a hierarchical storage system, *i.e.*, tertiary, disk and memory. There is an analogy between memory caching and proxy caching as it is shown in figure 2.3.

Webgroup CZ a.s. Ex. 1015, Page 55 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

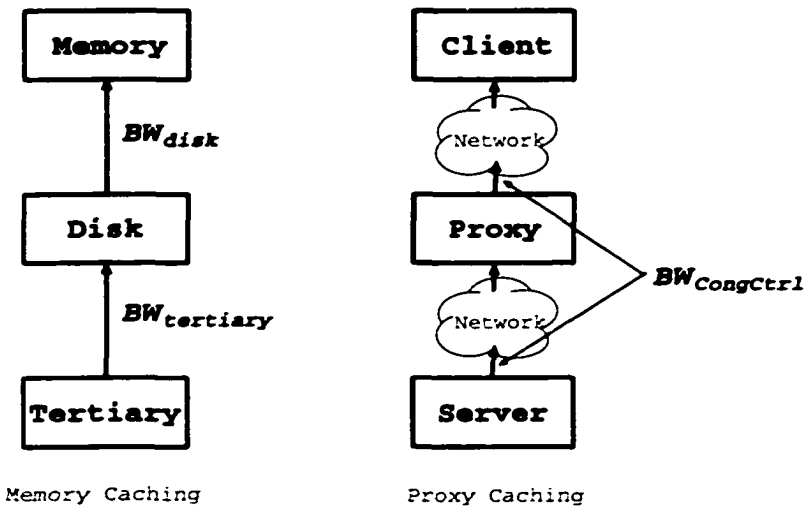


Figure 2.3: Memory Caching versus Proxy Caching of Multimedia Streams

In other words, object migration in a hierarchical storage system is similar to stream transmission among server, proxy and client. However, there is a fundamental difference between them: the available bandwidth between two levels of a hierarchical storage system is fixed and known *a priori*, whereas the available bandwidths among server, proxy and client unpredictably change with time. As a result, one can not simply apply these memory caching schemes for proxy caching of Internet streams.

There are numerous works on proxy cache replacement algorithms [21, 59, 113, 135, 134]. It is not clear how they will behave when a request sequence contains a significant number of requests to huge multimedia streams. To our knowledge, there is only one work that addresses the influence of multimedia streams on cache replacement algorithms [126]. They consider the impact of resource requirements (*i.e.*, bandwidth and space) on cache replacement algorithms.

Rapid increase in deployment of streaming applications during recent years motivated several products such as Streaming cache by Inktomi[57] and MediaMall by InfoLibria[56]. While there is no technical information about these products, they seem to simply advocate the idea of caching multimedia streams closer to the receivers to reduce congestion in the network and the server.

There has also been some work that dealt with caching issues for streaming applications. Here we briefly review all the relevant works of which we are aware:

Webgroup CZ a.s. Ex. 1015, Page 56 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Sen et al.[116] proposed a prefix caching scheme for multimedia streams. The idea is to cache the first few segments (i.e. prefix) of popular streams at a proxy close to the clients. The cached data is used for work-ahead smoothing and it also reduces start up latency.

Miao et al. [91] addresses both a prefix caching and a selective caching mechanisms. Given the client buffer size, the goal in selective caching approach is to exploit encoding specific information to increase the robustness of the stream against upcoming loss due to congestion, i.e. it tries to keep those frames that are more critical for the quality of the stream.

Ortega et al. proposed some image-specific caching strategies, called Soft caching[95]. The idea is to adjust the resolution of cached images with their popularity.

Acharya et al. proposed the MiddleMan architecture[1], a collection of cooperative proxy caches associated with a LAN to reduce startup latency and load on both the network and the server. They characterize video streams on the Web and analysis users' access pattern to these stream. This information is exploited to devise appropriate high performance caching techniques and load balancing strategies among proxies.

Hofmann et al. described the SOCCER architecture[55], a self-organizing cooperative caching architecture. They essentially proposed an architecture for delivery of multimedia streams over the Internet using cooperative proxies. However, they mainly focus on issues related to load distribution and coordination among proxies as well as different trade-offs associated to delivery from various proxies. They do not discuss replacement algorithms and their effect on delivered quality.

None of these contributions have addressed the idea of proxy caching for improvement of delivered quality and the need to perform congestion control for streaming applications over the Internet.

2.4.1 Summary of Proxy Caching for Multimedia Streams

Despite the success of caching for the Web, it has not been effectively used for multimedia streams. Current cache replacement algorithms are fine-tuned to achieve high performance for web objects. Given different characteristics of multimedia streams and

Webgroup CZ a.s. Ex. 1015, Page 57 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

their access pattern, it is unlikely that these algorithms result in high performance for multimedia streams. Although memory caching of multimedia streams has been extensively studied, it is not easily applicable to multimedia proxy caching because of the need to congestion control. Furthermore, proxy caching provides a good opportunity to improve the quality of delivered stream to the client despite the presence of a bottleneck along the path to the server. This aspect of multimedia proxy caching has never been addressed.

Webgroup CZ a.s. Ex. 1015, Page 58 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

The End-to-end Architecture

This chapter presents our design philosophy and provides a high level architectural view of the design of multimedia playback applications for the Internet. An example of our target environment is a video server that plays back video streams for a large group of clients with heterogeneous network capacity and processing power through the Internet where the dominant competing traffic is TCP-based. The server maintains a large number of video streams. Video clips are sufficiently large that their transmission time is longer than acceptable playback latency. As with current Internet video streaming, we expect the length of such streams to range from 30 second clips to full-length movies. Users expect startup playback latency to be low, especially for shorter clips played back as part of web surfing. Thus pre-fetching an entire stream before starting its playback is not an option. We believe that this scenario reasonably represents many of the current streaming applications in the Internet. The goal is to maximize the overall stable playback quality while obeying congestion control constraints. Furthermore neither the server nor the clients should require excessive processing power or storage space.

Addressing design principles for Internet applications led us to identify *congestion control*, *quality adaptation* and *error control* as three key components for any video streaming application. The key idea is to separate congestion control from error (and quality) control because the former depends on the state of the network while the latter is application specific. We explore the design space for each one of these components in the context of video playback applications for the Internet, suggest a mechanism for each component from its design space, and justify our design choices. Our main contribution

Webgroup CZ a.s. Ex. 1015, Page 59 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is to compose the three key components into a coherent architecture and describe the interaction among these components. We then argue that the architecture can be viewed as a generic architecture for video playback applications as long as the different modules are properly integrated.

3.1 Design Principles

In a shared best-effort network such as the Internet, there are several principles that must be followed in the design of any new application including streaming applications as follows:

3.1.1 Social Behavior

The Internet is a shared environment and does not micro-manage utilization of its resources. Since flows are not isolated from each other, a mis-behaved flow can affect other co-existing flows. Thus end systems are expected to be cooperative and react to congestion properly and promptly [41, 77]. The goal is to improve inter-protocol fairness and keep utilization of resources high while the network operates in a stable fashion.

The current Internet does not widely support any reservation mechanism or Quality of Service(QoS). Thus the available bandwidth is not known a priori and changes with time. This implies that applications need to experiment to learn about network conditions. A common approach is that applications gradually increase their transmission rates to *probe* availability of bandwidth without severely congesting the network[65]. When any indication of congestion is detected, they rapidly *back-off* their transmission rate. This process is known as end-to-end congestion control and is required for stability of the network. Because of the dynamics of the traffic, each flow continuously probes and backs off to adapt its transmission rate to the available bandwidth. It is crucial to understand that congestion control is a *network dependent* mechanism and must be equally deployed by all applications.

Webgroup CZ a.s. Ex. 1015, Page 60 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Even if the Internet eventually supports reservation mechanisms [138] or differentiated services[11], it is likely to be on per-class rather than per-flow basis. Thus, flows are still expected to perform congestion control within their own class.

3.1.2 Being Adaptive

With the Internet's best-effort service model there is neither an upper bound for delay nor a lower bound for available bandwidth. The quality of the service provided by the network changes with time. Furthermore, performing effective congestion control could result in random and wide variations in available bandwidth. Applications must be able to cope with these variations and adaptively operate over a wide range of network conditions.

Streaming applications are able to adjust the quality of delivered stream (and consequently its consumption rate) with long-term changes in available bandwidth and operate in various network conditions. However this mechanism is application specific. We call this *quality adaptation*.

3.1.3 Recovery From Loss

Packets are lost in the network mainly due to congestion. The loss pattern from the end points appears seemingly random[16]. Although streaming applications can tolerate some loss, it does degrade the delivered stream quality. To maintain reasonable quality, streaming applications need a way to recover from most losses before their playout time. Such a loss recovery mechanism is usually known as *error control*. The effect of loss on playout quality is also application specific.

3.2 Design Space

Before we describe our proposed architecture, we explore the design space for the key components and specify our design choices.

Webgroup CZ a.s. Ex. 1015, Page 61 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.1 Congestion Control

The most well understood algorithm for rate adaptation is Additive Increase, Multiplicative Decrease(AIMD) [25] used in TCP[65], where transmission rate is linearly increased until a loss signals congestion and a multiplicative decrease is performed.

A dominant portion of today's Internet traffic consists of a variety of TCP-based flows[26]. Thus TCP-friendly behavior is an important requirement for new congestion control mechanisms in the Internet otherwise they may shut out the well-behaved TCP-based traffic. By TCP-friendly we mean that a new application that coexists with a TCP flow along the same path should obtain the same average bandwidth during a session.

TCP itself is inappropriate for streaming applications with hard timing constraints because its in-order delivery could result in a long delay. Even modified version of TCP without retransmission [61] exhibits bursty behavior. SCP[22] and LDA[121]. protocols target streaming applications. Their goal is to be TCP-friendly, however they were not examined against TCP over a wide range of network conditions. RAP[109] is a rate-based congestion control mechanism that deploys an AIMD rate adaptation algorithm. RAP is suited for streaming applications and exhibit TCP-friendly behavior over a wide range of network conditions. Another potential class of rate-based congestion control schemes is based on modeling TCP's long-term behavior[96]. There is on-going work[50] to evaluate the stability of these mechanisms. However we have adopted RAP for congestion control in our architecture.

3.2.2 Quality Adaptation

Streaming applications are rate-based. Once the desired quality is specified, the realtime stream is encoded and stored. The output rate of the encoder is a direct function of the desired quality, the encoding scheme and the content of the stream. Although the output rate of the encoder could vary with time, for simplicity we assume that encoder generates output with a near-constant bandwidth. In the context of video, this typically implies that the perceived quality is inversely proportional to the motion in the video. Remaining small variations in bandwidth are smoothed over a few video frames using playout buffering.

Webgroup CZ a.s. Ex. 1015, Page 62 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In contrast, performing TCP-friendly congestion control based on an AIMD algorithm results in a continuously variable transmission rate. The frequency and amplitude of these variations depends on the details of the rate adjustment algorithm and the behavior of competing background traffic during the life of the connection. The main challenge for streaming applications is to cope with variations in bandwidth while delivering the stream with an acceptable and stable quality. A common approach is to slightly delay the playback time and buffer some data at the client side to absorb the variations in transmission rate [107]. The more data is initially buffered, the wider are the variations that can be absorbed, although a higher startup playback latency is experienced by the client. The main reason that we target playback applications is because they can tolerate this buffering delay. For a long-lived session, if the transmission rate varies widely and randomly, the client's buffer will either experience buffer overflow or underflow. Underflow causes an interruption in playback and is very undesirable. Although buffer overflow can be resolved by deploying a flow control mechanism it then means that the fair share of bandwidth is not fully utilized.

To tackle this problem, a complementary mechanism for buffering is required to adjust the quality(i.e. consumption rate) of streams with long term variations of available bandwidth. This is the essence of *quality adaptation*. A combination of buffering and quality adaptation is able to cope with random variations of available bandwidth. Short term variations can be absorbed by buffering whereas long term changes in available bandwidth trigger the quality adaptation mechanism to adjust the delivered quality of the stream.

There are several ways to adjust the quality of a pre-encoded stored stream, including adaptive encoding(i.e transcoding), switching between multiple encoded versions and hierarchical encoding. One may adjust the resolution of encoding on-the-fly by re-quantization based on network feedback[14, 94, 124]. However, since encoding is a CPU-intensive task, servers are unlikely to be able to perform on-the-fly encoding for large number of clients during busy hours. Furthermore, once the original data has been stored compressed, the output rate of most encoders can not be changed over a wide range.

Webgroup CZ a.s. Ex. 1015, Page 63 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In an alternative approach, the server keeps several versions of each stream with different qualities. As available bandwidth changes, the server switches playback streams and delivers data from a stream with higher or lower quality as appropriate.

With hierarchical encoding [76, 86, 89, 131], the server maintains a layered encoded version of each stream. As more bandwidth becomes available, more layers of the encoding are delivered. If the average bandwidth decreases, the server may drop some of the active layers. Layered approaches usually have the decoding constraint that a particular enhancement layer can only be decoded if all the lower quality layers have been received.

There is a duality between adding or dropping of layers in the layered approach and switching streams with the multiply-encoded approach. The layered approach has several advantages though: it is more suitable for caching by a proxy for heterogeneous clients[110, 111], it requires less storage at the server side and it provides an opportunity for selective retransmission of the more important information. The main challenge of a layered approach for quality adaptation is primarily in the design of an efficient add and drop mechanism that maximizes overall delivered quality while minimizing disturbing changes in quality.

3.2.3 Error Control

Streaming applications are semi-reliable, i.e. they require quality instead of complete reliability. However, with most encoding schemes, packet loss beyond some threshold will degrade the perceived playback quality because good compression has removed temporal redundancy and image corruption thus becomes persistent. Therefore these applications must attempt to limit the loss rate below that threshold for a given encoding.

Techniques for repairing realtime streams are well known[99], and include retransmission[98], FEC[17], inter-leaving and redundant transmission. The appropriate repair mechanism is selected based on the level of reliability that is required by the application codec, the delay that can be tolerated before recovery, and the expected or measured loss pattern throughout the session.

In the context of unicast delivery of playback video, retransmission is a natural choice. The only disadvantage of retransmission-based approach is the retransmission delay, but

Webgroup CZ a.s. Ex. 1015, Page 64 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in the context of non-interactive playback applications, client buffering provides sufficient delay to perform retransmission. Moreover retransmission can be performed selectively, which nicely matches our layered framework for quality adaptation where the lower layers are more important than the higher layers.

Missing packets are retransmitted only if there is sufficient time for retransmission before playout. With a layered codec, retransmission of packets from layer i have priority over both new packets from layer i and over all packets from layer $i + 1$. This is because immediate data is more important than future data, and the lower layers are more important for perceived quality.

3.3 The Architecture

In this section, we compose our design choices into a coherent end-to-end architecture and explain the interaction among different components. Figure 3.1 depicts the architecture. The three key components are labeled as rate adaptation, quality adaptation and error control.

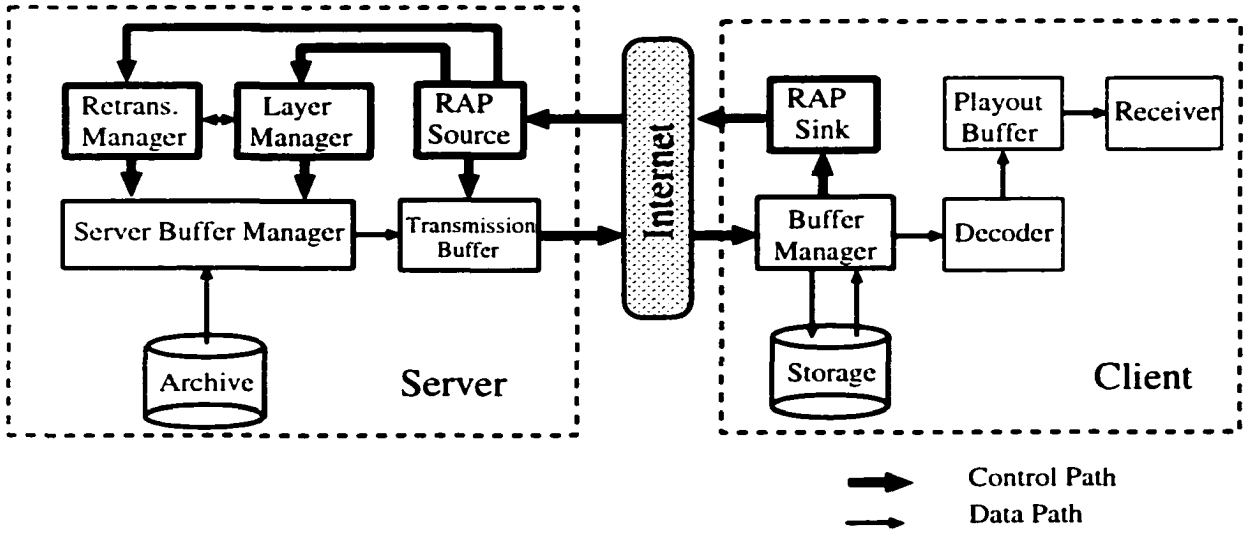


Figure 3.1: End-to-end architecture for playback streaming applications in the Internet

End-to-end congestion control is performed by the *rate adaptation* (RA) and *acker* modules at the server and client respectively. The RA module continuously monitors

Webgroup CZ a.s. Ex. 1015, Page 65 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the connection and regulates the server's transmission rate by controlling the inter-packet gaps. The acker module acknowledges each packet, providing end-to-end feedback for monitoring the connection. The acker may add some redundancy to the ACK stream to increase robustness against ACK loss. Moreover, each ACK packet carries the most recent playout time back to the server. This allows the server to estimate the client buffer occupancy and perform quality adaptation and error control more effectively.

The quality of the transmitted stream is adjusted by the *quality adaptation* (QA) module at the server. This module periodically obtains information about the available bandwidth and the most recent playout time from the RA module. Combining this information with the average retransmission rate provided by the error control module, the QA module adjusts the quality of the transmitted stream by adding or dropping layers accordingly.

Error control is performed by the *error control* (EC) module at the server. It receives information about available bandwidth, loss rate and recent playout time from the RA module. Based on this information, it either flushes packets from the server's buffer manager that were acknowledged or their playout time have passed, or schedules retransmission of a lost packet. The EC module can selectively retransmit those packets that have high priority such as losses from the base layer.

Since both quality adaptation and retransmission must be performed within the rate specified by the RA module, the EC and QA modules need to interact closely to share the available bandwidth effectively. The goal is to maximize the quality (taking into account packet losses of the final played-out stream) for the available bandwidth while minimizing any variations in quality. In general, retransmission has a higher priority than adding a new layer whenever extra bandwidth is available. These interactions among the RA, QA and EC modules are shown as part of the control path in figure 3.1 with thicker arrows.

The data path which is followed by the actual multimedia data, is specified separately with thinner arrows. The server maintains an archive of streams in local mass storage. A requested stream is pre-fetched and divided into packets by the server buffer manager just prior to the departure time of each packet. The resolution (i.e. number of layers) of the pre-fetched stream is controlled by the QA module. Moreover, the QA and EC modules cooperatively arrange the order of packets for upcoming transmission. In summary, the

Webgroup CZ a.s. Ex. 1015, Page 66 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

RA module regulates the transmission rate and QA and EC modules control the content of each packet. The client's buffer manager receives the packets and rebuilds the layered encoded stream based on the playout time of each packet before they are fed into the decoder. The playout time of the base layer is used as reference for layer reorganization. The buffered data at the client side is usually kept in memory, but if the client does not have enough buffer space, the data could be temporarily stored on a disk before it is sent to the decoder

3.4 Generalizing The Architecture

We outlined a sample architecture for a video playback server and its client to describe the interactions among different components in the previous section. This can be viewed as a generic architecture for a class of Internet video playback applications. The selected mechanisms we described for each component can be replaced by others from the corresponding design space as long as they are in harmony with other components. For example, one could deploy another technique such as FEC for error control on the base layer. Such a design choice would affect the buffer management scheme at the server and client side, and would change the interaction between QA and EC modules since there is no need to leave a portion of the available bandwidth for retransmission. Instead the base layer requires a higher bandwidth. Another example would be to replace RAP with a congestion control mechanism based on modeling TCP's long-term throughput. This implies that quality adaptation must be tuned based on the rate adaptation algorithm of the new congestion control mechanism. It is generally more effective to design a quality adaptation mechanism that is customized to the design choices for the other components of the architecture. For example, knowing the rate adaptation algorithm allows us to devise a more optimal quality adaptation mechanism.

A key property of this architecture is to separate different functionalities and assign each of them to a different component. Given this generic architecture, the natural steps for designing an end-to-end scheme for video playback applications are the following:

1. Select a TCP-friendly congestion control scheme.

Webgroup CZ a.s. Ex. 1015, Page 67 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. Select an error control scheme that satisfies the application requirement given the expected or measured characteristics of the channel.

3. Design an effective quality adaptation mechanism and customize it such that it maximizes the perceptual quality of the delivered video for a given encoding, rate adaptation algorithm and the choice of error control mechanism.

Congestion control is a network specific issue and it has been extensively studied. However work on congestion control for streaming applications is more limited. Error control is a well understood issue and one can plug in one of the well known algorithms from the design space that suites the particular application. The remaining challenge is to design application specific quality adaptation mechanisms that reconcile the constant-bit rate (or content-driven variable bit-rate) nature of video applications with the congestion-driven variable bandwidth channel. While doing this, it must interact appropriately with the error control mechanism toward the goal of maximizing the perceptual quality. We believe that quality adaptation is a key component of the architecture that requires more investigation.

3.5 Summary

In this chapter, we described our end-to-end architecture for multimedia playback applications. We argued that to satisfy design principles for Internet applications, these applications should include three main control modules as follows: 1. Congestion Control, 2. Quality Adaptation, and 3. Error Control.

We explored the design space for each one of these components, evaluated different design choices, and suggested a particular mechanism for each component. Furthermore, we addressed the implications of choosing a particular mechanism on other components. Finally, we argued that the architecture can be generalized. We presented some natural design steps to design such an architecture.

Webgroup CZ a.s. Ex. 1015, Page 68 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

The Rate Adaptation Protocol

This chapter presents the design and evaluation of the Rate Adaptation Protocol(RAP) through extensive simulation. RAP is an end-to-end rate-based congestion control mechanism that is suited for unicast playback of realtime streams as well as other semi-reliable (i.e. UDP-based) Internet applications. The goals of RAP are to be well-behaved and TCP-friendly.

It has been shown that the Additive Increase and Multiplicative Decrease (AIMD) algorithm efficiently converges to a *fair* state [25]. RAP adopts an AIMD algorithm for rate adaptation to achieve inter-protocol fairness and TCP-friendliness. RAP performs loss-based rate control and does not rely on any explicit congestion signal from the network since packet loss seems to be the only feasible implicit feedback signal in the Internet due to the presence of competing TCP traffic. However, if the network supported explicit congestion signaling[105], RAP could exploit this to behave more efficiently.

4.1 The RAP Protocol

The RAP protocol machinery is mainly implemented at the source. A RAP source sends data packets with sequence numbers, and a RAP sink acknowledges each packet, providing end-to-end feedback. Each acknowledgment (ACK) packet contains the sequence number of the corresponding delivered data packet. Using the feedback, the RAP source

Webgroup CZ a.s. Ex. 1015, Page 69 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can detect losses and sample the round-trip-time (RTT). To design a rate adaptation mechanism, three issues must be addressed [66]. These are the *decision function* , the *increase/decrease algorithm*, and the *decision frequency*.

4.1.1 Decision Function

The rate adaptation scheme can be summarized by its decision function as follow:

- If no congestion is detected, *periodically* increase the transmission rate;
- If congestion is detected, immediately decrease the transmission rate.

RAP considers losses to be congestion signals, and uses timeouts, and gaps in the sequence space to detect loss.

Similar to TCP, RAP maintains an estimate of RTT, called $SRTT$, and calculates the timeout based on the Jacobson/Karel's algorithm [1] However, it detects the timeout losses differently because RAP is *not* ack-clocked. Unlike TCP, a RAP source may send several packets before receiving a new ACK to update the RTT estimate. Therefore, a TCP-like timeout mechanism is not appropriate and results in detecting frequent incorrect losses that are in fact late ACKs. Thus RAP couples the timer-based loss detection to the packet transmission. The source maintains a record for each transmitted packet, containing the sequence number, departure time, transmission rate and status flag for each packet. The collection of records for outstanding packets is called the *transmission history*. Before sending a new packet, the source checks for a potential timeout among the outstanding packets using the updated value of the $SRTT$ estimate. Then it traverses through the transmission history and detects all the timeout losses using the following algorithm:

$$
\begin{aligned}
&WHILE(DepartTime_i + Timeout \leq CurrTime)\\
&\quad IF(Flag_i \neq Acked)\ THEN\\
&\qquad Seq_i\ is\ lost
\end{aligned}
$$

This mechanism may detect a burst of loss at once. Moreover because of the absence of

[1] $SRTT_{i+1} = \frac{7}{8}SRTT_i + \frac{1}{8}SampleRTT$, $Timeout = \mu * SRTT + \delta * VarSRTT$, $VarSRTT$ denotes variations of $SRTT$

Webgroup CZ a.s. Ex. 1015, Page 70 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ack-clocking, the RAP source may still receive a late ACK. Late ACKs are also used for updating different RTT estimates , i.e. SRTT, FRTT, XRTT.

The ACK-based loss detection mechanism in RAP is based on the same intuition as fast-recovery in TCP. To limit the amount of overshoot during the increase phase, the RAP source needs to detect congestion (i.e. packet loss) as early as possible. If a RAP source receives an ACK that implies delivery of three packets after the missing one, the packet is considered lost. RAP requires a way to differentiate the loss of an ACK from the loss of the corresponding data packet. We have added redundancy to the ACK packets to specify the last hole in the delivered sequence space and provide robustness against single ACK losses. [2]. An ACK packet contains the following information:

1. The sequence number, A_{curr}, of the packet being acknowledged,
2. The sequence number, N, of the last packet before A_{curr} that was still missing, or 0 if no packet was missing,
3. The sequence number, A_{last}, of the last packet before N that was received, or 0 if A_{curr} was the first packet.

Packet Loss Pattern	*ACK Information*			
	A_{last}	N	A_{curr}	
1 _	0	0	1	
1 _ _ 4	1	3	4	
1 _ _ 4 _ 6	4	5	6	A_6
1 _ _ 4 _ 6 _ _ 9	6	8	9	
_ 6 _ _ 9 10	6	8	10	
6 _ _ 9 10 11	6	8	11	

Figure 4.1: ACK-based loss detection in RAP

Figure 4.1 shows an example of how the information in the ACK packets are used to detect a packet loss. "_" in figure 4.1 denotes a missing packet. Using this information, a RAP sender can mark packets as having arrived even when the ACK is dropped because subsequent ACKs arrive in which their sequence number is greater than N and less than

[2]To achieve resilience to multiple ACK-loss, more information must be carried in ACK packets. We have not studied this issue any further

Webgroup CZ a.s. Ex. 1015, Page 71 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A_{curr}. Adding A_{last} provides sufficient redundancy in the ACK stream to recover loss of ACK_{i-1} when data packet with sequence number i, is lost. After each ACK arrives, the RAP source performs the following algorithm:

$$
\begin{array}{l}
\quad For\ each\ Seq_i\ in\ Trans.\ History \\
DO \\
\quad IF[((A_{curr} \geq Seq_i) AND (Seq_i > N))\ OR \\
\quad\quad (Seq_i = A_{last})]\ THEN \\
\quad\quad\quad Seq_i\ was\ received \\
\quad ELSE\ IF\ (A_{curr} - Seq_i \geq 3)\ THEN \\
\quad\quad\quad Seq_i\ is\ lost \\
WHILE(Seq_i \leq A_{curr})
\end{array}
$$

Note that the timeout mechanism is still required as a back up for critical scenarios such as a burst of loss.

4.1.2 Increase/Decrease Algorithm

RAP uses an AIMD increase/decrease algorithm. In the absence of packet loss, the transmission rate is periodically increased in a step-like fashion. The transmission rate is controlled by adjusting the inter-packet-gap(IPG). To increase the rate additively, IPG must be iteratively updated based on equation (1) [65]:

$$
\begin{aligned}
S_i &= \frac{PacketSize}{IPG_i} \\
IPG_{i+1} &= \frac{IPG_i * C}{IPG_i + C} \\
\alpha = S_{i+1} - S_i &= \frac{PacketSize}{C}
\end{aligned}
\tag{4.1}
$$

where S_i and α denote transmission rate and *step height* respectively. C is a constant with the dimension of time. We refer to the value of α as *step height*. Note that in the equation 4.1, C has the dimension of time and it determines the value of α. Upon detecting

Webgroup CZ a.s. Ex. 1015, Page 72 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

congestion, the transmission rate is decreased multiplicatively, by doubling the value of IPG We use a value of $\beta = 0.5$ which is a conservative choice to be similar to TCP:

$$\begin{aligned} S_{i+1} &= \beta S_i, \qquad 0 < \beta < 1 \\ IPG_{i+1} &= IPG_i/\beta \end{aligned} \tag{4.2}$$

4.1.3 Decision Frequency

Decision frequency specifies how often to change the rate. The optimal adjustment frequency depends on the feedback delay. The feedback delay is the time between changing the rate and detecting the network's reaction to that change. Feedback delay in ACK-based schemes is in the order of to one RTT. It is suggested that rate-based schemes adjust their rates not more than once per RTT [80]. Changing the rate too often results in oscillation whereas infrequent change leads to unresponsive behavior.

RAP adjusts the IPG once every $SRTT$ using equation 4.1. The time between two subsequent adjustment points is called a *step*. Because of the random nature of the RTT signal[16], using the recent sample RTT as the step length is likely to result in a poor behavior. We need a smoothed version of RTT that represents low frequency variation of RTT and filters out the transient (i.e. high frequency) changes. RAP uses the most recent value of SRTT as the step length. At the beginning of each step, a timer, called step-timer, is set to the recent value of SRTT and the IPG is decreased based on equation 4.1. The value of IPG remains unchanged until the step-timer expires or a packet loss occurs. If no loss is detected, IPG is decreased and a new step is started. Adjusting the IPG once every $SRTT$ has a nice property; packets sent during one step are likely to be acknowledged during the next step. This allows the source to observe the reaction of the network to the previous adjustment before making a new adjustment. Furthermore, choosing SRTT as the step length, the RAP source can adaptively adjust the slope of increase as congestion is formed or cleared up.

As we mentioned earlier, C in equation 4.1 has the dimension of time and is the only parameter that controls the rate of increase of the transmission rate. One immediate question is *"what is the right value for C?"*. Since our chief goal is to be TCP-friendly,

Webgroup CZ a.s. Ex. 1015, Page 73 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

C must be adjusted so that in the steady state, the number of packets transmitted per step is increased by one. Ideally, we want the slope of increase to be adaptively adjusted with characteristics of a connection such as RTT and volume of background traffic. Equation 4.1 allows us to achieve this goal. If the value of IPG is updated once every T seconds and we choose the value of C to be equal to T/k, the number of packets sent during each step is increased by k every step. If the value of IPG is updated once every $SRTT$ and we choose the value of C to be equal to $SRTT$, the number of packets sent during each step is increased by 1 every step [3].

RAP uses value of one for k in order to emulate the TCP window adjustment mechanism in the steady state. At each adjusting point, first the step length (i.e. the step timer) and the value of C are set to the recent value of $SRTT$, and then equation 4.1 is used to update the value of IPG.

Since the length of each step is $SRTT$ and the height of each step is inversely dependent on $SRTT$, the slope of the transmission rate is inversely related to $SRTT^2$.

$$\begin{aligned} Slope &= \frac{StepHeight}{StepLength} = \frac{\alpha}{SRTT} = \frac{PacketSize}{C * SRTT} \\ C = SRTT &\Rightarrow Slope = \frac{PacketSize}{SRTT^2} \end{aligned} \tag{4.3}$$

TCP's slope of linear increase is related to RTT in the same way in the steady state. Thus a RAP source can exploit RTT variations and adaptively adjust its rate in the same manner as TCP. The adaptive rate adjustment in RAP is meant to emulate the coarse-grain rate adjustment in TCP. The step length in RAP is analogous to the time it takes for TCP to send a full window worth of packets.

RAP is "unfair" to flows with longer RTT in the same way that inter-TCP unfairness has frequently been reported[40]. RAP connections with shorter RTTs are more aggressive and achieve a larger share of the bottleneck bandwidth. In general, other measures of fairness can be only achieved by implementing the required machinery in the network[117]. As long as the unfairness problem is not resolved among TCP flows, being TCP-friendly implies accepting this unfairness.

[3] See appendix C in [65] for justification of choosing $k = 1$ and $\beta = 0.5$ for TCP.

Webgroup CZ a.s. Ex. 1015, Page 74 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2 Auxiliary Mechanisms

Besides the three main components of the RAP protocol, there are several auxiliary mechanisms that are required to achieve the desired behavior. Here we describe *Clustered Losses* and *Fine grain Adaptation* as two added auxiliary mechanisms.

4.2.1 Clustered Losses

In a shared best-effort network with a high level of statistical multiplexing, the observed loss pattern has a near random behavior[16] that is determined by the aggregate traffic pattern. Thus it is generally hard for an end system to predict or control the loss rate by adjusting the transmission rate. End systems are required to react to congestion events instead of individual packet losses. It takes one RTT for end systems to detect and react to congestion. Once an end system reacts, it takes another RTT for the reaction to be effective. Thus an end-system only needs to react at most once per RTT as long as they react properly and promptly[80].

To achieve this, RAP requires a mechanism to identify a cluster of losses that are potentially related to the same congestion event. A simple approach is to ignore all losses that are detected during the first RTT after a back-off. RAP employs a slightly different approach. Right after loss of packet $Seq_{FirstLoss}$ that results in a back-off, the sequence number of the last packet that has been transmitted is called $Seq_{LastSent}$. The outstanding packets in the pipe, called a *cluster*, have a sequence number, Seq, within the following range:

$$Seq_{LastSent} \geq Seq > Seq_{FirstLoss} \tag{4.4}$$

Any packet in the cluster can be potentially dropped due to the recent congestion event that was detected by the loss of $Seq_{FirstLoss}$. As the source has already reacted to the congestion, loss of other packets from the cluster are silently ignored. This cluster-loss-mode is triggered by a back-off and terminated as soon as an ACK with sequence number

Webgroup CZ a.s. Ex. 1015, Page 75 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

greater or equal to $Seq_{LastSent}$ is received. This mechanism is similar to that employed in TCP-Sack.

4.2.2 Fine-Grain Rate Adaptation

TCP achieves some degree of congestion avoidance because of its ACK-clocking mechanism. As a result, TCP becomes very responsive to short-lived congestion events. If RAP performs only the coarse-grain AIMD rate adaptation on a per-RTT basis, it only reacts to packet losses. The main motivation for fine-grain rate adaptation is to emulate TCP's ACK-clock based congestion avoidance and make RAP more responsive to transient congestion while still performing the AIMD algorithm at a coarser granularity.

A short-term exponential moving average of the RTT captures short-term trends in congestion. However, we require a *dimension-less, zero-mean* feedback signal to be independent of the connection parameters and have wider applicability. The ratio of the short-term to the long-term exponential moving average of the RTT signal exhibits these desired properties. We have exploited the RTT signal and devised a *continuous* [4] feedback function that is defined as:

$$Feedback_i = \frac{FRTT_i}{XRTT_i} \tag{4.5}$$

where $FRTT_i$ and $XRTT_i$ are the value of short and long term exponential moving average of RTT samples respectively.

