

# Scalable Network Architectures, Protocols and Measurements for Adaptive Quality of Service

Xin Wang

Submitted in partial fulfillment of the  
requirements for the degree  
of Doctor of Philosophy  
in the Graduate School of Arts and Sciences

Columbia University

2001

© 2001

Xin Wang

All Rights Reserved

ABSTRACT

## **Scalable Network Architectures, Protocols and Measurements for Adaptive Quality of Service**

Xin Wang

New IP services and applications have diverse and stringent bandwidth and quality of service requirements. It is difficult and costly to predict these requirements and add sufficient capacity to provide reliable and high quality service. This thesis is broadly concerned with scalable and efficient architectures for delivering Internet applications reliably and with high quality. The work consists of two parts.

We have developed a flexible, scalable service framework to utilize existing network capacity efficiently. The framework is based on conditioning the network to provide multiple services with certain quality of service expectations even under high demand, using short-term, dynamic configuration of network resources for better response to user demand and more efficient resource usage. To support this framework, we develop a Resource Negotiation and Pricing Protocol (RNAP) to enable the user and the network (or two peer networks) to dynamically negotiate services, and to formulate and communicate prices and charges.

As part of the service framework, we also propose a pricing model in which services are priced based on QoS (resources consumed) and user willingness-to-pay. The model also motivates rate and service adaptation by applications with elastic demand through congestion-sensitive pricing of certain services, while providing more expensive services with static pricing for non-elastic users. We describe two mechanisms to develop a congestion-sensitive price component, one based on a tâtonnement process, and the other on a second-price, multiple-bid auction. We also describe a user rate-adaptation model in response to price changes, based on

maximization of the user-perceived benefit.

We have demonstrated the functionality of RNAP on a testbed network, and have shown that the framework can achieve high utilization and control congestion, while maintaining a stable price. We also present simulation results to show that our proposed framework achieves a lower connection blocking rate, higher user satisfaction based on user utility functions, and higher network revenue. The performance improvement increases with the number of connections. The auction-based congestion pricing mechanism is seen to provide higher network utilization than the tâtonnement-based mechanism, at the cost of higher complexity.

The second part of the thesis is a study of the performance of the Lightweight Directory Access Protocol (LDAP). LDAP is potentially useful for the management of network resources, and the administration of traffic-handling and pricing policies. We describe a benchmark tool to study LDAP performance in a dynamic environment. The tool provides a detailed profile of the latency and throughput contributions of system components. We report measured performance using a LDAP schema proposed for administration of Service Level Specifications in a differentiated network. In most cases, the connection management latency increases sharply at high loads and thus dominates the response time. The TCP Nagle algorithm is found to introduce a very large additional latency, and it appears beneficial to disable it in the LDAP server. The CPU capability is found to be significant in limiting the performance of the LDAP server, and for larger directories, which cannot be kept in memory, data transfer from the disk also plays a major role. The scaling of server performance with the number of directory entries is determined by the increase in back-end search latency, and scaling with directory entry size is limited by the front-end encoding of search results, and, for out-of-memory directories, by the disk access latency.

# Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>I</b>	<b>An Integrated Resource Negotiation, Pricing and QoS Adaptation Framework for Multimedia Applications</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>4</b>
2.1.	Motivation . . . . .	4
2.1.1	Status of the Current Internet . . . . .	4
2.1.2	Cost of Bandwidth . . . . .	6
2.1.3	More Efficient Service Models . . . . .	7
2.2.	Design Goals . . . . .	9
2.2.1	Support Signaling for Dynamic Service Selection and Resource Negotiation . . . . .	9
2.2.2	Support Differential Pricing for Differentiated Services . . . . .	10
2.2.3	Support Short-Term Resource Allocation and Congestion-sensitive Pricing . . . . .	11
2.2.4	Develop Reference User Adaptation Model . . . . .	11
2.3.	Our Contributions . . . . .	12
2.4.	Organization . . . . .	14

<b>3</b>	<b>Background and Related Work</b>	<b>16</b>
3.1.	Signaling for Resource Reservation and Allocation . . . . .	16
3.1.1	Integrated Services (IntServ) and the Signaling . . . . .	16
3.1.2	Differentiated Service (DiffServ) and Signaling . . . . .	18
3.1.3	Other Work on Resource Allocation . . . . .	23
3.1.4	Conclusion . . . . .	23
3.2.	Adaptive Internet Multimedia Applications . . . . .	23
3.2.1	Sender-driven Adaptation . . . . .	24
3.2.2	Receiver-driven Adaptation . . . . .	25
3.2.3	Transcoder-based Adaptation . . . . .	28
3.2.4	Signaling Mechanism . . . . .	30
3.2.5	Conclusion . . . . .	31
3.3.	Pricing and Billing in the Network . . . . .	32
3.3.1	Total User Benefit Maximization based on Welfare Theory . . . . .	32
3.3.2	Pricing for Efficient Resource Utilization and Congestion Control . . . . .	33
3.3.3	Pricing in Multi-service Environment . . . . .	34
3.3.4	Auction Mechanisms . . . . .	34
3.3.5	Signaling Support for Pricing and Charging . . . . .	37
3.3.6	Conclusion . . . . .	38
<b>4</b>	<b>RNAP: A Resource Negotiation and Pricing Protocol</b>	<b>39</b>
4.1.	Basic Characteristics . . . . .	41
4.1.1	Dynamic Re-negotiation Capability . . . . .	41
4.1.2	Pricing and Charging Capability . . . . .	42
4.1.3	Scalability . . . . .	43
4.1.4	Service Predictability . . . . .	43

4.2. Basic Model . . . . .	43
4.2.1 The RNAP Architecture . . . . .	44
4.2.2 Protocol Operation . . . . .	46
4.3. The Protocol . . . . .	49
4.3.1 Basic Objects . . . . .	49
4.3.2 Message Forwarding Mechanisms . . . . .	55
4.3.3 RNAP Message Format . . . . .	56
4.4. State Aggregation . . . . .	61
4.4.1 Overview of the Aggregation Scheme of RNAP . . . . .	62
4.4.2 Aggregation and De-aggregation in RNAP-D . . . . .	63
4.4.3 Aggregation and De-aggregation in RNAP-C . . . . .	65
4.4.4 Overhead Reduction due to Aggregation . . . . .	66
4.4.5 Resource Negotiation for Flow-Aggregates and Advance Reser- vation . . . . .	66
4.5. Receiver Negotiation . . . . .	68
4.6. Price and Charge Collation in RNAP . . . . .	69
4.6.1 Pricing Structure and its Use in RNAP Messages . . . . .	70
4.6.2 Shared Charging . . . . .	72
4.6.3 Multicast Charging . . . . .	73
4.7. NRN Location for a Neighboring Domain . . . . .	74
4.8. Other Issues . . . . .	76
4.8.1 Transport Protocol and Reliability . . . . .	76
4.8.2 Stability . . . . .	77
4.8.3 Security . . . . .	78
4.9. Protocol Comparison between RNAP and RSVP . . . . .	78

<b>5 Pricing Strategies</b>	<b>81</b>
5.1. Price and Charge Formulation in Network . . . . .	83
5.1.1 Price and Charge Formulation in RNAP-D . . . . .	84
5.1.2 Price Formulation in RNAP-C . . . . .	84
5.1.3 Charge Formulation for Multiple Domains and Flow Aggregates	88
5.2. Pricing for Differentiated Services . . . . .	88
5.2.1 Holding Charge . . . . .	89
5.2.2 Usage Charge . . . . .	90
5.2.3 Congestion Charge . . . . .	92
5.2.4 Total Charge . . . . .	99
<b>6 User Adaptation</b>	<b>101</b>
6.1. The Utility Function . . . . .	102
6.1.1 Utility as Perceived Value . . . . .	103
6.1.2 Utility as a Function of Bandwidth . . . . .	104
6.1.3 Effect of Scaling . . . . .	105
6.1.4 Time Dependence of Utility . . . . .	106
6.1.5 An Example Utility Function and User Adaptation . . . . .	106
6.2. Application Adaptation Under CPA-TAT Policy . . . . .	108
6.2.1 Adaptation of Single Application with Fixed Transmission Quality . . . . .	108
6.2.2 Adaptation of Single Application over Multiple Transmission Parameters . . . . .	110
6.2.3 Simultaneous Adaptation of Multiple Applications correspond- ing to Single Task . . . . .	111
6.2.4 Adaptation of Applications with Example Utility Functions .	114
6.3. Resource Allocation under CPA-AUC Policy . . . . .	115



6.3.1	Adaptation of Single Application over Multiple Transmission Parameters . . . . .	116
6.3.2	Auction for Multiple Applications corresponding to a Single Task . . . . .	116
6.3.3	Bidding over Multiple Hops . . . . .	116
6.4.	System Stability and Network Dynamics . . . . .	117
6.4.1	Price Stability . . . . .	117
6.4.2	Stability of User Bandwidth Requests . . . . .	119
<b>7</b>	<b>Simulation</b>	<b>122</b>
7.1.	Simulation Model . . . . .	123
7.2.	Performance Comparison of CPA-TAT, CPA-AUC and FP policies . .	127
7.2.1	Bottleneck Bandwidth Utilization . . . . .	128
7.2.2	User Request Blocking Rate . . . . .	129
7.2.3	Network Revenue . . . . .	130
7.2.4	Average and Total User Benefit . . . . .	130
7.2.5	Dynamics of the System Price and User Bandwidth . . . . .	131
7.2.6	A General Network Topology . . . . .	133
7.2.7	Result Summary . . . . .	134
7.3.	Effect of Session Multiplexing . . . . .	135
7.4.	User Behavior . . . . .	136
7.4.1	Effect of User Demand Elasticity . . . . .	137
7.4.2	Adaptive and Non-adaptive Users . . . . .	139
7.4.3	Initial Adaptation versus Continuous Adaptation . . . . .	142
7.5.	CPA-TAT Control Parameters . . . . .	144
7.5.1	Effect of Congestion Control Threshold . . . . .	144
7.5.2	Effect of Price Adjustment Step . . . . .	147

7.5.3	Effect of Price Adjustment Threshold . . . . .	150
7.5.4	CPA-TAT with Partial Admission . . . . .	151
7.6.	Packet-level Simulation of DiffServ Classes . . . . .	152
7.6.1	Effect of Traffic Burstiness . . . . .	153
7.6.2	Effect of Traffic Load . . . . .	155
7.6.3	Load Balance between Classes . . . . .	157
7.6.4	Effect of Admission Control . . . . .	160
7.7.	Other Mechanisms to Reduce Network Dynamics . . . . .	161
<b>8</b>	<b>Implementation</b>	<b>162</b>
8.1.	RNAP over RSVP . . . . .	162
8.1.1	Protocol Implementation . . . . .	163
8.1.2	Experimental Results . . . . .	166
8.2.	Stand-alone RNAP . . . . .	176
8.2.1	Architecture . . . . .	177
8.2.2	User Interface Design . . . . .	183
8.2.3	Results . . . . .	189
<b>9</b>	<b>Conclusion</b>	<b>192</b>
9.1.	Resource Negotiation and Pricing Protocol - RNAP . . . . .	193
9.2.	Pricing Model . . . . .	195
9.3.	User Adaptation Model . . . . .	196
9.4.	Performance of the Proposed Negotiation Framework . . . . .	197
9.5.	Future Work . . . . .	199
<b>II</b>	<b>Measurement and Analysis of LDAP Performance</b>	<b>201</b>
<b>10</b>	<b>Measurement and Analysis of LDAP Performance</b>	<b>202</b>

10.1. Introduction . . . . .	202
10.2. Background . . . . .	204
10.2.1 The LDAP Directory Service . . . . .	204
10.2.2 Using LDAP for SLS Administration . . . . .	205
10.3. Experimental Setup . . . . .	209
10.3.1 Hardware . . . . .	209
10.3.2 LDAP Server . . . . .	210
10.3.3 LDAP Client . . . . .	211
10.4. Test Methodology . . . . .	212
10.5. Result Analysis . . . . .	214
10.5.1 Overall LDAP Performance . . . . .	215
10.5.2 Improving the LDAP Search Performance . . . . .	216
10.5.3 Performance Limitations . . . . .	221
10.5.4 Session Reuse . . . . .	228
10.5.5 Performance under Update Load . . . . .	229
10.6. Related Work . . . . .	230
10.7. Conclusions . . . . .	231

<b>References</b>	<b>234</b>
-------------------	------------

## List of Figures

3-1	Linear regulator with dead zone (from [1]) . . . . .	24
3-2	Block diagram of transcoder proposed by Amir, et al. [2] . . . . .	29
3-3	Adaptation of packet size in the TCP-like congestion control scheme used by Kouvelas, et al. [3] . . . . .	31
4-1	RNAP-C architecture . . . . .	44
4-2	RNAP-D architecture . . . . .	46
4-3	RNAP messaging sequence between HRN and NRN. . . . .	46
4-4	Example of RNAP-D message aggregation. . . . .	62
4-5	Example of RNAP-C message aggregation. . . . .	65
5-1	Price formulation in RNAP-C . . . . .	85
6-1	Some example utility functions . . . . .	104
6-2	A perceived value based rate adaptation model . . . . .	110
7-1	Simulation network topology 1 . . . . .	124
7-2	Simulation network topology 2 . . . . .	124
7-3	Performance metrics of CPA-TAT, CPA-AUC, and FP policies as a function of offered load: (a) bottleneck utilization; (b) blocking probability; (c) total network revenue; (d) total user benefit; (e) average user benefit. . . . .	128

7-4 System dynamics under CPA-TAT and CPA-AUC: variation over time of system price (a), user demand (b) of CPA-TAT; time-average and standard deviation of system price of CPA-TAT and CPA-AUC at offered load 1.2 (c); variation over time of system price (d), user demand (e) of CPA-AUC; and average user demand and standard deviation of CPA-TAT and CPA-AUC at an offered load of 1.2 (f). 132

7-5 Performance metrics of CPA-TAT, CPA-AUC, and FP policies as a function of offered load with topology 2: (a) blocking probability; (b) average user benefit; (c) Time-average and standard deviation of system price of CPA-TAT and CPA-AUC at offered load 1.2. . . . . 134

7-6 Performance of CPA and FP with different number of customers sharing the system: (a) bottleneck utilization; (b) blocking probability; (c) average user benefit 135

7-7 Time-average and standard deviation of system price of CPA-TAT (a) and CPA-AUC (b), plotted against offered load. . . . . 136

7-8 Effect of the elasticity factor  $w$  on bandwidth allocation and user expenditure: (a) minimum, maximum and average requested bandwidth when users have the same utility function; (b) minimum, maximum and average requested bandwidth when users have three different utility functions, with  $w$  set to 0.20, 0.25, and 0.30 \$/min. respectively; (c) average bandwidth reserved by users with the three different values of  $w$ ; (d) average expenditure of users with the three different values of  $w$  . . . . . 138

7-9 Performance of CPA when only some of the users adapt their bandwidth requests: (a) bottleneck utilization; (b) blocking probability; (c) network revenue; (d) total user benefit, plotted against offered load. . . . . 140

7-10 Relative benefits and expenditures of adaptive and non-adaptive users, under three different proportions of non-adaptive to adaptive users: (a) average user benefit for the two groups of users; (b) average benefits and average expenditures of non-adaptive users relative to average benefits and expenditures of non-adaptive users . . . . . 140

7-11 Performance when CPA users select bandwidth only at session set-up, compared with performance when they continue to adapt during the session (a) bottleneck utilization; (b) blocking probability; (c) network revenue; (d) total user benefit. . . 143

7-12 Performance of CPA and FP policies at different values of target congestion control threshold  $\rho$ : (a) bottleneck utilization; (b) blocking probability; (c) total network revenue; (d) total user benefit. . . . . 145

7-13 System dynamics at different values of the congestion control threshold: variation over time of system price (a), and average user demand (b), at an offered load of 1.2; time-average and standard deviation of system price (c) and average user demand (d), plotted against offered load. . . . . 146

7-14 Performance of CPA and FP at different values of  $\sigma$ : (a) bottleneck utilization; (b) blocking probability; (c) total network revenue; (d) total user benefit. . . . 148

7-15 System dynamics at different values of  $\sigma$ : variation over time of system price (a), and average user demand (b), at on offered load of 1.2; time-average and standard deviation of system price (c) and average user demand (d), plotted against offered load. . . . . 149

7-16 Performance of CPA and FP at different values of  $\theta$ : a) bottleneck utilization; b) blocking probability; c) average net user benefit; (d) total net user benefit. . . . 150

7-17 System dynamics at different values of  $\theta$ : variation over time of system price (a), and average user demand (b), at on offered load of 0.9; . . . . . 151

7-18 Performance of CPA with and without partial admission: (a) bottleneck utilization; (b) blocking probability; (c) network revenue; (d) total user benefit.] . . . . 153

7-19 System dynamics under CPA with increase in AF traffic burst index: (a) price average and standard deviation of AF class; (b) variation over time of AF. Performance metrics of CPA and FP policies as a function of burst index of AF class: (c) average packet delay; (d) average packet loss; (e) average traffic arrival rate; (f) average user benefit. . . . . 154

7-20 System dynamics under CPA with increase in AF offered load: (a) average and standard deviation of AF class price; (b) variation over time of AF class price. Performance metrics of CPA and FP policies as a function of AF offered load: (c) average packet delay; (d) average packet loss; (e) average bottleneck traffic arrival rate; (f) average user benefit. . . . . 156

7-21 Performance metrics of CPA and FP policies with traffic migration between classes: (a) variation over time of AF class price; (b) ratio of AF class traffic migrating through class re-selection; (c) average packet delay of all classes; (d) average packet loss of all classes; . . . . . 158

7-22 System dynamics under CPA with access control CPA as AF offered load increases: (a) average and standard deviation of AF class price. Performance metrics of CPA and FP policies with access control as a function of AF offered load: (b) user requests blocking rate; (c) average packet delay; (d) average packet loss. 159

8-1 Testbed setup . . . . . 165

8-2 The architecture of the extended MInT system . . . . . 165

8-3 Utility functions used in the experiments of section 4.6.2 and 8.1.2.3 . . . . . 166

8-4	Allocation of bandwidth and surplus for three competing users sharing a link. a1, a2, and a3 show the results when the users all have the Utility 1 function from Fig. 8-3, and b1, b2, and b2 show corresponding results when the users have the Utility 2 function from the same figure . . . . .	167
8-5	Utility functions with different bandwidth sensitivities . . . . .	170
8-6	Bandwidth reservation a) and perceived surplus b) when the users have different demand elasticities . . . . .	170
8-7	Network charges for different users a) and the total network bandwidth demand and price b) when the users have different demand elasticities . . . . .	171
8-8	Bandwidth reservation a) and perceived surplus value b) for adaptation across media sessions in a system, all sessions having the same utility . . . . .	171
8-9	Resource reservation a) and perceived surplus value b) among sessions of a system with different bandwidth sensitivity . . . . .	172
8-10	Equivalent utilities under multiplicative scaling a) and additive shifting b) . . .	173
8-11	Bandwidth reservation and perceived surplus for utilities scaled multiplicatively by different amounts . . . . .	174
8-12	Bandwidth reservation and perceived surplus for utilities shifted additively by different amounts . . . . .	174
8-13	a) Audio and video utility functions used for adaptation by MInT b) Price and total bandwidth variation in the same experiment . . . . .	175
8-14	Individual bandwidth reservations and perceived surplus in the adaptation of MInT applications . . . . .	175
8-15	Testbed setup for stand-alone RNAP implementation . . . . .	177
8-16	The main window for user profile configuration . . . . .	183
8-17	The user budget distribution window . . . . .	184
8-18	The audio profile window . . . . .	185



8-19	The video profile window . . . . .	185
8-20	The service and application list window . . . . .	186
8-21	The application state window . . . . .	186
8-22	Configuration of different services . . . . .	187
8-23	Service statistics . . . . .	187
8-24	Flow profile . . . . .	188
8-25	Service statistics of a flow . . . . .	188
8-26	Accumulated charging information for a flow . . . . .	188
8-27	Service configuration at an interface . . . . .	189
8-28	Service statistics at an interface . . . . .	189
8-29	Measured traffic as compared with target utilization . . . . .	190
8-30	Service statistics when applications are not adaptive . . . . .	190
8-31	Service statistics when applications are adaptive . . . . .	191
10-1	An example of organization of data in a LDAP directory . . . . .	205
10-2	An architecture for network QoS control using LDAP . . . . .	207
10-3	LDAP tree structure in tests . . . . .	209
10-4	LDAP benchmarking testbed architecture . . . . .	212
10-5	Sequence of steps in a simple LDAP search operation . . . . .	213
10-6	Average connection time, processing time and response time (a) and average server throughput (b) shown as a function of the average request rate at the server. The directory had 10,000 entries and each entry size was 488 bytes. . . . .	215
10-7	Latency associated with the various server and client process modules in a LDAP <i>search</i> operation . . . . .	216
10-8	Variation of the <i>open</i> time and <i>bind</i> time components of the average connection time with average request rate at the server. The directory had 10,000 entries and each entry size was 488 bytes. . . . .	218

10-9	Comparison of the server performance with and without Nagle algorithm: (a) average connection time, processing time and response time; (b) average server throughput. The directory had 10,000 entries and each entry size was 488 bytes.	219
10-10	Comparison of the performance when the server is configured with 0 and 10,000 entry caches: (a) the response latency; (b) the throughput.	220
10-11	Effect of using a dual processor server on the response time and throughput, with a directory containing 10,000 entries, entry size 488 bytes: (a) connect, processing and response latencies versus request rate; (b) throughput versus request rate	222
10-12	Throughput of search and add operations for single and dual processor servers, with a varying number of LDAP clients generating queries continuously	224
10-13	Effect of directory size on the server connect and processing times (a) and the server throughput (b).	224
10-14	Effect of directory entry size on the server connect and processing times (a) and the server throughput (b) for 5000 entry directory and 5000 entry cache	226
10-15	Effect of directory entry size on the server connect and processing times (a) and the server throughput (b) for a 10,000 entry directory, 5000 entry cache	227
10-16	Effect of session reuse rate on the server processing times (a) and the server throughput (b).	228
10-17	Comparison of latency and throughput characteristics for search and add operations: variation of connect, processing and response latency with server load (a), and variation of server throughput with server load (b). (Directory had 10,000 entries, entry size 488 bytes)	229

## Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Henning Schulzrinne, for giving me the opportunity to pursue this work, and for providing me invaluable guidance during the course of my thesis. I found that his vision and experience were oriented towards finding solutions to real-life problems. His questions often opened up new angles on challenging research issues. Henning has always been a great inspiration and support, and a model to follow.

I would also like to thank Prof. Ioannis Stavrakakis, for leading me into network research. His expertise and guidance on queueing theory and network theoretical analysis established an important foundation for my research career.

I would like to thank Dr. Paul Stirpe of Reuters Inc. for the enjoyable joint work at the beginning of my Ph.D, and for providing Reuter sponsorship for two years of my work. I would also like to thank Dr. Dilip Kandlur and Dr. Dinesh Verma for giving me the chance to work at the IBM T. J. Watson Research center during the summer of 1998. This work also went on to form part of my thesis. Thanks also go to Dr. Yow-Jian Lin and Dr. Mon-choon Chan for a very fruitful summer experience at Bell-Labs in 1999.

I am very grateful to Dr. Ellen Hahne for the valuable discussions on my thesis work during my summer at Bell-Labs. Thanks also go to Dr. T.V. Lakshman and Anwar Elwalid for providing some theoretical insight on network quality of service; and to Dr. Steve Lanning and Qiong Wang for providing some new thoughts on network economics. I'd also like to thank Dr. Roch Guerin and the members in his group for many interesting talks and discussions during my summer at IBM, and Dr. Debanjan Saha for his great support in the early stages of my thesis work. Special thanks to Dr. Wai Chen at Applied Research of Telcordia for countless discussions on various aspect of network research, and for his great support and advice.

I would like to thank my thesis committee: Prof. Mischa Schwartz, Prof. Andrew Campbell, Dr. Dilip Kandlur, Prof. Dan Rubenstein, and Prof. Henning Schulzrinne for taking time from their busy schedules to review my thesis and provide many valuable suggestions.

I would also like to thank Tony Eyers of University of Wollongong, Australia, for his advice on my thesis during his visit at Columbia and for sparing his time to review some of my papers. Thanks also go to Prof. Hermann de Meer of University of Hamburg, Germany, for many valuable discussions at the early stages of my thesis work.

Many thanks to the members of the Internet Real-Time Lab of Columbia University who have made my time at Columbia very enjoyable. In particular, I would like to thank Ping Pan for many interesting discussions and for allowing me to use his investigation of the literature of quality of service work in my thesis, Santosh Krishnan for discussion of work on network QoS and buffer management, Jonathan Lennox for help with systems work in the lab, Jonathan Rosenberg for many useful and informative discussions, Maria Papadopouli, Wenyu Jiang, Weibin Zhao, Xiaotao Wu, Ashutosh Dutta and Kundan Singh for many discussions on various topics. I would like to thank Sankaran Narayanan and Kyung-Tae Kim, and other project students for help in building the testbed. Also many thanks to the members from the Comet group to which I also belong. In particular, Raymond Liao, Nemo Semret, Javier Gomez-Castellanos, Jean-Francois Huard, Constantin M. Adam, and Koon-Seng Lim.

Particular thanks go to my friends Phil Wang and Jianqi Yin for their continuous encouragement during my thesis work, and Phil's help in some of the systems work.

I am eternally grateful to a very special friend, Sumit Majumder, for being a constant source of encouragement, for always being willing to challenge me and being

the first reader of my papers, and for his crucial role in sparking many aspects of this research.

Special thanks go to my parents, Jianfeng Wang and Lihua Gu for their constant support, love, and encouragement to pursue my Ph.D. I would also like to thank my sister Heng Wang for her help during my study overseas. Finally, I'd like to thank my husband Dawei Wu for his understanding, love, and continuous encouragement. I would not be able to finish my study without his support. This thesis is dedicated to my parents and my husband.

This work was supported in part by NSF CAREER award, NSF CISE award, Hughes, Reuters, and New York Center for Advanced Technology (CAT).

# Chapter 1

## Overview

Many new applications begin to be widely used in the Internet. These applications include real-time audio, video, and mission-critical financial data. At the same time, revenue from the traditional connectivity services (raw bandwidth) is declining. The new value-added services provide new business opportunities, but also present new challenges. The providers have to be able to add new services quickly and efficiently, and equip the network to meet the high quality and reliability expectations and diverse requirements of the value-added services.

This thesis is broadly concerned with scalable and efficient architectures for delivering multimedia applications over the Internet reliably and with high quality. The work consists of the following two parts:

**An integrated resource negotiation, pricing and QoS adaptation framework for multimedia applications.** The framework has the objectives of improving network utilization and user connectivity, and meeting the QoS requirements of a range of user applications. The framework is based on dynamic resource negotiation to enable short-term commitment of resources by the network and demand adaptation by adaptive user applications, and the pricing of network resources based on usage, QoS and user demand. The work incorporates a dynamic resource negotiation protocol, RNAP. RNAP enables negotiation between user applications and the

access network, as well as between adjoining network domains, and also enables the distribution and collation of price and charging information. We describe detailed mechanisms for the formulation of end-to-end prices in the RNAP architectures. We also present a demand-sensitive pricing model based on the cost of resources and long-term user demand. The pricing model considers a congestion-sensitive price component to motivate user adaptation during high demand. Two methods are considered for determining the congestion price, one based on a tâtonnement process, and the other based on an auction mechanism. User demand and adaptation are modeled based on user utility functions. The above resource negotiation and pricing framework has been implemented on a test-bed network, and also studied using simulations. We discuss the performance benefits under high or bursty loads, particularly increased user benefit and network revenue, and lower service blocking rate.

**Measurements and analysis of LDAP performance.** The Lightweight Directory Access Protocol (LDAP) is being used for a number of distributed directory applications. In the current context, it is potentially useful for the management of network resources, and the administration of traffic-handling and pricing policies. In this work, we describe a tool to analyze the performance of LDAP under a variety of access patterns. We report measured performance using a LDAP schema proposed for administration of Service Level Specifications in a DiffServ network. We discuss how various system components affect latency and scalability under various access patterns, and investigate mechanisms for improving performance.

The first part occupies the bulk of this thesis, and the second part is presented in the last chapter.

Part I

**An Integrated Resource  
Negotiation, Pricing and QoS  
Adaptation Framework for  
Multimedia Applications**



## Chapter 2

### Introduction

New distributed applications are growing rapidly. Many of these applications, such as multimedia applications, are bandwidth-intensive, and may also require high reliability from the network. The Internet's lack of control over quality of service (QoS) has slowed the deployment of these applications. While the technology for voice over IP is now available and the service is much in demand, its deployment on the public Internet has been severely limited because of the congestion problem. Clearly, we either need to greatly increase existing network capacity, or develop more sophisticated network service schemes which utilize the existing capacity more efficiently and are able to provide QoS assurances under high load, or combine both of these approaches. We now look at the motivation behind developing more sophisticated service models.

#### 2.1. Motivation

##### 2.1.1 Status of the Current Internet

We first look at some examples of traffic volume and congestion patterns in the current Internet, given in [4].

SWITCH [5] is a regional ISP that provides Internet connectivity to

Swiss academic and research institutions. Presently, the backbone consists of E3 (34 Mbps) and OC-3 trunks. The measurements indicate that although the average link utilization is low, the peak utilization at the network access point (NAP) and the direct peering links is quite high. The peak utilization at CERN NAP has reached the maximum link capacity, with the NAP operating on an Ethernet.

NORDUnet [6] interconnects the Nordic national networks for research and education and connects these networks to the rest of the world. NORDUnet provides its services by a combination of leased lines and Internet services provided by other international operators. The backbone is a mix of OC-3 and OC-12 trunks. NORDUnet connects to several NAP's, including Chicago NAP, and is peering with several other large provider backbones, such as TeleGlobe, Telia and FUNET. Most of the access links are OC-3 trunks, except that the Chicago NAP link runs at DS-3 speed, and the link to DGIX NAP is a 100 Mb/s Ethernet connection. Similar to other providers, NORDUnet uses multiple links to peer with other providers, and traffic to and from other providers are evenly distributed across all the links.

The investigation indicates that both NAP and private peering links can get congested, contrary to the general perception that private peering links are better managed and thus have less congestion. Trans-Atlantic links are a part of the provider's backbone, and have had very high peak utilization. This simply implies that there are bottleneck links within provider networks, and user traffic that stays within the same provider's network can also get congested.

Even though the average link utilization is quite low, around 20% to 30%, the peak link utilization is high on all links. At times, utilization reaches 100% on some

of the links. The congestion occurrence varies depending on links. The Chicago NAP link operated at peak once in August, 2000, for approximately seven hours. The DGIX NAP link operated at peak (over 80% link utilization) five times in September, 2000, and only once in August, 2000. Each congestion event in DGIX lasted between around one hour to as many as ten hours. Frequent and long-lasting congestion on several peering links has also been observed. During August and September, 2000, the TeleGlobe links operated at peak (over 80% link utilization) almost every workday. Each peak lasted about 8 hours.

Even though the traffic condition in NORDUnet and SWITCH networks may not be typical of the rest of the Internet, the statistics released by several other ISPs, such as Above.net [7] and BBC [8], shows that they all have similar traffic behavior, i.e., the average link utilization is always reasonably low, and many links are lightly loaded at all times. Every network always has some busy links (particularly, access links at NAP's and peering points) that have long lasting high bandwidth utilization.

### **2.1.2 Cost of Bandwidth**

One obvious solution to congestion is to add enough bandwidth. However, this is probably not feasible or cost effective for a lot of current ISPs, especially the smaller ones. This is because most of ISP links are leased from telephone companies. Despite many predictions that bandwidth would become “dirt” cheap, the reality is that leased line prices have not been decreasing consistently and dramatically.

Andrew Odlyzko in [9] estimated the tariff per month price for a T1 link from New York City to Pittsburgh, a distance of about 300 miles, which is about the average length of long distance private line links. In 1987, the link was priced at \$10,000, and this price consistently decreased for five years to \$4,000 in 1992. But from 1992 to 1998, the link price has climbed by over 50%, to \$6,000 per month.

Not only are leased lines expensive, interconnecting ISP's can be costly as well. Recently, some US ISP's [10] have claimed that the cost for a transit DS-3 link was \$50,000 per month, and an OC-3 link costs up to \$150,000 per month. The market dominance of a relatively small number of telephone companies, and government regulation are also likely to impede the drop in prices, despite the low cost of fiber installation and maintenance.

### **2.1.3 More Efficient Service Models**

As we have seen, access networks get congested frequently, and the cost of bandwidth is not dropping rapidly. It is also difficult to predict the various user requirements, especially due to the rapid deployment of new applications. Also, recent history tells us that availability of more bandwidth will create its own demand through increasing utilization of bandwidth intensive applications, such as real-time video and virtual reality. Consequently, it makes economic sense to study and develop models that utilize resources more efficiently than a simple best-effort, flat-rate network, and deliver adequate QoS even under high utilization. Two approaches have been studied and used in recent times, one based on providing QoS support through some form of resource reservation or prioritization, and the other based on user adaptation of sending rates in response to network congestion.

#### **2.1.3.1 Models based-on Quality of Service**

Current research in providing QoS support in the Internet is mainly based on two architectures defined by the IETF: per-flow based *integrated services* (IntServ) [11], and class-based *differentiated service* (DiffServ) [12]. IntServ provides end-to-end guaranteed [13] or controlled-load service [14], while DiffServ deals with Per-Hop Behaviors (PHB) [15] operating on traffic aggregates. Examples of DiffServ PHBs

include assured forwarding (AF) [16] and expedited forwarding (EF)[17].

In general, adding multiple levels of QoS requires more network management, and greater user-network negotiation. Currently, the Resource Reservation Protocol (RSVP) [18] is being used for setting up IntServ services, whereas DiffServ generally assumes there are static profiles to indicate users' preferences.

In addition, providing different services also requires a differentiated pricing structure. Currently the predominant form of Internet retail pricing in the united state is flat-rate pricing, which most home users of the Internet are familiar with. That is, the user is charged a fixed fee for a set amount of bandwidth to access the network. A network supporting multiple classes of service also requires a differentiated pricing structure, to motivate users to use the services they require and can afford, rather than the highest quality service.

### **2.1.3.2 Models based-on Application Adaptation**

QoS-support models involve a traffic contract or service level agreement (SLA) between the user and the network. If the agreement, including price negotiation and resource reservation is done statically (before transmission), pricing, resource allocation and provisioning have to be conservative to be able to meet QoS assurances in the presence of short- and long-term network traffic dynamics during the life of the application. Many multimedia applications are long-lived, exacerbating the problem. Allowing only static resource reservation unavoidably imposes higher resource costs and hence higher charges to the users.

A number of protocols and algorithms have been proposed for multimedia applications to dynamically regulate the source bandwidth according to the existing network conditions (a survey of this work is given in [19].) Compared to resource reservation, the adaptation approach has the advantage of better utilizing available

network resources, which change with time. But if network resources are shared by competing users, users of rate-adaptive applications do not have any incentive to scale back their sending rate below their access bandwidth, since selfish users will generally obtain better quality than those who reduce their rate. Also, the work in the current literature generally assumes best-effort service from the network, and does not consider how to work with QoS-assured services.

## **2.2. Design Goals**

The objective of our work is to develop an efficient service model which combines QoS assurance and differentiation in the network with rate adaptation by adaptive users. We define certain design goals which would make such a model feasible.

### **2.2.1 Support Signaling for Dynamic Service Selection and Resource Negotiation**

As more services are implemented in the Internet, user applications should be able to request and use the delivery service appropriate to their requirements. Also, when multiple services are available at different prices, users should be able to demand particular services, signal the network to provision according to the requested quality, and generate accounting and billing records. We can view the selection and use of a specific delivery service as a negotiation process. The customer and network negotiate and agree upon specifications such as the type of service user packets will receive, the constraints the user traffic must adhere to, and the price to be charged for the service.

The first goal of our work is to develop a protocol which enables a user to select from a set of available network services with different QoS characteristics, and enables the user and network to dynamically re-negotiate the contracted service

parameters and price. The protocol should be generic and flexible enough to support multiple delivery services and environments (including IntServ, DiffServ, and best effort services), service negotiation at different levels of granularity (flow- and aggregate-based), negotiation by both sender and receiver, and “in-band” and “out-of-band” resource reservation mechanisms. It should allow the service provider to communicate service availability, estimated prices for available services and charges accruing to the user, and allow the user to request a specific service. It should also support dynamic service re-negotiation between the user and the network, allowing the network to adjust pricing in response to changes in network load, and allowing the user to respond to changes in application requirements.

### **2.2.2 Support Differential Pricing for Differentiated Services**

Offering various value-added services over IP networks has major technical challenges; however, business and economic challenges are equally important. Clearly, providing different network services requires a differentiated pricing structure. The lack of an appropriate pricing scheme is one of the reasons that prevents service providers from supporting quality of service in the Internet.

While network tariff structures are often dominated by business and marketing arguments rather than costs, we believe it is worthwhile to understand and develop a cost-based pricing structure as a guide for actual pricing. In economically viable models, the difference in the charge between different service classes would presumably depend on the difference in performance between the classes, and should take into account the average (long-term) demand for each class. In general, the level of forwarding assurance of an IP packet in a network supporting multiple classes of services depends on the amount of resources allocated to a class the packet belongs to, the current load of the class, and in case of congestion within the class, the drop

precedence of the packet. The pricing structure should reflect the differences in cost arising from these factors.

### **2.2.3 Support Short-Term Resource Allocation and Congestion-sensitive Pricing**

For efficient resource usage, the network should be able to commit resources for the short-term, and dynamically reconfigure resources based on demand, particularly if users are adaptive. Also, it should be able to adjust the price during network congestion, by setting a congestion-sensitive component on top of the long-term, relatively static price.

The dynamic configuration of resources and pricing of services allows a more competitive price to be offered, and allows the network to be used more efficiently. Differentiated and congestion-sensitive pricing also provides a natural and equitable incentive for applications to adapt their service contract according to network conditions. Users that are adaptation-capable can adapt to price changes by adjusting their sending rate or selecting a different service class. Users with stringent bandwidth and QoS requirements can maintain a high quality by paying more, while adaptation-incapable applications will choose a static service.

### **2.2.4 Develop Reference User Adaptation Model**

Adaptation-capable users would tend to adapt to price-changes so as to maximize their perceived “value for money”, subject to budget and minimum quality requirements. We develop a user adaptation model based on this principle.



### 2.3. Our Contributions

We propose a dynamic resource negotiation protocol, RNAP, to enable the collation and distribution of prices and charges, and to enable dynamic resource negotiation between user applications and the access network, as well as between adjoining network domains. The proposed protocol and architecture are intended to co-exist with current Internet QoS schemes (e.g., those proposed within the IntServ and DiffServ frameworks), and work in a scalable manner over a variety of network architectures. RNAP is designed to be lightweight and flexible: it can function as a stand-alone protocol, but it is also possible to implement some components of RNAP as a layer on top of RSVP or other hop-by-hop reliable signaling protocols. RNAP is intended for use by both adaptive and non-adaptive applications. It provides a framework within which an application can adapt so as to obtain the best value from the network.

Our proposed architecture integrates resource reservation, negotiation, pricing and adaptation in a flexible and scalable way. The framework offers a middle ground, where resources are reserved, but resource commitments are made only for short intervals, instead of indefinitely. Prices may vary for each interval, encouraging applications to adjust their resource demands to network congestion. Unlike best-effort adaptive approaches, applications are guaranteed resources and there is no assumption that applications are cooperative. Our model allows the network operator to make different trade-offs between blocking new resource requests and raising congestion prices to prevent overload.

In addition, we propose a pricing scheme in a DiffServ environment based on the cost of providing different levels of quality of service to different classes of users, based on long-term user demand, and based on the level of congestion in the network. The pricing scheme developed allows more efficient network usage. We also develop

the demand behavior of adaptive users based on a physically reasonable user utility function. We show how a set of user applications performing a given task (for example, a video conference) can adapt their sending rate and quality of service requests to the network in response to changes in service prices and subject to budget and minimum quality requirements, so as to maximize the benefit to the user.

We evaluate two mechanisms for determining the congestion-sensitive price, the first based on tâtonnement, and the second based on auction. In the tâtonnement process, the price is adjusted gradually to drive the user demand to the supply level of the network resources. In an auction process, the resources are allocated to the users that value the resource the most. Auction appears to be a useful concept for determining actual market prices in case of network congestion. However, the auction schemes proposed in the literature tend to be theoretical and leave a lot of practical issues open. In this work, we propose a new auction model, the  $M$ -bid auction, in which a user sends multiple bids at a time indicating its willingness to pay a premium for different amounts of bandwidth during congestion. The network evaluates the bids to obtain a congestion price, and allocates bandwidth to the users based on their bids. We discuss problems of the auction framework, and attempt to resolve them.

Using RNAP and an extended version of an existing DiffServ implementation, we develop a simulation framework to compare the performance of a network supporting congestion-sensitive pricing and adaptive service negotiation to that of a network with a static pricing policy. We evaluate the system performance and perceived user benefit (or value-for-money) under the dynamic and static systems. We also study the relative effects on system performance of rate adaptation, network load and traffic burstiness, dynamic load balancing between service classes and ad-

mission control. We simulate the tâtonnement-based congestion pricing model and the auction-based model, and compare their performance with respect to network utilization, connection blocking rate, user satisfaction and network revenue. We also study the stability of the dynamic pricing and service negotiation mechanisms. Although the simulation framework is based on the RNAP model, we try to derive results and conclusions applicable to static and congestion-driven, dynamic pricing schemes in general.

The proposed framework is shown to be able to control congestion and allow a service class to meet its performance assurances under large or bursty offered loads. Users see a reasonably stable service price and are able to maintain a very stable expenditure. Compared with simple admission control schemes, the framework developed leads to a greatly reduced connection blocking rate at high loads by driving down bandwidth requests of adaptive applications, resulting in a higher overall user satisfaction.

We also describe a prototype implementation of the resource negotiation framework on a test-bed network. The results confirm the performance benefits predicted by the simulations.

## **2.4. Organization**

The work on resource negotiation, pricing and QoS adaptation is organized as follows. We first describe some related work in Chapter 3. In Chapter 4, we discuss in detail the RNAP protocol and architectures, the roles of various RNAP agents, and the price and charge formulation and collation mechanisms of RNAP. Chapter 5 discusses our proposed pricing model in detail. Chapter 6 develops a physically realistic user utility function to represent user demand behavior in response to price changes. In Chapter 7, we describe our simulation model and discuss the simulation

results. We describe our prototype implementation of the proposed framework in Chapter 8, and summarize our work in Chapter 9.

## Chapter 3

### Background and Related Work

#### 3.1. Signaling for Resource Reservation and Allocation

IntServ and DiffServ are two basic models for providing QoS support in the current Internet. Although DiffServ is more scalable than IntServ in terms of implementation, services provided with existing DiffServ solutions usually have lower flexibility, utilization and assurance levels than IntServ services. In both architectures, implementations should include a mechanism by which the user can request specific network services, and thus acquire network resources. Any services with quality of service expectation also need admission control, where the network admission control module will grant or reject the user requests based on the network resources and network policies [20, 21]. Currently, resource reservation supported by both the services is normally static and long lived.

##### 3.1.1 Integrated Services (IntServ) and the Signaling

IntServ was proposed to be an Internet model that includes best-effort service, real-time service, and controlled link sharing. Currently two types of service are proposed in IntServ: guaranteed and controlled-load service.

Controlled-load service provides the client data flow with a quality of service

closely approximating the QoS that the same flow would receive from an unloaded network element, but uses capacity (admission) control to assure that this service is received even when the network element is overloaded. The controlled load service does not accept or make use of specific target values for control parameters such as delay or loss.

Guaranteed service (guaranteed delay and bandwidth) provides firm (mathematically provable) bounds on end-to-end datagram queuing delays. This service makes it possible to provide a service that guarantees both delay and bandwidth.