At each *tuning* point, the value of IPG_i is modulated by the fine-grain feedback signal and the resulting value, IPG'_i, is used for the transmission timer:

$$IPG'_i = IPG_i * Feedback_i \tag{4.6}$$

The value of IPG is adjusted once per step iteratively and as we explained earlier, and acts as a *base* transmission rate. Thus, during one step the base transmission rate remains unchanged. However, the actual inter-packet-gap, IPG', adaptively varies with the

[4]Since our proposed feedback function is continuous, we do not need to deal with specifying thresholds which can be problematic.

Webgroup CZ a.s. Ex. 1015, Page 76 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

short-term congestion state. Note that the fine-grain feedback does not have a cumulative effect. Fine-grain rate adaptation could be performed at several granularities, however the feedback signal and the rate adjustment must have the same granularity. We focused our attention on the per-ack scheme since it has the finest granularity. We have simulated fine-grain adaptation for two extreme granularities: adapting once per step , and adapting once per ACK. Although both schemes result in improvements, we focused our attention on the per-ack scheme since it has finer granularity, performs slightly better, and is intuitively closer to TCP. We plan to study coarser adaptation schemes by relaxing some restrictions of the per-ack scheme in our future work.

4.2.2.1 Per-step Fine-grain Rate Adaptation

This scheme has the granularity of one step. The fine-grain and coarse grain rate adjustment occur both at the same time. At the beginning of the ith step, first IPG_i is calculated based on the equation 4.1. Then the updated value of the feedback is applied (using the equation 4) to obtain the value of $IPG'i$. The rate remains unchanged for the entire step. The feedback must capture the trend in congestion that was observed during the last step. Since number of sample RTTs (i.e. received ACK) during one step varies, weight of exponential filters, K_{XRTT} and K_{FRTT}, must be accordingly adjusted for each step. We have adjusted the weights based on the following observation; The initial value of the FRTT at the beginning of each step contributes only 10% in the value of the FRTT at the end of that step whereas for XRTT the initial value contributes 90% in the final value. It means:

$$\begin{aligned} (1 - K_{FRTT})^n &= 0.1 \\ (1 - K_{XRTT})^n &= 0.9 \end{aligned} \tag{4.7}$$

Where n presents number of outstanding packets in the pipe that are expected to receive during the current step. The idea in K_{FRTT} adjustment is to give a higher weight to the recent RTT samples since they are more correlated with recent congestion state.

Webgroup CZ a.s. Ex. 1015, Page 77 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.2.2 Per-ack Fine-grain Rate Adaptation

With per-ack adaptation, the arrival of a new ACK causes FRTT and XRTT to be updated, and so hence a new value of IPG' is calculated and applied to the timer. Note that the transmission timer is already running for the next transmission, and so the new IPG' will be used as the next inter-packet gap. Since multiple ACKs may arrive before expiration of the transmission timer, the most recent value of IPG' is used.

The weights for XRTT and FRTT exponential average filters must be chosen so that the feedback signal captures the short-term congestion state since the last ACK. We have set the value of K_{XRTT} and K_{FRTT} to 0.01 and 0.9 respectively. These weights are constant. The per-ACK feedback mechanism is the finest granularity for rate adjustment. It is expected to be more responsive to a congestion event than the per-step scheme.

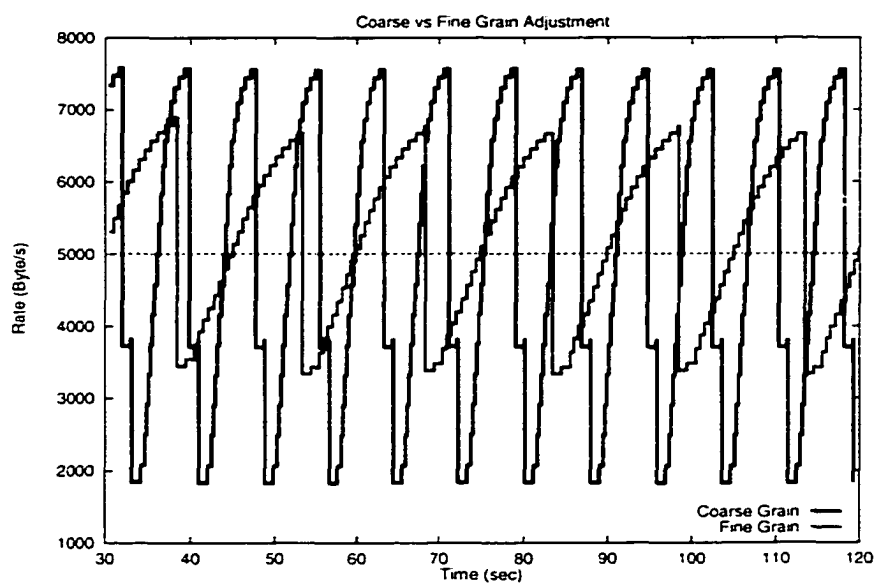


Figure 4.2: Effect of per-packet fine-grain rate adaptation on transmission rate of a single RAP flow

Figure 4.2 shows the effect of per-ack fine-grain adaptation on transmission rate of a single RAP flow. It clearly illustrates that RAP with fine-grain adaptation becomes less-aggressive(i.e. smaller overshoot) although it still follows the AIMD algorithm for coarse grain adaptation.

A TCP source adjusts its instantaneous transmission rate on a per packet basis and this reaction would only affect the next packet. A RAP source, however reacts only once

Webgroup CZ a.s. Ex. 1015, Page 78 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

every RTT (i.e. step) but it adjusts its rate based on the overall trend in congestion during the last step. The reaction only affects the next step. In other words, rate adjustment in RAP is equivalent to the average per-packet rate adjustments of a TCP flow during that interval. We recall that the length of each step is long enough (i.e. SRTT) that at the end of the ith step, a RAP source can detect the reaction of the network to its behavior during the $(i - 1)$th step before adjusting the rate for the $(i + 1)$th step. Transient congestion does not cause a major change (if any) in the value of FRTT. But if the congestion persists, the value of FRTT increases. In this case, although the value of IPG decreases linearly, the actual transmission rate is directly specified based on the recent level of congestion (i.e. FRTT). If the RTT signal exhibits an oscillatory behavior, FRTT will oscillate with a lower frequency. As a result, a RAP flow can cope with fluctuating background traffic. However, in a network with a large number of users, queue lengths along a path remain relatively constant due to statistical multiplexing among large numbers of flows.

4.3 Self-limiting Issues in RAP

Self-limiting behavior is the classic problem with rate-based schemes. In window-based schemes the source stops once it has a full window worth of data on the fly. This property makes the window-based schemes intrinsically stable. However, if the source allows retransmission beyond the current window, the stability is lost, and so the number of retransmitted packets must also be limited. Rate-based schemes need to find some variant analogous to the window to bound the volume of outstanding data in the network[62]. One way to achieve this goal is use of correctly implemented timers. In the absence of any feedback, the expired timer forces a source to drop its rate.

RAP achieves self-limiting by using the timeout mechanism for loss detection as a variant of window. In an extreme case when no ACK is received, the transmission rate drops to half once every RTT, until the rate falls below the minimum rate that the application can tolerate. This worst case scenario only happens if a connection fails.

The fine-grain rate adaptation mechanism in RAP effectively strengthens the self-limiting property in RAP and prevents the source from over running the network. During

Webgroup CZ a.s. Ex. 1015, Page 79 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

normal operation of a RAP connection in a network with a large number of well-behaved flows, the departure rate of data packets and arrival rate of ACKs are in balance. If the RTT suddenly increases, the arrival rate of ACKs decreases. therefore the number of out-standing packets grows and the balance would be lost. If a loss is detected, the rate is dropped to half. Otherwise since the values of new RTT samples is growing, the fine-grain feedback increases the value of IPG' effectively and controls the transmission rate.

4.4 Random Early Drop Gateways

There seems to be general agreement in the community on deploying Random Early Drop (RED)[43] gateways to improve both fairness and performance of TCP traffic. RED queue management tries to keep the average queue size low and, by preventing the buffer from overflowing, it also accommodates bursts of packets. One of the main problems for TCP's congestion control is to recover from multiple losses within a window [35]. This occurs mainly due to buffer overflow in drop-tail queues. Ideally, RED should be configured such that each flow experiences at most one single loss per RTT. Under these circumstances, TCP flows can efficiently recover from a single loss without experiencing a retransmission timeout. Intuitively, as long as a RED gateway operates in its ideal region, RAP and TCP obtain an equal share of bandwidth since both use the AIMD algorithm. Nevertheless, if the average queue length exceeds the maximum threshold, RED starts to drop packets with a very high probability. At this point, RAP and TCP start to behave differently. When regular TCP experiences multiple losses within a window, it undergoes a retransmission timeout and its congestion control diverges from the AIMD algorithm. RAP, however, follows the AIMD algorithm and reacts only once to the first loss in an RTT.

We expect to observe substantial improvement in fairness by deploying RED even if it only prevents the buffer from overflowing and causing burst of loss. This behavior limits the divergence of TCP's congestion control from the AIMD algorithm.

Since RED parameters are closely dependent on the behavior of aggregate traffic, it is hard to keep a RED gateway in its ideal region as the traffic changes with time. Thus, configuration of RED is still a research issue[37].

Webgroup CZ a.s. Ex. 1015, Page 80 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5 Startup Phase

Similar to slow-start in TCP, RAP needs a mechanism to quickly explore availability of bandwidth during the startup phase. However there is a trade-off: the more aggressive the rate is increased during the startup phase, the faster the source finds out about the available bandwidth, but the larger the overshoot it causes over its fair share.

In the context of long-lived sessions, performance of the startup phase is not crucial because the duration of this phase is negligible in comparison to the session length. However, both short-lived and interactive sessions need to detect available bandwidth quickly. The challenge is that a source that starts transmitting data or resumes transmitting after a long delay does not have any information about the level of congestion in the network. Thus, in the absence of any support from the network, there is no alternative approach but to experience the startup phase. The above trade-off exists for all the end-to-end congestion control mechanisms.

Using exponential increase during the startup phase of a rate-based congestion control such as RAP is more harmful than window-based scheme such as TCP. Window-based schemes can effectively limit the size of the resulting overshoot at the end of the startup phase because they directly control number of packets on the flight. However, the resulting overshoot could be larger for a rate-based scheme. Another alternative is to simply increase the slope of linear increase by proper adjustment of C during the startup phase. The startup phase is ended when first backoff occurs. We have not carefully studied the effect of increase algorithm during the startup phase. We believe that these trade-offs are rather generic for all end-to-end congestion control approach..

4.6 Simulations

In this section we present a summary of our simulation results. Our main goal is to explore the properties of RAP, namely TCP-friendliness, ability to cope with background TCP traffic, interaction with both drop-tail and RED gateways and the behavior of the fine-grain rate adaptation over a reasonable parameter space. Our simulations demonstrate that RAP is in general TCP-friendly.

Webgroup CZ a.s. Ex. 1015, Page 81 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We have simulated RAP using the ns2 simulator [6], and compared it to TCP Tahoe, Reno, NewReno [35], Sack [83] and also run real-world experiments. Fig. 4.3 shows the topology of our simulations. The link between SW_1 and SW_2 is always the bottleneck and SW_1 is the bottleneck point. All the other links have higher bandwidth and shorter delay than the bottleneck. The switches implement FIFO scheduling and drop-tail queuing except in RED simulations. m RAP connections from sources $R_1...R_m$ to receivers $P_1...P_m$ share the bottleneck bandwidth with n TCP flows from sources $T_1...T_n$ to receivers $S_1...S_n$. Data and ACK packet sizes are similar for RAP and TCP flows. For a fair comparison, all connections have equal end-to-end delay. The total delay of the side

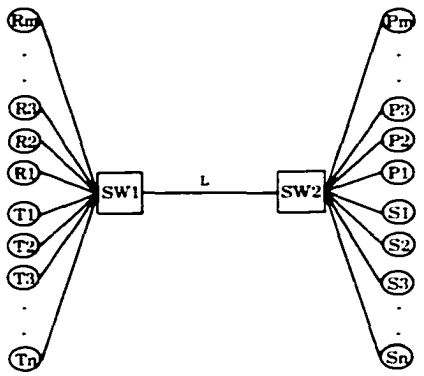


Figure 4.3: Simulation Topology

links for each flow is fixed, but is randomly split between $R_i \rightarrow SW_1$ and $SW_2 \rightarrow P_i$. The buffer size at SW_1 is four times the RTT-bandwidth product of the bottleneck link, except where otherwise stated. All simulations were run until they exhibited steady state behavior. All TCP flows are "FTP" sessions with an infinite amount of data. The TCP receiver window is large enough that TCP flow control is not invoked. TCP and RAP sources start in a random order with a uniform random delay between their start times. This random delay lessens the resonance between sources and reduces the duration of the initial transition phase. The average bandwidth for each flow is measured by the number of delivered packets during the last three quarters of the simulation time to ignore transient startup behavior. Simulation parameters are summarized in table 1. The average

Webgroup CZ a.s. Ex. 1015, Page 82 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Parameter	Value
Packet Size	100 Byte
ACK Size	40 Byte
Bottleneck Delay	20 ms
Bottleneck Buffers	4 * bottleneck B/W * RTT
B/W per Flow	5 KByte/s
B/W of Side Links	1.25 MByte/s
Tot. Delay of Side-Links	6 ms
Simulation Length	120 sec
TCP Maximum Window	1000
TCP Timeout Granularity	100 ms
TCP Maximum Window	1000

Table 4.1: Simulation setup for RAP evaluation

bandwidth for each flow is measured by the number of delivered packets [5] during the last three quarters of the simulation time to ignore transient startup behavior. Since loss only occurs at the bottleneck switch, the average goodput represents the average bottleneck bandwidth share for each flow. Typically we will graph the mean, min and max value for average bandwidths of the flows of each protocol to examine fairness.

4.6.1 Phase Effect

We initially observed severe phase effect phenomena in our simulations. Phase effects becomes more pronounced as the number of flows and the amount of resources (i.e. buffer size and bandwidth) increase. This occurs mainly because of Drop Tail gateways in our small deterministic network as was reported in [42]. Moreover, in our simulations all flows have the same packet size and observe similar RTT, which increases the probability of phase effects.

To eliminate this problem without changing our parameters, we have added a small uniform random delay before transmission of each TCP packet[6]. This delay ranges from

[5] Here we implicitly assumed that number of duplicate packets a TCP flow receives is negligible and our experiments confirm this assumption.

[6] This is possible by using *overhead* configuration parameter of TCP agent in ns.

Webgroup CZ a.s. Ex. 1015, Page 83 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

zero to the bottleneck service time and emulates the random packet-processing time of intermediate gateways [42] [7].

Obviously, adding this random delay slightly decreases the transmission rate of TCP because it always delayed the transmission. We have added similar randomness to RAP, not only to resolve the phase effect problem between RAP flows, but also compensate for the added delay to TCP flows. Each RAP source adds a random delay ranges from zero to the bottleneck service time, to the value of IPG' before scheduling transmission of the next packet[8]. In a real network, adding randomness is more crucial for RAP than TCP. Because of the ack-clocking and random change in RTT, TCP experiences some randomness. Since RAP is not ack-clocked, the RAP source needs to slightly randomize the IPG' to resolve the phase effect problem. Our fine-grain feedback seems to achieve this goal. Another solution to the phase effect problem is to replace the DropTail gateway with a RED gateway. We have explored this in our simulations as we report later. We believe that the phase effect problem deserves more attention. We plan to investigate the phase effect among RAP flows as well as phase effects between TCP and RAP flows in our future work.

4.6.2 Evaluation Methodology

In an environment with large numbers of parameters, it is generally hard to isolate a particular variable and study its relation with a particular parameter because of existing interdependency among variables. In particular, TCP is a moving target. Its behavior changes drastically with configuration parameters and it has some internal constraints. Therefore, it is crucial to distinguish an effect that is caused by TCP's performance constraints from those phenomena that are due to coexisting RAP flows. During our simulations, with some exceptions, we attempted to minimize these problems by using the following guidelines:

1. To identify the impact of TCP's constraints from the inter-protocol dynamics on our results, we have compared RAP with different flavors of TCP.

[7] Although adding the randomness substantially lessens the phase effect problem, the problem still occurs occasionally in big simulations.

[8] Although adding the randomness substantially lessens the phase effect problem, the problem still occurs occasionally in big simulations.

Webgroup CZ a.s. Ex. 1015, Page 84 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. We limited the side-effect of bottleneck bandwidth and buffer space contention by scaling up resources proportional to the number of flows so that the amount of resource share per flow remains fixed across simulations. Since the bandwidth and the buffer size of the bottleneck link are scaled up equally, the maximum queuing delay does not change across simulations. The impact of resource contention is also studied separately.

3. We chose configuration parameters so that the TCP congestion window tends to be sufficiently large and TCP remains in its well-behaved mode.

4. We have explored a reasonable portion of the parameter space to examine inter-protocol fairness over a wide range of circumstances.

5. As a baseline for comparison, we occasionally replaced all the RAP flows with TCP and ran the same scenario without any RAP flow. We call this *TCP base-case*. The TCP base case may help us to separate those phenomenon that are purely related to TCP traffic.

4.7 Experiments and Results

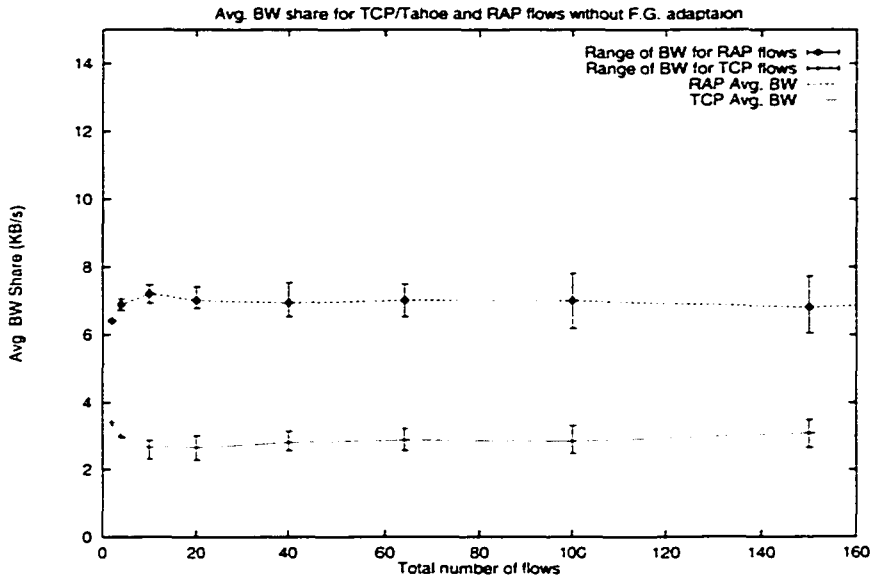
We have conducted a large number of simulations to evaluate different aspects of the RAP protocol. Each of these groups of simulations are presented in this section.

4.7.1 TCP-friendliness

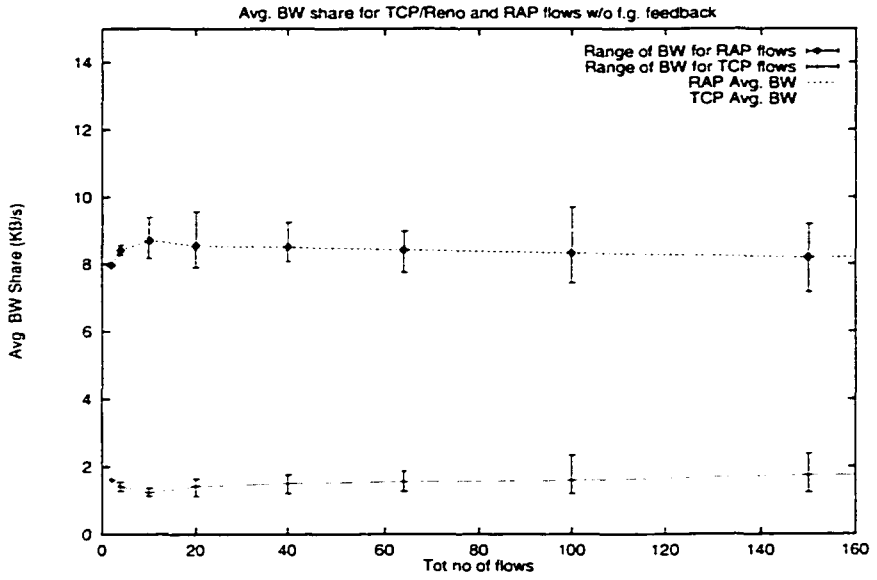
The first set of simulations examines the TCP-friendliness of RAP without fine-grain rate adaptation.

Fig. 4.4(a) shows the average bandwidth share of n RAP and n TCP Tahoe flows coexisting over the topology depicted in fig. 4.3. The resources (i.e. the bottleneck bandwidth and the buffer size) are scaled up linearly with the total number of flows. The range of the bandwidth share among RAP and TCP flows are represented by vertical bars around the

Webgroup CZ a.s. Ex. 1015, Page 85 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) RAP coexisting with Tahoe



(b) RAP coexisting with Reno

Figure 4.4: Comparison of RAP with TCP(Tahoe and Reno)

Webgroup CZ a.s. Ex. 1015, Page 86 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

average value. This result implies that RAP is not terribly TCP-friendly across these simulations. The observed unfairness can be due to TCP's inherent performance limitations, an artifact of configuration parameters, or unfairness imposed by coexisting RAP flows.

TCP suffers from some performance limitations[35]. In particular, when TCP experiences multiple losses within a window or the window is smaller than 4, it is constrained to either wait for retransmission timeout or go through slow-start. As a result, TCP may temporarily lose its ack-clocking and its congestion control mechanism diverges from the AIMD algorithm. The severity of the problem varies among different flavors of TCP and mainly depends on window size and loss patterns. TCP Sack is able to recover from the multiple loss scenarios easier than other flavors of TCP whereas Reno's performance is substantially degraded [35]. Generally, TCP's ability to efficiently recover from multiple losses increases with its window size. The more TCP diverges from the AIMD algorithm, the less bandwidth it obtains.

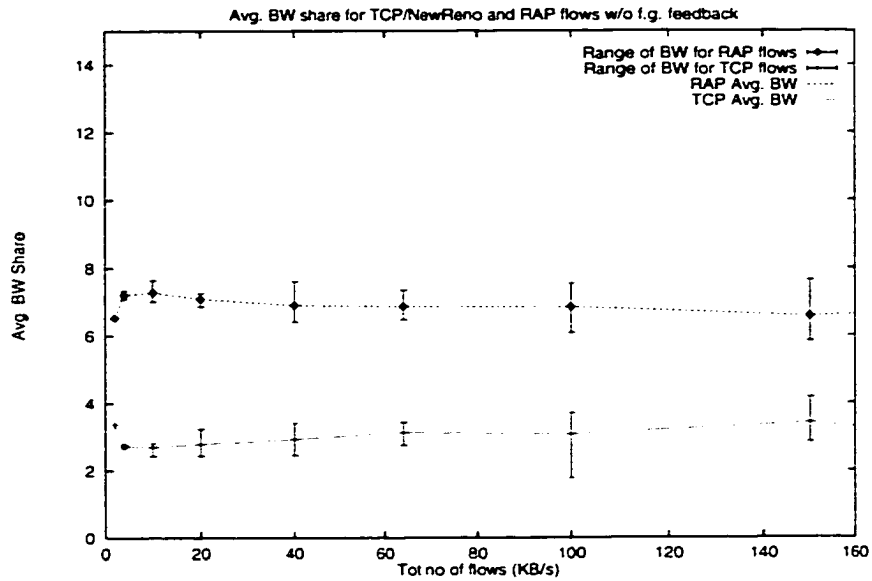
We exploited the difference among various TCP flavors to assess the impact of TCP's performance problem on the observed unfairness. We have repeated the same experiment with RAP against Reno, NewReno[35] [9], and Sack TCP. Results are shown in Figure 4.4(b), 4.5(a), 4.5(b) respectively. Figure 4.5(b) show the same experiment with TCP Sack. Our results confirm that the large-scale behavior of TCP traffic is in agreement with the behavior reported in [35]. These experiments also reveal that TCP's inherent performance problems partially contribute to unfairness. It is interesting to notice that the inter-protocol fairness remains unchanged across all simulations [10].

We would like to limit the impact of the TCP's performance problems and focus on the interaction between RAP and TCP traffic. Therefore, we chose TCP Sack as an ideal representative for TCP flows. For the rest of this paper whenever we refer to TCP, we mean TCP Sack unless explicitly stated otherwise. To attempt to ensure that we have not chosen an unrepresentative set of parameters, we have explored a wide range of different values.

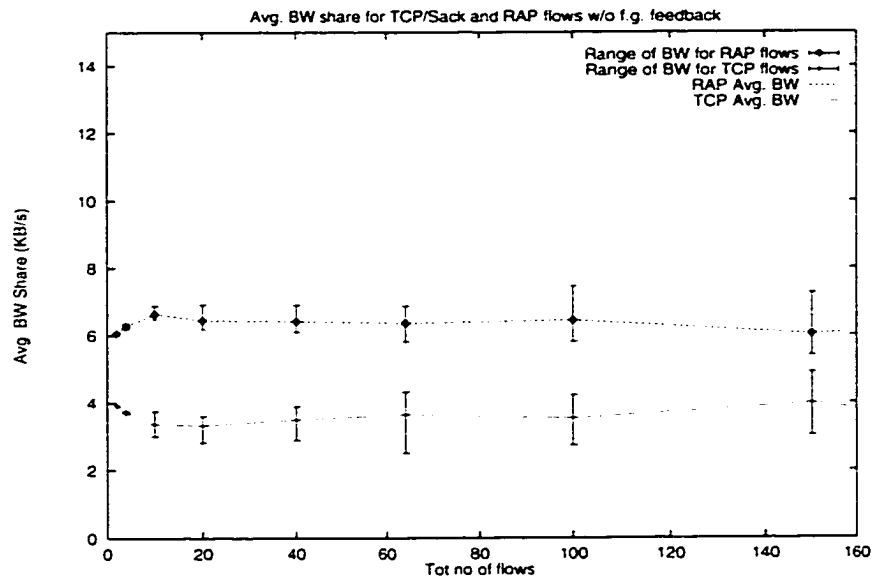
[9]NewReno is a modified version of Reno TCP that avoids some of the Reno's performance problems. For more details, refer to [35]

[10]Notice that some of our results seem to have minor phase effect, e.g. for 100 flows in figure 4.4(b).

Webgroup CZ a.s. Ex. 1015, Page 87 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) RAP coexisting with NewReno



(b) RAP coexisting with Sack

Figure 4.5: Comparison of RAP with TCP(NewReno and Sack)

Webgroup CZ a.s. Ex. 1015, Page 88 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Since we are unable to exhaustively examine the parameter space, we focus our attention on parameters that play key roles in protocols' behavior. RTT and TCP's congestion window are particularly important. RTT is crucial because it affects rate adjustment in both RAP and TCP. TCP's congestion window is a primary factor in the performance of the TCP protocol. We introduce the term *inter-protocol fairness ratio* that is the ratio of the average RAP bandwidth calculated across all the RAP flows over the average TCP bandwidth calculated across all the TCP flows. We changed the delay of the bottleneck link to control the value of RTT. The bandwidth was linearly scaled up with the total number of flows and the buffering was adjusted accordingly. Other parameters are the same as table 1. Fig. 4.6(a) depicts the fairness ratio as a function of the bottleneck link delay and the total number of flows. Figure 4.6(b) provides the side view (from the delay axis) of figure 4.6(a) for easier comparison. Each data point is obtained from an experiment where half of the flows are RAP and the other half are Sack TCP.

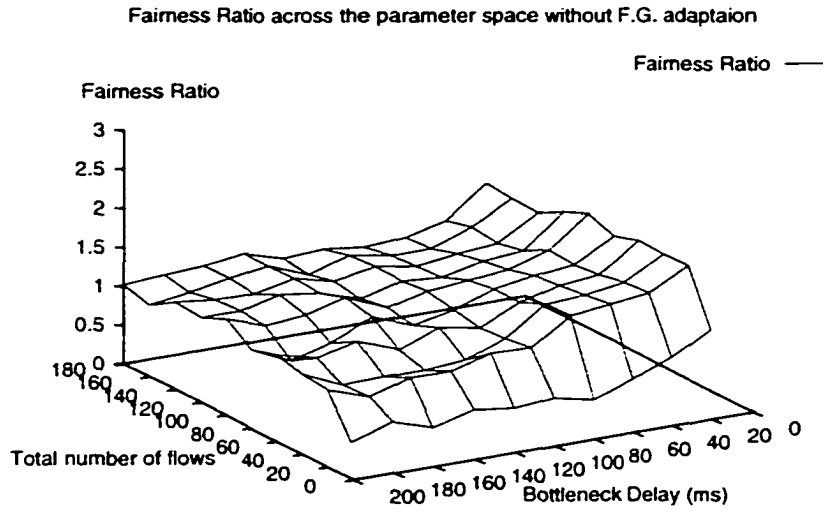
This reveals several interesting trends in the fairness ratio:

For a particular value of the bottleneck delay, increasing the number of flows improves the fairness ratio except for the smallest value of delay (20ms) in which the ratio never converges to one. This special case is, in fact, the problem that we have observed in the previous section. This figure illustrates that except for small simulations, RAP exhibits TCP-friendly behavior. The different behavior in small simulations has to do with TCP's burstiness and loss pattern in these scenarios We will address these problems shortly.

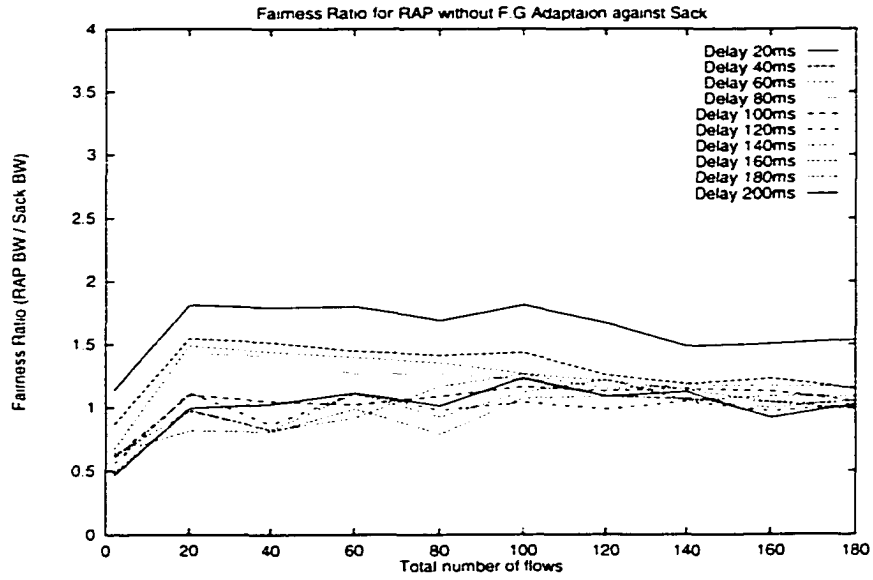
Excluding simulations with a small bottleneck delay as well as small simulations, the fairness ratio is mostly close to one and is not a function of the RTT. The problem with short bottleneck delay in small simulations has to do with the small size of TCP's congestion window. In these scenarios, TCP has a smaller congestion window and frequently experiences retransmission timeout. As the bottleneck delay increases, both the bottleneck pipe size and the buffer size increase [11]. This allows TCP flows to have a larger number of packets on-the-fly and maintain their ack-clocking.

[11] Note that we keep the ratio of the buffer size to the pipe size (i.e. link bandwidth, delay product) fixed for the bottleneck link across the simulations. Thus the maximum queuing delay increases with the bottleneck delay. This in turn, could further increase the average RTT depends on the behavior of the aggregate traffic.

Webgroup CZ a.s. Ex. 1015, Page 89 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Fairness Ratio across the parameter space



(b) Side view of the previous graph

Figure 4.6: Exploring the inter-protocol fairness across the parameter space

Webgroup CZ a.s. Ex. 1015, Page 90 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

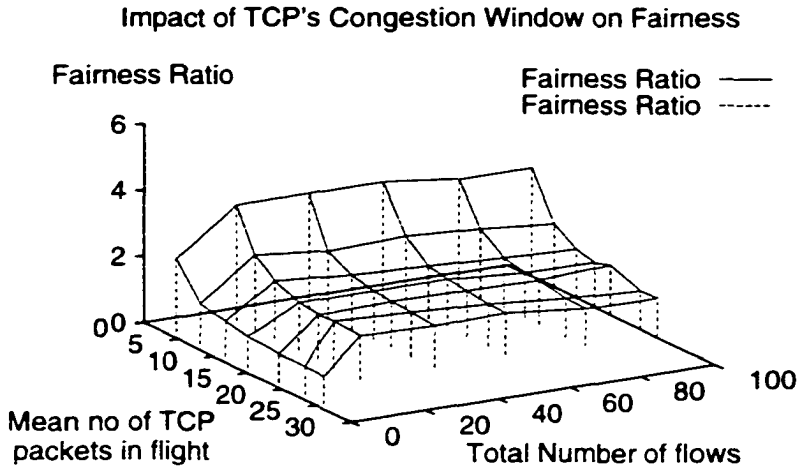
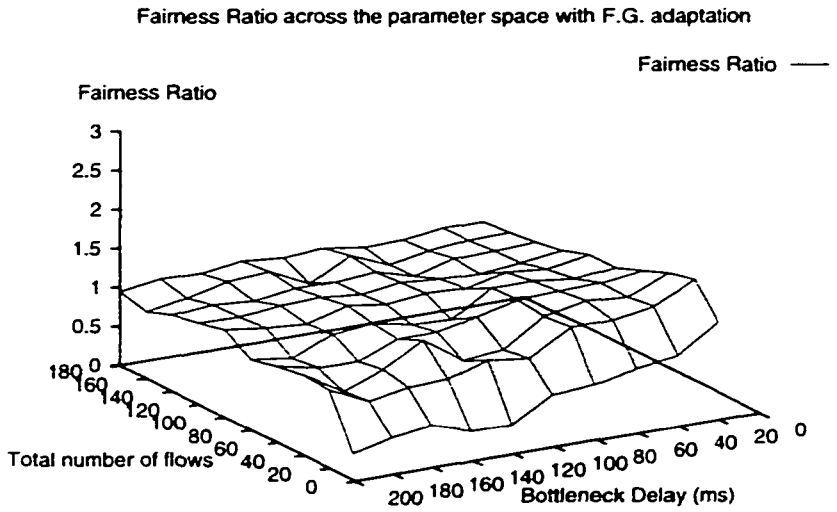


Figure 4.7: Variation of the Fairness ratio with TCP's congestion window

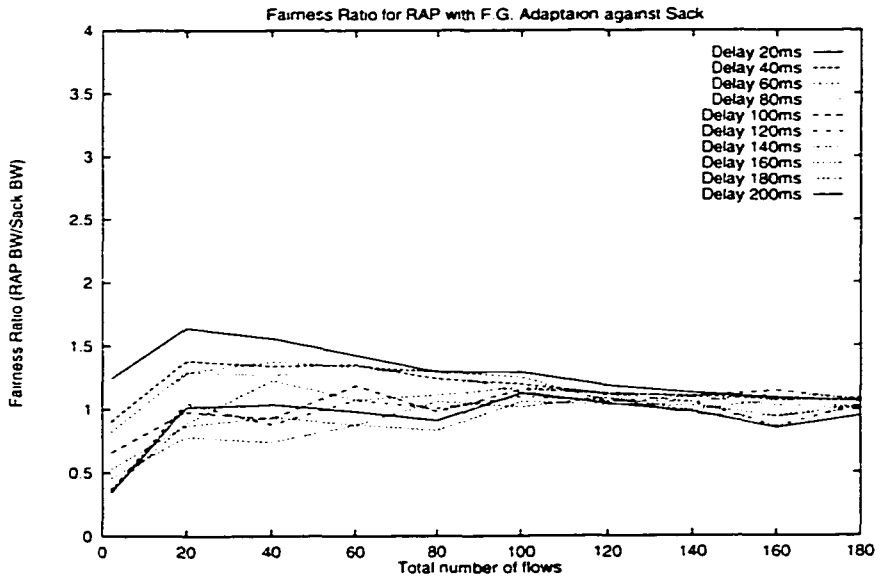
We conducted another set of simulations to observe the primary effect of TCP's congestion window on the fairness ratio. The congestion window is dependent on several parameters such as available bandwidth per flow, buffer size, mean queue size, queue management scheme and number of flows. We adjust the bottleneck bandwidth as a primary factor to control the value of congestion window. We decided to measure the number of outstanding TCP packets per flow instead of congestion window for two reasons. Firstly, TCP's congestion window may not be full during the fast-recovery period. In those cases, TCP's behavior depends on the number of outstanding packets. Secondly, since RAP is not a window-based mechanism, the number of packets on-the-fly seems to be the only common base of comparison from the network's point of view. Fig. 4.7 shows the variation of the fairness ratio as a function of the number of flows and the amount of allocated bandwidth per flow. Since the number of outstanding packets is dependent on both variables, we have used the mean number of outstanding packets (averaged across all the TCP flows in a simulation) as the x coordinate for the corresponding data point instead of the amount of allocated bandwidth per flow [12]. This graph clearly confirms our hypothesis that TCP's performance is directly influenced by the number of outstanding packets in transit. As the number of outstanding packets grows, the fairness ratio improves except

[12]This slightly changes alignment of the graph.