Per-flow resource reservation in IntServ is generally implemented through RSVP. RSVP was designed to enable the senders, receivers, and routers of communication sessions to communicate with each other in order to set up the necessary router state to support the services requested. There are two fundamental RSVP message types: **Path** and **Resv**. In the RSVP model, senders send the Path message towards the receivers following the data path, to install reverse routing state in RSVP-capable routers and provide the receivers with information about the characteristics of the sender traffic. In response, the receivers send the reservation requests (**Resv**) messages hop-by-hop following the reverse path formed by Path messages towards the sender. The two messages create and maintain “reservation state” in each node along the path (s). For robustness, RSVP takes a “soft state” approach to managing the reservation state in routers and hosts. RSVP soft state is created and periodically refreshed by Path and Resv messages and the state is deleted if no matching refresh messages arrive before the expiration of a “cleanup” timeout interval.

### 3.1.2 Differentiated Service (DiffServ) and Signaling

The DiffServ architectural model [22] provides multiple service classes with different QoS expectations. It improves the scalability of QoS provisioning by maintaining state information only at the edges of the network, and keeping classification and packet handling functions in the core network as simple as possible. Flows are classified, policed, marked and shaped at the edges of a DS domain. Based on this classification, the nodes at the core of the network forward the marked packets according to a corresponding *per-hop behavior* (PHB). The number of DS code points and the number of PHBs is limited and consequently this mechanism allows for a large number of individual flows to be aggregated from the point of view of the core router. Two kinds of PHB have been recommended: expedited forwarding (EF) [17] and assured forwarding (AF) [16].

The EF PHB is defined as a forwarding treatment for a particular diffserv aggregate where the departure rate of the aggregate's packets from any diffserv node must equal or exceed a configurable rate. It should average at least the configured rate when measured over any time interval equal to or longer than the time it takes to send an output link MTU sized packet at the configured rate. The EF PHB can be used to build a low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS domains. This service appears to the endpoints like a point-to-point connection or a "virtual leased line".

The AF PHB provides forwarding of IP packets in  $N$  independent AF classes. Within each AF class, an IP packet is assigned one of  $M$  different levels of drop precedence. Currently, four AF classes are defined, where each AF class in each DS node is allocated a certain amount of forwarding resources (buffer space and bandwidth). Within each AF class, IP packets are marked (again by the customer or the provider DS domain) with one of three possible drop precedence values. A

congested DS node tries to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value.

A Service Level Agreement (SLA) is a contract between a network service provider and a customer that specifies, usually in measurable terms, what services the network service provider will furnish. The customer can be an individual or a peer domain. A SLA provides a guarantee that traffic offered by the (peer) customer domain, that meets certain stated conditions, will be carried by the service provider domain to one or more appropriate egress points with one or more particular service levels. The guarantees may be quantitatively or qualitatively, may carry certain tariffs, and may also carry certain monetary or legal consequences if they are not met. An SLA may include PHBs to be applied, traffic conditioners and their parameters, and any applicable policies.

The Service Level Specification (SLS) contains the technical details of the agreement specified by the SLA. Specifically, the SLS asserts that traffic of a given class, meeting specific policing conditions, entering the domain on a given link, will be treated according to a particular PHB or set of PHBs. If the destination of the traffic is not in the receiving domain, the traffic will be passed on to another domain (which is on the path toward the destination according to the current routing table state) with which a similar (compatible and comparable) SLS exists specifying an equivalent set of PHBs.

Bandwidth Brokers (BBs) [12] are agents whose responsibilities include the implementation of the technical aspects of the agreements. In general, a bandwidth broker may receive a resource allocation request (RAR) from an element in the domain that the bandwidth broker controls or represents, or from a peer bandwidth broker. The bandwidth broker responds to this request with a confirmation of service or denial of service. This response is known as a Resource Allocation Answer



(RAA). If the RAR is admitted, traffic conforming to the RAR is handled according to the SLS between the provider and the customer. In response to the RAR, the BB may reconfigure routers in its domain, and possibly generate additional RAR messages requesting downstream resources. Implementation of the bandwidth broker for DiffServ is a subject of ongoing research, and various approaches have been proposed [23].

The signaling associated with BBs includes the user/application protocol, the intra-domain communication protocol and the inter-domain peer protocol. The user/application protocol is an interface provided for resource allocation requests from within the bandwidth broker's domain. These requests may be manual (e.g., via a web interface) or they may consist of messages from another setup protocol (for example RSVP messages). An intra-domain signaling protocol communicates BB decisions to routers within the bandwidth broker's domain in the form of router configuration parameters for QoS operation and (possibly) communication with the policy enforcement agent within the router. Current bandwidth broker implementations have a number of different protocols for communicating with routers, including COPS [24], DIAMETER [25], SNMP [26], and vendor command line interface commands. An inter-domain protocol provides a mechanism for peering BBs to request and send admission control decisions for aggregates and exchange traffic.

Full parameterization of SLSs is complex and is currently poorly understood, and there has been little work on SLS establishment and re-negotiation between customer and access domain bandwidth broker as well between neighboring domain bandwidth brokers. Some work has been done in the Qbone, an Internet2 initiative to build a testbed for new IP quality of service (QoS) technologies.

In the Phase 1 of Qbone implementation, a Simple Inter-domain Bandwidth Broker Signaling protocol (SIBBS) [23] was proposed. SIBBS is a sender-oriented,

request-response protocol between the bandwidth broker peers. A basic assumption of Phase 1 is that of a pure DiffServ environment, in which heterogeneous networks inter-operate at layer 3 and, specifically, achieve QoS inter-operability through DiffServ. The SLSs are assumed to have already been established (pairwise) between peer bandwidth brokers out-of-band, that is, without a SLS negotiation protocol. It is also assumed that there are globally well-known services (GWS) and service IDs (GWSID) referring to those services. The SLSs refer to these services and in addition, resource allocation requests use the well-known IDs. The BB handles end system requests for its domain, and may peer directly with non-adjacent BBs.

In the originating domain, the end system sends an RAR to the bandwidth broker. This message includes a globally well-known service ID and a destination IP address, a source IP address, an authentication field, times for which the service is requested and other parameters of the service. The bandwidth broker makes a number of decisions at this point, including request authorization, egress router location, admission decision for the flow based on the predefined SLS with neighboring successor domain en route to the destination, and domain policies. If these decisions all have a positive outcome, the bandwidth broker will modify the RAR by including the ID for the domain and sign the request with its own signature.

In a transit domain, the bandwidth broker receives an RAR from an adjacent bandwidth broker with a fully-specified destination address specification. The transit BB functions similar to the access domain BB, and also ensures that there are sufficient resources within the domain to support the flow from the ingress border router and possibly determine the intra-domain route. In case that all these decisions have positive outcomes, the transit bandwidth broker modifies the RAR as appropriate and sends it to the bandwidth broker of the following domain en route to the destination IP address.

In the destination domain, the bandwidth broker of the destination domain knows the address of the end system which is to receive the flow. As in the behavior just described, on the reception of the RAR, it checks that the requested resources fall within any possible SLS with the end system.

In case the decisions have negative outcomes in a domain, the bandwidth broker returns a Resource Allocation Answer (RAA) to the sending end system right away. If all these decisions have positive outcomes, the end system makes the determination whether it can receive the flow. This is signaled with an RAA to the bandwidth broker of the destination domain. This contains authentication of the end system, and parameters for the flow which the end system is willing to accept (which may be different from those received). In case the flow is rejected, the RAA contains a reason code and possibly hints about the set of service parameters that would be acceptable.

Upon receiving the RAA from the end system, the bandwidth broker authenticates the answer and forwards the RAA, with appropriate changes to the peer bandwidth broker that sent the RAR. Similarly, transit BBs en route to the originating domain forward the RAA towards the originating BB, which in turn forwards it to the requesting system. Along the way, a BB may modify traffic conditioners and PHB parameters in edge routers in response to the RAA, and may initiate resource allocation in internal routers.

The sender end system receives the RAA and is able to send the flow. The sender could send the flow earlier; however, the flow will not receive the requested service until the RAA is received and the DSCP of packets sent earlier than this will not be marked consistent with the service.

### 3.1.3 Other Work on Resource Allocation

Resource allocation schemes based on perceived-quality have been studied in [27][28][29]. These studies were limited to a local system, and did not address the interaction of the local system with a large network. Liao [30] allocates resources among users to achieve equal perceived quality. We will argue later that perceived quality does not directly represent the economic value of communications.

### 3.1.4 Conclusion

In general, RSVP and the implementations of DiffServ lack integrated mechanisms by which the user can select one out of a spectrum of services, and re-negotiate resource reservations dynamically. For efficient resource usage, the network should be able to commit resources for a short-term, and dynamically re-configure resources based on user demand and network conditions, particularly if users are adaptive. The protocols proposed currently also restrict the user requests to be either sender-driven (e.g., SIBBS) or receiver-driven (e.g., RSVP), while sometimes both senders and receivers would like to participate in the resource negotiation process. The protocols also do not integrate mechanisms to support pricing and billing, which are essential components of a multi-service framework.

## 3.2. Adaptive Internet Multimedia Applications

There has been a lot of recent research on adaptation of the sending rates of multimedia applications in response to available network resources [19]. In general, these adaptation algorithms rely on implicit signaling mechanisms such as packet loss rates for feedback.

Adaptive control schemes presented in the literature can be broadly classified into sender-driven, receiver-driven, and transcoder-based. Sender-driven schemes

require the sender to respond to fluctuations in the service available from the network, and adjust its transmission rate accordingly. Receiver-driven schemes specify a mechanism for each receiver to select transmission of a particular quality according to the service it receives from the network, typically by subscribing to different multicast groups. Transcoder-based schemes place gateways at appropriate locations to deliver different levels of quality to network regions with different types of connectivity or different levels of congestion.

### 3.2.1 Sender-driven Adaptation

Sender-driven adaptation schemes in the literature fall into two major categories: buffer-based adaptation schemes and loss based adaptation schemes. Adaptation schemes have been proposed based on other congestion indicators, including CSMA/CD collisions [31], packet delay [31] and delay jitter [32].

Buffer-based adaptation schemes [33][34][35] base the adaptation of the transmission rate on the occupancy of a buffer on the transmission path. Essentially, the goal of the control algorithm is to maintain buffer occupancy at a constant, desired level. When the buffer begins to fill up, the transmission rate is reduced in response, and when the buffer begins to empty, the transmission rate is increased.

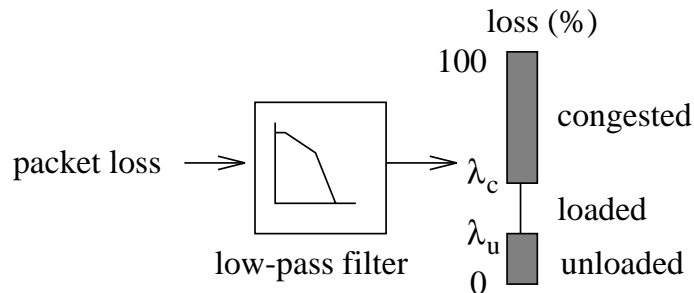


Figure 3-1: Linear regulator with dead zone (from [1])

Loss based adaptation schemes [36][1][37] regulate the transmission rate based on loss rate information reported by the receivers. Qualitatively, all three of the

adaptation schemes adopt the following approach (Figure 3-1). Based on feedback information from a receiver, the sender assumes that the receiver is in one of three states: *unloaded*, *loaded*, or *congested*. In the *unloaded* state, the sender progressively increases its transmission rate in an additive manner in response to feedback, until the network state is driven into the *loaded* state or the sender is sending the maximum useful rate. In the *loaded* state, the sender maintains a constant transmission rate. Depending on packet loss feedback, it can be driven into either the *unloaded* or the *congested* state. In the *congested* state, the sender progressively reduces its transmission rate multiplicatively until the reported loss decreases to the *loaded* state. Issues that need to be considered include the loss thresholds for determining a particular network state, and the parameters controlling the additive rate increase and multiplicative rate decrease. In a multicast environment with heterogeneously connected receivers, different receivers may experience widely varying degrees of congestion. The sender must deal with the problem of deciding upon an overall network state based on feedback from these receivers. The adaptation schemes take different approaches in tackling this problem.

### **3.2.2 Receiver-driven Adaptation**

In receiver-driven adaptation, receivers individually tune the received transmission according to their needs and capabilities. A number of receiver driven schemes use a combination of layered encoding, and a layered transmission scheme. The source data is encoded into a number of layers. A base layer provides the minimal QoS needed for an acceptable representation of the original data stream. Incrementally combining higher layers with the base layer results in a progressively higher QoS. Each encoded layer is transmitted to a separate multicast group. An alternative to this cumulative layering scheme is to encode and transmit multiple copies of the

source input; each copy is encoded to have a different level of QoS, and is sent to a separate multicast group. Although this approach (commonly referred to as simulcast) makes inefficient use of bandwidth, it may be more appropriate for layered transmission of audio using separate encoders for each layer, since audio encoders usually do not support layered encoding. Possibly due to the more demanding nature of distributed video applications relative to audio, layered adaptive schemes reported in literature generally use the former (cumulative layering) approach, and we restrict ourselves to this class of schemes in the present discussion.

The receiver selects a transmission quality appropriate to its requirements and constraints by subscribing to a certain number of multicast groups carrying different layers. The receiver monitors network congestion (based on parameters such as packet loss and throughput), and adapts to changes in network conditions by adding or dropping layers accordingly.

In the RLM scheme proposed by McCanne, Jacobson, and Vetterli [38], the sender takes no active role in the adaptation mechanism. It encodes the source signal into cumulative layers, and transmits each layer of the signal to a separate IP multicast group. Layered encoding of video usually results in a small number of high bandwidth layers. Adaptation by adding or dropping an encoded layer is of a correspondingly large granularity, and this may result in under-utilization of bandwidth, and sub-optimal quality of reception. One approach towards alleviating this problem is taken by the ThinStreams protocol [39]. An alternative approach is to have the source dynamically adjust the bandwidth of each encoded layer in response to feedback from the receivers or the network.

In the ThinStreams scheme, the granularity with which receivers may add/drop layers is decoupled from the granularity with which the source signal is encoded into layers. Each encoded layer at the sender is termed a thick stream, and this is split

up into several *thin streams* of a fixed, small bandwidth. Each thin stream is sent to a separate multicast group. A receiver drops and adds thin stream layers based on perceived network congestion.

Source adaptation schemes combine the layered encoding and transmission architecture with rate adaptation by the sender in response to feedback. Sisalem and Emanuel [40] propose an Adaptive Layered Transmission (ALT) protocol. The sender monitors loss information for each layer through periodic RTCP [41] receiver reports. The transmission rate of each layer is adapted using the additive increase/multiplicative decrease model. In addition to the rate adaptation by the sender, if a receiver experiences packet loss above a certain threshold, it drops a layer to avoid driving the transmission rate of the layer down too low. If the receiver determines that it has excess capacity, it adds a layer. If all the receivers drop the current highest layer, or if the transmission rate of the highest layer is reduced below the minimum transmission rate, the sender may choose to temporarily discontinue the layer.

In [42], Vickers, Albuquerque, and Suda propose a rate-based adaptation schemes, as well as a credit-based scheme (AMML). In response to receiver feedback, the sender decides the number of layers to encode, and the rate at which to transmit each layer. In the rate-based method, the sender receives feedback explicitly in the form of the desired transmission rate for each layer. The sender initiates the feedback process by multicasting a “forward feedback packet”. At each intermediate node, the ERICA algorithm [43] is used to calculate the fair share of link bandwidth of the connection, and this is entered in the explicit rate ( $R_E$ ) field of the forward feedback packet. When the packet reaches a receiver, the  $R_E$  field indicates the transmission rate the receiver’s connection can support. Based on this information, receivers send “backward feedback packets” to the sender requesting specific



transmission rates. Backward feedback packets are merged at intermediate nodes, concatenating the rate fields, and eliminating some if required according to specified criteria, so that the number of requested rates in a feedback packet does not exceed the number of layers the sender can support.

In the credit-based method, congestion feedback, as well as information about the number of receivers fully and partially receiving each video layer, propagates hop-by-hop back to the sender. The underlying principle is that an upstream node can send a certain number of packets to a downstream neighbor only if it has received an equal number of credits from the downstream node. The feedback packet eventually arriving at the sender indicates the total number of receivers fully and partially receiving each layer. The sender uses this information, as well as its buffer occupancy, to decide the number of video layers and the transmission rate of each layer.

### 3.2.3 Transcoder-based Adaptation

An alternative approach to layered encoding and transmission is to use *video* (or *multimedia*) *gateways* at appropriate locations in the network to convert through transcoding a high bandwidth transmission into a transmission with appropriate bandwidth to accommodate groups of poorly connected receivers. In addition to configuring a session appropriately with gateways during start-up, receivers may be allowed to adapt to network congestion by dynamically identifying and requesting service from a node with better reception to serve as the gateway. Alternatively, or in addition, the gateway may use an adaptive rate-control algorithm to adjust its transmission in response to receiver feedback. The two main considerations in developing a transcoder-based adaptation scheme are the design of the transcoding algorithm, and the placement or selection of the gateway to perform the transcoding.

In [2], Amir, McCanne and Zhang propose the following underlying model. The input format is converted into an intermediate representation by a decoder. This representation is transformed and delivered to the encoder, which produces a new bit stream in a new format (Figure 3-2). Multiple intermediate formats are supported by the proposed transcoding model, allowing flexibility in choosing an encoder/decoder pair, and optimizing performance by enabling a higher level of intermediate representation (such as DCT coefficients) to be used instead of decomposing the input stream into pixel format.

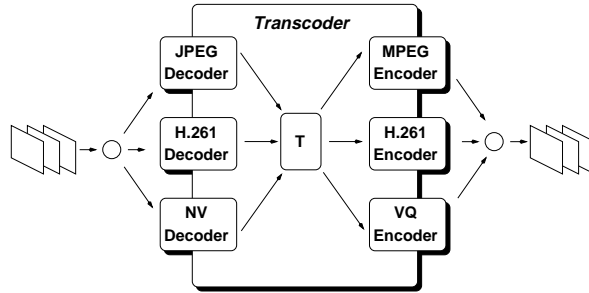


Figure 3-2: Block diagram of transcoder proposed by Amir, et al. [2]

Kouvelas, Hardman and Crowcroft [3] present a control scheme that automatically configures transcoders within the multicast tree to support branches with bad reception. A group of receivers affected by a bottleneck tries to locate an upstream receiver with better reception to provide a customized, transcoded version of the session stream by multicasting request messages. To prevent requests from multiple receivers in the group from proliferating, a requester delays its request by an interval proportional to its distance from the stream source plus a small random interval. If the requester receives an identical request during this delay, it cancels its own request.

### 3.2.4 Signaling Mechanism

A primary issue in rate adaptation schemes is the choice of the indicator which signals congestion in the network. The feedback signal may be explicit, or network assisted – for example, occupancy of a buffer along the transmission path, request for a specific transmission rate from the network or receivers, or source quench messages. The feedback signal may also be implicit, or sent directly from receiver to sender without network intervention – for example, packet loss, packet delay, delay jitter, throughput, or CSMA/CD collisions. The main criteria in evaluating the signaling mechanism are the promptness and reliability with which congestion is indicated.

- In the buffer based adaptation scheme of Kanakia, et al. [33], the sender requires state information about the bottleneck router. The state information propagates in either the forward or reverse direction, and each router along the transmission path either updates the state information, or passes it unmodified.
- In AMML [42], intermediate nodes are responsible for periodically collecting feedback messages from downstream routers, and merging the state information into a new feedback packet which is sent upstream. Additionally, in rate-based AMML, the sender multicasts a feedforward message; each switch implements a fairness algorithm (ERICA) to calculate bandwidth share for the connection, and updates state information in the feedforward message accordingly. In credit-based AMML, the switch maintains state information about the number of packets served at output links, and sends “credits” to the upstream router in the merged feedback message.

The adaptive scheme should detect and in react to congestion quickly, to minimize poor reception quality and interruptions at the receiver. At the same time, the adaptation should not result in abrupt changes or oscillations in the reception quality.

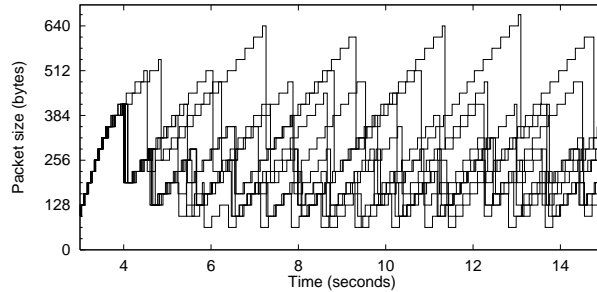


Figure 3-3: Adaptation of packet size in the TCP-like congestion control scheme used by Kouvelas, et al. [3]

### 3.2.5 Conclusion

The adaptive schemes which do TCP-like congestion control react to every packet loss by halving the rate. This can result in rapid changes in the transmission rate (for example, Figure 3-3) which may be perceived as unpleasantly abrupt changes in quality at the receiver. In the LDA scheme [37], a TCP-like congestion control mechanism is used in which the multiplicative rate reduction factor can be reduced to obtain a smoother change in the rate at the cost of a longer convergence time, but without sacrificing the goal of fairness to competing TCP connections. In the buffer-based scheme of Jacobs, et al. [34], abrupt changes may be smoothed out by the use of local buffering, but this is reported to add a delay of a few seconds, undesirable in real-time applications.

The loss-based, sender-driven schemes have relatively low overhead and simple implementations. In comparison, layered transmission schemes add complexity and delay to the encoding and decoding systems, and transcoder-based systems require

implementation of transcoding systems at routers, and have possible concerns about increased delay and security. At the same time, sender-driven schemes are limited in their ability to accommodate a heterogeneous group of receivers which differ in their connectivity or the amount of congestion on their delivery paths, since all receivers have to receive the same rate. They are better-suited to multicast sessions with a homogeneous group of receivers, distributed over a relatively small area. In larger, heterogeneous sessions, transcoder-based schemes and layered transmission schemes are better suited. Transcoder-based schemes may be preferred in a session which has diverse receivers with different connectivities. In an environment with dynamically changing congestion, layered transmission schemes may be preferable to avoid having to dynamically locate and configure transcoders.