Webgroup CZ a.s. Ex. 1015, Page 91 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Fairness Ratio across the parameter space



(b) Side view of the previous graph

Figure 4.8: Exploring the inter-protocol fairness across the parameter space

Webgroup CZ a.s. Ex. 1015, Page 92 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for simulations with a small number of flows (n=1). Therefore, under a heavy load, if the number of outstanding packets for a TCP flow drops below a threshold, its performance is substantially degraded. Under these circumstances, RAP can easily utilize the available bandwidth because it decouples congestion control from error control and only performs the former.

Fig. 4.7 also implies that the number of coexisting flows does not have a visible impact on fairness when resources are scaled appropriately, except for very small numbers of flows.

4.7.2 Fine-grain Rate Adaptation

We have theorized that fine-grain rate adaptation attempts to emulate a degree of congestion *avoidance* that TCP obtains due to ack-clocking. To investigate the effect of fine-grain rate adaptation on TCP-friendliness, we explored the parameter space over a wide range. Fig. 4.8(a) shows the fairness ratio as a function of bottleneck link delay and the total number of coexisting flows. Half of the traffic consists of RAP flows. Comparison with fig. 4.6(a) reveals that fine-grain rate adaptation only improves the fairness among connections with small RTT (i.e. small TCP window) while it does not affect other areas. This result implies that as long as TCP flows do not diverge from the AIMD algorithm, the fairness ratio is primarily determined by TCP's behavior and the large-scale behavior remains intact. This is indeed a desired property. However, for those scenarios where TCP traffic is vulnerable to loss of ack-clocking and achieves a smaller share of the bandwidth, the fine-grain rate adaptation enhances resolution of rate adaptation for RAP flows by preventing them from overshooting the available bandwidth share. This in turn, reduces the probability of experiencing loss of ack-clocking across all the TCP flows. Consequently, TCP traffic obtains a fair share of bandwidth.

4.7.3 Burstiness

We have observed two special cases where inter-protocol fairness was not achieved that have not been addressed yet. These cases are discussed in this section separately.

Webgroup CZ a.s. Ex. 1015, Page 93 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The first special case occurs in simulations with a relatively small number (10 to 50) of flows over a bottleneck link with a small delay value(Figure 4.8(a)). Although this scenario is not usually exercised over the Internet because of the high level of statistical multiplexing, the problem still deserves attention since RAP might be deployed over an ISDN line where it coexists with a small number of TCP flows.

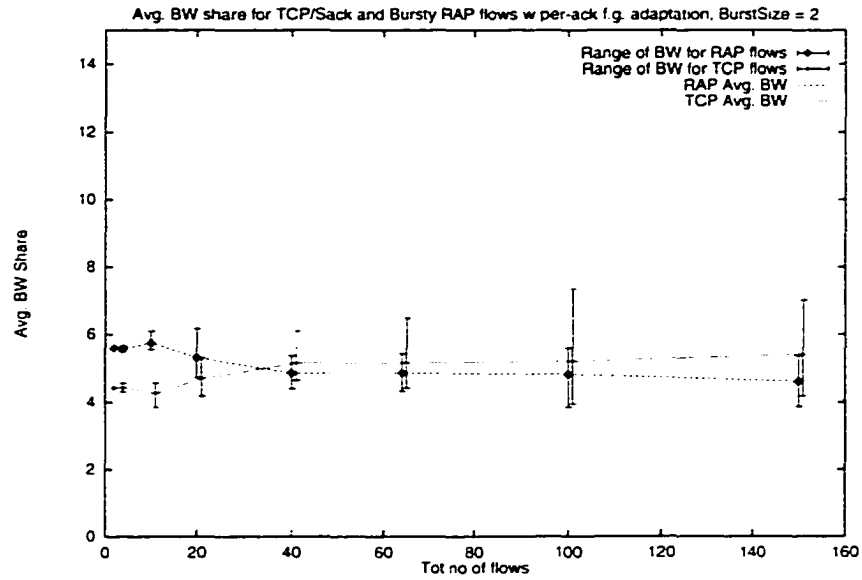


Figure 4.9: Bursty RAP with per-ack fine grain adaptation against Sack

The chief reason for the unfairness in these scenarios is interaction between TCP's burstiness and DropTail queue queues. Since TCP's behavior is more bursty than RAP, TCP has a greater probability of experiencing loss due to buffer overflow. These losses tend to be bursty in small simulations because of the DropTail queues and lack of sufficient statistical multiplexing. Thus TCP experiences more backoff and obtains less share of bandwidth.

To observe the effect of burstiness, we changed RAP to send a burst of size b packets every $b * IPG'$ seconds. Figure 4.9 shows the average bandwidth share of n TCP Sack flows coexisting with n bursty RAP flows and 20ms bottleneck delay. The RAP flows perform fine-grain rate adaptation and the burst size is 2 packets. The resources are linearly scaled up as the number of flows increases. This graph demonstrates two points. Firstly, adding burstiness to RAP's behavior increases the probability of loss and experiencing

Webgroup CZ a.s. Ex. 1015, Page 94 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

back off. Secondly, as the simulation size grows and the level of statistical multiplexing increases, TCP's burstiness gradually disappears and TCP's performance improves. However, RAP's burstiness does not depend on the observed level of statistical multiplexing, and so as we move toward bigger simulations, TCP gradually outperforms this bursty RAP.

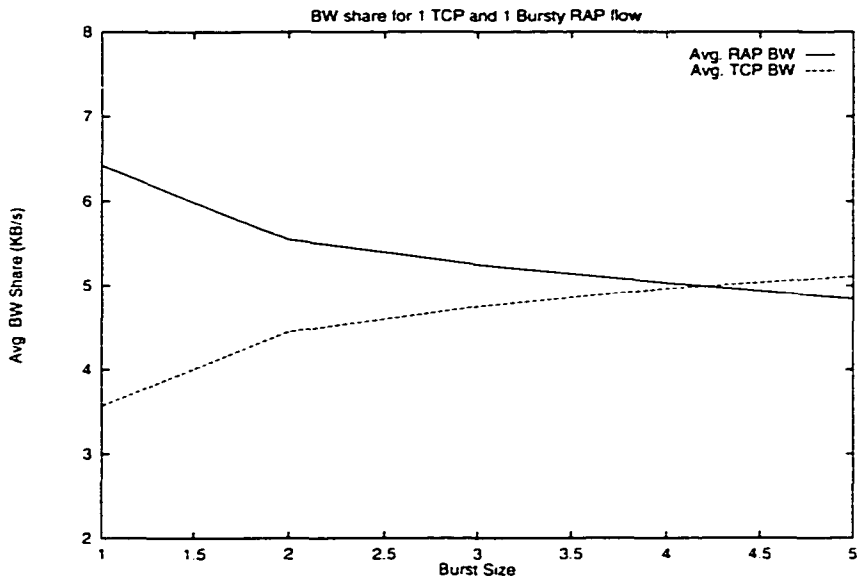


Figure 4.10: Impact of burstiness on RAP's behavior

We have conducted another set of simulations to assess the impact of burst size on performance. We have repeated the previous simulation for one bursty RAP against one TCP flow for different burst sizes. Figure 4.10 illustrates the average RAP and TCP bandwidth for these scenarios as the burst size increases. This figure clearly illustrates that burstiness is harmful to performance with DropTail queues.

The second special case is observed when the fairness ratio among a very small number of flows monotonically decreases as the bottleneck delay increases. This phenomenon was observed in figure 4.8(a) and 4.6(a) in simulation with two flows($n = 1$) as the bottleneck delay changes. We do not have enough evidence to provide a solid explanation for this case. However, we speculate that this has to do with impact of increasing buffer size on TCP's behavior. In the context of small simulations, if the buffer size at the bottleneck increases, TCP's burst can be absorbed at the bottleneck. This leads to a higher share of bandwidth for TCP.

Webgroup CZ a.s. Ex. 1015, Page 95 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

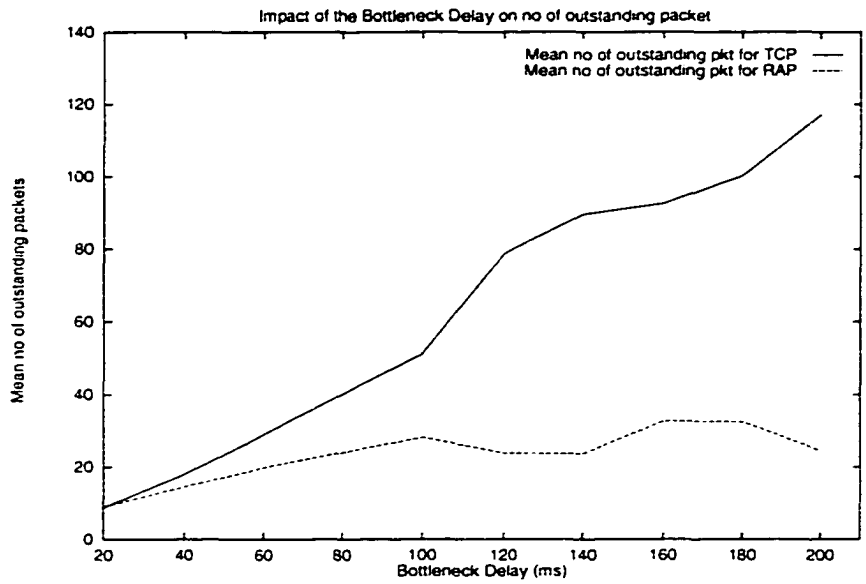


Figure 4.11: Mean number of outstanding packets for one RAP and one TCP

Figure 4.11 depicts the number of outstanding packets for RAP and TCP flows in the same set of simulations. The buffer size linearly increases with the bottleneck delay because we scale up the buffer size proportionally. It remains four times of the bottleneck pipe across our simulations. This figure reveals that the number of outstanding packets for TCP rapidly increases with buffer size while it remains unchanged for RAP. As the buffer size increases, TCP manages to rapidly inflate its window and obtain a bigger share of the buffer. RAP is not as sensitive to buffer size as TCP because of its smooth transmission. Thus it simply operates with the left over portion of the bottleneck buffer. This special case requires further investigation.

4.7.4 RED Gateways

The main challenge here was to configure the RED gateway so that it behaves uniformly across all simulations. RED's performance closely depends on the behavior of the aggregate traffic. Since this behavior could change with the number of flows, it is hard to obtain the same performance over a wide range without reconfiguring the gateway. Table 4.2 summarizes our configuration parameters: Half of the traffic consists of RAP flows with fine-grain adaptation. We provided sufficient buffer at the bottleneck to eliminate buffer

Webgroup CZ a.s. Ex. 1015, Page 96 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Parameter	Value
Min. Threshold	5 Packets
Max. Threshold	0.5 * Buffer
Bottleneck B/W	5 KByte/s * No. of Flows
Bottleneck Delay	20 ms
Buffer Size	12 * RTT * Bottleneck B/W
q_weight	0.002

Table 4.2: RED configuration for RAP evaluation

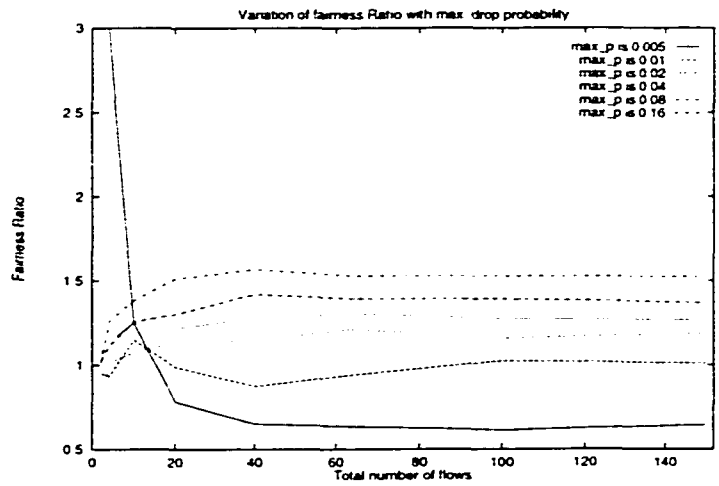


Figure 4.12: Impact of RED on the fairness

overflow. Fig. 4.12 shows the fairness ratio for different value of max_p (i.e. maximum probability of loss) as the number of flows changes. This graph clearly illustrates three interesting points:

1. There exists a range for max_p where RAP and TCP evenly share the bottleneck bandwidth.

2. Except for small simulations, the fairness ratio does not change with simulation size.

3. The behavior of the aggregate traffic is substantially different in small simulations.

Fig. 4.12 demonstrates that RED is able to evenly distribute the losses across all the flows and avoid buffer overflow over a wide range. Thus RED has eliminated the unfairness

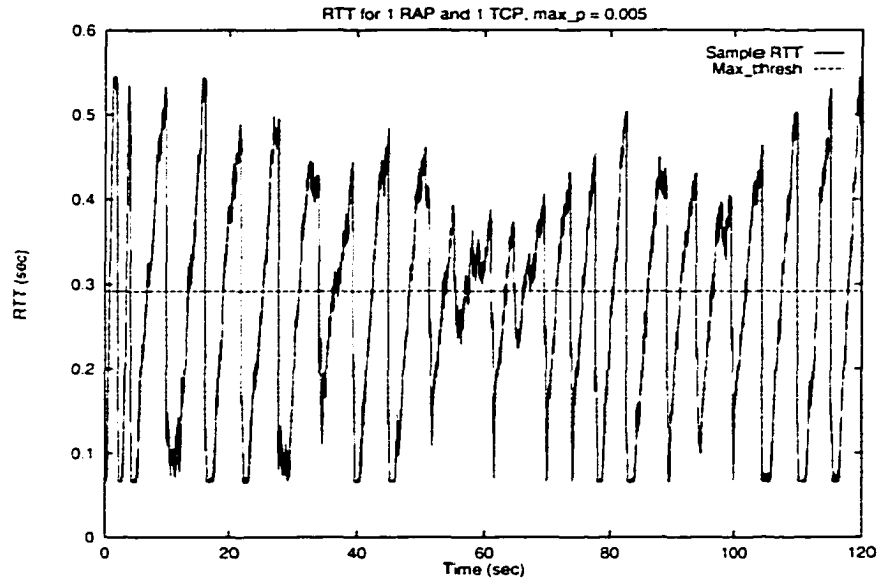
Webgroup CZ a.s. Ex. 1015, Page 97 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

caused by TCP's burstiness. The higher the value of max_p, the more likely RED is to drop a packet before the buffer becomes full, and so the lower the mean buffer utilization is. Fig. 4.7 has already shown that TCP performs poorly with small congestion window, and higher values of max_p tend to reduce TCP's mean congestion window. RAP takes advantage of this, and a degree of unfairness results. As long as the average queue size remains in RED's operating region (below max_{th}), the bandwidth share between RED and TCP is quite fair. However, if the value of max_p is too small, the average queue size reaches max_{th}, and RED then starts dropping all packets until the average queue size decreases below max_{th} again. This process repeats and oscillations occur, with the loss probability alternating between max_p and one. RED should not be operated in this region, and the curve in figure 4.12 shows this effect when $max_p = 0.005$. The differences between RAP and TCP are due to TCP's burstiness interacting with periodic oscillations of the average queue size about max_{th}. With small simulations, the oscillation period is long, and both TCP and RAP lose whole RTT worth of packets. TCP takes a very long time to recover, while RAP recovers comparatively easily. With large simulations, the period of these oscillations is much shorter, and although a few TCP's may lose, on average a TCP is less likely to be hit by one of the loss periods than a RAP flow which spaces its packets out evenly. Hence, on average TCP performs better than RAP. It should be emphasized that this RED regime will impose terrible loss bursts on realtime flows, and should be avoided at all costs. Figures 4.13(a) and 4.13(b) graph the measured RTT for small simulations, and demonstrate these oscillations in fig. 4.13(a) with $max_p = 0.005$ versus normal RED behavior in fig. 4.13(b) with $max_p = 0.16$. We conclude that, with appropriate tuning, RED can significantly improve the fairness between RAP and TCP. However that aggressively pushing for very low buffer utilization is counter-productive when RAP and TCP share a link because TCP then diverges from AIMD.

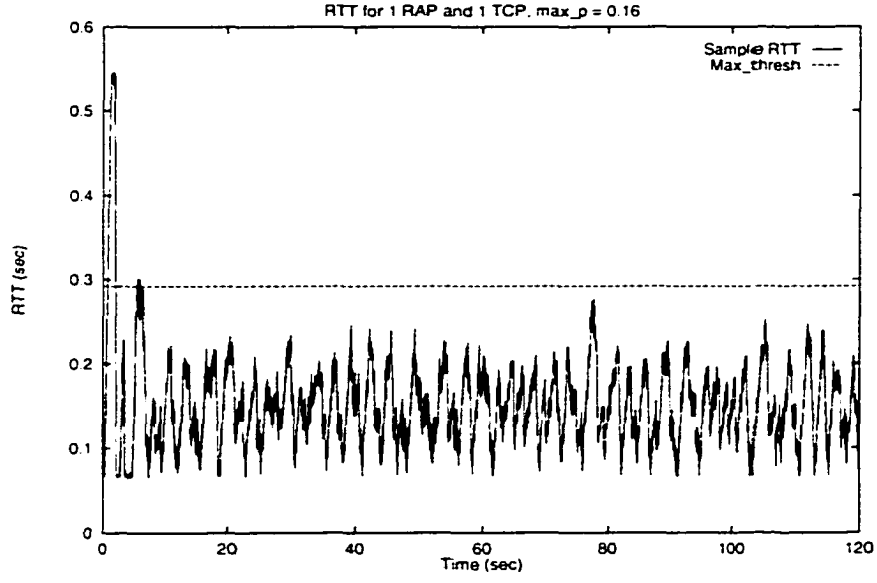
4.7.5 Intra-protocol Fairness

Figure 4.14 shows the average bandwidth share among n RAP flows as we increase the number of flows while the amount of resources remain fixed. The bottleneck bandwidth

Webgroup CZ a.s. Ex. 1015, Page 98 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) RTT for 1 RAP, 1 TCP and $max_p = 0.005$



(b) RTT for 1 RAP, 1 TCP and $max_p = 0.16$

Figure 4.13: Effect of RED configuration of fairness

Webgroup CZ a.s. Ex. 1015, Page 99 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is 250 KByte/s and the bottleneck buffer size is four times the bandwidth-bottleneck RTT product. The rest of the parameters are similar to table 1.

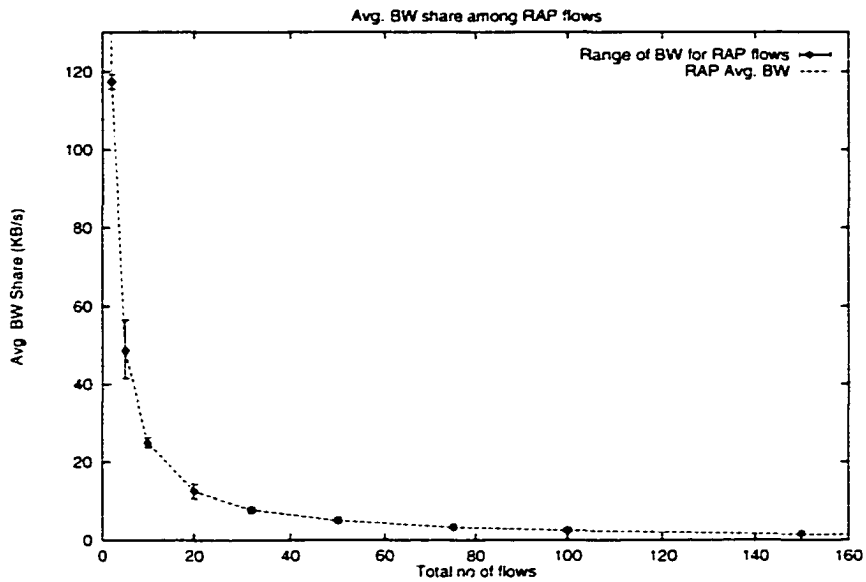


Figure 4.14: Intra-protocol fairness among RAP flows

We performed a number of such simulations to show that RAP flows divide the bandwidth fairly for a wide range of network loads. Note that we have already covered the impact of load on inter-protocol fairness in Figure 4.7.

4.7.6 Smoothness of transmission rate

Figures 4.16 and 4.15 show the variation of goodput for a sample RAP and TCP Tahoe[13] flow respectively from a simulation with 32 RAP and 32 TCP. Although the transmission rate of TCP flows has higher variance than that of RAP flows, both have the similar mean values.

Thus RAP satisfies our design goal of providing a smoother, more predictable congestion-control signal to our real-time applications than TCP does.

[13]TCP Sack is smoother than Tahoe, so this illustrates near worst case behavior by TCP

Webgroup CZ a.s. Ex. 1015, Page 100 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

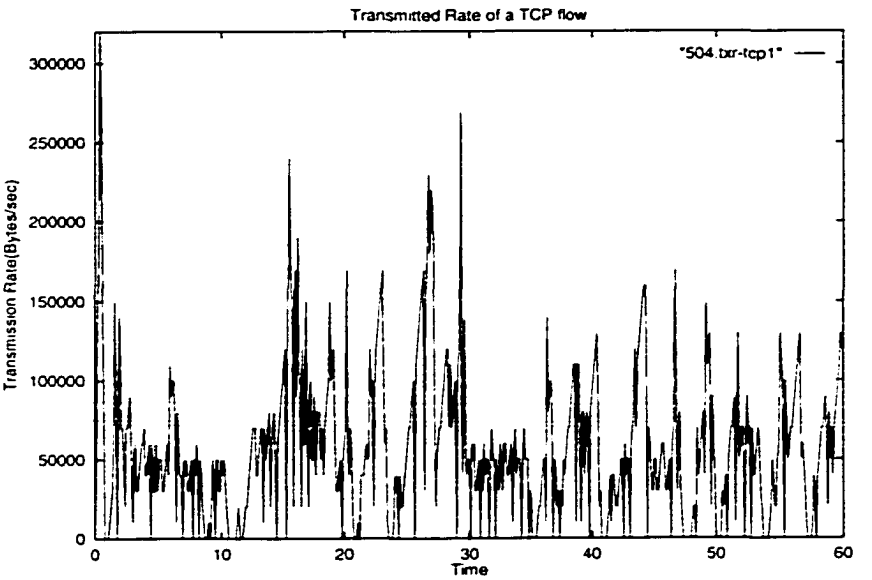


Figure 4.15: Transmission rate of a sample TCP flow

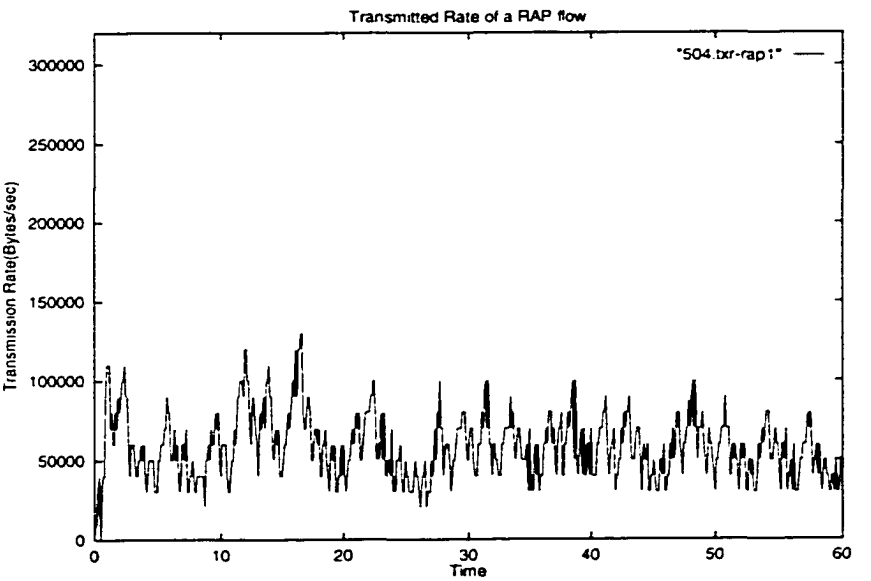


Figure 4.16: Transmission rate of a sample RAP flow

Webgroup CZ a.s. Ex. 1015, Page 101 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.8 Summary

In this chapter, we presented a rate-based congestion control mechanism, called RAP, and extensively examined its interaction with TCP through simulation. RAP performs loss-based congestion control using AIMD rate adaptation. To emulate the ACK-clocking mechanism of TCP and improve RAP's responsiveness to transient congestion, we devised an optional fine-grain rate adaptation on top of coarse-grain AIMD. Towards that goal, we exploited the RTT signal and devised a dimensionless, zero-mean, fine-grain feedback mechanism to detect short-lived congestion events.

The main challenge is that TCP itself is both a moving target and undergoing various design changes, thus it is hard to achieve TCP-friendliness over a wide range of network parameters. Furthermore, because of traffic dynamics, it is hard to differentiate unfair behaviors that are due to coexisting traffic from those due to internal performance limitations. We presented a methodology for evaluation of our simulation results in order to distinguish the effects that are caused by TCP's inherent performance constraints from those that are due to coexisting RAP flows. Our simulation reveals that RAP without fine-grain rate adaptation exhibits a TCP-friendly behavior over a rather wide range of network condition. The fine-grain adaptation further extends that range.

Occasional unfairness against TCP traffic occurs when TCP experiences multiple losses within a window and loses its ACK-clocking, and its congestion control mechanism diverges from AIMD. Our simulations showed that using RED switches with proper configuration can effectively limit the number of losses per window and result in a fair share of resources allocated between RAP and TCP traffic over a wide range. Finally, we also assessed the effect of TCP burstiness in some special cases.

Webgroup CZ a.s. Ex. 1015, Page 102 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

The Quality Adaptation

If video for playback is stored at a single lowest - common - denominator encoding on the server, high-bandwidth clients will receive poor quality despite availability of a large amount of bandwidth. However, if the video is stored at a single higher quality encoding (and hence higher data rate) on the server, there will be many low-bandwidth clients that can not play back this stream. In the past, we have often seen RealVideo streams available at 14.4 Kb/s and 28.8 Kb/s, where the user can choose their connection speed. However, with the advent of ISDN, ADSL, and cable modems to the home, and faster access rates to businesses, the Internet is becoming much more heterogeneous. Customers with higher speed connections feel frustrated to be restricted to modem-speed playback. Moreover, the network bottleneck may be in the backbone, such as at provider interconnects or links to the server itself. In this case, the user can not know the congestion level, and congestion control mechanisms for streaming video playback are critical.

Given a channel that changes its bandwidth over time due to congestion control, the server should be able to adjust the quality of the stream it plays back so that the perceived quality is as high as the available network bandwidth will permit. We term this *quality adaptation*.

Webgroup CZ a.s. Ex. 1015, Page 103 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1 Quality Adaptation Mechanisms

There are several ways to adjust the quality of a pre-encoded stored stream, including: adaptive encoding, switching among multiple pre-encoded versions, and hierarchical encoding.

One may re-quantize stored encodings on-the-fly based on network feedback[14, 94, 124]. However, since encoding is CPU - intensive, servers are unlikely to be able to do this for large numbers of clients. Furthermore, once the original data has been stored compressed, the output rate of most encoders can not be changed over a wide range.

In an alternative approach, the server keeps several versions of each stream with different qualities. As available bandwidth changes, the server plays back streams of higher or lower quality as appropriate.

With hierarchical encoding[76, 86, 89, 131], the server maintains a layered encoded version of each stream. As more bandwidth becomes available, more layers of the encoding are delivered. If the average bandwidth decreases, the server may then drop some of the layers being transmitted. Layered approaches usually have the decoding constraint that a particular enhancement layer can only be decoded if all the lower quality layers have been received.

There is a duality between adding or dropping of layers in the layered approach and switching streams in the multiply-encoded approach. However the layered approach is more suitable for caching by a proxy for heterogeneous clients[110]. In addition, it requires less storage at the server, and it provides an opportunity for selective retransmission of the more important information. The design of a layered approach for quality adaptation primarily entails the design of an efficient add and drop mechanism that maximizes quality while minimizing the probability of base-layer buffer underflow.

This chapter is organized as follows: first we provide an overview of the layered approach to quality adaptation and then explain coarse-grain adding and dropping mechanisms in section 5.2. We also discuss fine-grain inter-layer bandwidth allocation for a single backoff scenario. Section 5.3 motivates the need for smoothing in the presence of real loss patterns and discusses two possible approaches. In section 5.4, we sketch an efficient filling and draining mechanism that not only achieves smoothing but is also able

Webgroup CZ a.s. Ex. 1015, Page 104 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

to cope efficiently with various patterns of losses. We evaluate our mechanism through simulation in section 5.5.

5.2 Layered Quality Adaptation

Hierarchical encoding provides an effective way that a video playback server can coarsely adjust the quality of a video stream without transcoding the stored data. However, it does not provide fine-grained control over bandwidth, i.e. bandwidth changes at the granularity of a layer. Furthermore, there needs to be a quality adaptation mechanism to smoothly adjust the quality (i.e. number of layer) as bandwidth changes. Users will tolerate poor quality video, but rapid variations in quality are disturbing.

Hierarchical encoding allows video quality adjustment over long periods of time, whereas congestion control changes the transmission rate rapidly over short time intervals (several round-trip times,(RTTs)). The mismatch between the two timescales is made up for by buffering data at the receiver to smooth the rapid variations in available bandwidth and allow a near constant number of layers to be played.

Figure 5.1 graphs a simple simulation of a quality adaptation mechanism in action. The top graph shows the available network bandwidth and the consumption rate at the receiver with no layers being consumed at startup, then one layer, and finally two layers. During the simulation, two packets are dropped and cause congestion control backoffs, when the transmission rate drops below the consumption rate for a period of time. The lower graph shows the playout sequence numbers of the actual packets against time. The horizontal lines show the period between arrival time and playout time of a packet. Thus it indicates the total amount of buffering for each layer. This simulation shows more buffered data for Layer 0 (the base layer) than for Layer 1 (the enhancement layer). After the first backoff, the length of these lines decreases indicating buffered data from Layer 0 is being used to compensate for the lack of available bandwidth. At the time of the second backoff, a little data has been buffered for Layer 1 in addition to the large amount for Layer 0. Thus data is drawn from both buffers properly to compensate for the lack of available bandwidth.

Webgroup CZ a.s. Ex. 1015, Page 105 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

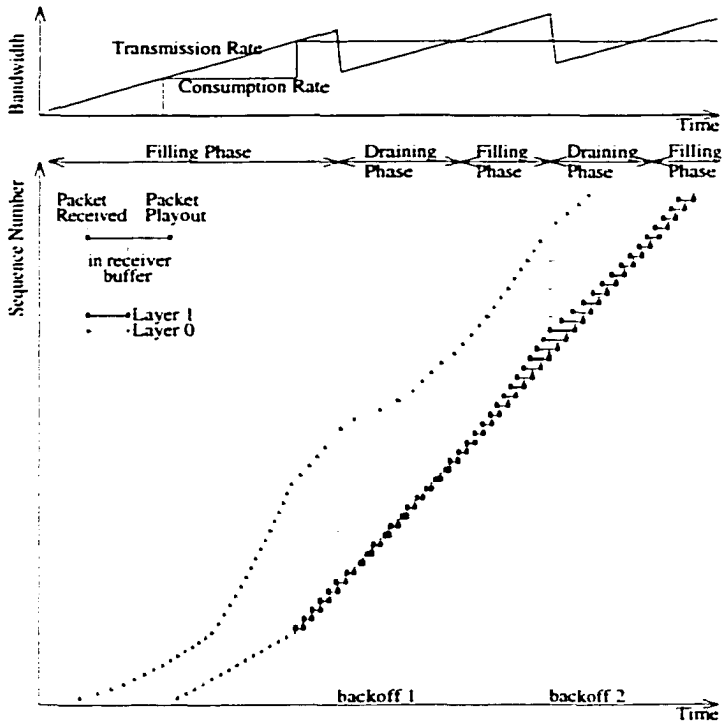


Figure 5.1: Layered encoding with receiver buffering

The congestion control mechanism dictates the available bandwidth [1]. We can not send more than this amount, and do not wish to send less[2]. In a real network even the average bandwidth of a congestion controlled flow changes over the session lifetime. Thus a quality adaptation mechanism must continuously evaluate the available bandwidth and adjust the number of active layers accordingly.

We assume that the layers are linearly spaced - that is each layer has the same bandwidth. This simplifies the analysis, but is not a requirement. In addition, we assume each layer has a constant consumption rate over time. In practice this is unlikely in a real codec, but to a first approximation it is reasonable. It can be ignored by slightly increasing the amount of receiver buffering for all layers to absorb variations in consumption rate.

Figure 5.2 shows a single cycle of the congestion control mechanism. The sawtooth waveform is the instantaneous transmission rate. There are n_a active layers, each of which

[1] Available bandwidth and transmission rate are used inter-changeably throughout this dissertation.

[2] For simplicity we ignore flow control issues but implementations should not. However our final solutions generally require so little receiver buffering that this is not often an issue.

Webgroup CZ a.s. Ex. 1015, Page 106 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

has a consumption rate of C. In the left hand side of the figure, the transmission rate is higher than the consumption rate, and this data will be stored temporarily in the receiver's buffer. The total amount of stored data is equal to the area of triangle abc. Such a period of time is known as a *filling phase*. Then, at time t_b, a packet is lost and the transmit rate is reduced multiplicatively. To continue playing out n_a layers when the transmission rate drops below the consumption rate, some data must be drawn from the receiver buffer until the transmission rate reaches the consumption rate again. The amount of data drawn from the buffer is shown in this figure as triangle cde. Such a period of time is known as a *draining phase*.

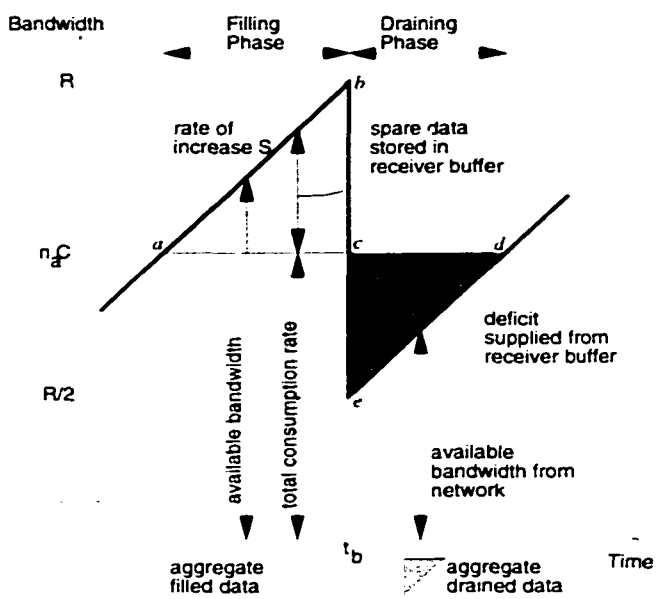


Figure 5.2: Filling and draining phase

5.2.1 Problem Overview

This section sketches an overview of the layered approach to quality adaptation in order to identify design parameters. Figure 5.3 depicts all the components related to the quality adaptation mechanism at the server and the client sides. Table 5.1 summarizes the notations used in this chapter.

Webgroup CZ a.s. Ex. 1015, Page 107 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

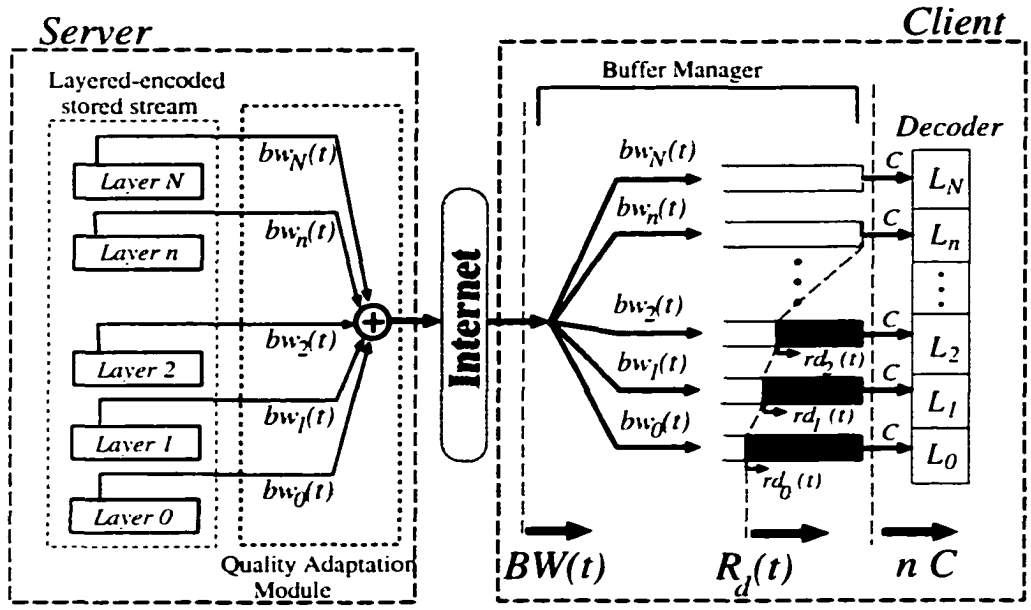


Figure 5.3: End-to-end components of quality adaptation mechanism

All the streams are linearly layered-encoded and stored at the server. Upon arrival of a request for a stream, the server multiplexes an appropriate number of layers by allocating a portion of the total available bandwidth (R) that is specified by the congestion control mechanism. Thus we have:

$$R = \sum_{i=0}^{n_a} bw_i(t) \tag{5.1}$$

At the other end, the client's buffer manager demultiplexes different layers and directs them to their corresponding buffers. While layer i is active, its buffer is continuously drained by the decoder with a constant consumption rate(C) and filled with the delivered data at the rate $bw_i(t)$ [3]. Thus the amount of buffered data for layer i is drained with overall rate $rd_i(t)$:

[3]Here we have ignored packet loss to simplify the problem. If packet loss occurs, the actual arrival rate is less than $bw_i(t)$.

Webgroup CZ a.s. Ex. 1015, Page 108 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Parameter	Value
$R(t)$	Current transmission rate
$R_d(t)$	Aggregate draining rate
$rd_i(t)$	Draining rate of layer i
$bw_i(t)$	Bandwidth share of layer i
$buf_i(t)$	Buffer share of layer i
S	Slope of linear increase
k	Back off factor
n_a	Number of active layers
n_b	Number of buffering layers
C	Consumption/Generation rate of a layer

Table 5.1: Parameter list for Quality adaptation

$$rd_i(t) = C - bw_i(t) \tag{5.2}$$

Equation (5.2) indicates that an active layer could experience one of the following conditions:

- $bw_i(t) = 0 \Rightarrow rd_i(t) = C$, An active layer without any share of bandwidth must only rely on its buffered data. Moreover its buffer is drained with the maximum rate(C).
- $0 < bw_i(t) < C \Rightarrow rd_i(t) = C - bw_i(t) \geq 0$, A portion of the consumption rate is compensated by the bandwidth share and the rest is drained from the buffer with the rate $rd_i(t)$.
- $bw_i(t) = C \Rightarrow rd_i(t) = 0$, The entire consumption rate is compensated with the bandwidth share. The amount of its buffered data does not change, i.e. this layer does not necessarily need any buffered data for quality adaptation.
- $bw_i(t) > C \Rightarrow rd_i(t) = C - bw_i(t) \leq 0$, The available bandwidth not only provides the consumption rate of the layer, but its buffer size is also increasing with the rate $-rd_i(t)$.

Webgroup CZ a.s. Ex. 1015, Page 109 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The amount of buffered data of layer i at time t is:

$$buf_i(t) = \int -rd_i(t).dt = \int (bw_i(t) - C)dt \tag{5.3}$$

To calculate the draining rate of the aggregate buffered data, we can sum up equation 5.2 across all the active layers:

$$\sum_{i=0}^{n_a} rd_i(t) = n_a C - \sum_{i=0}^{n_a} bw_i(t) \tag{5.4}$$

$$R_d(t) = \sum_{i=0}^{n_a} rd_i(t) \tag{5.5}$$

$$R = \sum_{i=0}^{n_a} bw_i(t) \tag{5.6}$$

$$R_d(t) = n_a C - BW(t) \tag{5.7}$$

Equation (5.7) implies that the aggregate buffered data for all active layers is consumed with the constant rate of $n_a C$ and filled with the rate BW(t). Finally, the volume of the aggregate buffered data can be calculated as follows:

$$\sum_{i=0}^{n_a} buf_i(t) = \int -R_d(t).dt = \int (R(t) - n_a C).dt \tag{5.8}$$

Equation (5.8) indicates that the amount of aggregate buffered data varies with the available bandwidth($R(t)$). However, the server can loosely adjust the amount of buffering by changing the number of active layers(n_a). This is the main flexibility that is gained from a layered approach. At the macro-level, equation (5.7) relates the add and drop mechanism(i.e. n_a) to the aggregate buffered data and the available bandwidth. Whereas

Webgroup CZ a.s. Ex. 1015, Page 110 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

equation (5.3) explains the dependency between the bandwidth and buffer share for each layer at the micro-level.

Note that the quality adaptation mechanism can *only* adjust the number of active layers and their bandwidth share. We attempt to derive efficient behavior for these two key mechanisms:

- A *coarse-grain* mechanism for adding and dropping layers. By changing the number of active layers, the server can perform coarse-grain adjustment on the total amount of receiver-buffered data.

- A *fine-grain* inter-layer bandwidth allocation mechanism among the active layers. If there is receiver-buffered data available for a layer, we can temporarily allocate less bandwidth than is being consumed while taking the remainder from the buffer. This smoothes out reductions in the available bandwidth. When spare bandwidth is available, we can send data for a layer at a rate higher than its consumption rate, and increase the data buffered for that layer at the receiver.

In the next section, we present coarse-grain adding and dropping mechanisms, and discuss their relation to fine-grain bandwidth allocation. We discuss fine-grain bandwidth allocation in the subsequent sections.

5.2.2 Adding a Layer

A new layer can be added as soon as the instantaneous available bandwidth exceeds the consumption rate (in the decoder) of the existing layers. The excess bandwidth could then be used to start buffering a new layer. However, this would be problematic as without knowing future available bandwidth we can not decide when it will first be possible to start decoding the layer. The new layer's *playout* is decided by the inter-layer timing dependency between its data and that in the base layer. Therefore we can not make a reasoned decision about which data from the new layer to actually send [4].

[4]Note that once the inter-layer timing for a new layer is adjusted, it is maintained as long as the buffer does not dry out.

Webgroup CZ a.s. Ex. 1015, Page 111 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A more practical approach is to start sending a new layer when the instantaneous bandwidth exceeds the consumption rate of the existing layers plus the new layer. In this approach the layer can start to play out immediately. In this case there is some excess bandwidth from the time the available bandwidth exceeds the consumption rate of the existing layers until the new layer is added. This excess bandwidth can be used to buffer data for existing layers at the receiver.

In practice, this bandwidth constraint for adding is still not conservative enough, as it may result in several layers being added and dropped with each cycle of the congestion control sawtooth. Such rapid changes in quality would be disconcerting for the viewer. One way to prevent rapid changes in quality is to add a buffering condition such that adding a new layer does not endanger existing layers. Thus, the server may add a new layer when:

1. The instantaneous available bandwidth is greater than the consumption rate of the existing layers plus the new layer, and,

2. There is sufficient total buffering at the receiver to survive an immediate backoff and continue playing all the existing layers plus the new layer.

To satisfy the second condition we assume (for now) that no additional backoff will occur during the draining phase, and the slope of linear increase can be properly estimated.

These are the minimal criteria for adding a new layer. If these conditions are held a new layer can be kept for a reasonable period of time during the normal congestion control cycles. We shall show later that we normally want to be even more conservative than this. Clearly we need to have sufficient buffering at the receiver to smooth out variations in the available bandwidth so that the number of active layers does not change due to the normal hunting behavior of the congestion control mechanism. Expressing the adding conditions more precisely:

$$\begin{aligned} &\text{Condition 1:} \quad R > (n_a + 1)C \\ &\text{Condition 2:} \quad \sum_{i=0}^{n_a - 1} buf_i \geq \frac{((n_a + 1)C - \frac{R}{2})^2}{2S} \end{aligned} \tag{5.9}$$

Webgroup CZ a.s. Ex. 1015, Page 112 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.3 Dropping a Layer

Once a backoff occurs, if the total amount of buffering at the receiver is less than the estimated required buffering for recovery, (i.e, the area of triangle *cde* in figure 5.2), the correct course of action is to immediately drop the highest layer. This reduces the consumption rate ($n_a C$) and hence reduces the buffer requirement for recovery. If the buffering is still insufficient, the server should iteratively drop the highest layer until the amount of buffering is sufficient. This rule clearly doesn't apply to the base layer which is always sent. Expressing the dropping mechanism more precisely:

$$\begin{aligned} &\textbf{WHILE } \left(n_a C > R + \sqrt{2S \sum_{i=0}^{n_a - 1} buf_i} \right) \\ &\textbf{DO } n_a = n_a - 1 \end{aligned} \tag{5.10}$$

This mechanism provides a coarse-grain criteria for dropping a layer. However, it may be insufficient to prevent buffer underflow during the draining phase for one of the following reasons:

- We may suffer a further backoff before the current draining phase completes.
- Our estimate of the slope of linear increase may be incorrect if the network RTT changes substantially.
- There may be sufficient total data buffered, but it may be allocated among the different layers in a manner that precludes its use to aid recovery(figure 5.4).

The first two situations are due to incorrect prediction of the amount of buffered data needed to recover, and we term such an event a *critical situation*. In such events, the only appropriate course of action is to drop additional layers as soon as the critical situation is discovered(figure 5.4).

The third situation is more problematic, and relates to the fine-grain bandwidth allocation among active layers during both filling and draining phases. We derive and evaluate a near-optimal solution to this situation.

Webgroup CZ a.s. Ex. 1015, Page 113 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.4 Inter-layer Buffer Allocation

Because of the decoding constraint in hierarchical coding, each additional layer depends on all the lower layers, and correspondingly is of decreasing value. Thus a buffer allocation mechanism should provide higher protection for lower layers by allocating a higher share of buffering for them.

The challenge of inter-layer buffer allocation is to ensure the total amount of buffering is sufficient, and that is properly distributed among active layers to effectively absorb the short-term reductions in bandwidth that might occur. The following two examples illustrate ways in which improper allocation of buffered data might fail to compensate for the lack of available bandwidth.

5.2.4.1 Dropping layers with buffered data

A simple buffer allocation scheme might allocate an equal share of buffer to each layer. However, if the highest layer is dropped after a backoff, its buffered data is no longer able to assist the remaining layers in the recovery. The top layer's data will still be played out, but it is not providing buffering functionality. This implies that it is more beneficial to buffer data for lower layers.

5.2.4.2 Insufficient distribution of buffered data

An equally simple buffer allocation scheme might allocate all the buffering to the base layer. Consider an example when three layers are playing, where a total consumption rate of $3C$ must be supplied for the receiver's decoder. If the transmission rate drops to C, the base layer (L_0) can be played from its buffer. Since neither L_1 nor L_2 has any buffering, they require transmission from the source. However available bandwidth is only sufficient to feed one layer. Thus L_2 must be dropped *even if the total buffering were sufficient for recovery*. In these examples, although buffering is available, it can not be used to prevent the dropping of layers. This is *inefficient* use of the buffering. In general, we are striving for a distribution of buffering that is most *efficient* in the sense that it provides

Webgroup CZ a.s. Ex. 1015, Page 114 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

maximal protection against dropping layers for any likely pattern of short-term reduction in available bandwidth.

These examples reveal the following tradeoffs for inter-layer buffer allocations:

- Allocating more buffering for the lower layers not only improves their protection but it also increases *efficiency* of buffering.
- Buffered data for each layer can not provide more than its consumption rate(i.e. C). Thus there is a minimum number of buffering layers that are needed to cope with short-term reductions in available bandwidth for successful recovery. This minimum is directly determined by the reduction in bandwidth that we intend to absorb by buffering.

Expressing this more precisely:

$$\begin{aligned} n_b &= \left\lceil n_a - \frac{R}{2C} \right\rceil ; \quad n_a > \frac{R}{2C} \\ n_b &= 0 ; \quad n_a \leq \frac{R}{2C} \end{aligned} \tag{5.11}$$

5.2.5 Optimal Inter-layer Buffer Allocation

Given a draining phase following a single backoff, we can derive the optimal inter-layer buffer allocation that maximizes buffering efficiency. Figure 5.5 illustrates an optimal buffer allocation and its corresponding draining pattern for a draining phase. Here we assume that the total amount of buffering at the receiver at time t_b is precisely sufficient for recovery(i.e. area of triangle afg) with no spare buffering available at the end of the draining phase.

To justify the optimality of this buffer allocation, consider that the consumption rate of a layer must be supplied either from the network or from the buffer or a combination of the two. If it is supplied entirely from the buffer, that layer's buffer is draining at consumption rate C. The area of quadrilateral $defg$ in figure 5.5 shows the maximum amount of buffer that can be drained from a single layer during this draining phase. If the draining phase ends as predicted, there is no preference as to buffer distribution among active layers as long as no layer has more than $defg$ worth of buffered data. However, if the situation

Webgroup CZ a.s. Ex. 1015, Page 115 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

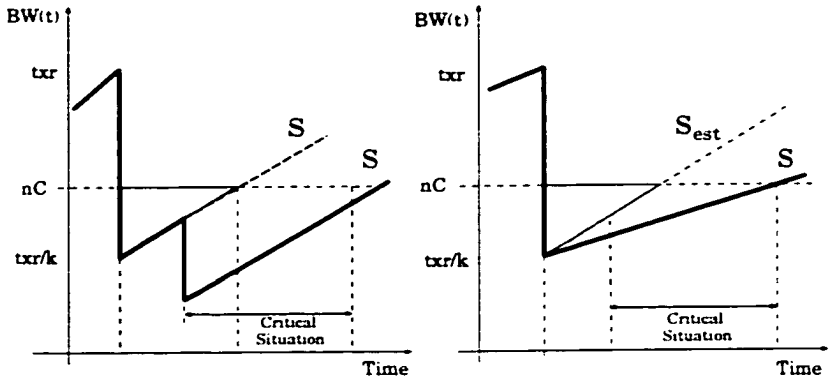


Figure 5.4: Critical situation due to a back-off or overestimated slope

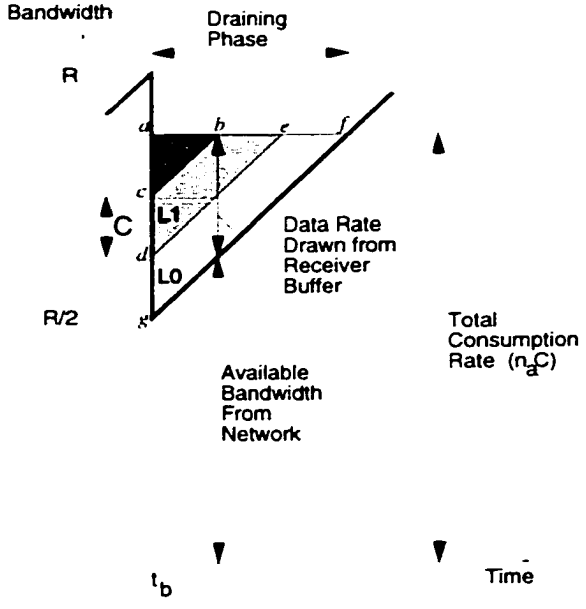


Figure 5.5: The optimal inter-layer buffer distribution

Webgroup CZ a.s. Ex. 1015, Page 116 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

becomes critical due to further backoffs, layers must be dropped. Allocating area $defg$ of buffering to the base layer would ensure that the maximum amount of the buffered data is still usable for recovery, and maximizes buffering efficiency.

By similar reasoning, the next largest amount an additional layer's buffer can contribute is quadrilateral $bcde$, and this portion of buffered data should be allocated to L_1, the first enhancement layer, and so on. This approach minimizes the amount of buffered data allocated for higher layers that might be dropped in a critical situation and consequently maximizes buffering efficiency.

The optimal amount of buffering for layer i is:

$$\begin{aligned} Buf_{i,opt} &= \frac{C}{2S}(C(2n_a - 2i - 1) - R)\ ; \quad i < n_b - 1 \\ Buf_{i,opt} &= \frac{C}{2S}(n_a C - \frac{R}{2} - iC)^2\ ; \quad i = n_b - 1 \end{aligned} \tag{5.12}$$

Although we can calculate the optimal allocation of buffered data for the active layers, a backoff may occur at any random time. To tackle this problem, during the filling phase, we incrementally adjust the allocation of buffered data so that the buffer state always remains as close as possible to an optimal state.

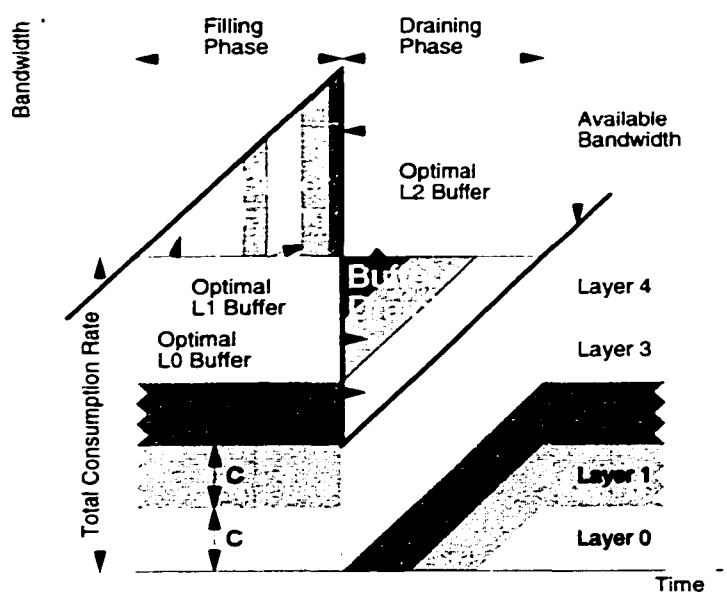


Figure 5.6: Optimal buffer sharing

Webgroup CZ a.s. Ex. 1015, Page 117 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Toward that goal, we assume that a single backoff will occur immediately, and ask the question: "if we keep only the base layer, is there sufficient buffering to survive?". If there is not sufficient buffering, then we fill up the base layer's buffer until it has enough buffering to survive a single backoff. Then we ask the question: "if we keep only two layers, is there enough buffering to survive with those buffers having optimal allocation?". If there is not enough base layer data, we fill the base layer's buffer up to the optimal level. Then we start sending L_1 data until both layers have the optimal amount of buffering to survive. We repeat this process and increase the number of expected surviving layers until all the buffering layers are filled up to an optimal level such that all active layers can survive from a single backoff. This approach results in a sequential filling pattern among buffering layers.

Figure 5.6 illustrates the optimal filling and draining scheme for a single backoff. If a backoff occurs exactly at time t_b, all layers can survive the backoff. Occurrence of a backoff earlier than t_b results in dropping one or more active layers. However the buffer state is always as close as possible to the optimal state without those layers. If no backoff occurs until adding conditions (section 5.2.2) are satisfied, a new layer is added and we repeat the sequential filling mechanism.

It is worth mentioning that the server can control the filling and draining pattern by proper fine-grain bandwidth allocation among active layers. Figure 5.6 illustrates that at each point of time during the draining phase, bandwidth share plus draining rate for each layer is equal to its consumption rate. Thus maximally efficient buffering results in the upper layers being supplied from the network during the draining phase while the lower layers are supplied from their buffers. For example, just after the backoff, layer 2 is supplied entirely from the buffer, but the amount supplied from the buffer decreases to zero as data supplied from the network takes over. Layers 0 and 1 are supplied from the buffer for longer periods.

Webgroup CZ a.s. Ex. 1015, Page 118 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Smoothness Constraints

In the previous section, we derived an optimal filling and draining scheme based on the assumption that we only buffer to survive a single backoff with all the layers intact. However, examination of Internet traffic indicates that real networks exhibit near-random[16] loss patterns with frequent additional backoffs during a draining phase. Thus, aiming to survive only a single backoff is too aggressive and results in frequent adding and dropping of layers.

5.3.1 Smoothing

To achieve reasonable smoothing of the add and drop rate, an obvious approach is to refine our adding conditions (in section 5.2.2) to be more conservative. We have considered the following two mechanisms to achieve smoothing:

- We may add a new layer if the *average* available bandwidth is greater than the consumption rate of the existing layers plus the new layer.
- We may add a new layer if we have sufficient amount of buffered data to survive K_{max} backoffs with existing layers, where K_{max} is a *smoothing factor* with value greater than one.

Although each one of these mechanisms results in smoothing, the latter not only allows us to directly tie the adding decision to appropriate buffer state for adding, but it can also utilize limited bandwidth links effectively. For example, if there is sufficient bandwidth across a modem link to receive 2.9 layers, the average bandwidth would never become high enough to add the third layer. In contrast, the latter mechanism would send 3 layers for 90% of the time which is more desirable. For the rest of this paper we assume that the only condition for adding a new layer is availability of optimal buffer allocation for recovery from K_{max} backoffs.

Changing K_{max} allows us to tune the balance between maximizing the short-term quality and minimizing the changes in quality. An obvious question is "What degree of smoothing is appropriate?" In the absence of a specific layered codec and user-evaluation,

Webgroup CZ a.s. Ex. 1015, Page 119 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

K_{max} can not be analytically derived. Instead it should be set based on real-world user perception experiments to determine the appropriate degree of smoothing that is not disturbing to the user. In practice, we probably also want to base K_{max} on the average bandwidth and RTT since these determine the duration of a draining phase.

5.3.2 Buffering Revisited

If we delay adding a new layer to achieve smoothing, this affects the way we fill and drain the buffers. Figure 5.7 demonstrates this issue.

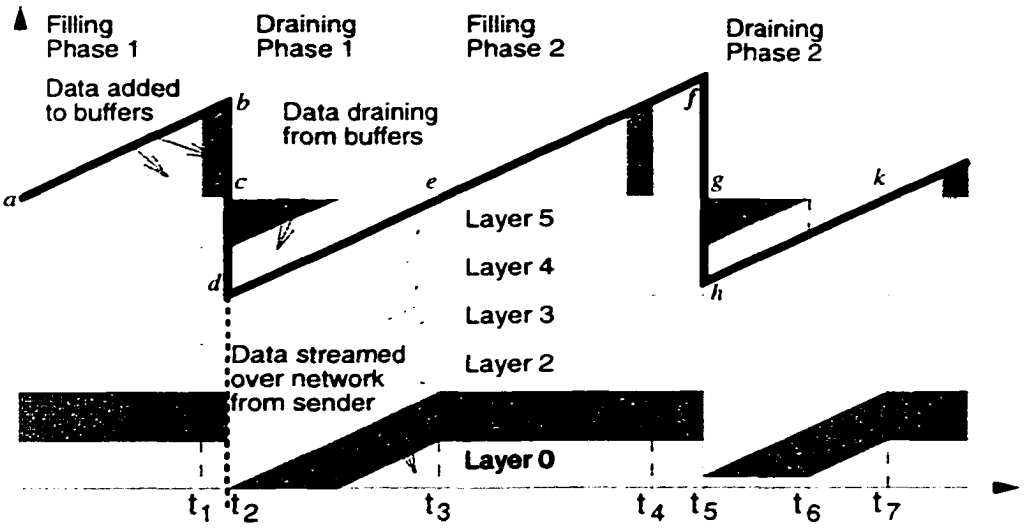


Figure 5.7: Revised draining phase algorithm

Up until time t_3, this is the same as figure 5.6. The second filling phase starts at time t_3, and at t_4 there is sufficient buffering to survive a backoff. However, for smoothing purposes, a new layer is not added at this point and we continue buffering data until a backoff occurs at t_5.

Note that as the available bandwidth increases, the total amount of buffering increases but the required buffering for recovery from a single backoff decreases. At time t_5, we have more buffering than we need to survive a single backoff, but insufficient buffering to survive a second backoff before the end of the draining phase. We need to specify how we allocate the extra buffering after time t_4, and how we drain these buffers after t_5 while maintaining efficiency.

Webgroup CZ a.s. Ex. 1015, Page 120 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Conceptually, during the filling phase, the server sequentially examines the following steps:

Step 1: enough buffer for one backoff with L_0 intact.

Step 2: enough buffer for one backoff with L_0 and L_1.

...

Step n_a: enough buffer for one backoff with L_0 through L_{n_a-1} intact.

Step n_a+1: enough buffer for one backoff with L_0 through layer L_{n_a-1} intact and two backoffs with L_0 intact.

At any point in the filling phase we have satisfied one step and are working towards the next step.

When a backoff occurs between steps, in this case between steps n_a and $n_a + 1$, we essentially reverse the filling process. First we identify between which two steps we're currently located. Then we traverse through the steps in the reverse order to determine which layers must be drained and by how much. In essence, during consecutive filling and draining phases, we traverse this sequence of steps (i.e. optimal buffer states) back and forth such that at any point of time the buffer state is as close to optimal as possible. In the next section, we describe this mechanism in more detail.

5.4 Buffer Allocation with Smoothing

To design an efficient filling and draining mechanisms in the presence of smoothing, we need to know the optimal buffer allocation among layers and the corresponding maximally efficient filling and draining patterns for multiple-backoff scenarios.

The optimal buffer allocation for a scenario with multiple backoffs is not unique because it depends on the time when the additional backoffs occur during the draining phase. If we have knowledge of future loss distribution patterns it might, in principle, be possible to calculate the optimal buffer allocation. In practice such a solution would be excessively complex for the problem it is trying to solve, and rapidly becomes intractable as the

Webgroup CZ a.s. Ex. 1015, Page 121 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

number of backoffs increases. Let us first assume that only one additional backoff occurs during the draining phase. The possible scenarios are shown in figure 5.8. This figure illustrates that the optimal buffer allocation for each scenario depends on the time of the second backoff, the consumption rate, and the transmission rate before the first backoff.

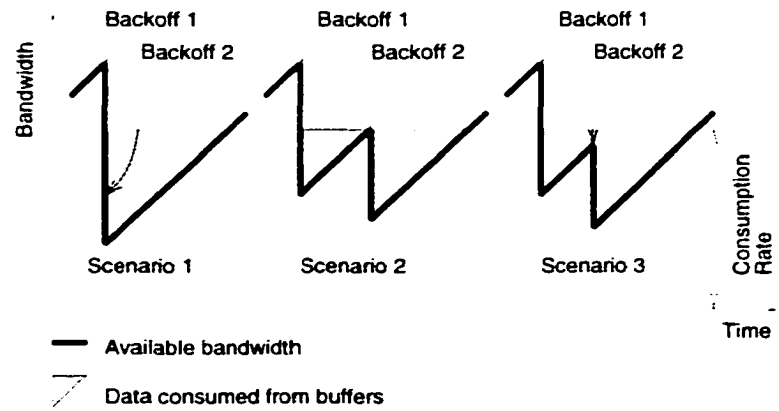


Figure 5.8: Possible double-backoff scenarios

We can extend the idea of optimal buffer allocation for a single backoff (section 5.2.5) to each individual scenario. Added complexity arises from the fact that different scenarios require different buffer allocations. For an equal amount of the total buffering needed for recovery, scenarios 1 and 2 are two extreme cases in the sense that they need the maximum and minimum number of buffering layers respectively. Thus addressing these two extreme scenarios efficiently should cover all the intermediate scenarios (e.g. scenario 3) as well.

We need to decide which scenario to consider during the filling phase. We make a key observation here. If the total amount of buffering for scenarios 1 and 2 are equal, having the optimal buffer distribution for scenario 1 is sufficient for recovery from scenario 2, although it is not maximally efficient. However, the converse is not feasible. The higher flexibility in scenario 1 comes from the fact that this scenario needs a larger number of buffering layers than does scenario 2. Thus, if we have a buffer distribution that can recover from a scenario 1, we will be able to cope with a scenario 2 that has the same total buffer requirement, but not vice versa.

This suggests that during the filling phase for the two backoff scenarios, first we consider the optimal buffer allocation for scenario 1 and fill up the buffers in a step by step

Webgroup CZ a.s. Ex. 1015, Page 122 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sequential fashion as described in section 5.3.2. Once this is achieved, then we move on to consider scenario 2.

5.4.1 Filling Phase with Smoothing

To extend this idea to scenarios of k backoffs, we need to examine the optimal buffer allocation for scenario 1 and 2 for each successive value of k. Figure 5.9 illustrates the optimal buffer state, including the total buffer requirement and its optimal inter-layer allocation in scenario 1 and 2, for different values of k. Ideally, we would like to fill the buffers during the filling phase such that we traverse through these buffer states in turn. Once k exceeds K_{max} (the smoothing factor), then we add a new layer and start the process again with the new sets of optimal buffer states.

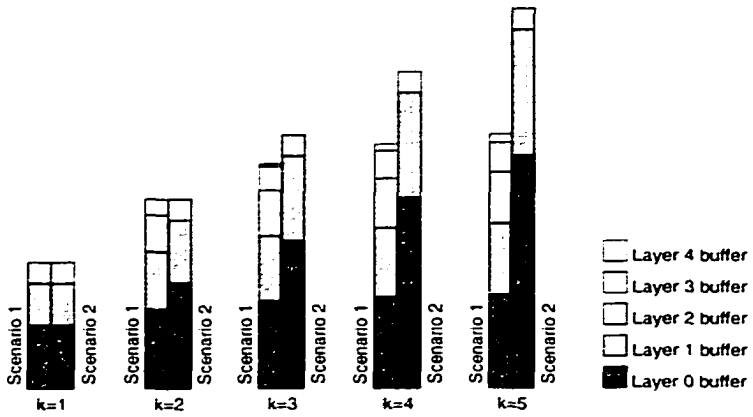


Figure 5.9: Buffer distributions for k backoffs

Toward this goal, we order these different buffer states in increasing value of total amount of required buffering in figure 5.10. Thus by traversing this sequence of buffer states, we always work towards the next optimal state that requires more buffering.

Unfortunately this requires us to occasionally drain an existing buffer in order to reach the next state[5]. Two examples of this phenomenon are visible in figure 5.10:

- Moving from the {scenario 2, k=2} case to the {scenario 1, k=2} case involves draining L_0's buffer.

[5]This means that the order of these states based on increasing value of total required buffering is different from their order based on increasing value of per layer buffering.

Webgroup CZ a.s. Ex. 1015, Page 123 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

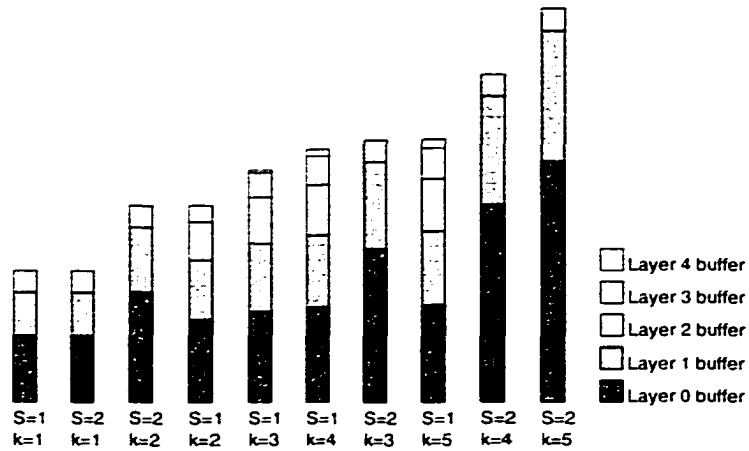


Figure 5.10: Distributions in increasing order of buffering

- Moving from the {scenario 1, k=4} case to the {scenario 2, k=3} case involves draining L_3's buffer.

We do not want to drain any layer's buffer during the filling phase because that buffering provides protection for a previous scenario that we have already passed. Thus we seek the maximally efficient sequence of buffer states *that is consistent with the existing buffering*. The total amount of required buffering and the per layer buffer requirement must be monotonically increasing as we go to the next buffer state.

The key observation that we mentioned earlier allows us to calculate such a sequence. We recall that having the optimal buffer distribution for scenario 1 is sufficient for recovery from scenario 2, although it is not maximally efficient. Given this flexibility, the solution is to constrain per layer buffer allocation in each scenario-2 state to be no less than the previous scenario-1 state, and no more than the next scenario-1 state (in the sequence of states in figure 5.10). Figure 5.11 depicts a sequence of maximally efficient buffer states after applying the above constraints where each step in the filling process is numbered. By enforcing this constraint, we can traverse through the buffer states such that buffer allocation for each state satisfies the buffer requirement for all the previous states. This implies that both the total amount of buffering and the amount of per layer buffering monotonically increase. Thus the per layer buffering can always be used to aid recovery. Once we have sufficient buffering for recovery from K_{max} backoffs in both scenarios, a new layer will be added.