In general, the multimedia adaptive schemes in the literature assume no QoS support. The frequent and passive rate adjustment can severely degrade multimedia quality, and sometimes can not guarantee that an application is able to maintain its minimum QoS requirement. The literature work also assumes the users are well-behaved and cooperative, and would adapt without any incentive.

### **3.3. Pricing and Billing in the Network**

Microeconomic principles has been applied to various network traffic management problems. In this section, we review some of the work.

#### **3.3.1 Total User Benefit Maximization based on Welfare Theory**

The studies in [44][27][45][46][47] are based on a maximization process to determine the optimal resource allocation such that the utility (a function that maps a resource amount to a satisfaction level) of a group of users is maximized. These approaches normally rely on a centralized optimization process, which does not scale. Also,

some of the algorithms assume some knowledge of the user's utility curves and truthful revelation by users of their utility curves, which may not be practical.

Kelly [48] showed that if users' utilities are concave functions of their attained throughput, then their aggregate utility is maximized by the network allocations of scarce network resources according to a weighted *proportional fairness* criterion, that is, the network shares resources in proportion to how much the users choose to pay. It was also shown in [47] that a weighted proportionally fair allocation could not be achieved by simple rate control algorithms, using increase and decrease rules similar to those described Chiu and Jain [49]. This work provides a sound theoretical analysis for the market-based network resource control. However, the implicit price calculation makes its application to real networks as an incentive to drive user traffic adaption an open question. The assumption of an extremely simple user utility function also restricts the model's application in real system.

### 3.3.2 Pricing for Efficient Resource Utilization and Congestion Control

In [50], packets are statistically marked similar to the ECN proposed for the Internet [51] and it was shown that the resulting system converges to a system optimal state as long as all utility curve are strictly concave. This model provides a simple pricing mechanism for packet transmission. However, the control is applied at the packet level and it is not clear whether the theoretical results hold in the presence of real timing and delay effects at the scale of a large network.

In [52][53][54][55][56], the resources are priced to reflect demand and supply. The pricing model in these approaches is usage-sensitive. The methods in [54][56][53] are limited by their reliance on a well-defined statistical model of source traffic, and are generally not intended to adapt to changing traffic demands.

The scheme presented in [55] is more similar to our work in that it takes into

account the network dynamics (session join or leave) and source traffic characteristics (VBR). It also allows different equilibrium prices over different time periods, depending on the user resource demand. However, congestion is only considered during admission control, and the study is restricted to single service class.

### 3.3.3 Pricing in Multi-service Environment

There is relatively little work on provider-oriented models for packet-switched multi-service networks.

Odlyzko [57] described an approach to Internet pricing, called *Paris Metro Pricing*, where logically separate networks each charge a fixed price per packet. The networks differ only in the price paid, and each user decides for each packet which price to pay. It is expected that the higher-priced classes will have less load and will provide better service. However, this approach provides no real service guarantee to the users, and there is nothing to prevent overloading of a service class.

Kumaran et al. [58] described the utility maximization by users and revenue optimization by service providers based on the quantitative admission control model proposed in [59][60]. The equivalent bandwidth estimation in [59][60] is under very specific assumptions about the traffic, and the analysis is also constrained to two classes, with users in the same class assumed to have the same bandwidth requirements. This prevents the application of the results to a general network service environment.

### 3.3.4 Auction Mechanisms

Several auction based mechanisms have been studied to elicit truthfully reported user utility functions and encourage the efficient utilization of scarce network resources. MacKie-Mason et al. [61] proposed implementation of congestion pricing

using a “smart market” model. In this model, the price for sending a packet varies on a very short time-scale (minute-by-minute), reflecting the current degree of network congestion. Each packet header contains a bid field, and the packet is admitted if the bid exceeds the current cutoff amount, determined by the marginal congestion costs imposed by the next additional packet. The user pays the cutoff amount, instead of her own bid. The optimal strategy for the user is to bid her true valuation. The mechanism only provides a priority relative to other users, and is not an absolute promise of service. Issues that need to be addressed include accounting complexity, user service interruptions during traffic peaks, and user response to fluctuations in price.

The model in [44] allows multiple QoS guarantees to be provided by scheduling resources in advance. The implementation scheme is again called “smart market”, and is actually a “generalized Vickrey auction” (GVA). The central idea in the well-known Vickrey auction [62] is to award the item on auction to the highest bidder, but charge the second highest bid as the price. Bidding one’s true valuation is a dominant strategy for each agent, and the resulting allocation is Pareto-efficient [62]. The generalized Vickrey auction extends the idea to allow agents to have preferences over more than one item, more than one unit of the item, and over the quantities of the items consumed by other agents (externalities). The model elicits the truthful revelation of user valuations, and preferentially allocates the scarce bandwidth to the users with the highest willingness to pay. The “second-price” analogue is to charge each agent the total social surplus that would be possible if that agent did not participate in the auction. The optimal solution requires substantial computation, which increases polynomially with the number of users, and the number of optimizations increases linearly with the number of users.

The “second-price” model is also used in [63], and the proposed auction scheme



is called progressive second price auction (PSP). The PSP model extends the traditional single non-divisible object auction to the allocations of arbitrary shares of the total available resource. Each user that receives resources pays the unit price of the maximum amount which the next highest bidder would pay if she is not denied.

A number of practical issues are not addressed in the above models. Other than computation and sorting complexity, potential problems include signaling bursts, set-up delay, and uncertainty of connection availability. Auctions taking place at intervals may cause signaling bursts around each auction moment. Set-up delays arise because new users have to wait until the next auction to receive requested resources. If the auction is implemented at multiple entities along a path to enable end-to-end reservation, the delay becomes cumulative. Since the refresh interval of current Internet reservation is normally of the order of tens of seconds, this would result in impractically long delays. Finally, if an auction is used for congestion control, the end-to-end connection will need to be refreshed from time to time based on the current traffic by performing new auction. At each refresh, a user risks losing its connection if its bid is rejected at any hop. Thus, the availability of the connection is always uncertain.

To address some of these concerns, Delta Auction (DA) [64] was proposed to allow auction to take place continuously. Arriving requests are processed immediately, and bids that are too low are refused right away. Sufficient bids are accepted provisionally, but there is the possibility that bids arriving later may exceed those that admitted and thus oust them from the auction. The advantage of this scheme is that the signaling traffic is distributed over time in a very uniform way, and a user is informed quickly about refused reservations.

A Connection-Holder-is-Preferred-Scheme (CHiPS) [65] was proposed to resolve the uncertainty of the connection. The auction mechanism is still used to deter-

mine a ranking among the bidders as well as the market-clearing price. Current connection-holders are preferred by allowing them to submit a second bid if their first bid is rejected. This still does not eliminate the uncertainty completely. It also appears to be inconsistent with the objective of the second-price model, which is to truthfully elicit the user valuation of a service, since the only choices for a users are either to withdraw, or to submit a second-chance bid with a price different from the user's real willingness to pay. If some users withdraw, other users may end up paying an unfairly higher price, compared to the price they would have paid if the withdrawals were taken into account at the outset. Similar to DA, CHiPS requires that resource auction take place regularly after each expiration of a given time period, and that reservations are possible and valid only for whole time periods. Moreover, each customer is allowed to send only one bid per time slot.

### **3.3.5 Signaling Support for Pricing and Charging**

Even though a fair amount of work is found in the literature addressing the resource optimization and pricing issues from an economic angle, little attention has been paid to the price quotation and collection mechanism, which becomes particularly necessary when the pricing is dynamic, based on the current network state.

In [66], a charging and payment scheme for RSVP-based QoS reservations is described. A significant difference from our work is the absence of an explicit price quotation mechanism — instead, the user accepts or rejects the estimated charge for a reservation request. Also, the scheme is coupled to a particular service environment (IntServ), whereas our goal is to develop a more flexible negotiation protocol usable with different service models.

### 3.3.6 Conclusion

In general, the work on network pricing in the literature is restricted to theoretical issues, and the results are not easily applicable to real networks. We design a pricing algorithm that takes into account the current Internet service infrastructure, and considers different service classes, the long-term user demand, and short-term network dynamics. The algorithm allows the network to optimize profit, and we also consider the user demand adaptation which would maximize the user benefit in such an environment. Also, the literature does not enter into detail about the negotiation process and the network architecture, and mechanisms for collecting and communicating locally computed prices. We address these drawbacks in our work by developing a resource negotiation framework with pricing and billing mechanisms. The framework is demonstrated in conjunction with our pricing model, but is general enough to be used with other models. We also propose an auction-based pricing model which minimizes the uncertainty of service availability inherent in most auction schemes, and addresses some of the other implementation issues.

## Chapter 4

# RNAP: A Resource Negotiation and Pricing Protocol

As growing numbers of “better-than-best-effort” (BBE) services are investigated and deployed in the Internet, user applications will be able to request and use the delivery service appropriate to their requirements. We may regard the selection and use of a specific delivery service as a negotiation process. The customer and network negotiate and agree upon specifications such as the type of service user packets will receive, the constraints the user traffic must adhere to, and the price to be charged for the service. We designed a Resource Negotiation And Pricing protocol, RNAP, which provides a framework through which this negotiation can take place.

The RNAP protocol is generic and flexible enough to support multiple delivery services and environments (including IntServ, DiffServ, and best effort services), service negotiation at different levels of granularity (flow- and aggregate-based), negotiation by both sender and receiver, and “in-band” and “out-of-band” resource reservation mechanisms. It allows the service provider to communicate service availability, estimated prices for available services and charges accruing to the user, and allows the user to request a specific service. It also supports dynamic service re-negotiation between the user and the network, allowing the network to adjust pricing

in response to changes in network load, and allowing the user to respond to changes in application requirements.

Based on the policy of each domain, different algorithms can be used for computation of a local or incremental price for a service at a given point in a network. RNAP as a signaling protocol does not depend on the actual pricing algorithms, and different domains can also have different pricing policies. The proposed protocol can be embedded in the current Internet QoS schemes (e.g., those proposed within the IntServ and DiffServ frameworks), or can be implemented stand alone. We present RNAP as a stand-alone protocol, but it is also possible to implement some components of RNAP as a layer on top of RSVP or other hop-by-hop reliable signaling protocols.

RNAP is intended for use by both adaptive and non-adaptive applications. Non-adaptive applications may choose services that offer a static price, or absorb any changes in price while maintaining their sending rate. Adaptive applications adapt their sending rate and/or choice of network services in response to changes in network service prices. RNAP provides a framework within which an application can adapt so as to obtain the best value from the network.

This chapter is organized as follows. We first describe the basic characteristics of the RNAP in Section 4.1. We then define two alternative protocol architectures, a centralized architecture (RNAP-C), and a distributed architecture (RNAP-D) in Section 4.2.2, followed by the introduction of the basic RNAP operations. We present a description of the basic RNAP protocol, including protocol objects and messages in Section 4.3. We then expand our discussion to the aggregation of RNAP messages in Section 4.4. In Section 4.5., we present the messaging scenario when a receiver initializes the negotiation. In Section 4.6., we discuss pricing and charging mechanisms in RNAP. The communication of pricing and charging information in

the various RNAP messages is discussed. In Section 4.7., we describe a scheme to locate the peering NRN address through DNS SRV [67]. Finally, in Section 4.8., we discuss some additional issues including transport protocol and reliability, the stability of the periodic negotiation mechanism, and security.

## **4.1. Basic Characteristics**

In this section, we introduce some basic characteristics of RNAP.

### **4.1.1 Dynamic Re-negotiation Capability**

Allowing customers to re-negotiate with the network dynamically and change their service requests during a session benefits both the customer and the network, particularly during network congestion.

In general, the network would like applications to acquire network resources so that there is high network utilization, but not at the expense of poor QoS. The real-time constraints of multimedia traffic make it difficult for these applications to estimate the bandwidth required for an application.

Also, many existing multimedia applications allow the media rate and quality to be adjusted over a wide range, allowing them to respond to network congestion by gracefully reducing their rate [19], possibly utilizing application-specific knowledge (a specific model for such user adaptation is discussed in Chapter 5). Such applications have the incentive to re-negotiate a service with lower QoS when network congestion results in the current service becoming more expensive, or if the network provider denies the requested service because of unavailability of the amount of resources requested.

The periodic nature of the RNAP message sequence provides a natural way for the customer and service provider to re-negotiate services. Possible re-negotiation

scenarios include periodic re-negotiation, in which the service contract expires after a period and is re-negotiated, and asynchronous re-negotiation initiated either by the customer or by the network provider. If the customer is an end-user running adaptive applications, these applications are able to respond to network congestion and also to changes in application task requirements by adjusting their sending rates, and hence network resource requirements. If the customer is an adjoining domain requesting network services for an aggregate of flows belonging to its customers, it would like to be able to re-negotiate its request if the size of the flow-aggregate changes significantly. Dynamic re-negotiation enables the service providing network to maintain a high network utilization without QoS degradation.

#### **4.1.2 Pricing and Charging Capability**

A network service model that provides a choice of delivery services must also incorporate a pricing system, so that users are charged appropriately for different levels of service. Usage and congestion sensitive pricing also signals network congestion to the customer. In combination with a periodic or dynamic service re-negotiation capability and adaptation-capable applications, it provides a congestion-control mechanism.

A pricing system should include monitoring of user traffic, price formulation at one or more points within the network, computation of a global, or end-to-end, price for a particular service, and a mechanism to communicate pricing information from the network to the customer. The RNAP protocol provides the means to communicate price quotations for different services and the charge rendered to the customer. It also supports different charging modes: charging the sender, or receiver, or both. The periodic re-negotiation framework provides a natural way to communicate periodic price quotations and cumulative charges to the customer.

### 4.1.3 Scalability

We envision that individual customer flows will be gradually aggregated towards the core of the network, and RNAP sessions will correspond to progressively larger granularity flows. We discuss a sink-tree based aggregation mechanism in Section 4.4..

### 4.1.4 Service Predictability

Predictability includes the quality expected from a service type, and the price charged for it. Each particular delivery service model assures different level of quality expectation by the user. The periodic price quotation mechanism of RNAP provides pricing predictability by keeping the price constant during a negotiation period.

## 4.2. Basic Model

RNAP supports service negotiation between the customer and network service provider as well as negotiation between two network domains. We represent the negotiating processes by two logical entities, a Host Resource Negotiator (HRN) that resides on the host machine, and a Network Resource Negotiator (NRN) that either resides on each network router or controls resources for an entire domain.

The HRN negotiates only with its access network to reserve resources, even if its flows traverse multiple domains. It obtains information and price quotations for available services from the network. It requests particular services, specifying the type of service (guaranteed [13], controlled load [14] (CL), expedited forwarding [17], assured forwarding [16], best effort, etc.), parameters to characterize the user traffic (e.g., peak rate, average rate and burst size) and QoS requirements (e.g., loss rate and delay). A customer (sender or receiver) may reserve network resources for



multiple flows, for example, flows corresponding to audio, video and white-board applications in a video-conference. In this case, the HRN can request a different service for each flow from the network through RNAP.

### 4.2.1 The RNAP Architecture

For negotiations by the network service provider, RNAP supports two alternative architectures, a centralized architecture, and a distributed architecture, described below.

#### 4.2.1.1 Centralized Architecture (RNAP-C)

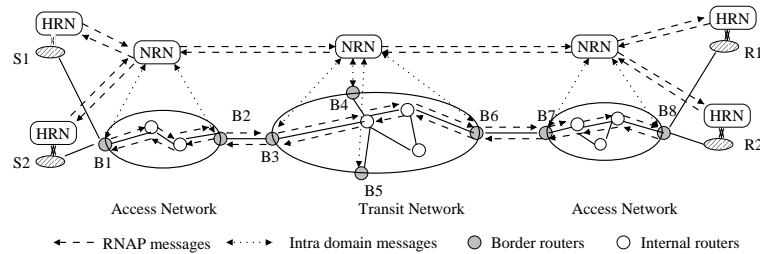


Figure 4-1: RNAP-C architecture

The RNAP-C architecture is based on an underlying network divided into Autonomous Systems (AS), and each domain negotiates through an NRN (Fig. 4-1). Protocol messages are sent between NRNs, or between HRNs and NRNs, and touch each AS once.

The NRN delivers price quotations for the different available service levels to customers, answers service requests from customers, and is also responsible for maintaining and communicating charges for a customer session.

The NRN may be an individual entity, or may be a complementary functional unit that works with other administrative entities. For example, the NRN can be part of (or function as) the Bandwidth Broker (BB) in the DiffServ model [12] and the PDP in the COPS architecture [24]. The NRN either has a well-known address,

or is located via the service location protocol (SLP) [68] or DNS SRV [67]. The NRN address of a neighboring domain can be pre-configured or obtained through DNS SRV. NRN location is described in more detail in Section 4.7..

Resource reservation and admission decisions may be performed by the NRN or by other entities, such as the BB of the DiffServ model. If they are performed by an NRN, the NRN will access intra-domain routing information to determine the path and therefore resource allocation and pricing information. The NRN will also store or have access to customer state information, and network administration information such as router configurations, current resource allocations, policy information, network statistics, and authentication information. Intra domain signaling, such as COPS can be used to send network statistics from domain entities to the NRN.

If reservation and admission decisions are performed by other entities, the NRN communicates requests for services to them individually or in aggregate, and receives admission and pricing decisions from them. The implementation of resource reservation and admission control, and the associated communication with administrative entities, is closely related to specific better than best effort (BBE) services, and is outside the scope of the RNAP protocol.

#### **4.2.1.2 Distributed Architecture (RNAP-D)**

In this architecture (Fig. 4-2), networks do not have a single negotiating entity. Instead, the RNAP protocol is implemented at each router, in the form of a Local Resource Negotiator (LRN). RNAP messages propagate hop-by-hop along the same path as customer data flows. We consider the messaging process in greater detail in Section 4.3.3.

The RNAP message format is independent of the architecture. Therefore, the two architectures can co-exist. For instance, a domain administered by a centralized

NRN can exchange RNAP messages with a neighboring domain which employs the distributed architecture. A HRN needs to know about the RNAP architecture of its local domain to decide where to send the messages to, but it receives and sends the same negotiation messages in either case.

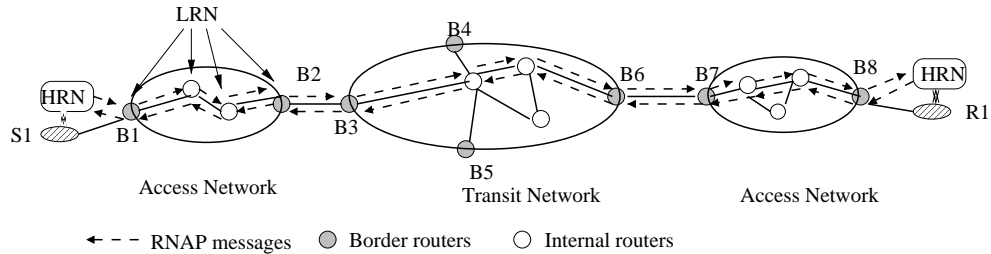


Figure 4-2: RNAP-D architecture

#### 4.2.2 Protocol Operation

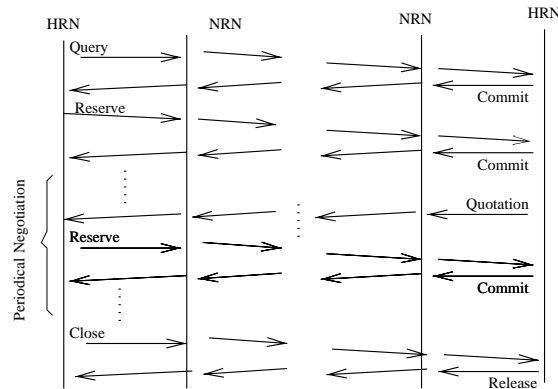


Figure 4-3: RNAP messaging sequence between HRN and NRN.

The basic RNAP message sequence is shown in Fig. 4-3. Typically, the sequence of Fig. 4-3 repeats periodically, with a pre-defined negotiation interval. This allows the protocol to maintain soft-state - state information that expires in the absence of any RNAP Reserve message. It also allows the customer and the network to easily re-negotiate services.

A network domain manages its own pricing scheme, which may be congestion sensitive or static, independent of other domains, and will have its own per-unit resource costs for each class. When a user flow traverses multiple domains, RNAP messaging collates pricing and billing information from each domain and determine the total price and charge for the user.

We first consider how a customer reserves resources for a flow **end-to-end**, to a particular destination address, assuming that the intervening domains implement RNAP-D. In general, the forwarding of RNAP messages in RNAP-D is similar to RSVP forwarding [18]. The router alert option is turned on in the RNAP message by the HRN, so that intermediate LRNs can intercept and process the message before forwarding it to the next-hop LRN.

1. The HRN sends a Query message towards the destination, requesting a price quotation from all the LRNs along the path for one or more services, for a flow or group of flows belonging to the customer. The HRN specifies a set of requirements (such as service time and QoS) with each service.

The destination initiates a Quotation message and sends it upstream towards the HRN following the recorded path of Query message. Each intermediate LRN verifies local availability of each service, and increments the price by the local price that it computes. When a Query message does not specify any service, the Quotation message contains pricing information for all available services using default values of unspecified service parameters. In addition to send Quotation messages in response to queries, the destination also sends out Quotation messages periodically, containing price quotations for all services requested by the customer.

2. The HRN sends a Reserve message towards the data destination to apply for services with specified service parameters for a flow or group of flows. A

Reserve message is sent at the beginning of a session to request services for the first time, and thereafter, periodically or by trigger to renew or change existing reservations.

In response to a Reserve message, the destination initiates a Commit message stating the admissibility of the flow. The Reserve request may be admitted or denied, or admitted partially if network resources are scarce and the provider admits the service request with a lower QoS or sending rate than requested. If the flows are admitted, the Commit message also contains a local price for the contracted service. For an on-going session, it also contains the accumulated local charge for a service. As the Commit message is forwarded upstream, the committed price and accumulated charge are incremented at each router.

Upon receiving an updated Quotation message periodically, the HRN renegotiates resources by sending Reserve messages and receiving Commit messages in return.

3. To terminate a session, the HRN sends a Close message, which is forwarded to the receiver. The receiver sends a Release message upstream, and delete the session states. Upstream routers forward the Release message towards the HRN and release the resources.