Webgroup CZ a.s. Ex. 1015, Page 124 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

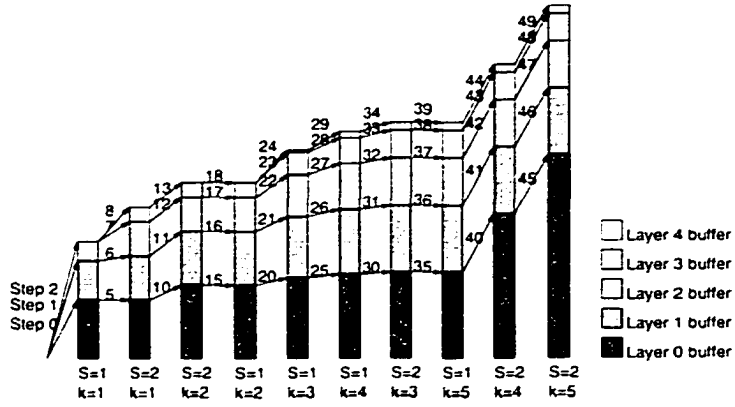


Figure 5.11: Step-by-step buffer filling

The following pseudo-code expresses our per-packet algorithm to ensure that buffer state remains maximally efficient during the filling phase[6]:

```
FUNCTION SendPacket

  S1Backoffs = 0; S2Backoffs = 0
  BufReq1 = 0; BufReq2 = 0

  WHILE (BufReq1 < TotBufAvailable) AND (S1Backoffs < K_max)
     INCREMENT S1Backoffs
     BufReq1 = TotalBufRequired(CurrentRate, Scenario=1,
                                S1Backoffs, ActiveLayers)

  WHILE (BufReq2 < TotBufAvailable)
     INCREMENT S2Backoffs
     BufReq2 = TotalBufRequired(CurrentRate, Scenario=2,
                                S2Backoffs, ActiveLayers)

  FOR Layer = 1 TO ActiveLayers
```

[6]The algorithm performs fine-grain bandwidth allocation by assigning the next transmitting packet to a particular layer.

Webgroup CZ a.s. Ex. 1015, Page 125 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
LayerBuf1 = BufRequired(CurrentRate, Scenario=1,
                        S1Backoffs, Layer, ActiveLayers)
LayerBuf2 = BufRequired(CurrentRate, Scenario=2,
                        S2Backoffs, Layer, ActiveLayers)
IF (BufReq1 < BufReq2) AND (S1Backoffs < K_max)
                                  #We're considering scenario 1
    IF (LayerBuf1 > BufAvailable(Layer)
        SendPacketFromLayer(Layer)
        RETURN

ELSE                              #We're considering scenario 2
    IF (LayerBuf2 > BufAvailable(Layer)) AND
      ((S1Backoffs > K_max) OR
      (LayerBuf1 < BufAvailable(Layer)))
            SendPacketFromLayer(Layer)
            RETURN
```

K_{max} is the smoothing factor, giving the number of backoffs for which we buffer data before adding a new layer.

The function TotalBufRequired returns the total amount of required buffering for all layers in the scenario in question, given the current sending rate, the number of active layers, and the number of backoffs being considered.

Webgroup CZ a.s. Ex. 1015, Page 126 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TotalBufRequired()

Scenario 1

$$Buf_{total} = 0 \quad ; \quad k \leq \log_2 \frac{R}{n_a C}$$

$$Buf_{total} = \frac{1}{2S}\left(n_a C - \frac{R}{2^k}\right)^2 ; \quad k > \log_2 \frac{R}{n_a C}$$

where k is the number of backoffs being considered

(5.13)

Scenario 2

$$Buf_{total} = 0 \quad ; \quad k \leq \log_2 \frac{R}{n_a C}$$

$$Buf_{total} = \frac{1}{2S}\left(\left(n_a C - \frac{R}{2^{k_1}}\right)^2 + (k - k_1)\left(\frac{n_a C}{2}\right)^2\right)$$

$$k_1 = \left\lceil \log_2 \frac{R}{n_a C} \right\rceil \quad ; \quad k > \log_2 \frac{R}{n_a C}$$

The function BufRequired returns the maximally efficient amount of required buffering for a particular layer in the scenario of the state we are currently working towards. The input parameters for this function are: the layer number, the current sending rate, the number of active layers, and the number of backoffs being considered.

Webgroup CZ a.s. Ex. 1015, Page 127 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BufRequired()

Scenario 1

$$
\begin{aligned}
Buf_{i,opt} =& 0 ; \quad k \leq \log_2 \frac{R}{n_a C} \\
Buf_{i,opt} =& \frac{C}{2S}(C(2n_a - 2i - 1) - \frac{R}{2^{k-1}}) \\
& k > \log_2 \frac{R}{n_a C} ; \quad 0 \leq i < n_b
\end{aligned}
\tag{5.14}
$$

Scenario 2

$$
\begin{aligned}
Buf_{i,opt} =& 0 ; \quad k \leq \log_2 \frac{R}{n_a C} \\
Buf_{i,opt} =& \frac{C}{2S}\left((C(2n_a - 2i - 1) - \frac{R}{2^{k_1-1}}) + (k - k_1)C(n_a - 2i - 1) \right) \\
& k > \log_2 \frac{R}{n_a C} ; \quad 0 \leq i < n_b
\end{aligned}
$$

5.4.2 Draining Phase with Smoothing

As we traverse through the maximally efficient states, one or more backoffs eventually move us into a draining phase. Given that we incrementally traverse the maximally efficient *path* of buffer states during the filling phase, we would like to traverse the same path, but in the reverse direction, during the draining phase. This approach guarantees that the highest layer buffers are not drained until they are no longer required, and the lowest layer buffers are not drained too early.

At the start of each step we have an efficient amount of protective buffering for one particular state, and regressively work toward the previous maximally efficient buffer state along the maximally efficient path. However, there is an additional constraint that we can not drain a layer's buffer faster than the layer consumption rate (i.e. C).

To achieve such a draining pattern, we periodically calculate the draining pattern for a short period of time, during which we expect to drain a certain number of packets. This number is based on the current estimate of slope of linear increase and the current

Webgroup CZ a.s. Ex. 1015, Page 128 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

consumption rate. We then calculate (using an algorithm similar to the above pseudo-code) the previous optimal state along the maximally efficient path that we can achieve with the current amount of buffering. Conceptually, then we consider draining data from each layer in turn, starting from the highest layer and working downwards, such that each layer's buffering does not drop below its buffer share at the previous optimal step we are draining towards. An added constraint is that we must limit the amount of drained data from a layer to the maximum amount that can be consumed during this period. If the buffer state reaches the previous optimal state being considered before we have allocated the number of packets that must be drained in this period, then we move on to consider the previous state along the maximally efficient path and so on. We repeat this process until a sufficient number of packets for draining during this period are identified. Then we allocate the bandwidth during the period such that each active layer receives the total amount of data that it must consume during this period, minus those packets we just allocated to drain during the period.

5.5 Simulations

We have evaluated our quality adaptation mechanism through simulation using bandwidth traces obtained from RAP in the ns2 [87] simulator and real Internet experiments.

Figure 5.12 provides a detailed overview of the mechanisms in action. It shows a 40 second trace where the quality-adaptive RAP flow co-exists with 10 Sack-TCP flows and 9 additional RAP flows through an 800 KB/s bottleneck with 40ms RTT. Figure 5.13 also showed the first 5 seconds of figure 5.13 for better demonstration. The smoothing factor was set to 2 so that it provides enough receiver buffering for two backoffs before adding a new layer($K_{max} = 2$). The consumption rate of each layer(C) is equal to 10 KB/s.

Figure 5.12 and 5.13 show the following parameters:

- The total transmission rate, illustrating the saw-tooth output of RAP. We have also overlaid the consumption rate of the active layers over the transmission rate to demonstrate the add and drop mechanism.

Webgroup CZ a.s. Ex. 1015, Page 129 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- The transmission rate broken down into bandwidth per layer. This shows that most of the variation in available bandwidth is absorbed by changing the rate of the lowest layers (shown with the light-gray shading).
- The individual bandwidth share per layer. Periods when a layer is being streamed above its consumption rate to build up receiver buffering are visible as spikes in the bandwidth.
- The buffer drain rate per layer. Clearly visible are points where the buffers are used for playout because the bandwidth share is temporarily less than the layer consumption rate.
- The accumulated buffering at the receiver for each active layer.

Graphs in figure 5.12 and 5.13 demonstrate that the short-term variations in bandwidth caused by the congestion control mechanism can be effectively absorbed by receiver buffering. Furthermore, playback quality is maximized without risking complete dropouts in the playback due to buffer underflow.

5.5.1 Smoothing Factor

To examine the impact of smoothing factor on the behavior, we repeated the previous simulation with different values of K_{max}. Figure 5.14 shows the number of active layers and buffer allocation across active layers for K_{max}=2, K_{max}=3, and K_{max}=4. As expected, higher values of K_{max} reduce the number of changes in quality at the expense of increasing the time it takes to first achieve the best short-term quality. This manifests itself in two ways. As K_{max} increases, first the total amount of buffering is increased. Second, more of the buffering is allocated for higher layers to cope with the larger variations in available bandwidth as a result of successive backoffs.

5.5.2 Responsiveness

We have also explored the responsiveness of the quality adaptation mechanism to large step changes in available bandwidth. Figure 5.15 depicts a RAP trace with the same

Webgroup CZ a.s. Ex. 1015, Page 130 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

parameters as figure 5.12 but a CBR source with a rate equal to half of the bottleneck bandwidth is started at t=30s and stopped at t=60s and K_{max}=4. The RAP congestion control mechanism rapidly responds to these changes by adjusting the average transmission rate. The quality adaptation mechanism closely follows the changes in bandwidth. L_3 and then L_2 are dropped when bandwidth reduces and then $L2$ is added when bandwidth becomes available again. Notice that every layer's buffer is involved in this process, but the reception of the base layer is never jeopardized. Thus, we have satisfied our original design goal of providing smoothing of quality while providing protection to the most critical layers.

5.5.3 Efficiency

The performance of our algorithms can be examined from the efficiency of the buffer allocation. The inter-layer buffer allocation is maximally efficient if the following conditions are both satisfied: (i) no data is buffered for a layer that is dropped, and (ii) the layer is only dropped because the *total* amount of buffering is insufficient. To quantify the efficiency of our scheme, we have calculated the percentage of remaining buffer for each dropped layer as follows:

$$e = \frac{buf_{total} - buf_{drop}}{buf_{total}} \tag{5.15}$$

where buf_{total} and buf_{drop} denote the total buffering and the buffer share of the dropped layer. Then we averaged out the value of e across all drop events during the simulation and use that as an evaluation metric for efficiency.

Table 5.2 shows these efficiency values for different values of K_{max} during two test, T1 and T2. T1 is the 10 RAP, 10 TCP test depicted in figures 5.12 whereas T2 is the 10 RAP, 10 TCP test with a large CBR burst shown in figure 5.15. These efficiency values show the mean percentage of remaining buffer after a layer is dropped. These results show that our scheme is very efficient - very little buffered data is still available in a layer that

Webgroup CZ a.s. Ex. 1015, Page 131 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

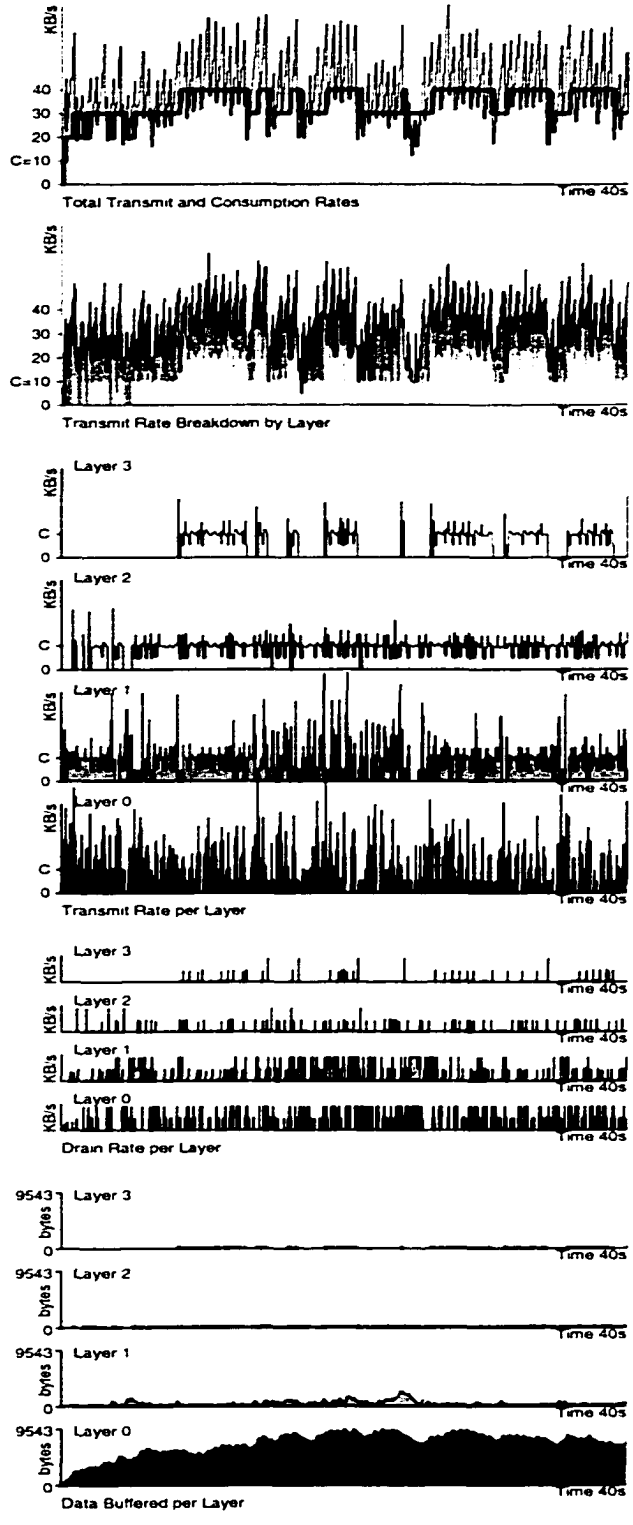


Figure 5.12: First 40 seconds of K_{max}=2 trace

Webgroup CZ a.s. Ex. 1015, Page 132 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

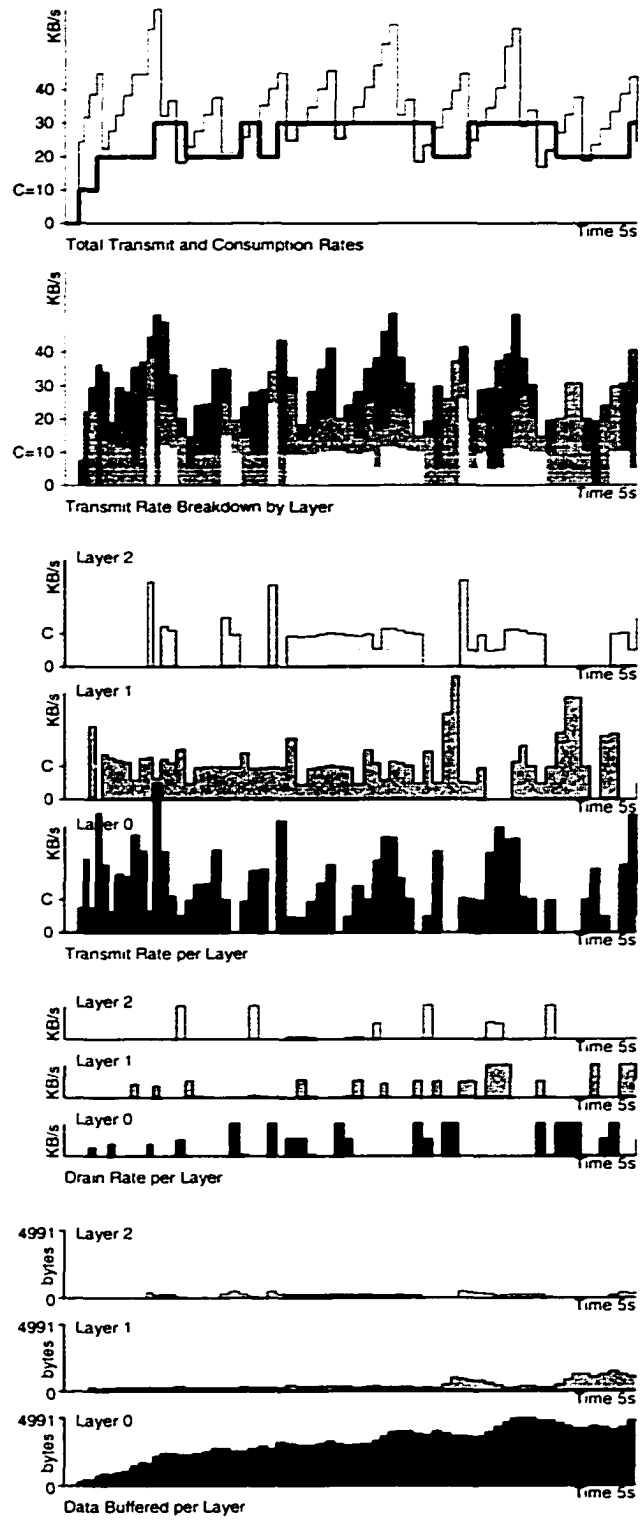


Figure 5.13: First 5 seconds of K_{max}=2 trace

Webgroup CZ a.s. Ex. 1015, Page 133 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

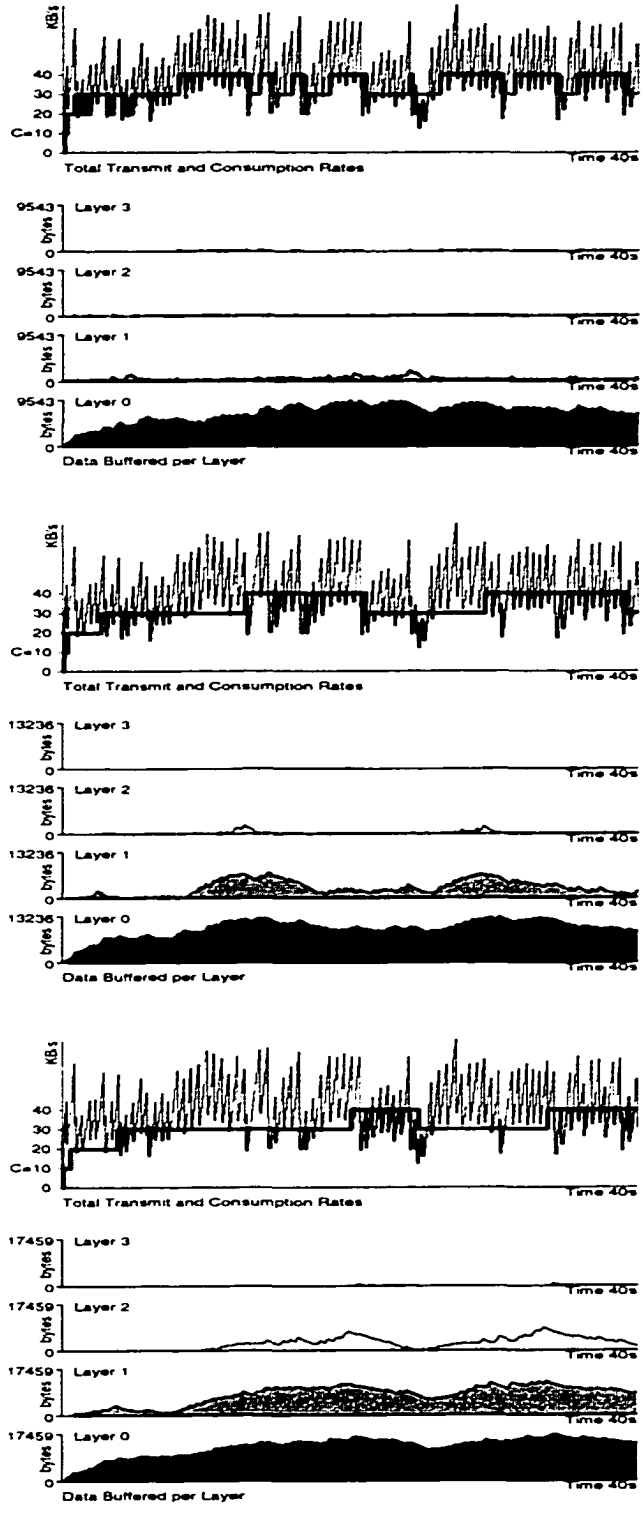


Figure 5.14: Effect of K_{max} on buffering and quality

Webgroup CZ a.s. Ex. 1015, Page 134 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

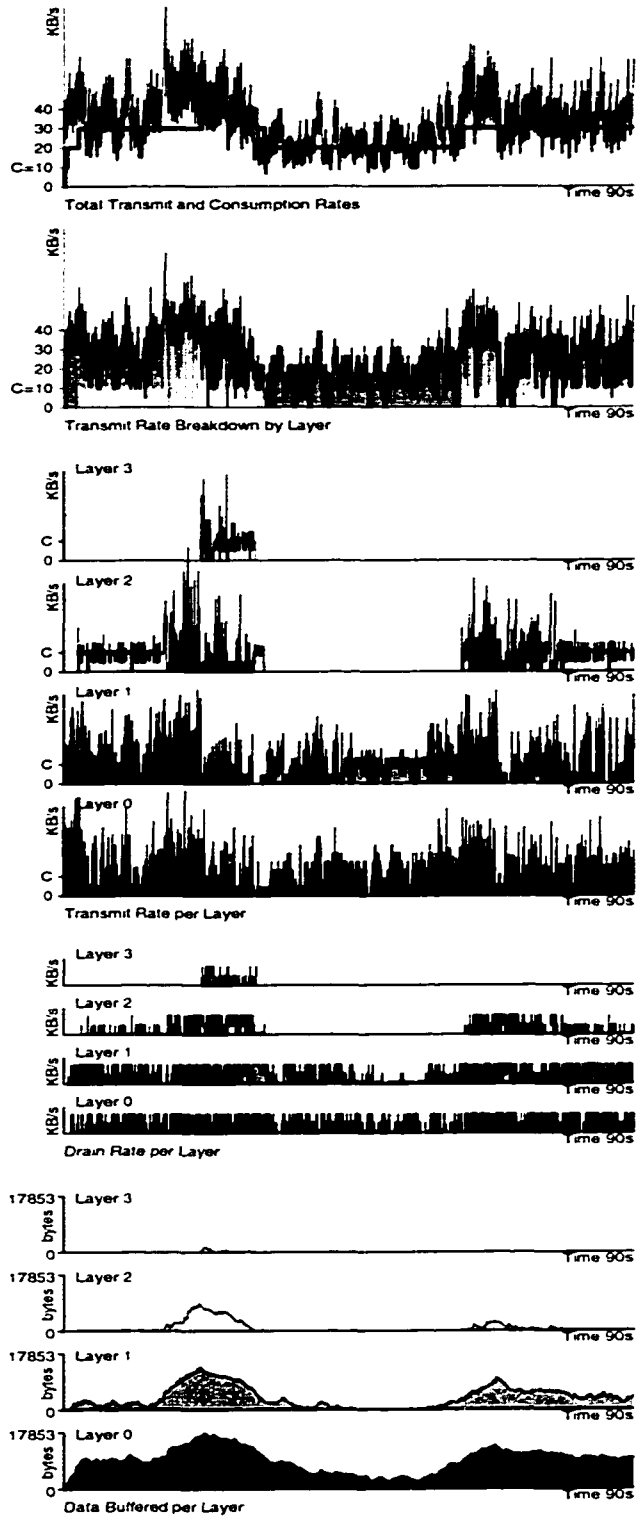


Figure 5.15: Effect of long-term changes in bandwidth

Webgroup CZ a.s. Ex. 1015, Page 135 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is dropped.

	K_{max}=2	K_{max}=3	K_{max}=4	K_{max}=5	K_{max}=8
T1	99.77%	99.97%	99.84%	99.85%	99.99%
T2	99.15%	99.81%	99.92%	99.80%	96.07%

Table 5.2: Efficiency of buffer allocation

Table 5.2 shows the percentage of drops due to poor buffer distribution in test T1 and T2. These are drops that would not have happened if the amount of buffered data that was at the receiver had been distributed differently. Our mechanism is completely efficient in this respect for the T1 tests, and performs fairly well for the T2 case. Clearly the mechanism becomes less efficient as K_{max} increases. The higher the value of K_{max}, the more buffering is allocated for higher layers. Hence there is a higher probability of dropping the highest layer with some buffering particularly after sudden drops in available bandwidth such as when a CBR source appears. In essence, conservative buffering(i.e. higher K_{max}) enables the server to cope with wider variations in bandwidth. However sudden drops of bandwidth in these situations results in lower efficiency.

	K_{max}=2	K_{max}=3	K_{max}=4	K_{max}=5	K_{max}=8
T1	0%	0%	0%	0%	0%
T2	2.4%	0%	4.8%	11%	-

Table 5.3: % drops due to poor buffer distribution

5.6 Summary

In this chapter, we presented a layered approach to quality adaptation that is well suited to congestion controlled multimedia playback. The quality adaptation mechanism adjusts

Webgroup CZ a.s. Ex. 1015, Page 136 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the quality of the delivered stream on-the-fly as the available bandwidth changes unpredictably.

Instead of solving the problem for a specific encoding, we addressed the general trade-off between short term improvement and long term smoothing in the delivered quality. The quality adaptation mechanism introduces a tuning parameter called Smoothing factor that allows the server to efficiently control the level of smoothing in the delivered quality. Thus one can tune the mechanism for a specific encoder to maximize the quality based on the applications needs.

The quality adaptation mechanism consists of two main components as follows:

1. Coarse grain add and drop mechanism.

2. Fine grain bandwidth allocation among active layers.

We assumed that the underlying congestion control mechanism performs AIMD rate adaptation and streams are linear layer encoded. We studied the buffer state that maximizes the efficiency of buffered data for any pattern of changes in bandwidth. Given th relationship between buffer state and bandwidth allocation among active layers, we inferred efficient solutions for two components of the mechanism.

We have evaluated the quality adaptation mechanism using simulations. Our results show that the mechanism can efficiently trade short-term improvement with long-term smoothing. Furthermore, the buffer requirement to achieve proper level of smoothing is relatively low(i.e. a few seconds of stream). Thus, we believe that this mechanism is applicable to live but non-interactive streams as well.

Webgroup CZ a.s. Ex. 1015, Page 137 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Multimedia Proxy Caching

The quality(i.e. bandwidth) of the delivered stream in the end-to-end client-server approach is limited to the bottleneck bandwidth between the server and the client. If the server is located across the Internet, a client with high bandwidth local connectivity to the network may still receive low quality streams due to congested links somewhere between the point of network attachment and the server. Clearly, if the client has only low-bandwidth connectivity to the network (i.e. the bottleneck is the last hop), the delivered quality can not be improved. However, clients with high bandwidth connectivity expect to receive high quality streams. There are two potential solutions to overcome bandwidth limitations between the server and these high bandwidth clients:

1. Replication, i.e. Mirror servers
2. Multimedia proxy caching

Having mirror servers scattered across the Internet improves the chance for a client to find a server that is reachable via a high bandwidth path. However, this static approach is expensive especially for those applications where the content of the server frequently changes, e.g. news server. Mirror servers must have the same amount of storage as the original server and must be updated by the original server after any change regardless of the level of interest (or disinterest) among associated clients of the mirror server.

In this chapter we explore an adaptive solution to this problem using multimedia proxy caching. A proxy server is a small server that resides close to a group of clients. The required amount of storage(i.e. cache space) for a proxy is proportional to the number of

Webgroup CZ a.s. Ex. 1015, Page 138 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

local clients and it is substantially lower than the original server. Requested streams are always delivered through the proxy, thus it is able to intercept and cache them. However, the cached streams and their corresponding qualities are adaptively adjusted based on both popularity of each stream among clients and available bandwidth between the proxy and interested clients. The proxy server can significantly increase the delivered quality of popular streams to high bandwidth clients despite the presence of a bottleneck on the path to the original server. Due to low storage and processing requirements for a proxy, any institution can easily deploy a proxy server to improve their perceived quality.

A primary challenge for multimedia proxy caching in the Internet is the need to operate within the context of congestion control. Performing TCP-friendly congestion control such as RAP results in random and potentially wide variations in transmission rate. To maximize the delivered quality to clients while obeying congestion controlled rate limits, streaming applications should perform quality adaptation- that is, they should match the quality of the delivered stream with the average available bandwidth on-the-fly. Thus the quality of cached streams will not only depend on the available bandwidth to the first client that retrieved the stream but also varies with time. Once the stream is cached, the proxy can replay it from the cache for subsequent requests but it still needs to perform congestion control and quality adaptation based on the state of the connection between the proxy and the client. This connection is likely to exhibit different characteristics(e.g. different average bandwidth, changes in background traffic) from previous sessions. For this reason, variations in the quality of the cached stream are not correlated with the required changes in playback quality due to quality adaptation during the new session. This implies that the proxy can not effectively perform quality adaptation to maximize delivered quality to the client.

Layered organization of the stream provides an opportunity to adjust the quality of the cached stream in a demand-driven fashion. To allow fine-grain adjustment of quality, each layer of the encoded stream is divided into equal-size pieces called *segments*. Thus the proxy can pre-fetch those segments that are required by the quality adaptation mechanism and are missing in the cache. If available bandwidth between the proxy and a client can support a stream with a higher quality, higher layers are gradually pre-fetched from the

Webgroup CZ a.s. Ex. 1015, Page 139 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

server to improve the quality. Thus the quality of the cached stream is adjusted with its popularity(i.e. with the number of times it is played back).

Rapid increase in the volume of multimedia traffic on the Internet justifies the need for multimedia proxy caching. Besides improving the delivered quality to high bandwidth clients, proxy caching of popular streams close to interested clients has several other advantages for both high and low-bandwidth(e.g. dial-up) clients:

- Supporting low-latency VCR-functions for clients.
- Supporting asynchronous access.
- Minimizing startup latency.
- Reducing load on the server and the network.

Proxy servers have not been widely used for caching of Internet multimedia streams such as audio and video yet. We believe this might be due to the following reasons:

- Realtime streams such as video are several orders of magnitude larger than typical web objects. Current replacement algorithms may discriminate against caching larger objects.
- The number of embedded multimedia streams on the Web has been fairly limited. Moreover, access patterns to these streams are not well understood.
- Lack of an open, well-accepted and well-behaved transport protocol for streaming applications.

However as bandwidth and sever capacity increase, the demand and ability to support streaming applications are expected to develop rapidly. Realtime streams have several inherent properties that can be exploited in the design of effective multimedia proxy caching mechanisms.

- Because of large size and duration of delivery, the entire object need not be sent at once. Instead, the server can pipeline the data to the client through the network.
- Multimedia streams are able to change their size by adjusting their quality.

Webgroup CZ a.s. Ex. 1015, Page 140 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The rest of this chapter is organized as follows: we present the proxy-based architecture in the next section. Then in section 6.2, we present the delivery procedures on cache-hit and cache-miss scenarios in order to demonstrate role of the proxy in each scenario. We also discuss the pre-fetching mechanism during the cache hit scenario to cope with variations in quality and address some of the related challenges and trade-offs. Different aspects of our replacement algorithm including fine and coarse-grain replacement pattern as well as popularity function are described in section 6.3. Finally, we present our simulation environment and some of our simulation results in section 6.4.

6.1 The Proxy-based Architecture

To include multimedia proxy caches, we extend our end-to-end client/server architecture that was presented in chapter 3. Figure 6.1 shows such an extended architecture. Notice that this architecture is still end-to-end (i.e. proxy servers are end systems) and do not require any support from the network. All streams are layered encoded and stored at the server's archive. Here we also assume linear-layered encoding where all layers have the same bandwidth just for the sake of simplicity, but the architecture and the caching scheme can be extended to other layered-encoding bandwidth distributions. Traffic is always routed through a corresponding proxy server that is associated with a group of clients in its vicinity. Thus the proxy is able to intercept each stream and cache it. All playback streams between the original server and the client or between the proxy server and the client must perform congestion control and quality adaptation. This implies that not only the original server, but also the proxy server, must be able to support congestion control and quality adaptation. We do not make any assumption about the inter-cache architecture. Our work is compatible with the various proposed inter-cache architectures proposed [23, 127, 139].

Replacement is performed at the granularity of a segment. Each segment can be as small as a single packet or as big as several minutes of a stream. Having small segments

Webgroup CZ a.s. Ex. 1015, Page 141 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

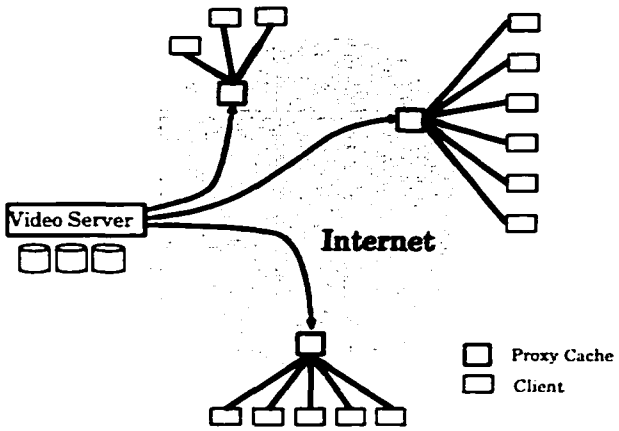


Figure 6.1: The end-to-end server/client/proxy architecture

prevents the cache space from becoming fragmented while large segments entail less co-ordination overhead. For a particular segment length, each segment is uniquely identified by the playout time of the first sample in that segment.

6.2 Delivery Procedure

Clients always send their requests for a particular stream to their corresponding proxy. When a proxy receives a request, it looks up the cache for availability of the requested stream. The rest of delivery procedure varies for a cache miss or a cache hit. We describe each scenario separately in the next two subsections.

6.2.1 Relaying on a Cache Miss

If the requested stream is missing from the cache, the request is relayed to the original server or neighbor caches, depending on inter-cache architecture. The original server plays back the stream to the client through the proxy. The proxy relays data packets toward the client and the ACK packets in the reverse direction. Thus the proxy remains virtually transparent from the end systems' point of view while it is able to intercept and cache each packet. The server performs end-to-end congestion control and quality adaptation based on the state of the session between the server and the client. The quality of the delivered stream is limited to the average bandwidth between the server and the client. Thus on a

Webgroup CZ a.s. Ex. 1015, Page 142 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cache miss scenario, the client does not observe any benefit(e.g. improvement in quality or lower startup latency) from the the presence of the proxy cache.

The proxy always caches a missing stream during its first playback. If cache space is exhausted, the replacement algorithm flushes a sufficient number of segments from the cache to make room for the new stream. Details of the replacement algorithm are discussed in section 6.3.

Since the original server performs quality adaptation, a cached stream has variable quality after its first playback. Furthermore, there might be occasional missing packets that have been lost and were not repaired during the first session because they have missed their playout times. Figure 6.2 shows variations in quality as well as missing packets for a portion of a sample cached stream. To perform quality adaptation effectively during subsequent playbacks from the cache, the proxy may smooth out the variations of the cached stream and repair the losses by pro-actively pre-fetching the missing segments during idle hours. Alternatively, the proxy may pre-fetch missing segments in a demand-driven fashion while it serves subsequent requests for the cached stream. We adopted the latter approach assuming the future access pattern is not predictable. We plan to investigate pro-active pre-fetching as part of our future work.

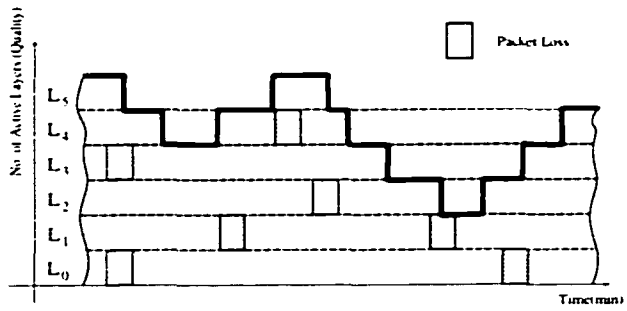


Figure 6.2: A sample quality adaptive stream in the cache

6.2.2 Pre-fetching on a Cache Hit

On a cache hit, the proxy acts as a server and starts playing back the requested stream from the cache. As a result the client observes shorter startup latency. The proxy must still perform congestion control and quality adaptation. However the connection between

Webgroup CZ a.s. Ex. 1015, Page 143 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the proxy and the client might have different characteristics(i.e. bandwidth, round trip time and background traffic). Moreover, there is no reason to expect that the variations in quality of the cached stream would be correlated with the variations in quality of the delivered stream (those that are due to quality adaptation by the proxy). This implies that the quality adaptation mechanism may be able to send some segments that do not exist in the cache. To maximize the delivered quality to the client, the proxy should pre-fetch the missing segments that are required by the quality adaptation mechanism from the server ahead of time. During each interval of the session two scenarios are possible:

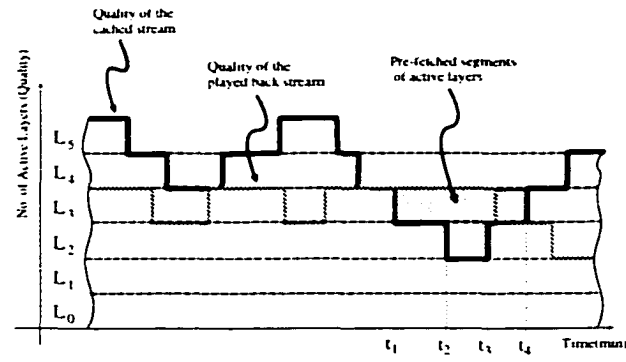


Figure 6.3: Delivery of lower bandwidth stream from the cache

1. $Playback_{AvgBw} \leq Stored_{AvgBw}$

2. $Playback_{AvgBw} > Stored_{AvgBw}$

where $Playback_{avgBw}$ and $Stored_{avgBw}$ denote average bandwidth of the playback session and the cached stream respectively. Note that during a complete session, the proxy may sequentially experience both of the above scenarios. Figure 6.3 and 6.4 illustrate these two scenarios. Figure 6.3 depicts a scenario where the average quality of the delivered stream is lower than the cached stream. However, there are segments that are required by the quality adaptation mechanism but are missing from the cache. For example, segments of layer 2 within the interval of $[t_2, t_3]$ and segments of layer 3 within the interval of $[t_1, t_4]$ are required by the quality adaptation mechanism but are not available in the cache. The second scenario is shown in figure 6.4 where the available bandwidth between the proxy and the client is sufficiently high to deliver a higher quality stream

Webgroup CZ a.s. Ex. 1015, Page 144 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

than the cached stream. Thus the proxy not only needs to pre-fetch missing segments of the lower layers, it may occasionally pre-fetch higher layers as soon as quality adaptation indicates possibility of adding a new layer in the future. All the pre-fetched segments during a session are cached in both scenarios.

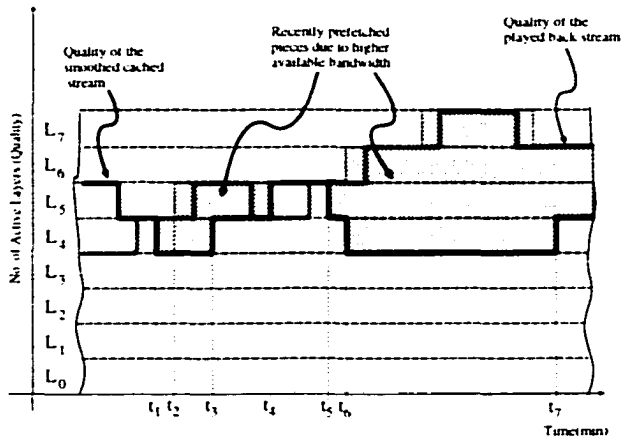


Figure 6.4: Delivery of higher bandwidth stream from the cache

6.2.3 Challenges and Trade-offs

During playback of a cached stream, the proxy needs to maintain two unsynchronized connections as shown in figure 6.5:

1. *Pre-fetching Stream*: the connection between the server and the proxy for pre-fetching and

2. *Playback Stream*: the connection between the proxy and the client for delivery of the stream.

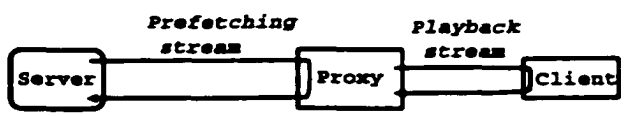


Figure 6.5: Pre-fetching and Playback Streams

Both connections are congestion controlled, however only the proxy performs quality adaptation for playback stream. Pre-fetching a segment from the server will take at least

Webgroup CZ a.s. Ex. 1015, Page 145 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

one RTT between the server and the proxy. To playback a missing segment during a session, the proxy must identify a missing segment that may be required by the quality adaptation mechanism in the future and send a pre-fetching request at least one RTT ahead of time.

Pre-fetching can be performed along two dimensions:

1. **Pre-fetching along time axis:** In this approach missing segments are pre-fetched based on their overall priorities. The proxy pre-fetches all the missing segments from the beginning to the end of a layer before pre-fetching any segments for higher layers. Since pre-fetching priority is determined based on only layer number, the pre-fetching and playback session are totally unsynchronized. Consequently, this approach does not necessarily improve the quality for all the playback sessions. This approach is more suited to off-line pre-fetching when the proxy pre-fetches data during idle hours without any timing constraint. Figure 6.6 shows the order of pre-fetching in this approach.

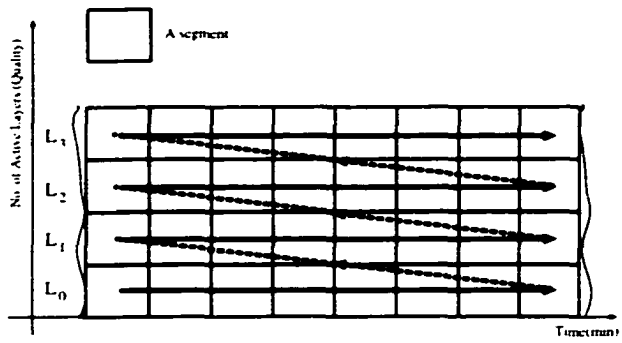


Figure 6.6: Pre-fetching along time axis

The main challenge is to determine the appropriate target quality, i.e. number of layers that must be pre-fetched. The proxy can use the bandwidth information of the previous sessions to estimate the number of useful layers in the cache. However, it is not clear how closely past information predicts future access.

2. **Pre-fetching along quality(i.e. space) axis:** In this approach, the goal is to improve short-term quality. Thus the pre-fetching bandwidth is used to maximize near-future

Webgroup CZ a.s. Ex. 1015, Page 146 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

quality along the quality axis. Figure 6.7 depicts pre-fetching pattern along quality axis.

The near-future target quality can be estimated by the quality adaptation mechanism. Since the quality adaptation mechanism adjusts the number of active layers with the random changes in available bandwidth, the time for the upcoming adjustment(i.e. adding or dropping of a layer) is not known a priori. This implies that the proxy is facing a trade-off, the earlier the proxy requests for pre-fetching of a missing segment, the less accurate the prediction would be, however the higher is the chance of receiving the requested segment in time. The main challenge in this approach is to synchronize pre-fetching and playback sessions.

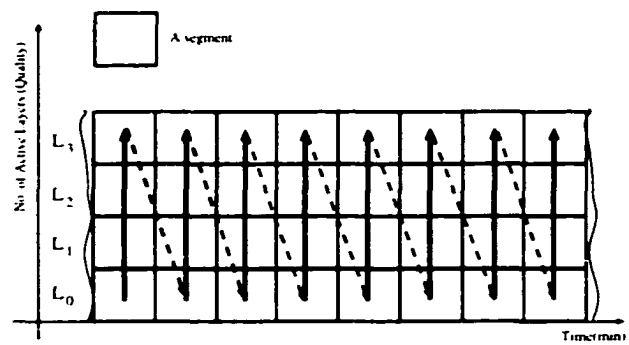


Figure 6.7: Pre-fetching along quality axis

The rate of pre-fetching requests from the proxy to the server depends on the available data in the cache as compared with available bandwidth between the proxy and interested client. However pre-fetched segments are delivered in a congestion controlled fashion from the server to the proxy. Thus if the requested rate for pre-fetching exceeds the available bandwidth between the server and the proxy, the server should deliver the requested segments based on their priority otherwise the pre-fetching stream will fall behind the playback stream and pre-fetched segments may miss their playout deadline.

To address this problem, we have devised a window-based pre-fetching mechanism. The main idea is to pre-fetch along the time axis within a window while sliding the window with playback speed to keep both sessions synchronized. Figure 6.8 shows the order

Webgroup CZ a.s. Ex. 1015, Page 147 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

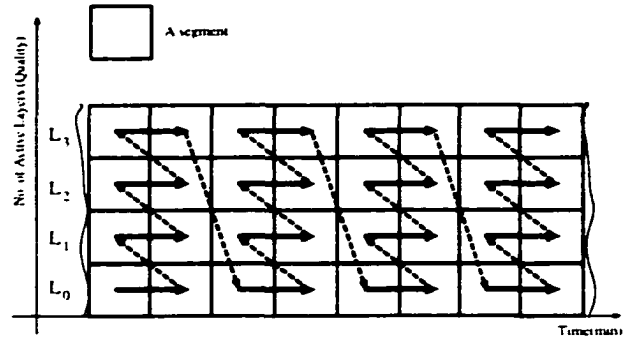


Figure 6.8: Pre-fetching pattern in a Win-based approach for window size of 2 segments

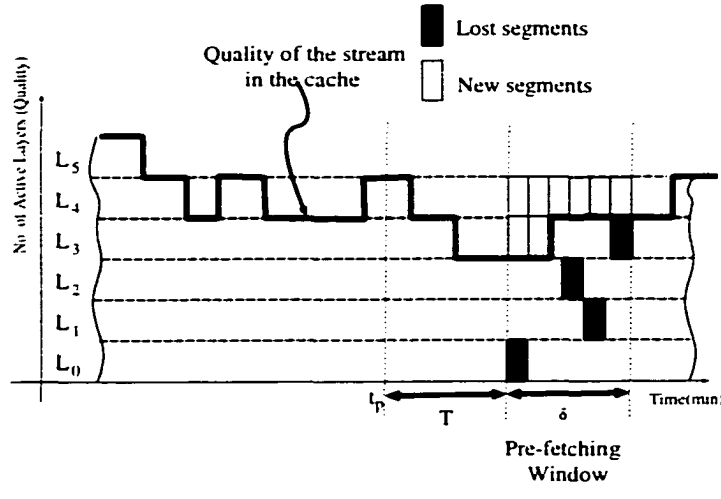


Figure 6.9: pre-fetching mechanism

Webgroup CZ a.s. Ex. 1015, Page 148 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of pre-fetching in the window-based approach for a window size of 2 segments. A window size of 1 segment or ∞ results in pre-fetching along the quality or time axis, respectively.

The window-based pre-fetching mechanism is illustrated in figure 6.9. The proxy maintains a playout time for each active client. At playout time t_p, the proxy examines the interval of $[t_p + T, t_p + T + \delta]$, called the pre-fetching window of the cached stream, and identifies all the missing segments within the pre-fetching window. The missing segments include any lost segments and the segments of current active layers within the pre-fetching window that have not been played back [1]. Furthermore, if the quality adaptation mechanism is close [2] to adding a new layer, any missing segment of the new layer within the pre-fetching window is included in the pre-fetching request as well. This mechanism enables the proxy to improve the quality during a playback if there is sufficient pre-fetching bandwidth available. Then the proxy sends a single pre-fetching request to the server that refers to a batch of missing segments in the current pre-fetching window. As we mentioned earlier, T must be larger than $RTT_{server-proxy}$ otherwise pre-fetched segments are useless for the active session.

To loosely synchronize pre-fetching stream with a playback stream, the pre-fetching window should slide as fast as the playout point. Thus after δ second, the proxy examines the next pre-fetching window and sends a new pre-fetching request to the server.

When the server receives a pre-fetching request, it starts sending the requested segments based on their priorities. Thus it first sends all the requested segments of layer 0, then requested segments of layer 1, etc. A new pre-fetching request preempts the previous one. If the server receives a new pre-fetching request before delivery of all the requested segments in the previous request, it simply ignores the old request and starts delivery of the segments in the new request based on their priorities. The preempting mechanism results in pre-fetching high priority segments while still limiting the pre-fetching rate to the congestion controlled rate limit. Furthermore, it causes the pre-fetching and the playback to proceed with the same rate. Notice that the average improvement in quality of a cached

[1] In the absence of an error control mechanism, our pre-fetching mechanism repairs loss segments as well. However adding an error control mechanism does not affect our pre-fetching scheme at all.

[2] The decision is made based on the state of receiver buffer for a given value of smoothing factor. For example, if buffer state is 90% close to adding condition, the proxy assumes that a new layer will be added in a near future.

Webgroup CZ a.s. Ex. 1015, Page 149 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

stream after each playback is determined by the average pre-fetching bandwidth. Thus it may take several playbacks until the quality of the cached stream reaches the maximum quality that can be viewed by a high bandwidth client.

If the proxy receives all the requested segments, it slides the pre-fetching window forward and sends a new pre-fetching request. As a result, the connection remains idle for one RTT. To avoid this, the proxy can always send a batch of requested segments for two consecutive windows. If the server delivers all the requested segments in the first window, it starts delivery of the segments in second window until a new pre-fetching request arrives. Within each window, pre-fetching order is similar to the above description. Since the proxy slides its window δ seconds at a time, every two consecutive pre-fetching requests have one window overlap. The pre-fetched segments are always cached even if they arrive after their playout times.

The value of δ might affect variations in playback quality from the cache. Small values for δ result in short-term improvement but might cause variations in quality. However large values for δ result in long-term smoothing.

When the proxy deals with several pre-fetching sessions simultaneously, some of the pre-fetching sessions might be multiplexed while a separate RAP connection is established for other sessions depending on its resource management policy and priority of different clients. We do not address this scenario in this dissertation.

6.3 Replacement Algorithm

This section addresses different aspects of our replacement algorithm. We exploit inherent properties of multimedia streams to design an effective replacement algorithm for layered encoded multimedia streams. For a given pre-fetching mechanism, The replacement algorithm must be designed such that its interactions with the deployed pre-fetching mechanism result in an appropriate state in the cache for the expected functionality from the proxy. We assume that it is generally prefered to have a complete stream in the cache and adjust its overall quality based on its popularity. This is in fact the only way to effectively hide low bandwidth connectivity to the server and improve overall quality. Thus

Webgroup CZ a.s. Ex. 1015, Page 150 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the chief goal of the caching mechanism is to converge the state of the cache to an efficient state after several playbacks. We define that the state of the cache is efficient if the following qualitative conditions are met:

1. The *average quality* of any cached stream is directly proportional to its popularity. Furthermore, the average quality of the stream must converge to the average bandwidth across the most recent playback for interested clients.

2. The *variations in quality* of any cached stream are inversely proportional to its popularity, i.e. the more popular a cached stream, the less variations in the quality of the stream.

We first describe coarse and fine-grain replacement patterns that are suited to layered-encoded streams. Then we extend semantics of a "hit" from Web caching and define a simple popularity function that captures both the level of interest among users who interactively perform VCR-functionalities and the value of a layer in the cache. Finally, we show how our replacement algorithm can be extended to caches with different functionalities.

6.3.1 Replacement Pattern

Current replacement algorithms do not exploit internal structure for some of the existing structured Web objects(e.g. layered JPEG images). This seems to be due to the following issues:

- Nature of access to the Web objects seems to be binary. The client is either interested in the entire object or is not interested at all.

- Most of the current Web objects are not structured, e.g.. Text documents.

Consequently each object is usually treated as an atomic object. A client requests and receives the entire object. Thus most of the replacement algorithms for Web caching make a binary decision for replacement of web objects, i.e. the least popular object is flushed in its entirety. However, layered encoded streams are inherently structured into