When a customer flow traverses a domain implementing RNAP-C, with a controlling NRN, the flow of messages is identical to that considered earlier for RNAP-D, if each domain is considered to be equivalent to a single node, with the NRN corresponding to the LRN for that node. Accordingly, the NRN is responsible for collecting and communicating admission and pricing and charging information for the domain as a whole instead of for a single node (mechanisms for doing this are discussed in Section 5.1.). The forwarding mechanism in RNAP-C is different from

RNAP-D. To forward a RNAP message downstream towards the destination HRN, the sending HRN or the intermediate domain NRN needs to find the targeted NRN address.

It is also possible that the flow traverses multiple domains some of which implement RNAP-C and others RNAP-D. In this case, the NRN of a RNAP-C domain would talk to the corresponding boundary LRN of an adjoining RNAP-D domain, and the messaging flow would be as before.

### 4.3. The Protocol

In this section, we start with an explanation of some basic objects used in the protocol messages, followed by a description of the protocol messages, and the typical negotiation sequence in which they are used. In the discussion that follows, we assume for convenience the RNAP-C architecture, and refer to the NRN as one of the negotiating entities. We later extend the discussion to the RNAP-D architecture, with the routers along the delivery flow path collectively playing the role of the NRN. Also for convenience, we assume that the other negotiating entity is a HRN, acting on behalf of the user application. As mentioned earlier, the RNAP protocol is also applicable to resource negotiation between two network domains, in which case, the first domain (through its NRN, in case RNAP-C is employed) plays the role of the HRN.

#### 4.3.1 Basic Objects

**Id:** The Id object contains two sub-fields: Flow Id, and Aggregate Flow Id. The Flow Id defines a flow for which services are negotiated. For individual flows, the Flow Id is based on one or a combination of destination IP address and port, source address and port, and transport protocol. The port number

may be null when only a subnet address is used. For aggregates of flows, it is based on the destination subnet addresses, determined by the HRN by a method such as RADB lookup [69]. The Aggregate Flow Id field is optional and is attached only if the messages is indicated to contain aggregation flow information. The purpose of the Aggregate Flow Id sub-fields is explained when we discuss message aggregation, in Section 4.4..

**RNAP\_Hop:** This object carries the IP address of the interface through which the last RNAP-knowledgeable hop (a node or an NRN) forwarded this message. The Logical Interface Handle (LIH) is used to distinguish logical outgoing interfaces. A node receiving an LIH in a Query message (or Quotation message when a receiver initiates the negotiation) saves its value and returns it in the *RNAP\_HOP* objects of subsequent messages sent to the node that originated the LIH. The LIH should be zero if there is no logical interface handle.

**Sender\_Identifier:** When an RNAP message is sent between domains, the sender domain may need to include authentication information for resource negotiation and accounting at the peer receiving domain. This information is contained in the Sender Id and Sender Signature fields of the Sender\_Identifier object.

**Service:** The *Service* object describes the service being negotiated. The HRN (or NRN for negotiation between domains) uses it to request a price quotation or reserve resources for a particular service with a set of associated parameters. The NRN uses it in the corresponding acknowledgment messages. A Service object consists of Service Independent Parameters (SIP), Service Specific Parameters (SSP), Service State, and Price objects. Multiple SSP-Price pairs or SSP-Service State pairs or SSP-Service State-Price triplets may be carried by

a RNAP message to convey price and service statistics.

### *Service Independent Parameters (SIP)*

*SIP* specifies a list of parameters that are generic to all service models and used to characterize a service. The service independent parameters include *Number of Services*, *Negotiation Capability*, *Preemption Capability*, *Reservation Coverage*, *Negotiation Interval*, *Expiration Action*, *Start Time*, and *End Time*.

#### Number of Services:

Specify how many different services are attached in the Service object. Some RNAP messages, such as a Quotation message, may carry multiple service information to allow an end user to select the best service each time. In this case, multiple SSP and Price pairs will follow the SIP field.

#### Negotiation Capability:

The Negotiation Capability flag is used by the HRN (or NRN) to signal its capability or willingness to negotiate. If the Negotiation Capability bit is set, service will be negotiated periodically between a HRN and NRN, or neighboring domain NRNs. Otherwise, the service confirmed will be effective until it is cancelled specifically.

#### Preemption Capability:

Preemption Capability defines whether the service is pre-emptable or non-pre-emptable. A non-pre-emptable service assures service to the user for the negotiated period. A pre-emptable service is subject to being terminated by the NRN asynchronously, instead of being allowed to expire at the end of a negotiation interval. For specific services, further refinements may be considered. For example, instead of all the reserved



resources being “at risk”, resources reserved above a certain base level, or just the cost of reservation may be “at risk”. These particulars can be negotiated and indicated in the Service Specific Parameter fields.

#### Reservation Coverage:

Reservation Coverage indicates the extent of reservation over the flow paths. The reservation can be end-to-end, over contiguous sub-trees where branches may not use or support reservations or for discontinuous segments. In the latter case, referred to as partial reservation, reservations may fail on a link, yet the resource reservation request will not be automatically removed for the remaining links.

#### Negotiation Interval:

For a client that would like to negotiate with the network periodically by setting the Negotiation Capability, the Negotiation Interval (in seconds) defines the length of time over which the negotiated service and price are valid. The negotiated service expires automatically at the end of the negotiation interval, and the HRN must periodically re-negotiate (by sending a Reserve message) before the expiration to ensure uninterrupted service. Specific services may define different actions on part of the provider regarding the treatment of user packets after the service has expired.

To reduce the signaling overhead, the negotiation interval for a service can be set equal to a multiple of time periods associated with an underlying protocol, for example, the TCP round-trip time, RSVP [18] refresh time, or RTCP [41] receiver report interval. To reduce control overhead, a minimum negotiation interval should be enforced. Multimedia services should not renegotiate too frequently, to avoid adjusting data rate too

often resulting in poor perceived quality.

The negotiation interval affects how a service is priced. A service with a longer negotiation interval will generally be charged more to protect against future congestion price increases.

#### Start Time and End Time:

Start Time and End Time specify the time period over which service is requested. The Start Time and End Time fields are optional and may be used to make either immediate or advance reservations. When immediate service is required, the Start Time and End Time fields are set to zero.

#### Expiration Action:

Re-negotiation requests may get lost, or a customer may choose not to re-negotiate even though it signs for the negotiation service with the benefit of lower average cost. When the negotiation period expires, possible actions are: maintaining the current service at the previously negotiated price, maintaining the current service but updating the price unilaterally as required, transmitting using best effort service, or halting the service.

#### *Service Specific Parameters (SSP)*

*SSP* consists a service type field, and a list of parameters used to characterize a service, specific to a particular service type. Examples of services are the Guaranteed and Controlled Load service models defined within the IntServ framework [13][14], and the Expedited Forwarding (EF) and Assured Forwarding (AF) models defined within the DiffServ framework [17][16]. The service type definition for services belonging to IntServ is the same as the service number defined in IntServ [70][71].

The service type definition for services belonging to DiffServ is the same as that defined for the Differentiated Services Field (DS Field) [72]. Typical service parameters define the traffic profile the user traffic should adhere to, such as average rate, peak rate, burst length, and parameters specifying the QoS provided by the service such as loss rate and maximum delay. For some services such as those belonging to DiffServ, the performance requested from a class may be in terms of a qualitative expectation. For example, service using EF PHB may be expected to have lower average loss, delay and jitter. The contents and format for *SSP* are specified in documents prepared by the IntServ working group, and *DiffServ* working group.

#### *Price*

The *Price* field contains two sub-fields: *NewPrice* and *Accumulated Charge*. The *NewPrice* carries the current price being quoted by the service provider for a service, and the *Accumulated Charge* carries accumulated charges corresponding to a particular customer session.

#### *Service State*

Service State is an optional field that can be used by the network provider to provide service statistics to the customer. Typical service statistics would be average packet loss rate, average queueing delay, and maximum queueing delay. The length of this field is variable, depending on the specific statistics being provided.

**HRN\_Data:** The HRN\_Data object contains information about the agent from which the message originates. The message originating agent is usually the negotiating HRN, but we will also discuss a situation in which the partner or

peer HRN originates the message.

**Status:** The Status object is carried by Commit and RNAPError messages. It has three values: Admit, Admit\_Incomplete, and Reject. It is used in Commit message to convey admission/denial decisions in response to Reserve messages. If a negotiation request is Admit\_Incomplete or Reject, the reasons will be indicated in the reason field. If the request is partially admitted, that is, admitted with one or more of the requested parameters, the modifications are indicated in the appropriate SSP/SIP fields. RNAPError messages always have a Status value of Reject.

### 4.3.2 Message Forwarding Mechanisms

Before we describe in details the transmission and processing of RNAP messages, we introduce the RNAP message forwarding mechanism in this section.

Forward messages (e.g., Query, Reserve, Close) are forwarded differently depending on whether RNAP-C or RNAP-D is used. For RNAP-C, the messages will be sent towards the destination domain and touch each AS once. The IP source address is the address of the sending RNAP node, either HRN or NRN, while the destination address must be the peer downstream domain NRN address. The determination of the neighboring domain NRN address will be described in Section 4.7.. If authentication is required, a sending domain NRN will insert its *Sender Id*, and a receiving domain NRN will authenticate the request.

For RNAP-D, the messages travel from a sender to receiver(s) along the same path(s) used by the data packets. The IP source address of a message in forward direction must be the address of the origin HRN, while the destination address must be the destination HRN. These addresses assure that the message will be correctly routed through a non-RNAP cloud. In either case, each RNAP node

stores the previous hop. For RNAP-D, this information needs to be recorded in the *RNAP\_Hop* field of when the RNAP message is passed from the previous hop.

Reverse messages (e.g., Quotation, Commit, Release) need to visit the same nodes as the forward messages and thus are sent hop-by-hop. Each node keeps a table of all RNAP sessions and their previous hop IP address. Reverse RNAP messages thus always use the IP address of the next hop as their IP destination address.

### 4.3.3 RNAP Message Format

RNAP supports service negotiation from sender, receiver, or both. In this section, we assume the negotiation is initiated by the sender side, and we will explain the receiver negotiation in Section 4.5.. We now describe the RNAP negotiation messages, with some explanation of the sequence in which they are used. The negotiation sequence is represented schematically in Fig. 4-3.

#### 4.3.3.1 Query Messages

The HRN uses Query messages to request a price quotation and/or quality statistics from the NRN for one or more services. The HRN will inform the NRN whether the HRN supports negotiation by setting the *Service.Negotiation Capability* bit appropriately. The HRN specifies a set of requirements with each service, by setting some or all of the *SIP* and *SSP* parameters in the corresponding service identifiers.

The HRN sends a Query message to the first-hop Local Resource Negotiator (FHL). The FHL forwards the Query message downstream to the last-hop Local Resource Negotiator (LHL), and the LHL forwards the message to the destination HRN.

Each RNAP-capable node or NRN along the path(s) captures a Query message and processes it to create RNAP state for the sender (HRN or NRN) identified *Id*

objects. The RNAP state will store the the unicast IP address of the previous hop or NRN and use it to route the messages hop-by-hop in the reverse direction. The IP address for RNAP-D is gained from the *RNAP\_Hop* object. Any *Service* objects are also saved in the RNAP state, to allow the proper service statistics or pricing information to be associated with a session.

For getting the price quote, a request may indicate the required service type only, without specifying the traffic format and amount of resource needed. Furthermore, the *Service* field can be set to NULL. A Query message with a null service list is interpreted by the NRN as a request for price quotations for all available services, for each *Id* specified in the Query message.

#### 4.3.3.2 Quotation Messages

We first consider the message flow in RNAP-D. Upon receiving a Query message, the receiver initiates a Quotation message and sends it hop-by-hop upstream. The IP destination address of a Quotation message is the unicast address of a previous-hop node, obtained from the RNAP state. The IP source address is an address of the node that sent the message. If price quote is requested by the HRN, each intermediate LRN verifies local availability of each service, and increments the price by the local price that it computes. If network service statistics is requested, the LHL and each intermediate LRN will attach the service statistics for the service type and set the corresponding parameters. The FHL returns the Quotation message to HRN. If periodical negotiation is requested, periodic Quotation messages are also sent by the receiver hop-by-hop upstream, as above.

For a customer flow traversing multiple network domains that implements RNAP-C and with a controlling NRN, the flow of messages is identical to that considered earlier for RNAP-D. If each domain is considered to be equivalent to a single node,

with the NRN corresponding to the LRN for that node, the NRN is responsible for collecting and communicating admission and pricing and charging information for the domain as a whole instead of for a single node. If the pricing scheme used by the domain is path dependent, the path between an egress and an ingress will be located through the domain routing table. The accumulated price along the path will be used as the domain price. The IP destination address of a Quotation message is the unicast address of a previous domain NRN, or HRN obtained from the RNAP state. The IP source address is the NRN address of the domain that sends the message.

In general, an NRN sends a Quotation message upon receiving a Query message, and upon expiry of the session quotation timer. The timer is reset whenever a new Quotation message is sent out, either synchronously, or asynchronously. A Quotation message sent periodically contains price quotations for all services requested by the customer. When a Query message has null Service list, the NRN returns quotations for all available services using default values of unspecified service parameters. It does not return quotations for services which have one or more mandatory parameters to be specified by the customer.

If the *Negotiation Capability* flag is false, the HRN and NRN could still exchange an initial pair of Query and Quotation messages, and negotiate a service with a set of parameters that remains unchanged for the rest of the session. A HRN may re-enable negotiation capability at any time during the session by sending a Query or Reserve message.

#### **4.3.3.3 Reserve Messages**

For any service that requires admission control, the HRN sends a Reserve message to apply for services with specified service parameters for a flow. A HRN makes decision on the service request based on the price quotation and user requirements

and willingness to pay. A Reserve message is sent at the beginning of a session to request services for the first time, and thereafter, periodically at negotiation intervals, and also asynchronously at any time. The parameters of the *Service* may be modified at each re-negotiation period.

The HRN applies for resources for a particular session through a corresponding service identifier, specifying the type of service, and a set of SIP and SSP parameters characterizing the user requirements from that service. As mentioned previously, some parameters may be mandatory for a particular service (typically parameters such as peak and average sending rates). In general, each Reserve message carries one *Id-Service* pair.

The Reserve message must be routed along the same path as the corresponding Query message. At each RNAP-knowledgeable hop, the Reserve request is authenticated, and its state information looked up by matching on the *Id* and *RNAP\_Hop* objects. If resources are available, the request is admitted, the corresponding state is updated, and the Reserve is forwarded to the next hop. If the Reserve request can not be accomplished at any hop, a Commit message with status *Reject* (as will be seen in Section 4.3.3.4) is sent back to the HRN. If block reservation is set between domains in advance as discussed in Section 4.4.5, the reservation request is authenticated to determine if it can use part of the reserved resource-block.

#### 4.3.3.4 Commit Messages

The Commit message is generated by the receiver in response to a Reserve message, and consists of *Id*, *Service* and *Status* triples. For each *Id*, the LRN/NRN determines the admissibility of the session, and returns this information in the *Status* field, as *Admit\_Complete*, *Admit\_Incomplete* or *Reject*. The admission policy is specific to the service, and need not be administered by the NRN. For instance, in a DiffServ



service, the bandwidth broker (BB) could make the admission decision, and the NRN simply communicates the admission decision through RNAP between the domains or between the access domain NRN and the HRN.

If the flow is admitted, the LRN/NRN returns the price for providing the service in the price parameters of the *Service* object (normally the same as the price quoted in the preceding *Quotation* message). If the Commit is in response to a re-negotiation Reserve request in an ongoing session, the LRN/NRN also returns the amount charged for each service in the preceding negotiation period, and the accumulated charge since the beginning of the session.

The *Status* is *Admit\_Incomplete* if network resources are scarce and the provider admits the service request with a lower QoS or sending rate than requested. The reduced QoS parameters or the sending rate will be set in the corresponding fields of the *Service* object.

If the flow is rejected at a point in the path, the Commit message will be sent back to the sender immediately. If partial reservation is indicated in the *Reservation Coverage* field of the Service object, the completed reservation before the failure node will be kept. Otherwise, the reservation state will be cancelled as the Commit message passes by.

#### 4.3.3.5 Preempt Messages

If a *Service* is set as preemptable (at the benefit of lower price), the NRN may preempt resources allocated previously to this service and make room for the other more important flows. Currently the *Preempt* field in the *Service* identifier is binary, i.e., preemptable or non-preemptable. More preemption priorities could be supported and allow different flows with different priority levels to be differentiated.

#### **4.3.3.6 Close Messages**

A Close message is sent from the HRN to the NRN to tear down the negotiation session between them.

#### **4.3.3.7 Release Messages**

The Release message acknowledges the Close message and optionally reports to HRN the cumulative charging information for the entire session. This information is for informational purposes, and may not be tied to the actual billing and payment procedures. The NRN releases the resources it had allocated for the session, and sends a Release message.

#### **4.3.3.8 RNAPError Messages**

RNAPError messages will be triggered when processing an RNAP message. If a error come from the processing of Query, Reserve or Close messages, the corresponding RNAPError messages are simply sent upstream to the sender that created the error, and they do not change RNAP state in the nodes through which they pass. At each hop, the IP destination address is the unicast address of a previous hop (a router or an NRN).

If an error arises during the processing of Quotation, Commit or Release, RNAPError messages travel downstream towards the appropriate receivers. At each hop, the IP destination address is the unicast address of a next-hop node or NRN.

### **4.4. State Aggregation**

If end-to-end RNAP reservation is carried out for each customer flow, RNAP agents in the core network may potentially need to process RNAP messages for hundreds

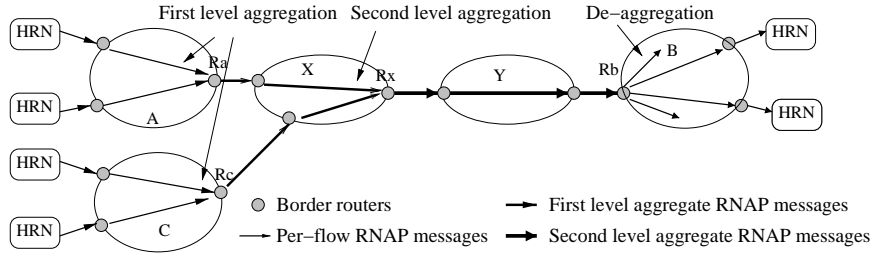


Figure 4-4: Example of RNAP-D message aggregation.

of thousands of flows, and maintain state information for each of them. In this section, we discuss how RNAP messages can be aggregated in the core of the network by allowing RNAP agents to handle reservations for flow-aggregates instead of individual flows.

#### 4.4.1 Overview of the Aggregation Scheme of RNAP

All RNAP messages have an *Id* field identifying the corresponding data flow; it contains three sub-fields: *Flow Id*, *Aggregation Flag*, and *Aggregate Flow Id*. The merging point aggregates RNAP messages for user flows which request the same or similar services and have similar negotiation intervals.

We consider the aggregation of RNAP messages belonging to senders sharing the same destination network address, forming a “sink tree”. Sink tree based aggregation has also been discussed in [73][74].

RNAP messages will be merged by the source domain and split again for each individual HRN at the border router (for RNAP-D) or NRN (for RNAP-C) of the destination domain. The merging point in the HRNs home network forwards two messages: one that travels directly to the destination network, without visiting any of the RNAP agents in between, and an aggregated-resource message that reserves resources and collects prices in the “middle” of the network.

The merged resource message requests resources equal to the sum of all the branch resource requests further up in the sink tree. At each merging point, up-

stream flow arrivals, departures and reservation changes will trigger the update of the downstream merged request. To avoid frequent re-negotiation, the merging point may decide to reserve more resources than the sum of the upstream requests and add resources in larger increments if the current downstream allocation has been reached or is about to be reached. (BGRP [74] analyzes the trade-off in some detail.) We consider aggregation first for RNAP-D, and then for RNAP-C.

#### 4.4.2 Aggregation and De-aggregation in RNAP-D

Fig. 4-4 illustrates how RNAP message aggregation works in the RNAP-D architecture. Consider the aggregation of Reserve messages (this also applies to Query messages). At access network A, the border router  $R_a$  creates an aggregate Reserve message, with the source address of the *Id* object set to 'a', the interface address the aggregator  $R_a$ , and the destination address of the aggregate message set to the network address B. It also sets the aggregation flag, which marks the message as aggregate.  $R_a$  then forwards the aggregate Reserve message hop by hop downstream.  $R_a$  also turns off the router alert option of the incoming per flow messages and tunnels the per-flow Reserve messages down to the de-aggregation point ( $R_b$  in Fig. 4-4), so that per-flow reservation can resume in the destination network. In each per-flow Reserve message, the address of the aggregator will be included in the *Aggregate Flow Id* field, to enable proper mapping at the de-aggregation point. A per-flow Reserve message is encapsulated in an UDP packet with the destination network address set as B, and the port number set to a port reserved for RNAP, and forwarded.

A border router of a domain is a potential de-aggregation point for RNAP messages to that domain. Therefore, filters are set up at border routers of a domain so as to intercept aggregate RNAP messages as well as tunneled per-flow RNAP

messages. For instance, the border router  $R_b$  (Fig. 4-4) of domain B is set up to intercept UDP packets with destination address set to the network address B and port number set to the RNAP port. Once intercepted, aggregate Reserve messages and tunneled per-flow messages are sent up to the transport layer. The de-aggregation point will record the mapping between an aggregation flow and per flow messages, by checking the *Aggregation Flow Id* field. The router alert option will be turned on for per-flow Reserve messages arriving at  $R_b$ , and the messages will be forwarded, allowing per-flow resource reservation within domain B. The aggregate Reserve message (identified as such by its *Aggregation Flag*) terminates at the de-aggregation router.

In response, a Commit message will be sent upstream for the aggregate Reserve message as well as each per-flow Reserve message. The de-aggregation point  $R_b$  will decide that the destination address for the per flow Commit message is 'a', by checking the mapping between the aggregate message and the per flow messages. Each per flow Commit message is then encapsulated in a UDP message with destination address 'a' and tunneled back to its aggregation point  $R_a$ . The aggregate Commit message will be forwarded hop by hop upstream until it reaches the aggregation point, and confirms the aggregate Reserve request sent by the aggregation agent. There is a similar message flow for RNAP Quotation messages in the upstream direction.

The aggregation entity on the source network side is also responsible for de-aggregation of RNAP response messages. It checks the mapping between an aggregate session and per-flow RNAP response messages. If it is the origination point for the corresponding aggregate session, it will map the aggregate-level pricing and charging (returned by the aggregate session Quotation and Commit messages) to the corresponding per-flow prices and charges for individual flows based on the local

policy.

Multiple levels of aggregation can occur, so that aggregate messages are aggregated in turn, resulting in a progressively thicker aggregate “pipes” towards the root of the sink-tree. For a level two aggregation of several level one RNAP aggregate requests as shown in Fig. 4-4, node  $R_x$  in domain X forms a level two aggregate message with the source address in the *Flow Id* set to ‘x’. Node ‘x’ also records the level one requests, and terminates these messages instead of forwarding them. In response, the RNAP agent at the de-aggregation node  $R_b$  sends response messages for the level two aggregate towards point ‘x’. At point  $R_x$ , the level one response messages are formed by mapping the pricing and charge data from level two aggregate message to individual level one aggregate response messages to send towards  $R_a$  and  $R_c$ . All the per flow request messages are tunneled downstream to node  $R_b$ , and per-flow response messages are tunneled from  $R_b$  directly either to  $R_a$  or  $R_c$ .

#### 4.4.3 Aggregation and De-aggregation in RNAP-C

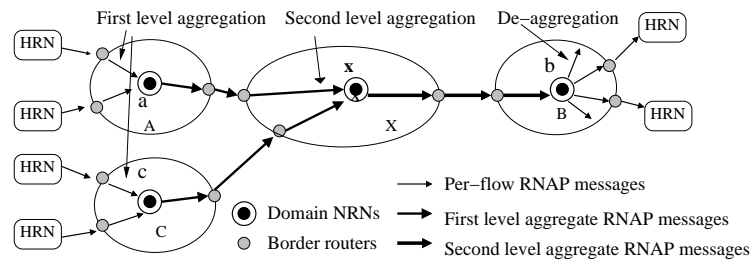


Figure 4-5: Example of RNAP-C message aggregation.

In the RNAP-C architecture of Fig. 4-5, the aggregation and de-aggregation entity are NRNs. Once again, we consider the aggregation of Reserve messages. At an aggregating NRN ‘a’, the aggregate Reserve message will be formed and sent domain by domain towards the destination domain NRN ‘b’, as in Section 4.3.3. In addition, the destination domain NRN is located through DNS SRV [67], and the aggregating NRN encapsulates the per flow Reserve messages in UDP packet

headers and tunnels them directly to the destination domain NRN ‘b’.

The destination domain NRN sends a Commit messages “hop by hop” (each hop is one domain) upstream towards ‘a’ in response to an aggregate Reserve message. It also receives the encapsulated per flow Reserve messages from ‘a’, processes them to perform per-flow reservation in the destination domain, and determines from the *Aggregate Flow Id* field what per-flow response messages are to be encapsulated and tunneled back to ‘a’. There is a similar message flow for RNAP Quotation messages in the upstream direction. The mapping of pricing and charging information from aggregate session to per flow message is similar to that in RNAP-D.

#### **4.4.4 Overhead Reduction due to Aggregation**

As a result of the aggregation of RNAP messages, the message processing overhead and the storage of the RNAP state information are greatly reduced in the core network. Since per flow messages need to be tunneled to the destination network, the RNAP message transmission bandwidth is not reduced, and actually slightly increased because of the extra aggregation messages. But since RNAP messages are updated with a relatively long interval, this is not a major concern compared with the bandwidth that will be consumed by the data flows.

#### **4.4.5 Resource Negotiation for Flow-Aggregates and Advance Reservation**

At the aggregation point, the NRN acts as the client negotiator for an aggregate session in negotiations with the downstream NRN (for example, referring to Fig. 4-5 again, the aggregating NRN ‘a’ negotiates with NRN ‘x’). In general, the client negotiator will negotiate resources for an aggregate session corresponding to the per-flow reservation requests. To avoid frequent re-negotiation, however, it is likely that

the client negotiator will increment or decrement the requested resources with some minimum granularity. When the sum of per-flow requests approaches the resources reserved (or reaches some threshold) for the aggregate, the client negotiator reserves an additional block of resources. Similarly, the requested reservation is decremented in blocks as required. The larger the block, the less frequently the aggregate session needs to be re-negotiated, but a higher cost is incurred for resources which may be under-utilized. In general, negotiation of resources in blocks results in a fairly static service, and periodic re-negotiations, if any, would occur with a much longer negotiation interval. Hence price Quotation messages for the aggregate session will probably only be sent in response to Query messages, when an additional block of resources needs to be reserved or removed.

The NRN at an aggregation point may also forecast a certain demand to a particular destination network, and could negotiate a large block of resources in advance, using the advance reservation mechanism. The HRN or client NRN indicates an advance reservation using the *Start Time* and *End Time* fields in the Service description. The server NRN initializes session state at the conclusion of the advance negotiations, and maintains the state until the actual transmission has been completed.

If the client negotiator chooses to cancel part or all of the reservation made in advance, it can re-negotiate with the server negotiator and try to ‘sell back’ previously reserved resources at an agreed price. The price eventually agreed upon would probably reflect any cancellation or holding cost fee the server negotiator wishes to charge. The server negotiator may also offer to buy back resources reserved in advance, for more important usage.



## 4.5. Receiver Negotiation

When the HRN of the receiver side initiates the negotiation, the message sequence will be similar to that described in Section 4.2.2, but in the reverse direction. However, the routing of the message will be different, and thus the message processing is slightly different.

In RNAP-D, a receiver side will initiate the negotiation by sending a Query message with the sender as the destination address. No processing will be done as the message passes the intermediate routers, until the sender-side HRN receives the message.

Upon receiving a Query message, the sender side HRN will send the quotation message back hop-by-hop towards the receiver. The Quotation message travels from a sender to receiver(s) along the same path(s) used by the data packets. The IP destination address of a Quotation message is the unicast address of the receiver, while the the IP source address of a Quotation message must be the address of the sender. Each RNAP-capable node (or NRN) along the path(s) processes it to create RNAP state for the sender defined by the *Id* objects. Any service objects are also saved in the RNAP state.

If a price quote is requested by the HRN, the first-hop LRN (FHL) and each intermediate LRN verifies local availability of each service, and increments the price by the local price that it computes. If network service statistics are requested, each intermediate LRN will attach the service statistics for the service type and set the corresponding parameters in the *Service* field. The sender HRN sends the message to the FHL, and the FHL forwards the Quotation message downstream to LHL. The LHL forwards the Quotation message to receiver HRN. If re-negotiation is requested, periodic Quotation messages are also sent by the sender side HRN hop-by-hop downstream towards the receiver HRN, as above.

Based on the network price quotation and service statistics, sender traffic specification and receiver requirement, the receiver HRN sends a Reserve message to apply for services with specified service parameters for a flow or group of flow. The IP destination address of a Reserve message is the unicast address of a previous-hop node, obtained from the RNAP state. The IP source address is an address of the node that sent the message. A Reserve message is sent at the beginning of a session to request services for the first time, and thereafter, periodically at *Negotiation Intervals*, and also asynchronously at any time.

The Commit message is generated by the sender HRN in response to a Reserve message. For each *Id*, the LRN determines the admissibility of the session, and returns this information in the *Status* field.

In RNAP-C, the IP source address of a Query or Reserve message must be the address of the receiver HRN (or an NRN), and the destination address is the unicast address of a NRN of the destination domain (or the neighboring domain towards the source), obtained from the RNAP state. The IP source address of a Quotation or Commit message must be the address of the sender HRN, while the destination address must be the downstream neighboring domain NRN address.

## **4.6. Price and Charge Collation in RNAP**

The main RNAP messages, Query, Reserve, Quotation and Commit, all contain price fields in the *Service* objects, used to convey pricing and charging information. We first briefly discuss the purpose of the various price fields, and how they are used in RNAP messaging. We then consider the scenario in which sender and receiver HRNs share the charges for services used, and consider charging in a multicast session.

#### 4.6.1 Pricing Structure and its Use in RNAP Messages

The price fields in the *Service* object carried by RNAP messages consists of the sub-fields *NewPrice* and *Accumulated Charge*. The *NewPrice* field contains the price quoted by the network provider to the negotiating HRN for the next negotiation period. The units of the quoted price are service-specific. A reasonable unit could be “currency/Mb”, so that the charge is computed according to the volume of the data transmitted. Alternatively, the unit could be “currency/time”, so that the charge is computed according to the time of usage at a specific data transmission rate. In this case, the HRN can expect to be charged an amount equal to the *NewPrice* multiplied by the length of the negotiation period.

The *Accumulated Charge* field contains the total amount charged by the network provider since the beginning of the session. The accumulated charge is carried to protect against the loss of Commit messages. This field should have an unit of currency, but the specific unit is service specific.

In addition to the price fields, a message can also carry a *HRN\_Data* object with information related to HRN (at sending end or receiving end). The *HRN\_Data.Account* field identifies the account to which charges are to be debited. The corresponding *HRN\_Data.Charging Fraction* field indicates the fraction of the total charge to be borne by the HRN. If for example, the negotiating HRN wishes to be responsible for half of the charges, (in the understanding that the peer HRN will be responsible for the other half), it sets the *HRN\_Data.Charging Fraction* to 0.5. We return to this issue in more detail in Section 4.6.2. The *HRN\_Data.MinimumDataRate* and *HRN\_Data.MaximumDataRate* fields are included to allow the sender and receiver HRNs to reach a basic agreement about the desired transmission rate. With respect to the sender HRN, the data rates represent the minimum and maximum sending rates the sender is willing and able to transmit. With respect to the receiver HRN,

these rates indicate the minimum and maximum data rates the receiver is willing and able to receive. *HRN\_Data.Budget* field is used to indicate the budget a negotiator would afford. The unit for *HRN\_Data.Budget* is the same as that indicated in the price parameters. The *HRN\_Data.OtherData* field is intended to carry other information that could be used to facilitate negotiation.

In general, pricing and charging information follows the following basic flow: after a session has been opened, the negotiating HRN sends a Query message. The HRN indicates how much of the charge for each service it is willing to bear by setting the *HRN\_Data.ChargeFraction* field accordingly, and may also indicate its budget for a particular service by setting the *HRN\_Data.Budget* field. If the Query message has a null Service list, the HRN may still wish to indicate how much of the total charge it is willing to bear by including a single *HRN\_Data* structure by itself in the Query message, with the *HRN\_Data.ChargeFraction* set. The network responds to the Query message with a Quotation message in which the *Service.NewPrice* fields are set to the price quoted for each service, if it is possible to determine it on the basis of the received Query. The HRN then requests one or more services through a Reserve message. As with the Query message, it can use the *HRN\_Data.ChargeFraction* field in the Reserve message to indicate the fraction of the charge it is willing to bear for each service. The network responds with a Commit message, committing or denying the requests, and setting the *Service.NewPrice* field for each *Flow Id-Service* pair to the committed price.

Subsequently, the network sends periodic Quotation messages to quote the updated price for available services, and the HRN and network re-negotiate services by exchanging Reserve and Commit messages. The price parameters in these messages are used as before. In addition, the *Service.Accumulated Charge* field in the Commit message is used to carry the charges for the corresponding *Flow Id-Service* pairs in

the preceding negotiation interval.

#### 4.6.2 Shared Charging

Let us assume that the sender HRN negotiates services, but the receiver pays part of the bill. We consider end-to-end services across multiple domains, and assume for convenience the centralized architecture in each domain - the equivalent situation in a distributed architecture can be understood by replacing the NRN with a router.

The sender HRN sets the *HRN\_Data.ChargeFraction* fields in the service identifier in Query and Reserve messages according to the fraction of total charges it is willing to bear. In response, the receiver HRN generates a Quotation message, and indicates its willingness to pay by setting the *HRN\_Data.ChargeFraction* field to  $(1 - \text{negotiating } HRN\_Data.ChargeFraction)$ . It may indicate its unwillingness to be responsible for the entire amount by setting *HRN\_Data.ChargeFraction* to a smaller value. It could also agree to bear the entire charge, but indicate an upper limit on the price it is willing to pay by setting the *HRN\_Data.Budget*.

The Quotation message is forwarded by intermediate NRNs back to the sender, where it serves as feedback to the sender HRN about the willingness to pay of the receiver. A modified Reserve message is similarly read by the receiver and used to generate a Commit message, with the last hop NRN either accepting or denying the service requested by the Reserve message. If the respective *HRN\_Data.ChargeFraction* fields in the Reserve messages received from the sender and receiver HRNs add up to less than 1, the service request is denied, and the *Status.Reasons* field is set accordingly. The Commit message is forwarded upstream through intermediate NRNs, updating *Status* and *Price* fields along the way.

Other than indicating its willingness to pay, the receiver could also set the *HRN\_Data.MaximumDataRate* and *HRN\_Data.MinimumDataRate* fields to convey

to the sender the minimum sending rate it requires and the maximum rate that it can handle. This allows the receiver, for example, to indicate to the sender that it cannot handle a rate offered by the sender, and in general, provide the sender guidelines for the negotiation process.

In receiver negotiation with sender bearing part of the charges, a similar sequence of messages is used, except that the flow of information is in the reverse direction.

### 4.6.3 Multicast Charging

In a multicast session, either sender or the receivers could negotiate separately, or they could both participate in negotiation.

If the sender is solely responsible for negotiation and payment, messaging sequence is similar to the simple scenario considered for unicast. The sender HRN determines a service request based on price quotations from the NRN, and on feedback about received quality from the receivers. A similar messaging sequence is also followed when the receiver negotiates and is responsible for payments. The receiver's Reserve message is based on its knowledge of sender traffic formats. The price splitting among receivers are based on the individual domain policy. In both the above case, the sender and receiver can learn about each other's capabilities and requirements by end-to-end Query and Quotation messages contained in the *HRN Data* fields.

In receiver negotiation with partial or full sender payment, the receiver HRN learns about the sender's willingness to pay through end-to-end Query and Quotation messages, as in Section 4.6.2. The sender may specify a maximum expenditure through the *HRN\_Data.Budget*. An example of this kind of negotiation may be when a company multicasts a commercial advertisement. The receiver adjusts its received transmission according to the sender budget and network conditions.

If the sender negotiates, with partial or full receiver payment, the sender receives feedback about each receiver's willingness to pay, as discussed in Section 4.6.2. The sender negotiates resources based on the overall demand willingness to pay, and may request partial reservation on some paths on which receivers have a low willingness to pay.

## 4.7. NRN Location for a Neighboring Domain

When a HRN wishes to send a Query, it either sends the request to a locally configured NRN if it is known, or needs to look up the port and IP address of the local domain NRN first. Similarly, a NRN needs to determine the address of the neighboring domain's NRN when forwarding a request. We use `client` in the remainder of this section to represent either a HRN or a NRN, which needs to locate the address of the peering NRN server.

A client should be able to interpret explicit network notifications (such as ICMP messages) which indicate that a server is not reachable, rather than relying solely on timeouts. (For example, when UDP is the transmit protocol, the socket needs to be bound to the destination address using `connect()` rather than `sendto()` or similar so that a second `write()` fails with `ECONNREFUSED` if there is no server listening). If the client finds the server is not reachable at a particular address, it should try the RNAP-D mode by sending messages to the destination address of the data packets with the port number set as the designated port number for RNAP.

The client tries to find one or more addresses for the NRN server by querying the DNS. If one or more addresses are found, but no RNAP server at any of those addresses responds, then the client concludes the server is not available, and does not continue on to the next step. Before concluding the server is down, a client should try the RNAP-D mode by sending messages to the destination address of

the data packets with the port number set as the designated port number for RNAP.

The service identifier for DNS SRV records is “rnap”. The client queries the DNS server for address records for the corresponding host name. There are no mandatory rules on how to select a host name for a NRN server. For locating a neighboring domain NRN address, a client needs to find out the neighboring domain number first through BGP4 [75]. Users are hence encouraged to name their NRN servers using the as-number.rnap.arpa convention, as specified in RFC 2219 [76]. SRV records contain port numbers for servers, in addition to IP addresses; the client always uses this port number when contacting the RNAP server NRN. If there is no port number, the default port is used. The format of these queries is defined in RFC 2782 [67]. The results of the query are merged and ordered based on priority. Then, the searching technique outlined in RFC 2782 [67] is used to select servers in order. If a list of addresses are returned, the user attempts to contact each server in the order listed. If no server is contacted, or the DNS server returns no address records, the client tries the RNAP-D mode. By address record, we mean A RR’s, AAAA RR’s, or other similar address records, chosen according to the client’s network protocol capabilities.

A client may cache the list of DNS query results if one of the addresses was contacted successfully. Request for the same transaction should be sent to the same network address. Other requests from the same client select a server from the list of addresses cached, using the SRV load-balancing mechanism if applicable. The client must invalidate this list and retry the DNS query according to the rules in RFC1035 [77] when the DNS time-to-live expires. If the client does not find a RNAP server among the addresses listed in the cached answer, it starts the search described above.

For example, consider a client that wishes to send a RNAP Query. The client



does a DNS query of QNAME = “as-number.rnap.udp.arpa”, QCLASS = IN, QTYPE = SRV. If there is no address in the DNS response, the client will do a DNS query for A records. When an address is found, the client attempts to contact a NRN at that address at the specified port if it exists in the srv record or default port.

## 4.8. Other Issues

### 4.8.1 Transport Protocol and Reliability

RNAP messages are sent using the UDP protocol. In both RNAP-C and RNAP-D models, periodic RNAP messages provide a natural way of protecting against loss. RNAP also allows the HRN to solicit any service related information at any time during the negotiation session. In this case, the response message will serve as a confirmation message. For example, the receipt of Quotation message will indicate that the Query is sent correctly. To protect against triggered message loss and as an additional protection against periodic message loss, the RNAP client continues to retransmit a request with exponential back-off (for congestion control). That is, the re-transmission interval starts from a time interval T1 which is larger than the round-trip time, doubles the interval for each packet sent, and stops when a response message is received or the timeout interval T2 is exceeded. Default values for T1 and T2 are 500 ms and 30 s (default negotiation period) respectively. If one or more Reserve messages are lost in transmission, the network will provide service based on contracted rules from previous negotiation period as in Section 4.3.1 or local policy.

In addition to the periodical transmission and retry mechanism for UDP, the charge information in RNAP is transmitted as a cumulative value, that the current charge can be recovered after the successive response information is received. Also,

when multiple levels of quality of service are available from the network, RNAP messages can be sent with a high quality service to protect against their loss.

Failure of a negotiation server or of a device storing RNAP state information, device reboots, and network partitions need also to be considered. A back-up NRN may be needed for the RNAP-C model. If a request is timed out, a new connection needs to be opened to the backup server. A re-negotiation process is triggered from the failure domain to the corresponding neighboring domains to allow new resource agreements to be reached. Use of non-volatile storage is recommended in critical elements to protect against the element failure and device reboots. When a device that stores customer charging information is recovered or the network connectivity is re-established, the charge for the period can be retrieved.

The network should also be able to track the liveness of an application using RNAP by tracking periodic RNAP messages and also by monitoring the data flow. This would avoid charging a terminated application and wasting network resources.

#### **4.8.2 Stability**

We prove the stability of the framework through analysis at Section 6.4.. The stability of the framework is further proved through simulation in Section 7. The proposed RNAP negotiation is intended for period in the unit of minute or longer, and hence the transmission delay is not a big concern. To reduce the oscillation of the resource request due to the large number of re-negotiation, resource negotiation between domains can be in the granularity of block (detailed in Section 4.4.5), at the cost of slightly lower bandwidth utilization. In addition, the network could set some traffic modification threshold beyond which the price will be adjusted, and an end host could set up its own threshold to avoid unnecessary re-negotiation.

### 4.8.3 Security

For security, RNAP messages may be authenticated and encrypted similar to RSVP [78] or using IPSEC [79].

## 4.9. Protocol Comparison between RNAP and RSVP

RNAP shares an important functionality with RSVP: performing end-to-end resource reservations. Consequently, they share some common features, and, as we have pointed out earlier, it is possible to embed some components of RNAP as part of RSVP. In Chapter 8, we explain the extensions needed for RSVP to support RNAP.

However, RNAP has a broader range of functions, and differs from RSVP in a number of ways. We compare the two protocols in this section.

**Transport Mechanism:** RSVP operates on top of IPv4 or IPv6, occupying the place of a transport protocol in the protocol stack. However, RSVP does not transport application data but is rather an Internet control protocol, like ICMP, IGMP, or routing protocols. As described in Section 4.8., RNAP operates on top of UDP.

**Transmission Architectures:** RSVP messages are forwarded hop-by-hop along the same path as data flows, i.e., in the distributed format. RNAP is designed to support both centralized and distributed architectures. Correspondingly, RNAP messages may either be routed independently, or along the same path as data flows. This allows for flexibility in supporting different types of services and various requirements of service providers.

**Reliability:** RSVP sends its messages as IP datagrams with no reliability enhancement, and relies on periodic transmission of refresh messages to recover from a

lost message. This will lead to a delay in recovery for at least a RSVP refreshment interval. RNAP supports a special re-transmission mechanism when a message is not received as expected.

**Protocol Initiation:** RSVP is designed to operate with both unicast and multicast routing protocols. In order to efficiently accommodate large groups, dynamic group membership, and heterogeneous receiver requirements, RSVP makes receivers responsible for requesting a specific QoS. Therefore, RSVP is receiver-oriented, i.e., the receiver of a data flow initiates and maintains the resource reservation used for that flow.

RNAP is also designed to work with both unicast and multicast routing protocols. However, RNAP allows either data sender or receiver to initiate a reservation request. In general, it is reasonable for the entity paying for the service to initiate the negotiation and resource reservation process. Sender-driven negotiation can greatly reduce the signaling complexity; for example, the sender does not need to inform the receiver about the data path before sending a resource request. Allowing traffic initialization to be either sender-driven, receiver-driven, or from both sides adds flexibility, and supports different business models.