Webgroup CZ a.s. Ex. 1015, Page 151 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

separate layers. Furthermore, each layer is divided into the same number of segments with a unique segment ID. This organization provides a good opportunity not only to make a *multi-valued* replacement decision but also to perform replacement with different granularities. The stream popularity primarily affects the quality of a cached stream and then its status of residency in the cache. As the popularity of a cached stream decreases, its quality and consequently its size is reduced in several steps before it is completely flushed out.

Figure 6.10 depicts the replacement pattern for segments within a single cached stream. The coarse-grain replacement is achieved by dropping the highest layer, called the *victim layer*, from the cache. However, to maximize the efficiency of the cache and avoid fragmentation of the cache space, the victim layer is dropped with the granularity of a segment.

It is generally prefered to cache a contiguous portion from the beginning of a layer to absorb startup latency and minimize the variations in quality. Thus once a victim layer is identified, its cached segments are flushed from the end towards the beginning in a demand-driven fashion. If flushing all segments of the victim layer does not accommodate sufficient space for a new stream, the proxy identifies a new victim layer and repeats the fine-grain replacement process.

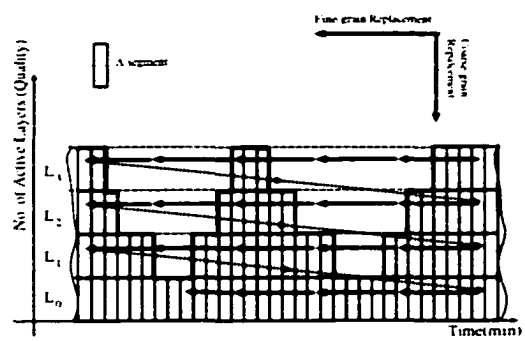


Figure 6.10: Replacement priority within a cached stream

Webgroup CZ a.s. Ex. 1015, Page 152 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.3.2 Popularity Function

We initially used number of hits(i.e. requests) for a cached resident stream during an interval, called *popularity window*, as a metric to measure its popularity. Most of the current Web caching schemes assign a binary value to a hit, i.e. 0 for lack of interest and 1 for each request. This model perfectly suits the binary nature of Web access. However, in the context of streaming applications, the client can interact with the server and perform VCR-functionalities (i.e. Stop, Fast forward, Rewind, Play). Intuitively, the popularity of each stream should reflect the level of interest that is observed through this interaction. We assume that the total duration of playback for each stream indicates the level of interest in that stream. For example if a client only watches half of one stream, its level of interest is half of a client who watches the entire session. This approach can also include weighted duration of fast forward or rewind with proper weighting.

Based on this observation we extend the semantics of a hit and define the term *weighted hit* (*whit*) which is defined as follows [3]:

$$whit = \frac{PlaybackTime}{StreamLength}, \quad 0 \leq whit \leq 1 \tag{6.1}$$

where $PlaybackTime$ and $StreamLength$ denote total playback time of a session and length of the entire stream, respectively. Both $PlaybackTime$ and $StreamLength$ have dimension of time (i.e. measured in seconds).

Notice that adding and dropping layers by the quality adaptation mechanism results in a different $PlaybackTime$ for various active layers in a session and consequently affects the value of a cached *layer*. For example, even if all layers of a stream are available in the cache and the client watches the entire stream, the quality adaptation mechanism may only send Layer 0, 1 and 2 for 100%, 80%, 50% of the play back time respectively. Since the available bandwidth directly controls the number of active layers, the longer a layer is played back for interested clients during recent sessions, the higher is the probability of

[3]The term "weighted hit" have been used in caching literature [21] to extend the definition of hit in the context of Web caches. However we have introduced a new definition for this term in the context of proxy caching for multimedia streams.

Webgroup CZ a.s. Ex. 1015, Page 153 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

using that layer in future sessions. To capture the value of a layer, the server calculates the value of weighted hits on a per-layer basis for each session. The total playback time for each layer is recorded and used to calculate the *whit* for that layer at the end of the session. The cumulative value of *whit* during a recent window is used as a popularity index of a layer of a cached stream. The popularity of each layer is recalculated at the end of a session as follows:

$$P_i(t) = \sum_{\tau=t-\Delta}^{t} whit_i(\tau) \tag{6.2}$$

where $P_i(t)$, $whit_i(\tau)$ and Δ denote popularity of layer i at time t, value of weighted-hit for layer i for playback session at time τ and the width of the popularity window respectively. Applying the definition of popularity on a per-layer basis is in fact compatible with our proposed fine-grain replacement mechanism because layered decoding guarantees that popularity of different layers of each stream monotonically decreases with the layer number[4]. Thus a victim layer is always the highest layer of one of the cached stream. Notice that length of a cached stream does not affect its popularity because replacement is performed at the granularity of a segment instead of a layer.

P	Lock	Stream name	Layer no.
5.85	1	*Titanic*	L_0
4.92	1	*Titanic*	L_1
4.76	0	*Amistad*	L_0
3.70	1	*Titanic*	L_2
3.50	0	*Contact*	L_0
3.33	0	*Apollo* 13	L_0
2.30	0	*Titanic*	L_3
1.28	0	*Amistad*	L_1

Table 6.1: Sample of a popularity table

[4]The decoding constraint requires that to decode a segment of layer i, corresponding segments for all the lower layers must be available

Webgroup CZ a.s. Ex. 1015, Page 154 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To implement this scheme, the proxy maintains a popularity table such as Table 6.1. Each table entry consists of stream name, layer number, popularity of the layer and a locking flag. Once the cache space is exhausted, the proxy flushes the last segments of the least popular layer (e.g. L_1 of "*Amistad*") until sufficient space becomes available. Popularity of this layer could be low due to lack of interest among clients, or lack of sufficient bandwidth to play this layer for interested clients, or both[5]. Except for the base layer of each stream, all segments of other layers can be flushed out if they are the least popular layer. The first few segments of the base layer for each stream are kept in the cache for a long period to hide the startup latency of possible future requests.

6.3.3 Locking Mechanism

The replacement for a web object is an atomic operation. However, in the context of multimedia streams, replacement is a timely process that proceeds gradually as the session continues. This causes the potential for thrashing where the tail of the highest layer of the cached stream (e.g. L_3) is flushed to make room for the initial segments of a higher layer (e.g. L_4) of the same stream during a playback session with higher proxy-client bandwidth. To avoid this, while a particular stream is played back from the cache, its layers are locked in the cache and can not be replaced. In practice, each layer is locked as soon as it is played back for the first time and remains locked until the end of the session. Thus if a layer has never been played during the session then it does not need to be locked. At the end of the session, the weighted hit of each layer is calculated and consequently the popularity value of each layer is updated in the popularity table. Then all the locked layers of that stream are unlocked.

6.3.4 Supporting Other Caching Functions

As we mentioned earlier, both the replacement pattern and popularity function are directly determined by the expected functionality from the proxy. For example the main

[5]Note that these three scenarios can be easily recognized from the distribution of popularity values among layers. Close popularity values imply lack of clients interest whereas widely variable popularity values imply lack of available bandwidth to play the less popular layers.

Webgroup CZ a.s. Ex. 1015, Page 155 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

functionality of the proxy might be to cache the most popular chunks of different streams where each chunk consists of a group of adjacent segments. We can simply achieve this by treating every chunk of each stream as an individual stream and deploy our fine-grain replacement and popularity function without any modifications. Clearly, this scheme requires larger popularity table and may result in more variations in quality.

A special case is a proxy that is only expected to hide the startup latency[116], the proxy should adopt a different replacement pattern to maintain initial segments of all layers. The new replacement pattern flushes ending segments of lower layers before initial segments of higher layers.

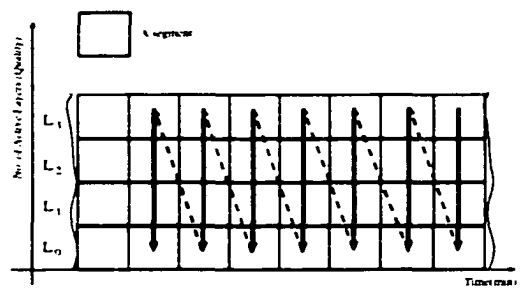


Figure 6.11: Vertical pattern of replacement

Figure 6.11 shows another potential flushing pattern that is more suited for caches where high quality startup with potential variations in quality is more important than completeness and smooth but lower quality stream. A trade-off between horizontal and vertical flushing is a snake-like pattern that is shown in figure 6.12. Evaluation of these patterns on cache performance remain as a future work.

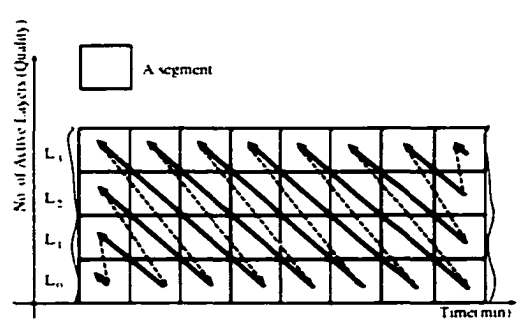


Figure 6.12: Snake-like pattern for replacement

Webgroup CZ a.s. Ex. 1015, Page 156 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.4 Simulations

We evaluate our multimedia caching mechanism via simulation using the *ns* [7] simulator. We use RAP (for congestion control) along with layered quality adaptation 5 as transport protocol for multimedia streams. Note that we did not include error control mechanisms.

As discussed earlier, our caching mechanism has two major design goals: prefetching during playback to enhance cached stream quality, and fine-grain replacement to adjust stream quality based on per-layer popularity. In this paper, we are merely interested in qualitatively evaluating the correctness of this mechanism and in verifying whether and how this mechanism satisfies these design goals. It remains as future work to examine the *performance* of this mechanism, such as the byte hit ratio and the convergence of our replacement algorithm, under realistic background traffic.

6.4.1 Evaluation Metrics

To examine the correctness of our mechanism, we choose the following two metrics that collectively represent the resulting quality of a cached stream:

- *Completeness* measures the percentage of a stream residing in the cache. This metric allows us to trace the quality evolution of a cached stream after each playback. Because our replacement algorithm is layer-based, we define the completeness on a per-layer basis. The completeness of layer l in cached stream s is defined as the ratio of the layer's size in cache to its "official size":

$$Cp(s,l) = \frac{\sum_{\forall i \in Chunks(l)} L_{l,i}}{RL_l} \tag{6.3}$$

  Here we define a *chunk* as a continuous group of segments of a single layer of a cached stream, and denote the set of all chunks of layer l as $Chunks(l)$. $L_{l,i}$ is the length (in terms of segments) of the ith cached chunk in layer l, and RL_l is the "official" length of the layer. Obviously the value of completeness always falls within [0,1]. If every byte of a stream is cached, each of its layers has completeness value of 1.

Webgroup CZ a.s. Ex. 1015, Page 157 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- *Continuity* measures the level of smoothing of a cached stream. Completeness alone does not capture this, because it does not reflect the number of "holes" in a cached stream. Continuity is also defined on a per-layer basis. The continuity of layer l in cached stream s is defined as the average number of bytes between two consecutive layer breaks, *i.e.*, average chunk size:

$$Ct(s,l) = \frac{\sum_{\forall i \in Chunks(l)} L_{l,i}}{LayerBreak + 1} \tag{6.4}$$

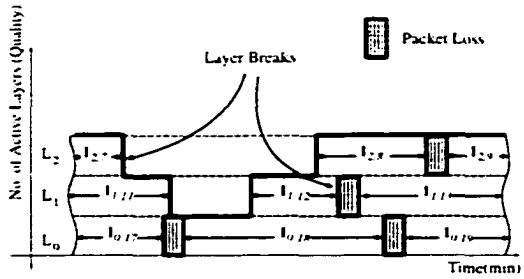


Figure 6.13: Average quality and layer breaks for a cached stream

A layer break occurs when there is a missing segment in a layer. It may be due to either quality adaptation dropping a layer or a packet loss. Fig. 6.13 illustrates this for a portion of a cached stream. Although layer-drop and packet loss are two fundamentally different phenomena, our prefetching algorithm does not distinguish between them. The prefetching mechanism copes with both of them similarly in a priority-based fashion. In the absence of an error control mechanism, our results represent worst case scenarios for convergence of the quality. Including error control in the transport protocol will speed up the prefetching process to fill these missing segments.

6.4.2 Simulation Setup

We have conducted two sets of simulations. The first focuses on evaluating the prefetching algorithm, and the second on the replacement algorithm. They both use a simple network topology shown in Fig. 6.14. BW_{sp} denotes the average available bandwidth be-

Webgroup CZ a.s. Ex. 1015, Page 158 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

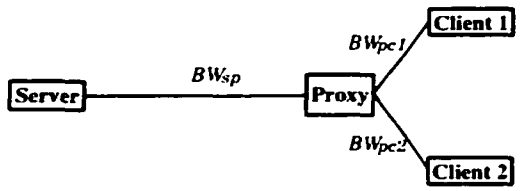


Figure 6.14: Simulation topology

tween server and proxy whereas BW_{pc1} and BW_{pc2} are physical link bandwidths between proxy and two clients, respectively. One may construct two interesting scenarios from this simple topology:

- Scenario I: $BW_{sp} < BW_{pc1}$, the server-proxy connection is the bottleneck.
- Scenario II: $BW_{sp} \geq BW_{pc1}$, the proxy-client connection are the bottleneck.

We are particularly interested in scenario I because in scenario II the quality of cached streams will always be higher than what the client can afford and leave no room for the proxy to gradually improve the quality. When there are multiple clients, it is interesting to mix Scenario I and II by having $BW_{sp} < BW_{pc1}$ and $BW_{sp} \geq BW_{pc2}$. Different client bandwidth will affect the resulting quality of cached streams, as we will discuss later in this section.

There are other parameters controlling our simulations, such as cache capacity, segment size, etc. To limit the number of parameters, we let all streams have 8 layers, the same segment size of 1KB, and layer consumption rate of 2.5KB/s. Changing these parameters will not qualitatively change our results as long as they are changed proportionally.

The server-proxy link is shared by 10 RAP and 10 long-lived TCP flows (carrying FTP traffic) except if it is explicitly stated. One of the RAP flows is used to deliver multimedia streams from server to cache; the other 19 flows present background traffic, whose dynamics result in available bandwidth changes that will trigger adding and dropping of layers. Bandwidth of the server-proxy link is set to 1.12Mbps (20*56Kbps). Since RAP is TCP-friendly, each flow obtains an even share of bandwidth on average, thus the average bandwidth of the server-proxy connection (BW_{sp}) is 56 Kbps which can afford 2.8 layers.

Webgroup CZ a.s. Ex. 1015, Page 159 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In order to generate a request sequence, we need to know two factors: the number of requests for each stream (*i.e.*, stream popularity), and the temporal distribution of these requests. First, we assume that the stream popularity conforms to the Zipf's law, which was observed in web page requests [19].[6] Given the number of total requests R and total number of streams N, we let the mth popular stream have $\frac{1}{m}\Omega R$ requests, where $\Omega = \sum_{i=1}^{N} \frac{1}{i}$. Second, it is non-trivial to generate a request sequence that exhibits temporal locality while the stream popularity follows the Zipf's law [10]. A request sequence that lacks temporal locality may lead to different cache performance, *e.g.*, hit ratio. Since performance analysis is not our primary concern, we are able to simplify the request sequence generation by assuming that different requests are served sequentially by the proxy, *i.e.*, the proxy transmits at most one multimedia stream at any time. This excludes the situations of simultaneous playbacks from the cache. However, the only added complexity in these situations is that two or more streams compete for the prefetching bandwidth between the server and the proxy; ignoring this does not affect the evaluation of correctness of our algorithms. Avoiding this situation reduces the number of variables that affect the replacement and helps us to understand the simulation results because we are then able to assess the effect of more important parameters. The distribution of requests between the two clients that have different bandwidth to the proxy varies in simulations as we will discuss next.

In all of these simulations, we maintain an infinite popularity window, because we expect that in reality the popularity window should be much larger than the time-scale in our simulations. We leave it as future work to investigate the impact of the popularity window.

6.5 Experiments and Results

We examine different aspects of the caching mechanism in the following order:

[6]There exist many web traces that do not exactly follow Zipf's law, instead, they exhibit *Zipf-like behavior* [19]. Here, we use Zipf's law for simplicity.

Webgroup CZ a.s. Ex. 1015, Page 160 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. **Pre-fetching**: This experiment shows the effectiveness of the pre-fetching mechanism in the absence of replacement.

2. **Replacement Algorithm**: These experiments show the effect of popularity and client-proxy bandwidth distribution separately and together with a simple background traffic. of our replacement algorithm

   (a) Effect of Popularity

   (b) Effect of Bandwidth

   (c) Mixing Together

3. **Replacement Algorithm with Realistic Background Traffic**: These experiments also evaluate the replacement algorithm with more realistic background traffic.

   (a) Effect of Popularity

   (b) Effect of Bandwidth

   (c) Mixing Together

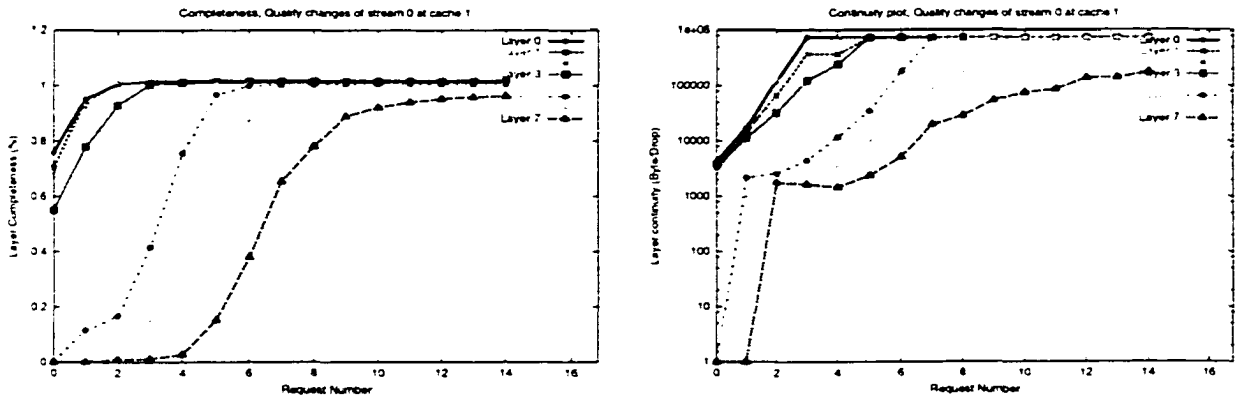
6.5.1 Prefetching

This simulation is intended to demonstrate that the prefetching algorithm results in gradual improvement in quality of cached streams. To disable the cache replacement mechanisms, we set cache size to infinity and use only a single client and a single stream. The stream size is set to 5 minutes. As discussed above, we choose scenario I where BW_{sp} is 56Kbps and BW_{pc1} is 1.5Mbps. The simulation ran for 125 minutes, containing 15 completed requests.

Fig. 6.15 shows the evolution of completeness and continuity of every layer of the cached stream. Each point in the figure represents the status of a particular layer after one playback. Since continuity inversely depends on the number of layer breaks, we plot it on a log-scale. It takes about 4 requests for the 4 lowest layers to achieve maximum quality. The higher the layer, the more playbacks it takes to improve that layer's quality. The convergence speed is not constant, rather, it exhibits a thresholded pattern. For each

Webgroup CZ a.s. Ex. 1015, Page 161 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

layer, there are several requests that greatly enhanced its quality, while other requests just marginally improve its quality. This can be explained by the layer dependence during prefetching: a higher layer is only pre-fetched when corresponding data of all lower layers are available. Currently we only repair for packet losses by prefetching. We expect that the convergence of continuity will be much faster if an error control mechanism is provided by the transport protocol.



(a) Completeness

(b) Continuity

Figure 6.15: Quality improvement due to prefetching.

6.5.2 Replacement Algorithm

This set of simulations is intended to examine that the state of the cache gradually converges to an efficient state as result of the interaction between prefetching and replacement algorithms. The resulting quality due to cache replacement depends on two factors: stream popularity and the bandwidth between the requesting client and the proxy. If we assume $BW_{sp} \gg BW_{pc1}$, stream popularity will be dominant. In the other extreme, $BW_{sp} \leq BW_{pc1}$, the client bandwidth overshadows stream popularity. To examine the impact of different parameters, we studied both extremes and an intermediate case using two clients with different bandwidths to the proxy. BW_{sp}, BW_{pc1} and BW_{pc2} are set to

Webgroup CZ a.s. Ex. 1015, Page 162 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

56Kbps, 1.5Mbps and 56Kbps respectively. By distributing the number of requests between client 1 and client 2, we are able to go from one extreme ($BW_{sp} \ll BW_{pc1}$) to the other ($BW_{sp} \geq BW_{pc2}$).

Without statistical knowledge about the size distribution of real Internet multimedia streams, we choose 10 streams with lengths uniformly distributed between 1 and 10 minutes(the size in byte can be obtained by multiplying the stream length, number of layers and layer consumption rate). Their popularity decreases with their index, *i.e.*, stream 0 is the most popular one. Streams longer than 10 minutes can be viewed as combinations of several shorter streams with the same popularity.

In order to show the effect of cache replacement, we set the cache size to be half the total size of all 10 streams. This cache size is chosen heuristically: we want a moderate number of replacements, but not so many as to cause frequent oscillations in quality.

The simulations ran for 44 (virtual) hours and contains 310 requests. We only show the first half (*i.e.*, first 22 hours); the rest exhibits the same trend and is omitted for clarity. Note that time is used as x-axis in our results. With replacement, a stream's quality not only changes with its own requests but also with requests to other streams. Thus, it is easier to represent this relationship using time as x-axis.

6.5.2.1 Effect of Popularity

In order to emphasize the influence of stream popularity on cache replacement, we should reduce the influence of client bandwidth to the minimum. We achieved this by assigning 95% of all requests to the high bandwidth client 1 and only 5% to client 2.

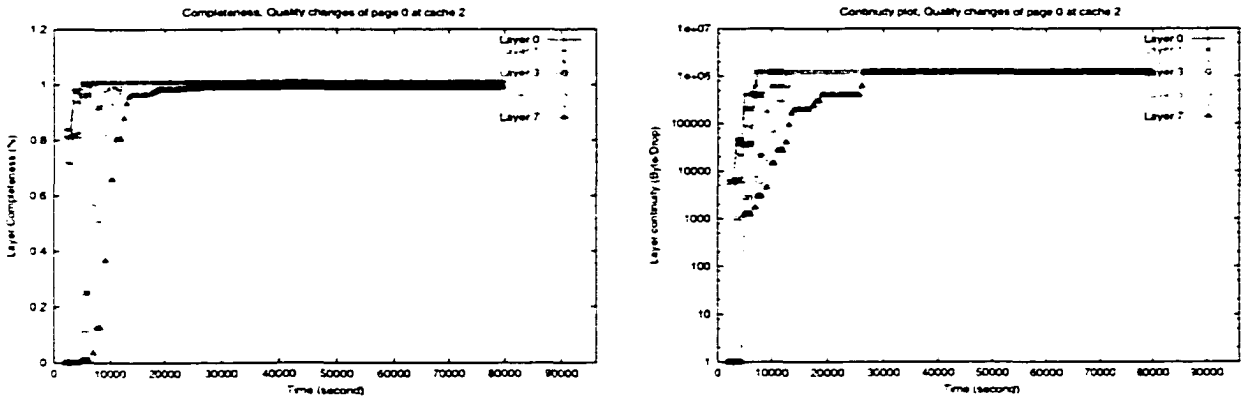
Figs. 6.16 and 6.17 show the quality change of the most popular stream 0 and the least popular stream 9, respectively. Stream 0 is eventually able to cache all of its layers after 20 requests. Notice that the Continuity of all layers of stream 0 is monotonically increasing. Compared to the most popular stream, the least popular stream is not able to keep adequate quality in the cache, and it was even completely flushed out during the interval [36000s, 72000s]. Stream 0 and 9 have the maximum and minimum quality respectively. Quality of all other streams are sorted based on their index number between these two extreme cases.

Webgroup CZ a.s. Ex. 1015, Page 163 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

These figures show that qualities of popular streams are gradually improved. They are likely to have more layers in cache, and these layers tend to have higher quality both in terms of completeness and continuity. In contrast, less popular streams have a lower quality with more variations. In other words, our algorithms successfully accomplished their goals.

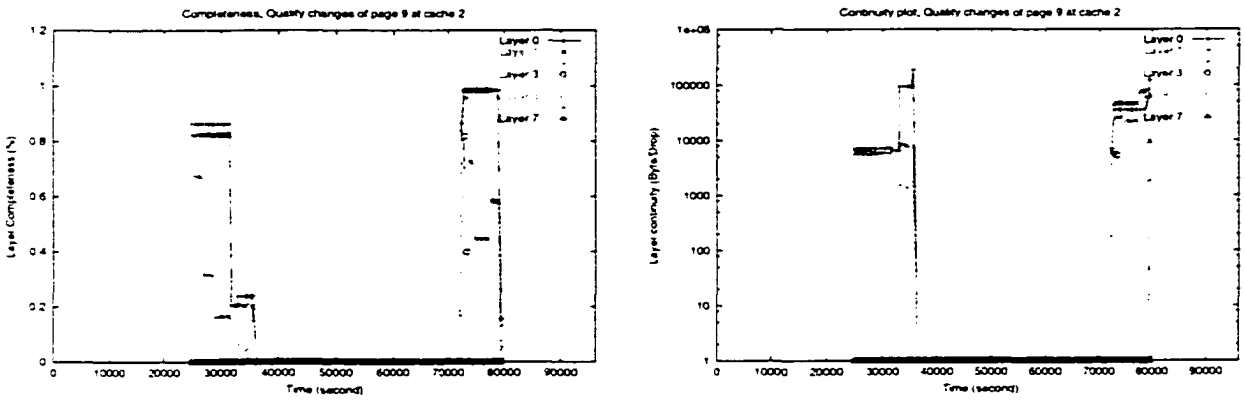
Note that Continuity does not always increase monotonically. Continuity may decrease when proxy pre-fetch a small number of segments that are not adjacent to other existing segments of that layer. As a result, the average size of all segments may decrease. This reveals that Continuity alone does not perfectly capture the distribution of cached chunks for a layer. We can add variations of Continuity to carefully identify these cases.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Completeness (b) Continuity

Figure 6.16: Effect of popularity on cache replacement, *The most popular stream.*



(a) Completeness (b) Continuity

Figure 6.17: Effect of popularity on cache replacement, *The least popular stream.*

Webgroup CZ a.s. Ex. 1015, Page 165 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

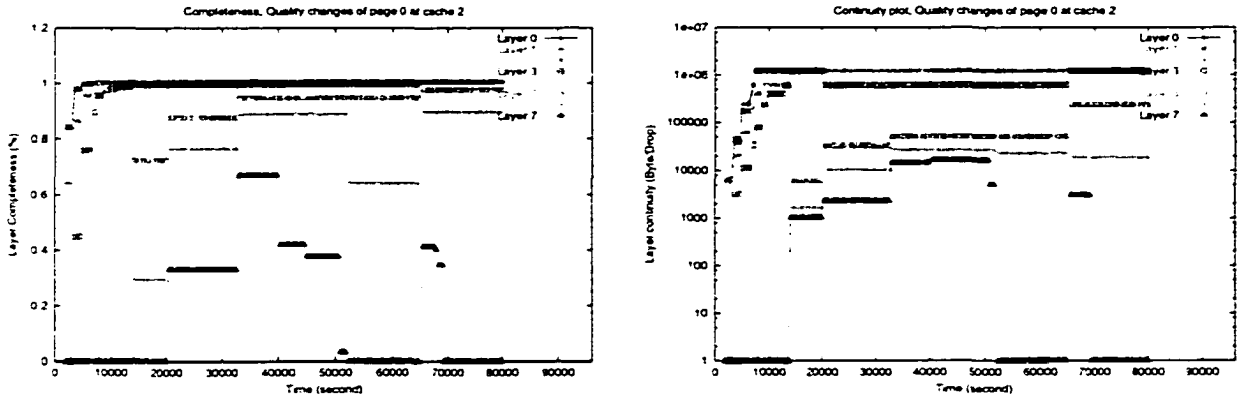
6.5.2.2 Effect of Client-side Bandwidth

Since we calculate popularity on a per-layer basis, if most clients who requested a stream have limited bandwidth, only the lower layers of the stream should be kept in cache. In order to examine this effect of client-side bandwidth on replacement, we assigned 95% of all requests to the low bandwidth client 2 and 5% to the high bandwidth client 1. Fig. 6.18 and 6.19 show the quality change of the most and least popular stream respectively.

Comparing Fig. 6.18 with Fig. 6.16 reveals that when clients have limited bandwidth, the maximum quality of the popular stream drops significantly. In the previous case where most requests came through a 1.5Mbps link, the most popular stream 0 could keep all 8 layers in the cache. However, in this case it is able to cache only 4 layers as most requests come from a 56Kbps link. Its highest 3 layers exhibit oscillating behavior because overall they were accessed less frequently. To the contrary, comparing Fig. 6.17 with Fig. 6.19, now the least popular stream 9 has improved quality and is able to keep the 2 lowest layers in cache most of the time. This is because the higher layers of more popular streams were accessed less frequently, thus the lower layers of the less popular streams became relatively more popular and are able to stay in cache.

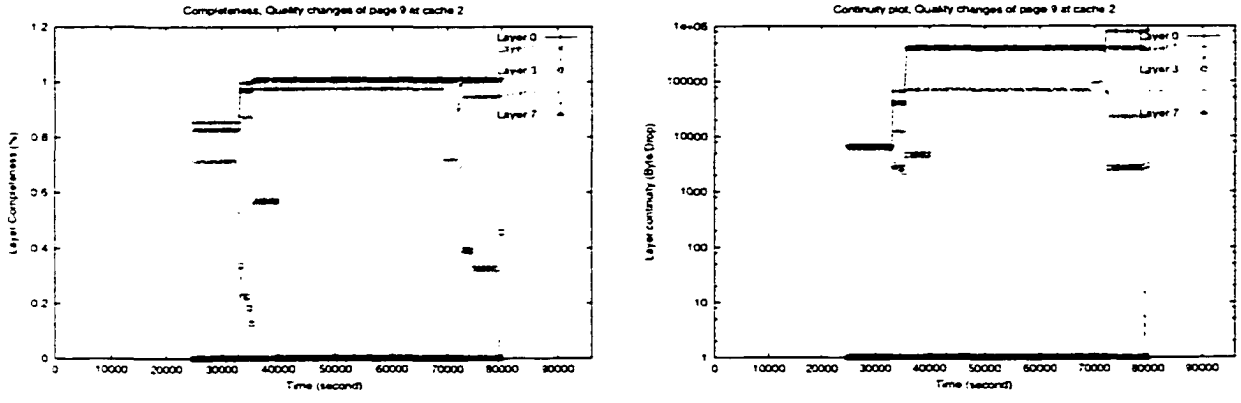
One interesting phenomenon in Fig. 6.18 is that the highest 4 layers of the most popular stream finally converged to maximum. This suggests that if there is no replacement and the simulation ran long enough, quality of cached streams converges to average bandwidth among interested clients. Furthermore prefetching will effectively fill every lost segment in a stream after several playbacks. Thus the continuity of every layer will be finally maximized.

Webgroup CZ a.s. Ex. 1015, Page 166 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Completeness (b) Continuity

Figure 6.18: Effect of client bandwidth on cache replacement, *The most popular stream.*



(a) Completeness (b) Continuity

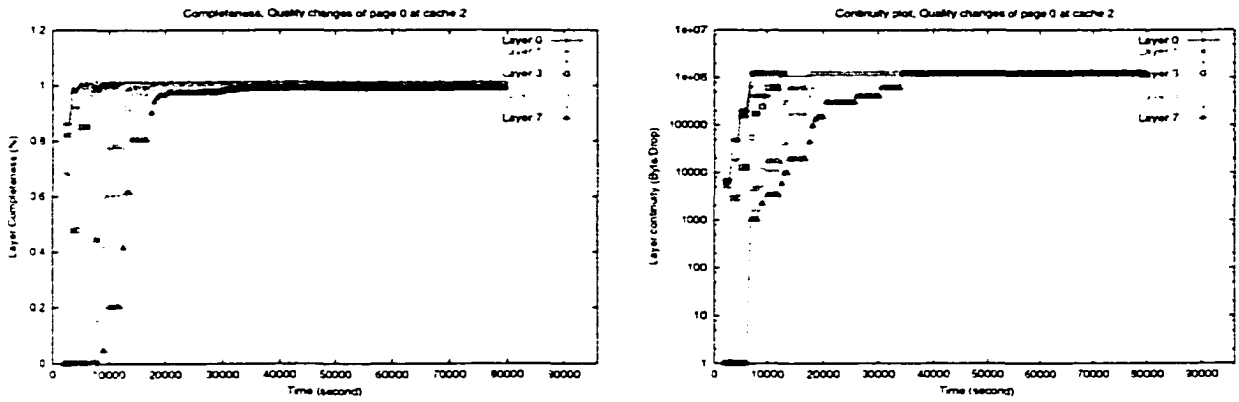
Figure 6.19: Effect of client bandwidth on cache replacement, *The least popular stream.*

Webgroup CZ a.s. Ex. 1015, Page 167 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.5.2.3 Mixing Together

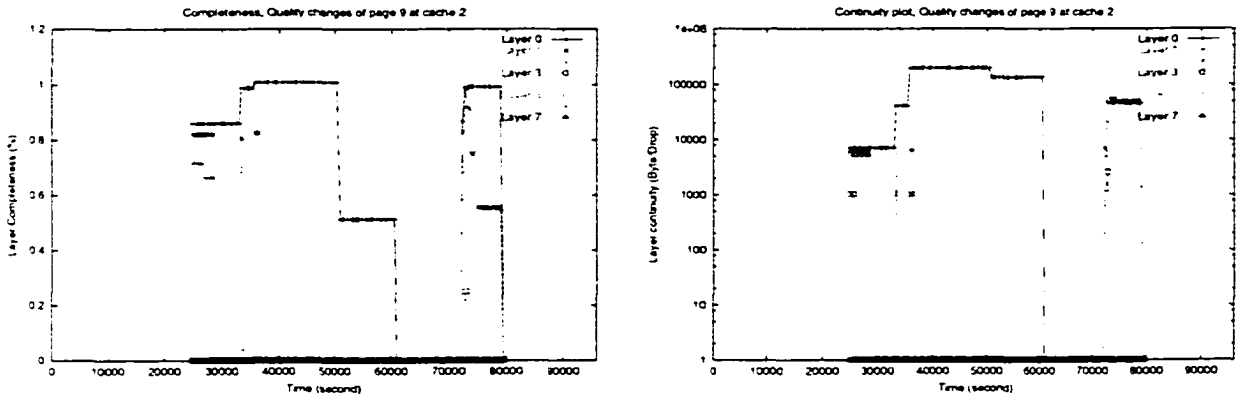
Having examined the above two extreme cases, we now examine an intermediate case where requests are evenly distributed between the low bandwidth client and the high bandwidth client. Figs. 6.20 and 6.21 show the quality change of the most popular stream 0 and the least popular stream 9, respectively. Comparing Fig. 6.20 to Figs. 6.16 and 6.18, we found that the average quality of the popular stream is higher than that in the low bandwidth client case, but lower than that in the high bandwidth client case. A similar situation is observed for the least popular stream. This is exactly what our cache replacement algorithm is intended to achieve, *i.e.*, converging the resulting quality of cached streams to the average quality that has been accessed by clients. Notice that the quality convergence here is closer to the high bandwidth case (Fig. 6.16) than the low bandwidth case (Fig. 6.18). This implies that the impact of client bandwidth limitation may be less than that of stream popularity.

Webgroup CZ a.s. Ex. 1015, Page 168 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Completeness (b) Continuity

Figure 6.20: General case of cache replacement. *The most popular stream.*



(a) Completeness (b) Continuity

Figure 6.21: General case of cache replacement, *The least popular stream.*

6.5.3 Replacement Algorithm with Realistic Background Traffic

We have conducted another set of simulations to show that our results do not depend critically on the selected simulation parameters. In particular, we studied the impact of background traffic on the resulting quality evolution using a statistically realistic bursty Web traffic model [36]. We expected that dynamics of background traffic to result in

Webgroup CZ a.s. Ex. 1015, Page 169 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

more realistic variations in quality of the initial playback stream and change the prefetching bandwidth. It is certainly possible to increase the smoothing factor of the quality adaptation mechanism to smooth out some of these variations. However, our goal was to observe how these changes affect the behavior caching mechanism.

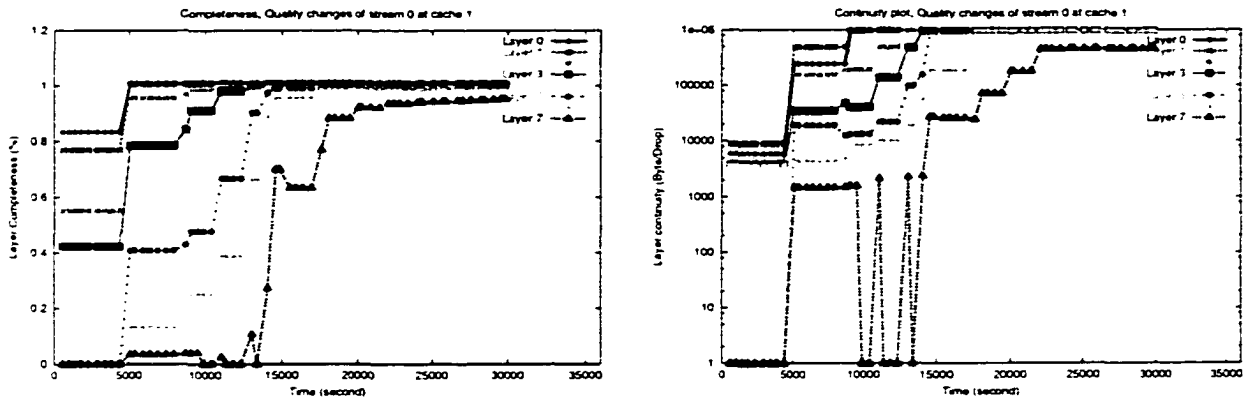
We repeated our previous simulations while a statistically realistic bursty Web traffic model existed along the path between the server and the proxy. Although the RAP connection between the server and the proxy obtained the same amount of bandwidth in average(i.e. BW_{sp}= 56 Kbps) as in previous simulations, the variations of server-proxy bandwidth in these simulations were statistically closer to Internet traffic. Note that these simulations were shorter due to the memory required for generating realistic background traffic.

6.5.3.1 Effect of Popularity

Figure 6.22 and 6.23 depict the effect of stream popularity on variations of completeness and continuity for the most popular and the least popular streams respectively. The simulation setup is similar to section 6.5.2.1 except that background traffic exhibits a statistically realistic bursty behavior similar to the Web traffic. Comparing figure 6.22 with figure 6.16 and figure 6.23 with figure 6.17, we observe that overall pattern of changes in completeness and continuity is relatively the same. All layers of the most popular stream eventually cache at the proxy in both cases. However, in the presence of realistic background traffic, it takes longer for each layer(specially pronounced for higher layers) to be completely cached. There might also be some oscillations in quality while it converges to a maximum.

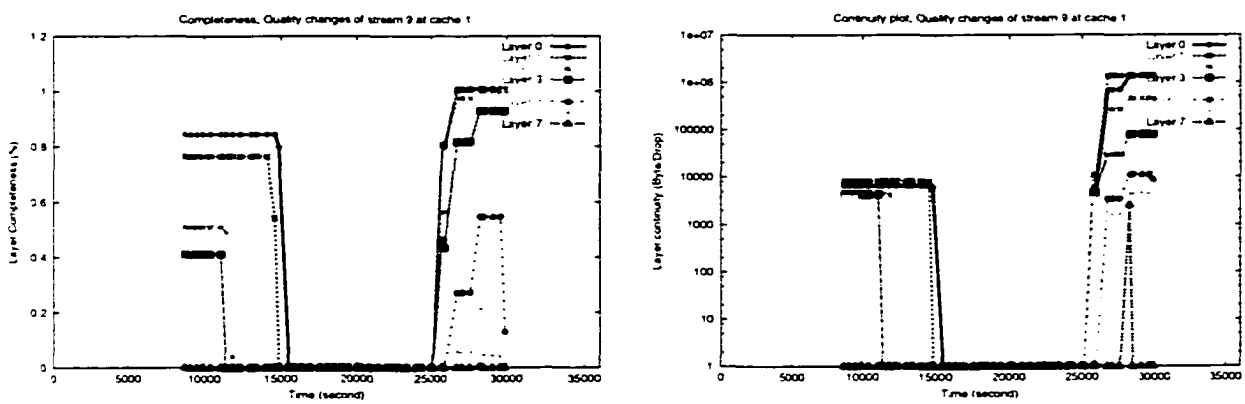
Evolution of quality of the least popular stream in both cases are pretty similar except that the realistic background traffic causes some more variations in quality of higher layers. This has to do with the dynamics of background traffic and frequent changes in available bandwidth.

Webgroup CZ a.s. Ex. 1015, Page 170 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Completeness (b) Continuity

Figure 6.22: Effect of popularity on cache replacement with realistic background traffic, *The most popular stream.*



(a) Completeness (b) Continuity

Figure 6.23: Effect of popularity on cache replacement with realistic background traffic, *The least popular stream.*

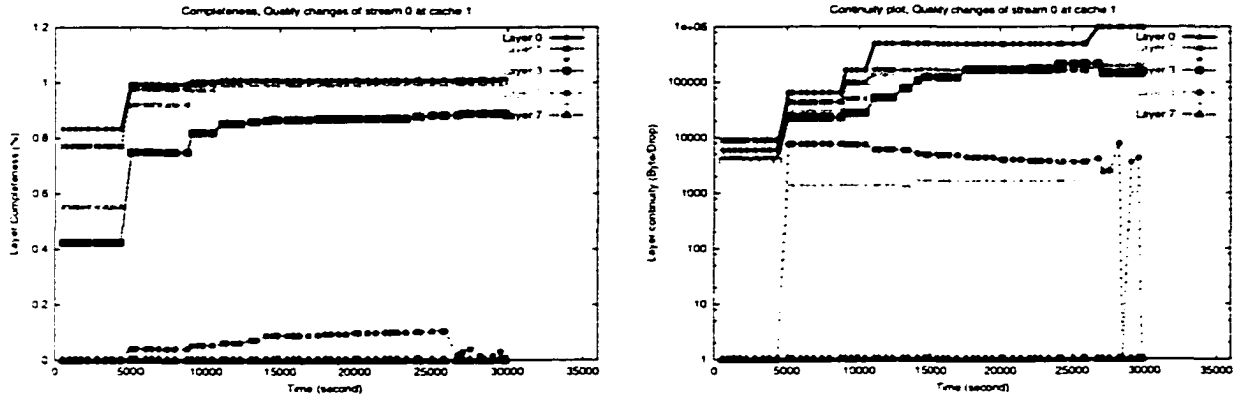
Webgroup CZ a.s. Ex. 1015, Page 171 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.5.3.2 Effect of Client-side Bandwidth

Figure 6.24 and 6.25 depict the effect of client bandwidth on variations of completeness and continuity for the most popular and the least popular streams respectively. The simulation setup is similar to section 6.5.2.2 except that background traffic exhibit a statistically realistic bursty behavior similar to the Web traffic. Comparing figure 6.24 with figure 6.18 and figure 6.25 with figure 6.19, we observe that overall pattern of changes in completeness and continuity is relatively the same for this case as well.

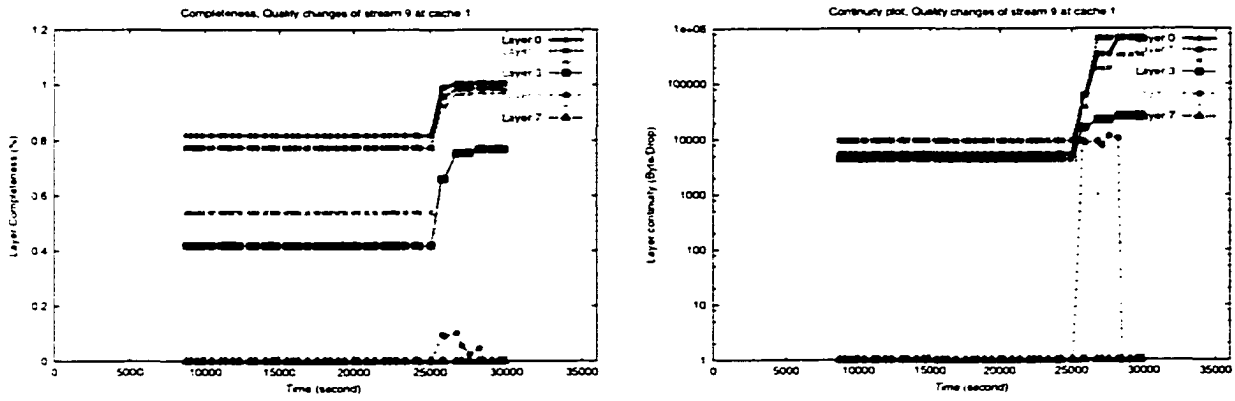
A few lower layers of the most popular stream are completely cached and other higher layers are cached only partially. The quality of the least popular stream has the same overall variations in both, except that in the presence of realistic background traffic some of the higher layers are partially cached. This is again related to the wider variations in bandwidth caused by background traffic which in turn triggers more add and drop events by both the quality adaptation and pre-fetching mechanisms.

Webgroup CZ a.s. Ex. 1015, Page 172 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Completeness (b) Continuity

Figure 6.24: Effect of client bandwidth on cache replacement with realistic background traffic, *The most popular stream.*



(a) Completeness (b) Continuity

Figure 6.25: Effect of client bandwidth on cache replacement with realistic background traffic, *The least popular stream.*

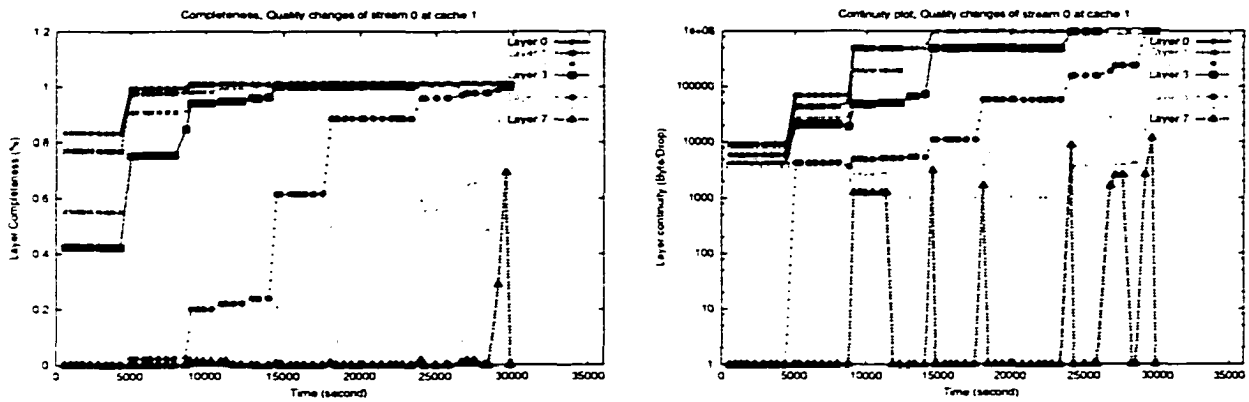
Webgroup CZ a.s. Ex. 1015, Page 173 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.5.3.3 Mixing Together

Figure 6.26 and 6.27 depict the overall effect of popularity and client bandwidth on completeness and continuity for the most popular and the least popular streams, respectively. The simulation setup is similar to section 6.5.2.3 except that background traffic exhibits a statistically realistic bursty behavior similar to the Web traffic. Comparing figure 6.26 with figure 6.20 and figure 6.27 with figure 6.21 clearly reveals the effect of realistic background traffic on changes of quality of the cached streams. In the latter case, higher layers of the most popular stream are added to the cache slower than the previous case. Furthermore, there are more oscillations, particularly in the quality of higher layers in the presence of the realistic background traffic.

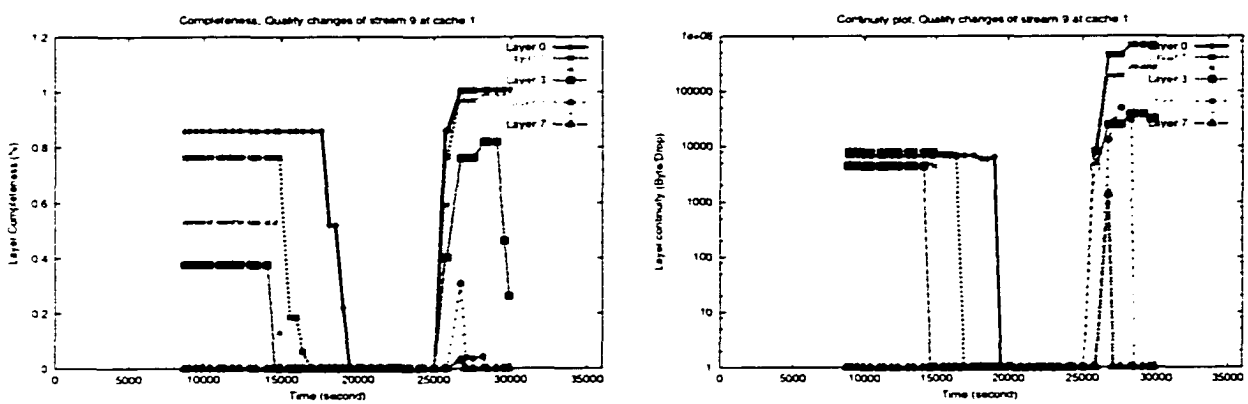
As for the least popular stream, we see the same phenomenon that we described in the previous section(sec. 6.5.3.2). In the presence of realistic background traffic, each layer is partially cached based on its importance (i.e. a higher portion of lower layers are cached in comparison to higher layers).

Webgroup CZ a.s. Ex. 1015, Page 174 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Completeness (b) Continuity

Figure 6.26: General case of cache replacement with realistic background traffic, *The most popular stream.*



(a) Completeness (b) Continuity

Figure 6.27: General case of cache replacement with realistic background traffic, *The least popular stream.*

Webgroup CZ a.s. Ex. 1015, Page 175 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.6 Summary

This chapter presented a novel idea for multimedia proxy caching of layer encoded streams. We addressed the inherent limitation of delivered quality in an end-to-end approach. To overcome this limitation, we extended our architecture to include multimedia proxy cache. Multimedia proxy caching not only improves delivered quality but it also provides an opportunity to perform VCR-function more interactively and results in less load on the network and the server. Thus it allows large scale deployment of streaming applications. Furthermore, multimedia proxy caching seems to be the only way to support unsynchronized delivery of multimedia streams in an efficient manner. The main challenge is to replay a quality variable cached stream while performing quality adaptation effectively for subsequent playbacks.

We described the delivery procedure and identified two major components for multimedia proxy caching:

1. Pre-fetching mechanism
2. Replacement algorithm

Pre-fetching gradually improves the quality of a cached stream during any subsequent playback while replacement tries to maximize overall caching value of stored streams. The goal is that interactions between pre-fetching and replacement converges the state of the cache to an efficient state.

We have devised a pre-fetching mechanism and described various challenges and trade-offs. We also described a replacement algorithm and showed that its interaction with pre-fetching results in an efficient cache state. Additionally, we briefly addressed several alternative replacement patterns to support other caching functionalities.

Our qualitative evaluation shows that the caching mechanism correctly achieves its design goals in the presence of realistic background traffic. We did not attempt to evaluate performance aspects of the mechanism. Multimedia proxy caching is a rich problem and different aspects of it deserve more through analysis. This initial work explored the feasibility and correctness of the idea.

Webgroup CZ a.s. Ex. 1015, Page 176 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 7

Conclusions and Future Work

We conclude this dissertation with a summary of our work. We then enumerate number of research problems that can be addressed in future work.

7.1 Conclusion

This dissertation presented an end-to-end client-proxy-server architecture for delivery of multimedia streams in the Internet.

We described and justified our design philosophy for streaming applications in the Internet. The key issue is to separate network-dependent congestion control from application-specific reliability. We provided architectural insights into the design of Internet video playback applications. Toward that goal, we justified the need for three crucial components:

- End-to-end congestion control,
- Quality adaptation,
- Error control.

We believe that the majority of current Internet video playback applications are missing one or more of these components. Given the rapid increase in deployment of these applications and the severe consequences of ignoring these issues, it is important to understand these issues and apply them.

Webgroup CZ a.s. Ex. 1015, Page 177 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We limited the design space for each of these components based on requirements that are imposed either by applications or by the network and indicated the natural choices for each one. Our main contribution is in combining these components into a coherent architecture and studying the interactions among them. As well as describing possible specific mechanisms for each component, we attempted to generalize the architecture by providing guidelines for design of each component, and highlighted some of the implications on the rest of the architecture.

We focused our attention on two main building blocks of the architecture, 1. Congestion control, 2. Quality adaptation.

We initially investigated congestion control as the underlying rate control mechanism to ensure network friendly, and in particular TCP-friendly behavior. We have presented the Rate Adaptation Protocol and extensively examined its interaction with TCP through simulation. Although achieving TCP-friendliness over a wide range of network parameters is extremely challenging, RAP reasonably achieves this goal. Our simulations reveal that TCP performs fine-grain rate adaptation during its congestion avoidance phase due to its ACK-clocking property. We devised and evaluated a fine-grain rate adaptation mechanism to emulate TCP's ack-clocking property. Our results show that the fine-grain rate adaptation extends inter-protocol fairness to a wider range. Divergence of TCP's congestion control from the AIMD algorithm is often the main cause for the unfairness to TCP in special cases. This problem is pronounced more clearly with Reno and Tahoe while it has a more limited impact on SACK. We observed that the bigger TCP's congestion window is, the closer it follows the AIMD algorithm. Properly configured RED gateways can result in an ideal inter-protocol sharing.