**Dynamic Negotiation Support:** RSVP allows a sender to modify the reservation request at any time. However, it does not facilitate selection from a group of services, or modification of existing requests. Supporting dynamic resource negotiation is one of the design goals of RNAP. RNAP can provide users with updated service statistics, as well as updated price information, to allow the users to select services and service parameters according to their needs. Support for dynamic negotiation is particularly useful for adaptive

applications, which would adapt their sending rate in response to changes in price and/or network statistics.

**Price and Charge Collection Support:** There is currently no mechanism in RSVP for supporting price and charge collection, which may be an important requirement for providing better than best effort services. As indicated earlier, the RNAP protocol provides the means to communicate price quotations and charges incurred. It also supports different charging modes: charging the sender, or receiver, or both.

## Chapter 5

### Pricing Strategies

A few pricing schemes are widely used in the Internet today [80]: access-rate-dependent charge (AC), volume-dependent charge (V), or the combination of the both (AC-V). An AC charging scheme is usually one of two types: allowing unlimited use, or allowing limited duration of connection (typically per month), and charging a per-hour fee for additional connection time. Similarly, AC-V charging schemes normally allow some amount of volume to be transmitted for a fixed access fee, and then impose a per-volume charge. Although time-of-day dependent charging is commonly used in telephone networks, it is not generally used in the current Internet. User experiments [81] indicate that usage-based pricing is a fair way to charge people and allocate network resources. Both connection time and the transmitted volume reflect the usage of the network. Charging based on connect-time only works when resource demands per time unit are roughly uniform. Since this is not the case for Internet applications and across the range of access speeds, we only consider volume-based charging.

In this work, we study two kinds of volume-based pricing: a fixed-price (FP) policy with a fixed unit volume price, and a congestion-price-based adaptive service (CPA) in which the unit volume price has a congestion-sensitive component. In the fixed price model, the network charges the user per volume of data transmitted,

independent of the congestion state of the network. The per-byte charge can be the same for all service class (“flat”, FP-FL), depend on the service class (FP-PR), depend on the time of day (FP-T) or a combination of time-of-day and service class (FP-PR-T).

If the price does not depend on the congestion conditions in the network, customers with less bandwidth-sensitive applications have no motivation to reduce their traffic as network congestion increases. As a result, either the service request blocking rate will increase sharply at the call admission control level, or the packet delay and dropping rate will increase greatly at the queue management level. Having a congestion-dependent component in the service price provides a monetary incentive for adaptive applications to adapt their service class and/or sending rates according to network conditions. In periods of resource scarcity, quality sensitive applications can maintain their resource levels by paying more, and relatively quality-insensitive applications will reduce their sending rates or change to a lower class of service. The total price under CPA is composed of a component that depends on congestion and a fixed volume-based charge. Thus, with four variations on the fixed volume-based charge as outlined above, we have the pricing models CP-FL, CP-PR, CP-T, CP-PR-T.

Based on the policy of each domain, different algorithms can be used for computation of a local or incremental price for a service at a given point in a network, and RNAP can be used to facilitate the price distribution, collation, and dynamic resource negotiation.

In this chapter, we first consider the issues of formulating end-to-end prices and charging customer flows accordingly. We address these issues within both the RNAP-D and RNAP-C architectures, and also discuss pricing and charging across multiple network domains. We then propose a pricing scheme in a DiffServ environ-

ment based on the cost of providing different levels of quality of service to different classes, on long-term demand, and on the level of congestion in the network. Finally, we describe two methods to derive the congestion price component of the total price, one based on a tâtonnement process and the other based on a multiple-bid, second-price auction system. We also discuss a slightly modified RNAP messaging sequence which can be used to implement the proposed auction scheme.

## 5.1. Price and Charge Formulation in Network

In the previous chapter, we discussed how price and charge information are communicated to the HRN through RNAP messages. We now consider the issue of arriving at the contracted price to be quoted for a flow receiving a particular service in a given negotiation period, and computing the charge for the service at the end of the period.

A network pricing entity, a router or an NRN, needs to maintain the price quotation for each service type. In general, the service prices are re-calculated periodically, based on network traffic characteristics, and this period is independent of the RNAP negotiation interval. The entities that are responsible for session price quotation and accounting (edge routers or NRNs) need also to maintain price quoted and charge accumulated for the active sessions.

The price structure maintained in the pricing entity may consist of several fields in order to reflect a complex pricing strategy such as that to be presented in Section 5.2., and is hence more complicated than the single *NewPrice* field carried in RNAP messages, which simply quote the estimated price to the HRN. The price quoted to a session remains unchanged during a negotiation interval, and is updated at the end of a negotiation period if the price of a service type has changed at some time during the interval. At the end of each negotiation period, the *AccumulatedCharge*



for a session is re-computed using the current price quoted for that period.

The service price quotation and session charging information are maintained by different entities, and used in different ways, depending on the RNAP architecture. We consider the centralized and distributed architectures separately.

### 5.1.1 Price and Charge Formulation in RNAP-D

In the distributed architecture, each LRN of an edge router maintains charging state information for the flows passing through it, based on prices quoted from the domain. At the beginning of a negotiation period (and also in response to a *Query* message), the destination HRN originates a *Quotation* message. The *Quotation* message is sent hop-by-hop back towards the HRN. At each LRN, the *Service.NewPrice* fields in the message are incremented according to the current price computed for the corresponding service at the LRN. In Section 5.2., we discuss a specific local pricing strategy in which a set of prices is computed for each service. In this case, some mapping behavior may have to be defined to obtain a single increment for the quoted *Service.NewPrice*. When the *Quotation* message arrives at the negotiating HRN, it carries the total quoted price for each service.

Similarly, *Commit* messages originate at the destination HRN, and are sent hop-by-hop back to the HRN. In this case, the *Service.NewPrice* and *Service.AccumulatedCharge* fields are all incremented at each router-LRN on the way.

### 5.1.2 Price Formulation in RNAP-C

When the centralized negotiation architecture is used, the local charging state information for a domain is maintained by the NRN. The price formulation strategy is a much more open-ended problem. Various alternatives may be considered, and different domains may apply different local policies. The NRN may compute a price

based on the service specifications alone. The price could be fixed, or modified based on the time of day. In general, if the price charged to a flow needs to depend on the network state and the flow path, we consider the following three approaches:

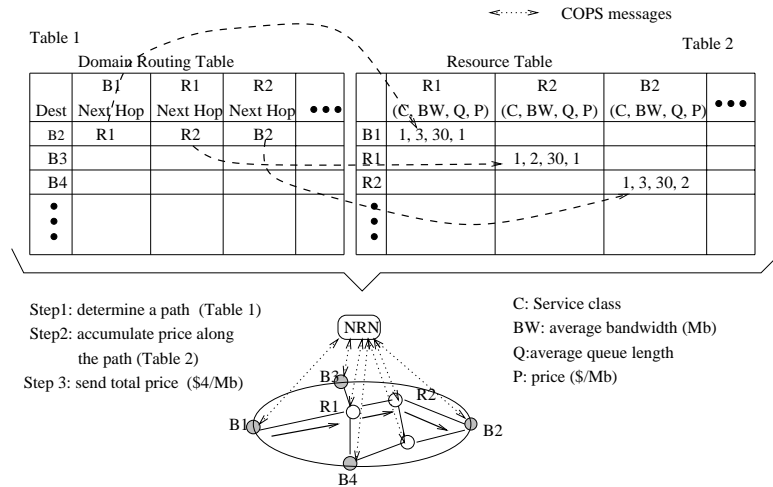


Figure 5-1: Price formulation in RNAP-C

1. The NRN makes the admission decision and decides the price for a service, based on the network topology, routing and configuration policies, and network load. In this case, the NRN sits at a router that belongs to a link-state routing domain (for example an OSPF area) and has an identical link state database as other routers in the domain. This allows it to calculate all the routing tables of all other routers in the domain using Dijkstra's algorithm. A similar idea has been explored in [73] in a different context.

The NRN maintains a domain routing table which finds any flow route that either ends in its own domain, or uses its domain as a transmit domain (Fig. 5-1). The domain routing table will be updated whenever the link state database is changed. A NRN also maintains a resource table, which allows it to keep track of the availability and dynamic usage of the resources (bandwidth, buffer space). In general, the resource table stores resource information for each

service provided at a router. The resource table allows the NRN to compute a local price at each router, for instance, using the usage-based pricing strategy to be described in Section 5.2.. For a particular service request, the NRN first looks up the path on which resources are requested using the domain routing table, and then uses the per-router prices to compute the accumulated price along this path. The resource table also facilitates monitoring and provisioning of resources at the routers. To enable the NRN to collect resource information, routers in the domain periodically report local state information (for instance, average buffer occupancy and bandwidth utilization) to the NRN. A protocol such as COPS [24] can be used for this purpose.

To compute the charge for a flow, ingress routers maintain per-flow (or aggregated flow from neighboring domains) state information about the data volume transmitted during a negotiation period. This information is periodically transmitted to the NRN, allowing the NRN to compute the charge for the period. The NRN uses the computed price and charge to maintain charging state information for each RNAP session.

2. Prices are computed at the network boundary, and communicated to the NRN. For price calculation, there are two alternatives.

One alternative is that the ingress router periodically computes a price for each service class and ingress-egress pair. The calculation is based on service specifications and local per-service demand at the ingress router; internal router states along the flow path are not taken into account.

The other alternative allows internal router load to be taken into account. Probe messages are sent periodically from an egress router to all ingress routers. A probe message carries per-service *Price* structures which accu-

mulate prices hop-by-hop at each router in a similar manner to Section 5.1.1. In both of the above cases, the ingress router maintains per-flow state information that includes the per-flow price (the price charged to the service class the flow belongs to), as well as the per-flow data volume entering the domain. This information is transmitted every negotiation period to the NRN, which computes the charge and is responsible for the messaging.

3. Price formulation takes place through an intra-domain signaling protocol. If resource reservation for a particular service in a domain is performed through a dynamic resource reservation protocol, such as RSVP or YESSIR[82], the price information is collected through the periodic messages of the reservation protocol, and stored at the ingress router. For example, the RSVP PATH message and RTCP [41] messages in YESSIR can collect pricing information. If the ingress router is responsible for sending the price information to the NRN, the price accumulated from a domain will be sent back to the ingress router along with the RSVP RESV message. Such an implementation, utilizing RSVP, is described in [83]. Communication between the ingress router and NRN occurs as discussed in the first scenario.

In the above schemes, we assume that a domain has one NRN. A domain could also have multiple NRNs, each NRN residing at an ingress router. In this case, the ingress router does not need to send periodic per-session reports to a centralized NRN, and pricing, charging, and RNAP messaging are done directly from the ingress router. Reliability concerns make a more distributed architecture (multiple NRNs, or RNAP-D) preferable. But some management goals (for instance, all NRNs in one domain need to have a coherent view of the resources at internal routers to allow them to make correct admission decisions) may make a centralized policy more attractive.

### 5.1.3 Charge Formulation for Multiple Domains and Flow Aggregates

When a customer flow spans more than one administrative domain, each domain computes incremental prices and charges for the flow using its own pricing strategy and architecture, and the total end-to-end price and charge are obtained by addition in a hop-by-hop manner (with each domain representing a single hop) as in Section 5.1.1.

When a set of flows enter a domain as a flow-aggregate, the NRN (or network domain as a whole in RNAP-D) carries out messaging and charging as if the aggregate belonged to a single customer. The NRN in the aggregating domain (or LRN at the aggregation point) is responsible for mapping the total charge into charges for individual customer flows or flow-aggregates.

## 5.2. Pricing for Differentiated Services

We assume that routers support multiple service classes and that each router is partitioned to provide separate link bandwidth and buffer space for each service, at each port. We use the framework of the competitive market model [62]. The competitive market model defines two kinds of agents: consumers and producers. Consumers seek resources from producers, and producers create or own the resources. The exchange rate of a resource is called its price. The routers are considered the producers and own the link bandwidth and buffer space for each output port. The flows (individual flows or aggregate of flows) are considered consumers who consume resources. The congestion-dependent component of the service price is computed periodically, with a price computation interval  $\tau$ . The total demand for link bandwidth is based on the aggregate bandwidth reserved on the link for a price computation interval, and the total demand for the buffer space at an output port is the average buffer occupancy during the interval. The supply bandwidth and buffer space need not

be equal to the installed capacity; instead, they are the targeted bandwidth and buffer space utilization. The congestion price will be levied once demands exceeds a provider-set fraction of the available bandwidth or buffer space. We now discuss the formulation of the fixed charge, which we decompose into *holding charge* and *usage charge*, and the formulation of the *congestion charge*.

### 5.2.1 Holding Charge

A service may enforce admission control to ensure some level of performance. In this case, the applications admitted into the network will impose some potential cost by depriving other applications the opportunity to be admitted. Hence, it is fair to charge the admitted applications a holding price. The holding charge can be justified as follows. If a particular flow or flow-aggregate does not utilize the resources (buffer space or bandwidth) set aside for it, we assume that the scheduler allows the resources to be used by excess traffic from a lower level of service. The holding charge reflects revenue lost by the provider because instead of selling the allotted resources at the usage charge of the given service level (if all of the reserved resources were consumed) it sells the reserved resources at the usage charge of a lower service level. The holding price ( $p_h^j$ ) of a service class  $j$  is therefore set to be proportional to the difference between the usage price for that class and the usage price for the next lower service class.

The holding price can be represented as:

$$p_h^j = \alpha^j (p_u^j - p_u^{j-1}), \quad (5.1)$$

where  $\alpha^j$  is a scaling factor related to service class  $j$ . The holding charge  $c_h^{jj}(n)$

when the customer  $i$  reserves a bandwidth  $r^{ij}(n)$  from class  $j$  is given by:

$$c_h^{ij}(n) = p_h^j r^{ij}(n) \tau^j \quad (5.2)$$

where  $\tau^j$  is the negotiation period for class  $j$ .  $r^{ij}(n)$  can be a bandwidth requirement specified explicitly by the customer  $i$ , or estimated from the traffic specification and service request of the customer.

### 5.2.2 Usage Charge

The usage charge is determined by the actual resources consumed, the average user demand, the level of service guaranteed to the user, and the elasticity of the traffic. The usage price ( $p_u$ ) will be set such that it allows a retail network to recover the cost of the purchase from the wholesale market, and various fixed costs associated with the service. In a network supporting multiple classes of service, the difference in the charge between different service classes would presumably depend on the difference in performance between the classes. The model we consider is a network supporting  $J$  classes of services, the service price for class  $j$  is  $p_u^j$ , the long time user bandwidth demand is known (e.g., through statistics) and can be represented as  $x^j(p_u^1, p_u^2, \dots, p_u^J)$ , and the cost of having capacity  $C$  during one unit of time is  $f(C)$ . The provider's decision problem is to choose the optimal prices for each class that optimize its profit:

$$\begin{aligned} & \max_{p_u^j} \left[ \sum_j^J x^j(p_u^1, p_u^2, \dots, p_u^J) p_u^j - f(C) \right], \\ & \text{subject to: } r(x^j(p_u^1, p_u^2, \dots, p_u^J)) \leq R, j \in J \end{aligned} \quad (5.3)$$

where  $r$  represents the bandwidth requirement for all classes, and  $R$  is the total bandwidth availability of the network. Assuming users can select individually a

class, the total demand for a class over a long enough time period depends only on the price for that class. If we assume the users have the utility functions of Chapter 6, the total demand of service class  $j$  can be represented as a constant elasticity model:  $x^j(p_u^j) = A^j/p_u^j$ , which varies inversely with the price of the service class.  $A^j$  reflects the total willingness to pay of users belonging to service class  $j$ .

*Service pricing for differentiated service*

DiffServ supports SLA negotiation between the user and the network. An SLA generally includes traffic parameters, which describe the user's traffic profile, and performance parameters, which characterize the level of performance that the network promises to provide to the conforming part of the user's traffic. A widely used descriptor for a user's traffic profile consists of a peak rate, a sustainable rate, and a maximum burst tolerance. The generally considered QoS parameters are delay and loss. Mechanisms, such as weighted fair queuing (WFQ) and class based queuing (CBQ) can be used to provision resources for different service classes. In general, a class with lower load leads to lower delay expectation. A higher level of service class is expected to have a lower average load, and hence lower average delay. If we do not consider the difference in element costs for different classes, charging services based on their individual expected load seems to reasonably reflect the cost of providing the services and the differences between their performance. Assuming that unit bandwidth of a service class would be charged a basic rate  $p_{basic}$  if all its bandwidth were used, and the expected load ratio of service class  $j$  is  $\rho^j$ , the unit bandwidth price for service class  $j$  can then be estimated as  $p_u^j = p_{basic}/\rho^j$ . The effective bandwidth consumption of an application with rate  $x^{ij}$  can be represented as  $x^{ij}/\rho^j$ . For constant elasticity demand,  $x^j(p_u^j) = A^j/p_u^j$ , and the effective bandwidth consumption is  $A^j/(p_u^j\rho^j)$ . Then the price optimization problem of equation 5.3 can be written as:



$$\begin{aligned} \max_{p_u^j} & \left[ \sum_j^J \frac{A^j}{p_u^j} p_u^j - f(C) \right], \quad p_u^j = \frac{p_{basic}}{\rho^j}, \\ \text{subject to: } & \sum_j^J \frac{A^j}{p_u^j \rho^j} \leq C \end{aligned} \quad (5.4)$$

The Lagrangian for the problem can be represented as:

$$\max_{p_{basic}} \left[ \sum_j^J A^j + \lambda \left( C - \frac{\sum_j^J A^j}{p_{basic}} \right) - f(C) \right] \quad (5.5)$$

The optimal solution is:

$$p_{basic} = \frac{\sum_j^J A^j}{C}, \quad p_u^j = \frac{p_{basic}}{\rho^j} = \frac{\sum_j^J A^j}{C \rho^j} \quad (5.6)$$

The bandwidth provisioned for each service class will be given by  $A^j/p_{basic}$ , and is hence proportional to total user willingness to pay for that class. The usage charge  $c_u^{ij}(n)$  for class  $j$  over a period  $n$  in which  $v^{ij}(n)$  bytes were transmitted is given by:

$$c_u^{ij}(n) = p_u^j v^{ij}(n) \quad (5.7)$$

### 5.2.3 Congestion Charge

Simple usage-based charging schemes monitor the data volume transmitted and in principle charges users based on their average rate. Charging according to the mean rate, though encouraging the user to use network bandwidth more efficiently, does not discourage users from selecting large traffic contracts and sending the worst-case traffic allowed by their contract, which can create problems for network traffic management. An appropriate pricing scheme should provide users incentives to select traffic contracts that reflect their actual needs. Effective bandwidth [84][85] and

pricing based on effective bandwidth [86] have been proposed in a multiple-service-class environment. However, effective bandwidth normally accounts for the worst case traffic subject to the traffic profile. The contract for typical users has an effective bandwidth much larger than the mean rate. Provisioning based on equivalent bandwidth is not economically efficient. Performance guarantees in DiffServ are qualitative and can be very loose. This may make it difficult to evaluate the equivalent bandwidth. Also, DiffServ does not allocate resources to applications based on their effective bandwidth. Therefore, it appears unfair to charge users based on their profile declaration only, though the charge should take the profile into account. To encourage users to reduce their resource requirements under network resource contention, we propose an additional congestion-sensitive price component.

With congestion price for class  $j$  over a period  $n$  represented as  $p_c^j(n)$  and the volume transmitted as  $v^{ij}(n)$ , the total congestion charge for customer  $i$  is given by

$$c_c^{ij}(n) = p_c^j(n)v^{ij}(n). \quad (5.8)$$

We now consider two methods for deriving the congestion price  $p_c^j(n)$ . The first method is based on a tâtonnement process and denoted as CPA-TAT. The second method is based on auction, and is denoted by CPA-AUC.

### 5.2.3.1 Congestion Pricing under CPA-TAT

The general network resources considered are bandwidth and buffer space. Therefore, two kinds of congestion pricing can be considered: pricing when the expected load bound is exceeded, or pricing when buffer occupancy reaches certain level. In the first case, when the average demand for a certain class exceeds a threshold, an additional congestion price is charged all users of that class.

In the case of priority dropping for AF class, the dropping precedence is only con-

sidered when the buffer occupancy reaches different thresholds. The same thresholds can be associated with different congestion or buffer prices. When each threshold is reached, user packets with the corresponding precedence level begin to be dropped with a certain probability, and users with higher precedence levels are charged the additional buffer price. Therefore, the higher precedence users pay the sum of buffer prices corresponding to all the exceeded thresholds. During congestion, lower precedence users will suffer lost packets, or reduce their rate, or smoothen their traffic at the source (at the cost of higher delay due to buffering), or change to a higher precedence and pay a higher price.

Both kinds of congestion price for a service class can be calculated as an iterative tâtonnement process [62]:

$$p_c^j(n) = \min[\{p_c^j(n-1) + \sigma^j(D^j, S^j)(D^j - S^j)/S^j, 0\}^+, p_{max}^j] \quad (5.9)$$

where  $D^j$  and  $S^j$  represent the current total demand and supply respectively, and  $\sigma^j$  is a factor used to adjust the convergence rate.  $\sigma^j$  may be a function of  $D^j$  and  $S^j$ ; in that case, it would be higher when congestion is severe.  $D^j$  and  $S^j$  will be different for bandwidth and buffer space congestion. The router begins to apply the congestion charge only when the total demand exceeds the supply. Even after the congestion is removed, a non-zero, but gradually decreasing congestion charge is applied until it falls to zero to protect against further congestion. In our simulations, we also used a price adjustment threshold parameter  $\theta^j$  to limit the frequency with which the price is updated. The congestion price is updated if the calculated price increment exceeds  $\theta^j p_c^j(n-1)$ . The maximum congestion price is bounded by the  $p_{max}^j$ . When a service class needs admission control, all new arrivals are rejected when the price reaches  $p_{max}^j$ . If  $p_c^j$  reaches  $p_{max}^j$  frequently, it indicates that more resources are needed for the corresponding service, or usage price for a class needs

to be adjusted to reflect the new demand statistics.