We have presented a quality adaptation mechanism to bridge the gap between short-term changes in transmission rate caused by congestion control and the need for stable quality in streaming applications. We exploit the flexibility of layered encoding to adapt the quality along with long-term variations in available bandwidth. The key issue is appropriate buffer distribution among the active layers. We have described an efficient mechanism that dynamically adjusts the buffer distribution as the available bandwidth changes by carefully allocating the bandwidth among the active layers. Furthermore, we

Webgroup CZ a.s. Ex. 1015, Page 178 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

introduced a smoothing parameter that allows the server to trade short-term improvement for long-term smoothing of quality. The strength of our approach comes from the fact that we did not make any assumptions about loss patterns or available bandwidth. The server adaptively changes the receiver's buffer state to incrementally improve its protection against short-term drops in bandwidth in an efficient fashion. Our simulation and experimental results reveal that with a small amount of buffering the mechanism can efficiently cope with short-term changes in bandwidth due to AIMD congestion control. The mechanism can rapidly adjust the quality of the delivered stream to utilize the available bandwidth while preventing buffer overflow or underflow. Furthermore, by increasing the smoothing factor, the frequency of quality variation is limited effectively.

Given that buffer requirements for quality adaptation are not large, we believe that these mechanisms can also be deployed for non-interactive *live* sessions where the client can tolerate a short (e.g. a few seconds) delay in delivery.

We extended our end-to-end architecture to include proxy caches. Our goal is to improve the delivered quality of a popular stream, despite the presence of a bottleneck between a server and interested clients. Performing quality adaptation results in variable quality streams cached at a proxy. As a result, performing quality adaptation during subsequent playback from the cache could be problematic because there is no correlation between the variations in quality of the cached stream and required quality for the new session.

Our layered approach to quality adaptation provides a perfect opportunity to cope with quality variations of cached streams in a demand-driven fashion by prefetching required segments that are missing from the cache. We have also exploited inherent properties of multimedia streams and devised a fine-grain replacement algorithm. Simulation-based evaluation of our mechanism reveals that interaction between the replacement and prefetching mechanism causes the state of the cache to converge to an efficient state. In such a state, the quality of every cached stream is directly determined by its popularity, and its upper limit is defined by the average available bandwidth between the proxy and its clients.

Webgroup CZ a.s. Ex. 1015, Page 179 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7.2 Future work

In this section, we identify several new research problems as future work for this dissertation.

On Congestion Control

- *TCP-friendliness under Realistic Background Traffic*: Most of our simulations assume long-lived TCP flows(i.e. FTP traffic). However, a reasonable portion of today's Internet traffic consists of short-lived TCP connection generated by web-based applications. It is crucial to examine the TCP-friendliness of the RAP protocol in the presence of more realistic background traffic. More auxiliary mechanisms can be added to RAP to more closely emulate TCP's behavior. These mechanisms can be used optionally depending on the required level of TCP-friendliness.

- *Real world Experiments and Validations*: A large number of experiments should be conducted to examine RAP's performance in real networks, first in a controlled physical environment such as CAIRN[82], and subsequently over the Internet. These experiments would validate simulation results and help to identify some of the actual issues that exist in the Internet and can not be easily captured in a simulation environment.

- *Sharing Congestion State*: Web-based applications usually establish several logical flows that are simultaneously active between two end-points. This can be supported either by multiplexing all flows into a single transport connection or establishing a separate transport connection for each flow. While the latter approach is destructive with respect to congestion control, the former scheme adds complexity due to multiplexing. One approach to tackle this problem is to establish parallel connections that share the congestion state. However, it is still not clear how the congestion state can be properly and effectively shared[9].

- *Intra-class Congestion Control*: It is likely that the Internet will support different classes of services or class-based reservation in the future. While the presence

Webgroup CZ a.s. Ex. 1015, Page 180 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of these services does not replace the need for end-to-end congestion control, it provides a better opportunity to perform end-to-end congestion control in a more effective manner since end systems are aware of the range of expected quality of service a priori based on purchased service profile. We should study how the congestion control mechanism can exploit this information to perform more effectively and smoothly. The implication of the service profile on level of interactivity, from the end-to-end point of view, is another interesting problem. In general, the study of interactive applications across a network with a range of quality of services is one of my immediate research goals. However this depends on the way QoS profiles are defined in the future, which in turn depends on the economic model of future services.

On Quality Adaptation

- *Applying to other Congestion Control Mechanisms*: The idea of quality adaptation can be extended to other congestion control schemes that employ AIMD algorithms and investigate the implications of the details of rate adaption on the mechanism.

- *Extending to Non-linear Encoding*: The quality adaptation mechanism should be extended to non-linear layered encoding since most of the available layered codecs generate non-linear layered streams.

- *Adaptive Smoothing*: Another interesting issue is to use a measurement-based approach to adjust K_{max} on-the-fly based on the recent history. The source maintains a short history for effectiveness of the current value of K_{max}. If it experiences frequents add and drop events, the value of K_{max} is increased. The server also needs to examine the value of K_{max} and decrease its value if it is too conservative.

- *Encoding/Application Specific Quality Adaptation*: Encoding-specific parameters can be exploited to fine-tune the quality adaptation mechanisms, i.e. the server uses these parameters to estimate visual effect of adding or dropping a layer and tries to adjusts its behavior for smoother variations in the quality. In general, any available

Webgroup CZ a.s. Ex. 1015, Page 181 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

metric for measuring delivered quality should be fed into the quality adaptation mechanism to be used as an input for layer add decision.

- *Integration of Error Control Mechanism*: A natural step is to plug in various repair approaches such as retransmission and redundant coding in the architecture and explore its behavior on the overall end-to-end architecture. One of the remaining open issues that has not been completely explored is the bandwidth sharing policy between error control and quality adaptation in order to maximize the delivered quality.

On Multimedia Proxy Caching

- *Performance Evaluation*: As we mentioned earlier, different component of multimedia proxy caching mechanisms deserve further performance evaluations. Note that evaluation of these mechanisms are slightly different from Web caching mechanisms because the primary goal of a multimedia proxy cache is to maximize the quality whereas the primary goal of the Web cache is to improve hit rate. While these two values are not independent, there are scenarios where details of replacement algorithms only improve one of these parameters. The key point is to improve delivered quality while achieving a high hit rate.

- *Demand-driven pre-fetching for popular streams*: The proposed demand-driven multimedia caching mechanisms can be combined with a proactive prefetching mechanisms whenever access pattern is known a priori. If the server knows about the popularity of one or several streams that will be requested during busy hours (e.g. a major sport event or a requested movie by a group of clients for Saturday night), this stream can be prefetched during idle hours ahead of time to avoid any potential degradation of delivered quality during playback.

- *Providing Various Classes of Services*: If the server provides various classes of services to different clients and charges them accordingly, it must 1) weight the value of popularity of each stream for each session according to the purchased service by

Webgroup CZ a.s. Ex. 1015, Page 182 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the corresponding client, and 2) deploy proper resource management strategies to guarantee the promised services.

On Supporting Future Applications :

- *Interactive Multi-player Games*: A long term goal is to extend the architecture to interactive multi-player games. Clearly, there are so many challenges and it is not a straight forward extension. However, we believe our design philosophy must be applicable. First, network-dependent congestion control for multicast distribution should be addressed. Then delay-quality trade-off for a given application must be identified. Finally, a layered quality adaptation mechanism must be designed to efficiently map application requirements into network requirements.

Webgroup CZ a.s. Ex. 1015, Page 183 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A

A Simple Model for RAP

This appendix presents a model for simplified version of RAP. Here we assume:

1. RTT is constant.
2. RAP performs only coarse grain rate adaptation.
3. Bandwidth does not change with time.

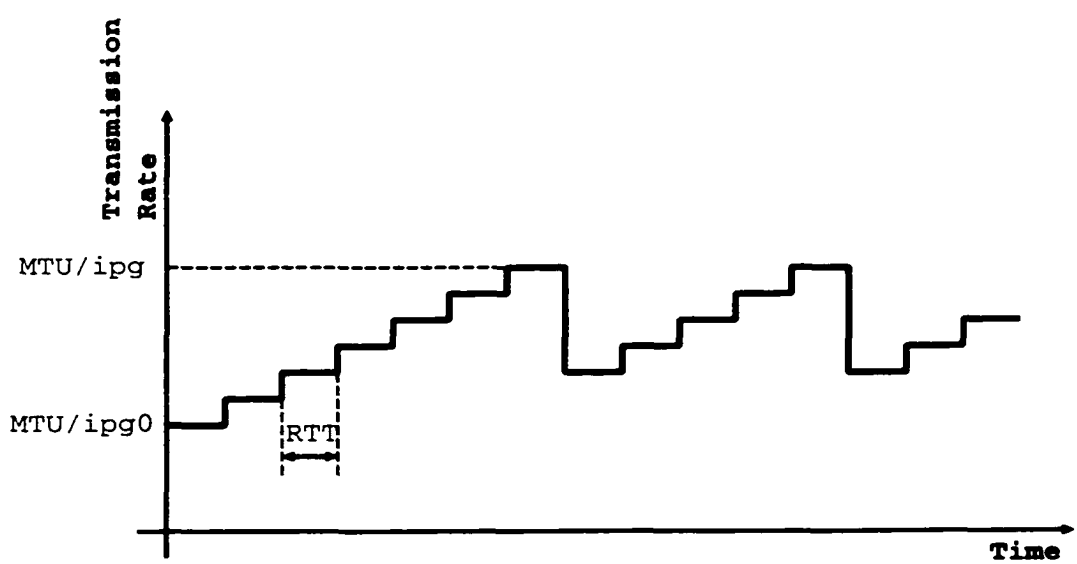


Figure A.1: Transmission rate of a single RAP flow

Figure A.1 shows variations of transmission rate for a single RAP flow without fine-grain adaptation assuming constant RTT. The list of parameters are given in table A.1. Notice that ipg is the value of inter-packet-gap(IPG) when a loss occurs.

Webgroup CZ a.s. Ex. 1015, Page 184 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

MTU	Packet Size
RTT	Round trip time
C	Constant with time dimension
r	Transmission rate
r_{max}	Max transmission rate
ipg	Inter-packet-gap when back off occurs
ipg_0	Initial value of Inter-packet-gap at startup
N_0	Number of steps before first loss
N	Number of steps between two consecutive backoff
β	Backoff factor

Table A.1: Simulation setup for RAP evaluation

The main equation for updating the value of inter-packet-gap from the previous value is:

$$IPG_{i+1} = \frac{IPG_i * C}{IPG_i + C} \tag{A.1}$$

Notice that C has dimension of time. We also have:

$$r = \frac{MTU}{IPG_i} \Rightarrow r_{max} = \frac{MTU}{ipg} \tag{A.2}$$

If we derive IPG_{i+2} as function of IPG_i, we have:

$$IPG_{i+2} = \frac{IPG_{i+1} * C}{IPG_{i+1} + C} = \frac{\frac{IPG_i * C}{IPG_i + C} * C}{\frac{IPG_i * C}{IPG_i + C} + C} = \frac{IPG_i * C}{2IPG_i + C} \tag{A.3}$$

If we iteratively apply the formula, we can derive IPG_{i+k} as function of IPG_i:

$$IPG_{i+k} = \frac{IPG_i * C}{kIPG_i + C} \tag{A.4}$$

Question 1: How many RTTs(i.e. steps) does it take to reach from IPG_2 to IPG_1, given that $IPG_2 \leq IPG_1$?

Webgroup CZ a.s. Ex. 1015, Page 185 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Using equation A.4, we have:

$$IPG_2 = \frac{IPG_1 * C}{nIPG_1 + C} \tag{A.5}$$

If we solve equation A.5 for n, we have:

$$n = \frac{C}{IPG_2} - \frac{C}{IPG_1} \tag{A.6}$$

Question 2: How many steps does it take before the first loss occurs, i.e. how many RTT is the startup phase?

Given the initial value of inter-packet-gap(ipg_0) and its value at the time of the first backoff (ipg), we can simply plug them into equation A.6. Hence we have

$$N_0 = \frac{C}{ipg} - \frac{C}{ipg0} \tag{A.7}$$

Question 3: How many steps do exist between two consecutive packet losses?

After each back IPG is divided by β. Thus the initial value of IPG for each ramp up is ipg/β. Having the initial and final value of IPG, we can derive the number of steps between two backoffs similar to the previous case:

$$N = \frac{C}{ipg} - \frac{C}{\frac{ipg}{\beta}} = \frac{(1-\beta)C}{ipg} \tag{A.8}$$

Question 4: How many packets are sent between two consecutive packet losses?

The number of transmitted packets during one step in steady state (i.e. not during a start up step) can be calculated as follows

$$M_k = \frac{RTT}{IPG_k} = \frac{RTT}{\frac{C\frac{ipg}{\beta}}{C+k*\frac{ipg}{\beta}}} \tag{A.9}$$

Webgroup CZ a.s. Ex. 1015, Page 186 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus the total number of transmitted packets(M) is:

$$M = \sum_{k=0}^{N} \frac{RTT}{\frac{\frac{ipg}{\beta}C}{C+k\frac{ipg}{\beta}}} = \frac{RTT}{C\frac{ipg}{\beta}} \sum_{k=0}^{N} (C + k * \frac{ipg}{\beta}) \tag{A.10}$$

$$M = \frac{RTT}{\frac{ipg}{\beta}}(N+1) + \frac{N(N+1)}{2}\frac{RTT}{C} \tag{A.11}$$

We can also replace value of N using equation A.9. Equation A.11 presents number of transmitted packets as a function of RTT, C, ipg and β.

Question 5: What is the loss rate?

If we assume only once loss occurs per congestion event, then loss rate (L) can be simply calculated from M as follows:

$$L = \frac{1}{M} \tag{A.12}$$

Question 6: What is the average transmission rate during steady state?

We can follow the same approach as in [80]. Since the transmission rate sawtooth varies between r_{max} and $r_{max}/2$, the average transmission rate is 0.75*r_{max}.

$$R = 0.75 * r_{max} = 0.75\frac{MTU}{ipg} \tag{A.13}$$

If we assume, $C = RTT$, we can rewrite equation A.11 as follows:

$$M = \frac{RTT}{\frac{ipg}{\beta}}(N+1) + \frac{N(N+1)}{2}\frac{RTT}{C} \tag{A.14}$$

$$M = \frac{RTT}{\frac{ipg}{\beta}}(N+1) + \frac{N(N+1)}{2} \tag{A.15}$$

Webgroup CZ a.s. Ex. 1015, Page 187 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thus we can rewrite equation as follows:

$$M = \frac{RTT}{\frac{ipg}{\beta}} + \frac{RTT}{\frac{ipg}{\beta}} + 1 + \frac{RTT}{\frac{ipg}{\beta}} + 2 + \ldots + \frac{RTT}{\frac{ipg}{\beta}} + N \tag{A.16}$$

Equation A.16 implies that for $C = RTT$, number of transmitted packet is increased by one per step.

$$M = \frac{1}{L} =\sim \frac{3}{8}(\frac{2 * RTT}{\frac{ipg}{\beta}})^2 \tag{A.17}$$

$$L \sim \frac{1}{\frac{3}{8}(\frac{2RTT}{\frac{ipg}{\beta}})^2} \tag{A.18}$$

if we solve equation A.19 for ipg as function of L, we have:

$$ipg \sim 1.22\sqrt{L} * \beta * RTT \tag{A.19}$$

Replacing value of ipg in equation A.13, we have

$$R \sim 0.61\frac{MTU}{\sqrt{L} * \beta * RTT} \tag{A.20}$$

The value of β must be 0.5 so that RAP and TCP behave similarly,

$$R \sim 1.22\frac{MTU}{\sqrt{L} * RTT} \tag{A.21}$$

This is consistent with the result derived by Mahdavi and Floyd in [80].

Webgroup CZ a.s. Ex. 1015, Page 188 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A.1 Summary

This result shows that in the absence of network dynamics, as long as both TCP and RAP follow the AIMD algorithm, they achieve the same average bandwidth over a reasonably long time scale(e.g. several RTT).

Webgroup CZ a.s. Ex. 1015, Page 189 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reference List

[1] Soam Acharya. Techniques for improving multimedia communication over wide area networks. *Ph.D. Thesis, Cornell University*, January 1999.

[2] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan. Experiences with TCP vegas: Emulation and experiment. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA., August 1995.

[3] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA., September 1999.

[4] X. Li amd M. Ammar and S. Paul. Layered video multicast with retransmission(LVMR): Evaluation of hierarchical rate control. In *Proceedings of the IEEE INFOCOM*, San Fransisco, CA, March 1998.

[5] E. Amir and S. McCanne nd Hui Zhang. An application level video gateway. *Proceedings of ACM Multimedia*, November 1995.

[6] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Scott Shenker, Kannan Varadhan, Haobo Yu, Ya Xu, and Daniel Zappala. Improving simulation for network research. Technical Report 99-702, University of Southern California, July 1999.

[7] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Scott Shenker, Kannan Varadhan, Haobo Yu, Ya Xu, and Daniel Zappala. Virtual InterNetwork Testbed: Status and

Webgroup CZ a.s. Ex. 1015, Page 190 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

research agenda. Technical Report 98-678, University of Southern California, July 1998.

[8] Sandeep Bajaj, Lee Breslau, and Scott Shenker. Uniform versus priority dropping for layered video. In *Proceedings of the ACM SIGCOMM*, Vancouver, Canada, September 1998.

[9] Hari Balakrishnan, Hariharan Rahul, and Srinivasan Seshan. An integrated congestion management architecture for internet hosts. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA., September 1999.

[10] Paul Barford and Mark Crovella. Generating representative web workloads for network and server peformance evaluation. In *Proceedings of the ACM SIGMETRICS*, pages 151–160, June 1998.

[11] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *RFC 2475*, October 1998.

[12] J. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-based error control for interactive audio in the internet. In *Proceedings of the IEEE INFOCOM*, New York, NY., March 1999.

[13] J. Bolot and Turletti. Experience with rate control mechanisms for packet video in the internet. *ACM Computer Communication Review*, pages 4–15, January 1998.

[14] J. Bolot and T. Turletti. A rate control mechanism for packet video in the internet. In *Proceedings of the IEEE INFOCOM*, pages 1216–1223, June 1994.

[15] J. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the internet. In *Proceedings of the ACM SIGCOMM*, pages 58–67, London, UK, September 1994.

[16] J. C. Bolot. Characterizing end-to-end packet delay and loss in the internet. *Journal of High Speed Networks*, 2(3):289–298, September 1993.

Webgroup CZ a.s. Ex. 1015, Page 191 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[17] J. C. Bolot and A. Vega Garcia. The case for FEC-based error control for packet audio in the internet. In *ACM Multimedia Systems*, Boston, MA., 1996.

[18] L. S. Brakmo and L. L. Peterson. TCP vegas: End to end congestion aviodance on a global internet. *IEEE Journal of Selected Areas in Communication*, 13(8):1465–1480, October 1995.

[19] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. On the implications of Zipf's Law for Web caching. In *Proceedings of the Third International Web Caching Workshop*, Warsaw, Poland, October 1998.

[20] Ingo Busse, Bernd Deffner, and Henning Schulzrine. Dynamic QoS control of multimedia applications based on RTP. In *Workshop on Network and Operating System Support for Digital Audio and Video*, St. Petersburg, Russia, June 1995.

[21] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206, December 1997.

[22] S. Cen, C. Pu, and J. Walpole. Flow and congestion control for internet streaming applications. *Proceedings Multimedia Computing and Networking*, January 1998.

[23] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *USENIX Conference Proceedings*, pages 153–63, 1996.

[24] Z. Chen, S-M Tan, R. H. Campbell, and Y. Li. Real time video and audio in the world wide web. *Fourth International World Wide Web Conference*, December 1995.

[25] D. Chiu and R. Jain. Analysis of the increase and decrease algorithm for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

Webgroup CZ a.s. Ex. 1015, Page 192 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[26] K Claffy, G. Miller, and K. Thompson. The nature of the beast: Recent traffic measurements from an internet backbon. *Proceedings INET, Internet Society*, December 1998.

[27] D. D. Clark, M. L. Lambert, and L. Zhang. NETBLT: A high throughput transport protocol. In *Proceedings of the ACM SIGCOMM*, Stanford, CA., August 1988.

[28] D. D. Clark, S. Shenker, and L. Zhang. Supporting realtime applications in an integrated service packet network: Architecture and mechanism. In *Proceedings of the ACM SIGCOMM*, pages 14–26, Baltimor, MD., August 1992.

[29] D. Cohen. On packet speech communication. In *Proceedings of the Fifth International Conference on Computer Communications*, pages 271–274, Atlanta, GA., October 1980.

[30] W. Dabbous. Analysis of delayed-based congestion avoidance algorithm. In *Proceedings 4th IFIP Conference on High Performance Newtworking*, Liege, Belgium, December 1992.

[31] A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari. Buffering and caching in large scale video servers. In *Proceedings of IEEE COMPCON*, pages 217–224, 1995.

[32] A. Dan and D. Sitaram. A generalized interval caching policy for mixed interactive and long video environments. In *IS&T SPIE Multimedia Computing and Networking Conference*, San Jose, CA., January 1996.

[33] A. Dan and D. Sitaram. Multimedia caching strategies for heterogeneous application and server environments. In *Multimedia Tools and Applications*, volume 4, pages 279–312, 1997.

[34] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings SIGCOMM Symposium on Communications Architectures and Protocols*, pages 1–12, Austin, TX, September 1989.

Webgroup CZ a.s. Ex. 1015, Page 193 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[35] K. Fall and S. Floyd. Simulation-based comparison of tahoe, reno and sack TCP. *Computer Communication Review*, 26(3):5–21, July 1996.

[36] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA., September 1999.

[37] W. Feng, D. Kandlur, D. Saha, and K. Shin. A self-configuring red gateway. In *Proceedings of the IEEE INFOCOM*, New York, NY., 1999.

[38] W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev. A priority-based technique for the best-effort delivery of stored video. *Proceedings of Multimedia Computing and Networking*, January 1999.

[39] N. R. Figueira and J. Pasquale. Leave-in-time: A new service decipline for real-time communications in a packet-switching network. In *Proceedings of the ACM SIGCOMM*, pages 207–218, Cambridge, MA., September 1995.

[40] S. Floyd. Connections with multiple congested gateways in packet-switched networks. *Computer Communication Review*, 21(5):30–47, October 1991.

[41] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *Under submission*, February 1998. http://www-nrg.ee.lbl.gov/floyd/papers.html/end2end-paper.html.

[42] S. Floyd and V. Jacobson. On traffic phase effect in packet-switched gateways. *Internetworking: Research and Experiences*, 3(3):115–156, September 1992.

[43] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[44] R. Fredrick. nv: Network video. *Unix Manual Page, Xerox Palo Alto Research Center*, 1993.

Webgroup CZ a.s. Ex. 1015, Page 194 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[45] M. W. Garrett and M. Vetterli. Joint source/channel coding of statistically multiplexed real time services on packet networks. *ACM/IEEE Transactions on Networking*, 7(1):71–80, February 1993.

[46] M. Gilge and R. Gusella. Motion video coding for packet-switching networks- an integerated approach. In *Proceedings og the SPIE Conference on Visual Communications and Image Processing*, Boston, MA., November 1991.

[47] S. J. Golestani and S. Bhattacharyya. A class of end-to-end congestion control algorithms for the internet. In *IEEE International Conference on Network Protocols*, Austin, TX., 1998.

[48] Jefferson Han and Brian C. Smith. CU-SeeMe vr: Immersive desktop teleconferencing. In *ACM Multimedia*, Boston, MA., 1996.

[49] Mark Handely. Personal communications. 1999.

[50] M. Handley and Sally Floyd. TCP-friendly congestion control. *Work in progress*, 1999.

[51] Mark Handley. An examination of mbone performance. Technical Report ISI/RR-97-450, USC/ISI Research Report, 1997.

[52] V.J. Hardman, M.A. Sasse, M. Handley, and A. Watson. Reliable audio for use over the internet. In *Proceedings of INET*, Reston, VA, July 1995.

[53] E. Hashem. Random drop congestion control. *M.S. Thesis, Massachusetts Institute of Technology, Department of Computer Science*, 1990.

[54] Juha Heinanen, Fred baker, Walter Weiss, and John Wroclawski. Assured forwarding phb group. *Internet draft*, January 1999.

[55] Markus Hofmann, T.S. Eugene Ng, Katherine Gue, Sanjoy Paul, and Hui Zhang. Caching techniques for streaming multimedia over the internet. Technical report, Lucent Technology, 1999.

Webgroup CZ a.s. Ex. 1015, Page 195 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[56] Infolibria Inc. Mediamall. 1999. http://www.infolibria.com.

[57] Inktomi Inc. 1999. http://www.inktomi.com.

[58] Microsoft Inc. Netshow service, streaming media for business. http://www.microsoft.com/NTServer/Basics/NetShowServices.

[59] S. Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the Annual ACM Symposium on the Theory of Computing*. March 1997.

[60] S. Jacobs and A. Eleftheriadis. Real-time dynamic rate shaping and control for internet video applications. *Workshop on Multimedia Signal Processing*, pages 23–25, June 1997.

[61] S. Jacobs and A. Eleftheriadis. Real-time dynamic rate shaping and control for internet video applications. *Workshop on Multimedia Signal Processing*, pages 23–25, June 1997.

[62] V. Jacobson. Email on the end-to-end email list. February 1997.

[63] V. Jacobson and S. McCanne. vic: a flexible framework for packet video. *Proceedings of ACM Multimedia*, November 95.

[64] V. Jacobson, Kathleen Nicols, and Kedernath Poduri. An expedited forwarding PHB. *Internet draft*, February 1999.

[65] Van Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM*, pages 314–329, Stanford, CA., August 1988. ACM.

[66] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM Computer Communication Review*, 19(5):56–71, October 1989.

[67] Sugih Jamin, Peter B. Danzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA, September 1995.

Webgroup CZ a.s. Ex. 1015, Page 196 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[68] K. Jeffy, D. L. Stone, T. Talley, and F. D. Smith. Adaptive, best-effort delivery of digital audio and video across packet-switched networks. In *Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, CA., November 1992.

[69] M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In *Proceedings of the 4th International Conference on Database Systems for Advanced Applications*, April 1995.

[70] H. Kanakia, P. P. Mishra, and A. Reibman. An adaptive congestion control scheme fro real-time packet video transport. In *Proceedings of the ACM SIGCOMM*, pages 20–31, San Francisco, CA., September 1993.

[71] Gunnar Karlsson and Martin Vetterli. Packet video and its integration into the network architecture. *IEEE Journal of Selected Areas in Communication*, 7(5):739–751, June 1989.

[72] S. Keshav. A control-theoretic approach to congestion control. In *Proceedings of the ACM SIGCOMM*, pages 3–16, Zurich, Switzerland, September 1991.

[73] S. Keshav. Packet-pair flow control. *ACM/IEEE Transactions on Networking*, 1997.

[74] T. V. Lakshman and U. Madhow. Performance analysis of window-based flow control using TCP/IP: the effect of high bandwidth-delay products and random losses. In *IFIP Transactions C-26, High Perfomance Networking V*, North Holland, 1993.

[75] S. S. Lam, S. Chow, and D. K. Y. Yau. A lossless smoothing algorithm for compressed video. *ACM/IEEE Transactions on Networking*, 4(5), October 1996.

[76] Jae-Yong Lee, Tae-Hyun Kim, , and Sung-Jea Ko. Motion prediction based on temporal layering for layered video coding. *Proceedings ITC-CSCC*, 1:245–248, July 1998.

Webgroup CZ a.s. Ex. 1015, Page 197 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[77] C. Leffehocz, B. Lyles, S. Shenker, and L. Zhang. Congestion control for best-effort service: why we need a new paradigm. *IEEE Network*, 10(1), January 1996.

[78] Didier LeGall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 4(34):47–58, April 1991.

[79] J. Koruse M. Yajnik and D. Towsley. Packet loss correlation in the mbone multicast network. In *Proceedings IEEE GLobal Internet Conference*, November 1996.

[80] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. *Technical note sent to the end2end-interest mailing list*, January 1997. http://www.psc.edu/networking/papers/tcp-friendly.html.

[81] A. Mankin. Random drop congestion control. In *Proceedings of the ACM SIGCOMM*, Philadelphia, PA., 1990.

[82] A. Mankin. Collaborative advanced interagency research network. *Slide Presentation*, February 1998.

[83] M . Mathis, S. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. *RFC 2018*, April 1996.

[84] M. Mathis and J. Mahdavi. Forward acknowledgment: Refining TCP congestion control. *Proceedings of the ACM SIGCOMM*, August 1996.

[85] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review*, 27(3), July 1997.

[86] S. McCanne. Scalable compression and transmission of internet multicast video. *Ph.D. thesis, University of California Berkeley, UCB/CSD-96-928*, December 1996.

[87] S. McCanne and S. Floyd. Ucb/lbnl/vint network simulator - ns(version 2). *Software on-line*, 1999. http://www-mash.cs.berkeley.edu/ns.

Webgroup CZ a.s. Ex. 1015, Page 198 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[88] S. McCanne, V. Jacobson, and M. Vettereli. Receiver-driven layered multicast. In *Proceedings of the ACM SIGCOMM*, pages 117–130, Stanford, CA., August 1996.

[89] S. McCanne and M. Vetterli. Joint source/channel coding for multicast packet video. In *Proceedings of the Proceedings of the IEEE International Conference on Image Processing*, pages 776–785, Washington D.C., October 1995.

[90] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity video coding for receiver-driven layered multicast. *IEEE Journal of Selected Areas in Communication*, 16(3), August 1997.

[91] Zhourong Miao and Antonio Ortega. Proxy caching for efficient video services over the internet. In *9th International Packet Video Workshop (PVW '99)*, New York, April 1999.

[92] P. Mishra and H. Kanakia. A hop-by-hop rate-based congestion control scheme. In *Proceedings of the ACM SIGCOMM*, Baltimor, MD., August 1992.

[93] Real Networks. HTTP versus realaudio client-server streaming. http://www.realaudio.com/help/content/http-vs-ra.html.

[94] A. Ortega and M. Khansari. Rate control for video coding over variable bit rate channels with applications to wireless transmission. In *Proceedings of the Proceedings of the IEEE International Conference on Image Processing*, Washington, DC, October 1995.

[95] Antonio Ortega, Fabio Carignano, Serge Ayer, and Martin Vetterli. Soft caching: Web cache management for images. In *IEEE Signal Processing Society Workshop on Multimedia*, Princeton, NJ, June 1997.

[96] Padhye, J. Kurose, D. Towsley, and R. Koodli. TCP-friendly rate adjustment protocol for continuous media flows over best effort networks. Technical Report 98-11, UMASS CMPSCI, 1998.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[97] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM*, Vancouver, Canada, September 1998.

[98] C. Papadopoulos and G. M. Parulkar. Retransmission-based error control for continuous media applications. *Workshop on Network and Operating System Support for Digital Audio and Video*, April 1995.

[99] C. Perkins. Options for repair of streaming media. *Internet Draft*. March 1998.

[100] M. Podolsky, S. McCanne, and M. Vettereli. Soft ARQ for layered streaming media. Technical report, UC Berkeley, 1999.

[101] M. Podolsky, C. Romer, and S. McCanne. Simulation of FEC-based error control for packet audio on the internet. In *Proceedings of the IEEE INFOCOM*, San Francsico, CA., March 1998.

[102] W. Prue and J. Postel. Something a host could do with source quench: The source quench introduced delay (squid). *RFC 1016*, 1987.

[103] K. K. Ramakrishan and Raj Jain. A binary feedback scheme for congestion avoidance in computer networks with connectionless network layer. In *Proceedings of the ACM SIGCOMM*, pages 303–313, Stanford, CA., August 1988.

[104] K. K. Ramakrishan and Raj Jain. A binary feedback scheme for congestion avoidance in computernetworks. *ACM Transactions on Computer Systems*, 8(2):158–181, 1990.

[105] K.K. Ramakrishnan and Sally Floyd. A proposal to add explicit congestion notification(ECN) to ip. *RFC 2481, Experimental*, January 1999.

[106] G. Ramamurthy and D. Raychaudhuri. Performance of packet video with combined error recovery and concealment. In *Proceedings of the IEEE INFOCOM*, April 1995.

Webgroup CZ a.s. Ex. 1015, Page 200 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[107] R. Ramjee, J. F. Kurose, D. Towsley, and H. Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. *Proceedings of the IEEE INFOCOM*, 1994.

[108] ITU-T Recommendation. Video codec for audiovisual services at p * 64 kb/s. *ITU-T Recommendation*, 1993.

[109] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proc. IEEE Infocom*, New York, NY., March 1999.

[110] Reza Rejaie, Mark Handley, Haobo Yu, and Deborah Estrin. Proxy caching mechanism for multimedia playback streams in the internet. In *Proceedings of the 4th International Web Caching Workshop*, San Diego, CA., March 1999.

[111] Reza Rejaie, Haobo Yu, Mark Handley, and Deborah Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. In *Under Submission*, July 1999.

[112] Injong Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proceedings of the ACM SIGCOMM*, Vancouver, Canada, September 1998.

[113] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. Technical Report RN/98/13, UCL-CS, 1998.

[114] J. Salehi, Z.-L. Zhang, J. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal. In *ACM SIGMETRICS*, Philadelphia, PA., May 1996.

[115] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for realtime applications. In *Internet Engineering Task Force, Audio-Video Transport Working group*, January 1996. RFC 1889.

[116] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of the IEEE INFOCOM*, 1999.

Webgroup CZ a.s. Ex. 1015, Page 201 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[117] S. Shenker. A theoretical analysis of feedback flow control. In *Proceedings of the ACM SIGCOMM*, pages 156–165, Philadelphia, PA., September 1990.

[118] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal of Selected Areas in Communication*, 13(7):1176–1188, 1995.

[119] S. Shenker, L. Zhang, and D. Clark. Some observations on the dynamics of a congestion control algorithm. *ACM Computer Communication Review*, 2(4), October 1990.

[120] Henning Shulzrinne. Voice communication across the internet. Technical Report TR 92-50, University of Massachusets, Amherest, July 1992.

[121] D. Sisalem and H. Schulzrinne. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. *Workshop on Network and Operating System Support for Digital Audio and Video*, 1998.

[122] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of the ACM SIGCOMM*, pages 118–130, Vancouver, Canada, September 1998.

[123] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *Proceedings of the ACM SIGCOMM*, Cambridge, MA., September 1999.

[124] W. Tan and A. Zakhor. Error resilient packet video for the internet. In *Proceedings of the Proceedings of the IEEE International Conference on Image Processing*, Chicago, Illinois, October 1998.

[125] D. L. Tannenhouse and D. J. Wetherall. Towards an active network architecture. *ACM Computer Communication Review*, 3(2):5–18, April 1996.

[126] R. Tewari, H. Vin, A. Dan, and D. Sitaram. Resource based caching for web servers. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking*, San Jose, CA., 1998.

Webgroup CZ a.s. Ex. 1015, Page 202 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[127] J. Touch. The LSAM proxy cache - a multicast distributed virtual cache. In *Proceedings of the Third International WWW Caching Workshop*, June 1998.

[128] T. Turletti. The INRIA videoconferencing system (IVS). *ConneXions - The Interoperability Report Journal*, 8(10):20–24, October 94.

[129] T. Turletti and C. Huitema. Videoconferencing in the internet. *ACM/IEEE Transactions on Networking*, pages 340–351, June 1996.

[130] Lorenzo Vicisano, Luigi Rizzo, and Jon Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of the IEEE INFOCOM*, 1998.

[131] M. Vishwanath and P. Chou. An efficient algorithm for hierarchical compression of video. In *Proceedings of the Proceedings of the IEEE International Conference on Image Processing*, Austin, TX, November 1994.

[132] Y. Wang, Z.-L. Zhang, D. Du, and D. Su. A network conscious approach to end-to-end video delivery over wide area networks using proxy servers. In *Proceedings of the IEEE INFOCOM*, April 1998.

[133] Z. Wang and J. Crowcroft. A new congestion control scheme: Slow start and search (Tri-S). *ACM Computer Communication Review*, 21(1):32–43, January 1991.

[134] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM SIGCOMM*, pages 293–305, Stanford, CA., 1996.

[135] R. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proceedings of the Sixth International WWW conference*, April 1997.

[136] L. Wu, R. Sharma, and B. Smith. Thin streams: An architecture for multicasting layered video. *Workshop on Network and Operating System Support for Digital Audio and Video*, May 1997.

Webgroup CZ a.s. Ex. 1015, Page 203 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[137] R. Xu, C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *Workshop on Network and Operating System Support for Digital Audio and Video*, St. Louis, May 1997.

[138] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: a new resource ReSerVation Protocol. *IEEE Network*, 7:8–18, September 1993.

[139] L. Zhang, S. Michel, K. Nguyen, and A. Rosenstein. Adaptive web caching: Towards a new global caching architecture. In *Proceedings of the Third International WWW Caching Workshop*, June 1998.

[140] L. Zhang, S. Shenker, and D. Clark. Observations on the dynamics of a congestion control algorithm: The effect of two-way traffic. In *Proceedings of the ACM SIGCOMM*, Zurich, Switzerland, September 1991.

Webgroup CZ a.s. Ex. 1015, Page 204 of 204
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.