### 5.2.3.2 Congestion Pricing under CPA-AUC

In our proposed framework, the network commits resources only for short intervals, and periodically re-negotiates contracts with users, enabling optimal usage of network resources. Auction schemes proposed in the literature normally do not consider the short-term resource reservation scenario. The auction price either varies on a packet level [61], or remain constant throughout a flow's life time [87][63]. Only the DA and CHiPS models proposed in [64][65] require that resource auctions take place periodically, and that reservations are possible and valid only for a time period. However, neither the DA nor the CHiPS model addresses the user response to price fluctuations. In addition, the auction schemes in the literature generally assume that a user has an unique service requirement, and bids for a fixed amount of bandwidth at a certain price. An adaptive application could have very elastic service requirements, and different preferences for different levels of service. A user's preference can therefore be represented as a utility function, as will be discussed in Section 6.

Users' utility functions are generally concave, that is the perceived value per unit bandwidth decreases as the total bandwidth increases. Consequently, when the user application learns of an increase in service price under CPA-TAT, it tends to reduce its bandwidth request in order to maximize the perceived surplus. However, in a periodic auction scheme in which users submit single bids, the user does not learn about increased competition for scarce bandwidth ahead of the auction, and may learn about it after its bid is rejected. The CHiPS auction scheme [65] gives current connection holders a second chance, as mentioned in section II. But this does not eliminate completely the uncertainty of service availability, and has other problems

discussed in section II.

### *M*-bid Auction

We propose an auction scheme called *M*-bid auction. The underlying principle is that users express their willingness to pay for a number of levels of bandwidth in advance of the auction. Each user sends a *M*-bid, which consists of multiple (price, bandwidth) pairs. The bid price represents the per-unit premium a user is willing to pay during the congestion to receive the corresponding bandwidth, and the bid is derived by sampling the user's utility function. Users who have elastic bandwidth requirements but highly value an uninterrupted connection will ensure a high probability of receiving at least the minimum required bandwidth during congestion by bidding a high price (per unit bandwidth) on their minimum bandwidth bids, and a relatively low price on their higher-bandwidth bids. Users with higher budgets and less elastic requirements will bid a relatively high price at all bandwidths. At low levels of congestion, the former group of users benefit by paying less for a high bandwidth. During congestion, the network preferentially reduces the bandwidth allocation of users with elastic bandwidth requirements, while maintaining the bandwidth allocation of users with more willingness to pay. In this way, the network utilizes its capacity more efficiently and obtains a higher overall user satisfaction.

When an auction is performed at a network entity, such as a link, all the bids from all the users are ranked based on their individual bid price. Bandwidth is allocated starting with the highest bid, until the target utilization is met. The service congestion price (5.2.3.2) is set to the highest rejected bid price, in accordance with the second-price concept. If more than one bid price from a user is higher than the cutoff price, only the one with the lowest bid price (potentially the highest bandwidth) is ultimately selected.

Bid Price	Bid Bandwidth	Bidder	Bid Selection
5	10	1	
4	10	2	
4	15	1	←
3.5	20	3	←
3	25	2	←
2	30	3	×

Table 5.1: Example  $M$ -bid auction. Selected bids are marked as '←', rejected bids are marked as '×'

Table 5.1 shows an example  $M$ -bid auction. There are three users, each submitting two bids to a network entity. The total available network bandwidth is 70. All the user bids are ranked based on their bidding price. The example shows that all the users are accepted, and the total allocated bandwidth is 60. The lowest bids from users 1 and 2 are selected. The lower bid of user 3 is rejected. The per-unit bandwidth congestion-price is set as 2, the highest rejected bid price.

It is convenient to organize all the bids in a binary tree. New bids are inserted into the tree upon a session's arrival and all the bids from a session will be removed from the tree upon its departure. If the total number of bids in the tree is  $N$ , and a new user submits  $N_u$  bids, the insertion and deletion complexities are each  $N_u \log N$  separately. The complexity for calculating the total bandwidth is  $O(N)$ .

### **RNAP Messaging Sequence for CPA-AUC**

In Section 4, we discussed the formulation of end-to-end prices by accumulating the local (node- or domain-wise) prices using the RNAP query-response message sequence. We now describe a slightly modified sequence of RNAP messages to facilitate sending of CPA-AUC  $M$ -bids, and the accumulation of the end-to-end charge in this case.

1. The HRN sends a *Query* message towards the destination, along with a set of  $M$  bids for each service. The *Query* message collects the number of hops

along the user path until it reaches the destination. The receiver calculates the per-hop bids by averaging the set of  $M$  bids over the number of hops, and initiates a *Quotation* message containing the per-hop bids, as well as fields for QoS specifications, the allocated bandwidth and the service price, or a rejection decision. Subsequently, the receiver continues to initiate a new *Quotation* message periodically, with per-hop bids based on the last received *Query* message. The *Quotation* message is sent towards the HRN along the reverse path of the *Query* message. If the user is submitting  $M$ -bids for the first time, the *Quotation message* is forwarded upstream by each on-the-path LRN after accepting the  $M$ -bid for its next scheduled auction. For an existing connection, the LRN on each hop accepts the  $M$ -bid and also updates the *Quotation* message with the allocated bandwidth and increments the service price in the message based on its latest auction, and then forwards it upstream. When it reaches the HRN, the *Quotation* message either contains the minimum allocated bandwidth and the cumulative service price along the data path, or a rejection decision indicating that one or more LRNs along the path has rejected the user bids.

2. If the *Quotation* message rejects all the submitted bids, the HRN either submits a fresh set of bids in a *Query* message, or transmits data using best-effort service, or terminates the application. Also, the HRN may transmit data before service negotiation is complete end-to-end, and it will be served best-effort on part or all of the data-path. If the received *Quotation* message admits one of the  $M$  bids, the HRN learns how much bandwidth it can obtain end-to-end, and sends a *Reserve* message towards the destination to reserve this bandwidth. Subsequently, *Reserve* messages are sent in response to received *Quotation* messages, and may also be sent by trigger in some special

circumstances. Each *Reserve* is forwarded hop-by-hop to the destination. The LRN at each hop modifies its resource allocation if necessary, and reserves the corresponding bandwidth. The destination initiates a *Commit* message to confirm the resource allocation and forwards it upstream. At each hop, the service price and *AccumulatedCharge* are incremented appropriately by the LRN based on its local service price.

3. To reduce the setup delay of a new session, we propose the following mechanism when a LRN receives a set of bids from a new user, and it determines that the wait until the next auction is longer than a certain threshold. The LRN will allocate bandwidth to the user immediately if it is available, and if at least one of the user bids exceeds the service price from the preceding auction. The first *Quotation* message which installs the bids at a hop will also communicate the allocated bandwidth and service price when it is sent upstream towards the HRN. The bids will also be entered in the next scheduled auction, and the messaging sequence described above will take over. Bandwidth may be available for immediate allocation in various ways - due to the demand during the previous auction being less than the capacity, due to sessions being terminated since the last auction, or due to users reserving less bandwidth than they were allocated by auction (generally due to being allocated less bandwidth at other nodes along their data paths).

#### 5.2.4 Total Charge

Based on the price formulation strategy described above, a router arrives at a cost structure for a particular RNAP flow or flow-aggregate at the end of each price update interval. The total charge for a session is given by



$$c_s^{ij} = \sum_{n=1}^N [p_h^j r^{ij}(n) \tau^j + (p_u^j + p_c^j(n)) v^{ij}(n)] \quad (5.10)$$

where  $N$  is the total number of intervals spanned by a session.

In some cases, the network may set the usage charge to zero, imposing a holding charge for reserving resources only, and/or a congestion charge during resource contention. Also, the holding charge would be set to zero for services without explicit resource reservation or admission control, for example, best effort service.

## Chapter 6

### User Adaptation

In a network with congestion dependent pricing and dynamic resource negotiation (through RNAP or some other signaling protocol), *adaptive* applications with a budget constraint will tend to adjust their service requests in response to price variations. Adaptive applications could maintain a constant charging rate without worrying about the price change by varying the service requests in response to price changes. We call this service congestion-price-based adaptive service (CPA).

Although we focus on adaptive applications as the ones best suited to a dynamic pricing environment, the negotiation framework does not require adaptation capability. Applications may choose services that provide a fixed price and fixed service parameters during the duration of service. Generally, the long-term average cost for a fixed-price service will be higher, since it uses network resources less efficiently. Alternatively, applications may use a service with usage-sensitive pricing, and maintain a high QoS level, paying a higher charge during congestion.

In Chapter 5, we introduced two alternative methods based on tâtonnement and  $M$ -bid auction to determine the congestion price. We refer to the congestion-price based adaptation processes based on tâtonnement and auction as CPA-TAT and CPA-AUC, respectively.

In this chapter, we consider how network bandwidth is allocated among compet-

ing users under the two policies. Under the CPA-TAT policy, we assume that the user chooses a sending rate and level of service so as to maximize her “satisfaction”, subject to the service price and user budget. Under the CPA-AUC policy, each user submits bids in the form of willingness to pay for different sending rates and levels of service, and the network entity sets the price and allocates bandwidth based on the bids. In either case, the application source rate is adapted accordingly. We also discuss the stability of the adaptation process, including price changes in the network due to changes in user demand, and rate adaptation by the user in response to price changes.

## 6.1. The Utility Function

We consider a set of user applications, required to perform a task or *mission*, for example, audio, video, and whiteboard applications for a video-conference. The user would like to determine a set of transmission parameters (sending rate and QoS parameters) from which it can derive the maximum benefit.

We assume that the user defines its preferences or willingness to pay quantitatively, through a *utility function*. The utility function represents the perceived monetary value (say, 15 cents/minute) provided by the set of transmission parameters towards completing the user task.

The *utility* of a reservation request denotes how beneficial the corresponding network resource allocation would be towards completing the mission. The utility function is therefore a function in a multi-dimensional space, with each dimension representing a single transmission parameter allocation for a particular application. The objective of the adaptation is to select a set of transmission parameters that gives the maximum possible utility relative to the cost of obtaining this service, subject to the user budget constraint.

### 6.1.1 Utility as Perceived Value

Clearly, the utility of a transmission depends on its quality as perceived by the user. However, since the user is paying for the transmission, it appears reasonable to define the utility as the *perceived value* of that quality to the user. For example, an audio transmission requiring a certain sending rate and certain bounds on the end-to-end delay and loss rate may be worth 10 cents/minute to the user. The perceptual value is strongly correlated to the perceptual quality, but is not exactly the same. For example, a pair of audio transmissions encoded identically and with the same transmission QoS parameters also have the same perceived quality, but their perceived values may differ according to the application requirements. For example, the requirement of a weekly meeting between native speakers will probably have a lower quality requirement than a conferencing system teaching a foreign language and hence the users will see a higher value for using high quality transmission for the latter.

A number of researchers have measured the subjective quality of multimedia transmissions [88][89][90]. Generally, these experiments were intended to derive the Mean Opinion Score (MOS), which is measured as an average perceptive quality across a number of test subjects. But in our framework, perceived value very strongly reflects individual user preferences, and the application task being performed. We consider it likely, therefore, that an user application will have one or more of the following features:

- allow the user to customize utility function(s);
- allow the user to define “scenario”-specific utility functions; a particular scenario may be selected by the user during a session, or may be deduced by the application based on user actions;

- allow user to specify a certain time-dependence of the utility function.

### 6.1.2 Utility as a Function of Bandwidth

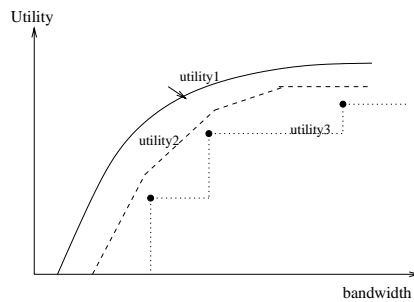


Figure 6-1: Some example utility functions

It is likely that only a few service alternatives will be available to a multimedia application on the Internet - at the current stage of research, some possible services are guaranteed [13] and controlled-load service [14] under the IntServ model, and Expedited Forwarding (EF) [17] and several classes of service under Assured Forwarding (AF) [16], also under DiffServ. A particular user application would be able to choose from a small subset of the available services. Each such service would probably provide some qualitative or quantitative guarantee for loss and delay. It seems likely, therefore, that the user would develop an utility function as a function of the transmission bandwidth (which in turn would depend on specific encoding parameters such as frame rate, quantization, etc.), at different discrete levels of loss rate and delay.

We can make some general assumptions about the utility function as a function of the bandwidth, at a fixed value of loss and delay. The application has a minimum transmission bandwidth, and the utility is zero for bandwidth below this threshold. Also, user experiments reported in the literature [89][90] suggest that utility functions typically follow a model of diminishing returns to scale, that is, the

marginal utility as a function of bandwidth diminishes with increasing bandwidth and eventually goes to zero, defining a maximum QoS requirement.

Fig. 6-1 shows some possible utility vs. bandwidth curves. Utility1 is a smooth function. User experiments for deducing the utility function would be performed at discrete bandwidths, and some form of interpolation, such as a piecewise linear function (utility2), can be used to approximate the utility function. In addition, in some multimedia applications, only discrete bandwidths are feasible. For example, audio codecs can only operate at certain bandwidth points (Utility3).

### 6.1.3 Effect of Scaling

In this section, we consider how changes in the utility function may influence the resource distribution. The operations we consider are an offset applied uniformly to the utility over all bandwidths, and multiplicative scaling of the utility function. We discuss the operations qualitatively here, and present some experimental results in Chapter 8. The utility function represents the relative preference of the user for different bandwidths. A constant (bandwidth-independent) offset to the utility function will not influence the resource distribution as long as the valuation of a bandwidth is higher than its cost.

On the other hand, a constant offset of the utility function changes the minimum perceived value. The minimum perceived value represents how much the user is willing to pay to just keep the application alive. Lowering this value allows the application to be terminated more readily during congestion (high cost). If a user values an uninterrupted service very highly, he increases the perceived value of the “keep alive” bandwidth.

A multiplicative scaling of the utility function by a factor greater than one tends to increase its bandwidth share, since it results in a bigger additive increase in

perceived surplus at higher bandwidth than at lower bandwidths. Effectively, the demand elasticity of the application is reduced. The opposite effect is observed when the scaling factor is less than 1.

#### **6.1.4 Time Dependence of Utility**

For a particular application, the value of the information may vary with time. An user may perceive a higher value initially after the connection is established, and a lower value after a certain duration (typically, a phone call is very important to the user in the first one minute, compared to one that has lasted 30 minutes), or the reverse (for a movie, the ending is usually more interesting than the introduction). The relative importance of individual applications in a system may also evolve with time.

The evolution with time of the application utilities may be defined based on various user-defined scenarios. A simple way of representing the time evolution is to represent the multiplicative scaling and additive offset in Section 10.5.1, with a pair of time dependent parameters,  $\alpha$  and  $\beta$ , so that the time-dependent utility can be represented as  $\alpha_j(t) * U_j(\cdot) + \beta_j(t)$ , where  $j$  represents a task performed at a time  $t$ .

#### **6.1.5 An Example Utility Function and User Adaptation**

Based on the previous discussion, we assume that the utility function is concave, with minimum and maximum constraints. The minimum perceived value for a task can be regarded as an “opportunity” value, and this is the perceived utility when the application receives just the minimum required bandwidth. The user terminates the application if its minimum bandwidth requirement can not be fulfilled, or when the price charged is higher than the opportunity value derived from keeping the

connection alive. Hence, a utility function can be represented in a general form as a function of bandwidth as:

$$U(x) = U_0 + w \log \frac{x}{x_m}, \quad (6.1)$$

where  $x_m$  represents the minimum bandwidth the application requires,  $w$  represents the sensitivity of the utility to bandwidth, and  $U_0$  is the monetary “opportunity” that the user perceives at the lowest QoS level  $x_m$ .

The utility function is also sensitive to network transmission parameters such as loss and delay. In our work, we rely on the experimental results in [91] which show that users’ perceived quality for interactive audio decreases almost linearly with either delay or loss, with a minimum acceptable quality requirement. More subjective tests are needed for other application types. Currently, we assume a similar linear dependence for all applications. Accordingly, we represent the utility function as:

$$U(x) = U_0 + w \log \frac{x}{x_m} - k_d d - k_l l, \quad \text{for } x \geq x_m, \quad (6.2)$$

where  $k_d$  and  $k_l$  represent respectively the user’s sensitivity to delay and loss. In some cases, the user’s perceived sensitivity to loss and delay may depend on the bandwidth used, for example, when using different speech codecs. Since we are not assuming any particular application model, we assume that the users’ delay and loss sensitivity are independent of bandwidth in our simulations. A user with a higher sensitivity to delay or loss will tend to select a higher service class rather than request more bandwidth.



## 6.2. Application Adaptation Under CPA-TAT Policy

Consumers in the real world generally try to obtain the best possible “value” for the money they pay, subject to their budget and minimum quality requirements; in other words, consumers may prefer lower quality at a lower price if they perceive this as meeting their requirements and offering better value. Intuitively, this seems to be a reasonable model in a network with QoS support, where the user pays for the level of QoS he receives. In our case, the “value for money” obtained by the user corresponds to the surplus between the utility  $U(\cdot)$  with a particular set of transmission parameters, and the cost of obtaining that service. The goal of the adaptation is to maximize this surplus, subject to the budget and the minimum and maximum QoS requirements.

We assume the applications belong to a single user, and have piecewise linear utility functions. We first consider the adaptation strategy of a single application when its utility is a function only of bandwidth (at a fixed loss and delay). We then discuss the adaptation strategy when the utility is a function of multiple transmission parameters (bandwidth, loss and delay). Finally, we consider the problem of maximizing the *mission-wide* utility of a system comprising multiple applications performing a certain task. We also use the example user utility function from Section 6.1.5 in Section 6.2.4, to model user adaptation in response to a changing price and discuss the user adaptation strategy.

### 6.2.1 Adaptation of Single Application with Fixed Transmission Quality

If the quality of transmission is fixed (a particular delay and loss), the application utility (that is, the user-perceived value) increases monotonically with the bandwidth. Hence the maximization problem for the user can be written as:

$$\begin{aligned}
& \max [U(x) - C(x)] \\
& \text{s. t. } C(x) \leq b \\
& x_{min} \leq x \leq x_{max},
\end{aligned} \tag{6.3}$$

where  $x$  is the bandwidth under consideration,  $C(x)$  is the cost for the requested bandwidth,  $b$  is the budget of user,  $x_{min}$  is the minimum bandwidth requirement, and  $x_{max}$  represents the maximum bandwidth requirement. Note that  $U$ ,  $b$  and  $c$  are in units of money/time.

One way of carrying out this optimization is to fit the utility function to a closed form function. The optimal solution is then obtained by using Kuhn-Tucker conditions for a maximum subject to inequality constraints.

As mentioned earlier, the application utility is likely to be measured by user experiments and known at discrete bandwidths. In this case, it is convenient to represent the utility as a piecewise linear function, as shown in Fig. 6-2. The figure also assumes a constant unit bandwidth cost  $C$ , so that the cost-vs-bandwidth function is a straight line with slope equal to  $C$ . The budget is shown as a horizontal line intercepting the cost/utility axis. From the figure, it is evident that the optimal bandwidth is

**either** the segment end-point with the highest surplus, if this end-point meets the budget constraint ( $b$  in Fig. 6-2 case A)

**or else** the bandwidth corresponding to the intersection point of the cost line with the budget line ( $b'$  in Fig. 6-2 case B).

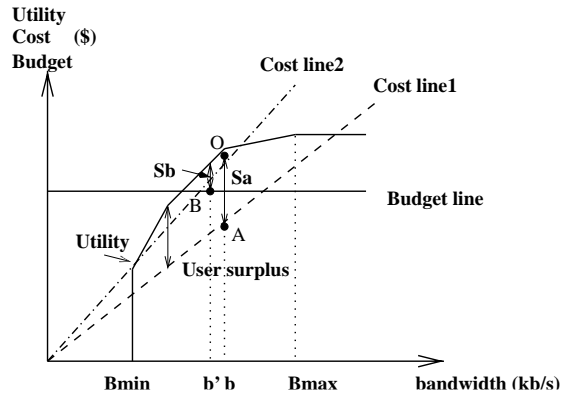


Figure 6-2: A perceived value based rate adaptation model

## 6.2.2 Adaptation of Single Application over Multiple Transmission Parameters

We now consider the maximization of the application surplus over a set of transmission parameters (usually, the bandwidth, loss rate and delay). The objective function is as shown earlier in equation 6.3, but  $x$ ,  $x_{min}$  and  $x_{max}$  are now vectors corresponding to the set of transmission parameters. If a complete quality of service parameter space is considered, the searching cost can be prohibitive. As briefly explained however, we believe it is likely that the application utility will take the form of a small set of utility versus transmission bandwidth functions, each at a different level of loss rate and delay, corresponding to a particular service. In this case, the optimization routine is as follows:

1. For each available service, use the corresponding utility versus bandwidth function to determine the optimal bandwidth, as in Section 6.2.1.
2. Select the service which gives the highest surplus at its optimal bandwidth.

### 6.2.3 Simultaneous Adaptation of Multiple Applications corresponding to Single Task

We now consider the simultaneous adaptation of transmission parameters of a set of  $n$  applications performing a single task. The transmission bandwidth and QoS parameters for each application are selected and adapted so as to maximize the mission-wide “value” perceived by the user, as represented by the surplus of the *Total Utility*,  $\hat{U}$  over the total cost  $C$ . We can think of the adaptation process as the allocation and dynamic re-allocation of a finite amount of resources between the applications.

A number of researchers have noted the interaction between the perception of the different component media in a multimedia system, such as a video conference [90][92][93][94]. For example, an investigation of video phone systems indicated that any increase in visual representation of the speaker increases the viewer’s tolerance to audio noise [92]. To take into account the interdependencies among applications, the utility of the  $i_{th}$  application should, in general, be written as  $U^i(x^1, ..x^i, ..x^n)$ , where  $x^i$  is the transmission parameter tuple for the  $i_{th}$  application. The total utility function of a system consisting of  $n$  individual application streams can be represented in general as  $\hat{U}(U^1(\cdot), \dots, U^n(\cdot))$ , where  $U^i(\cdot)$  represents the utility of stream  $i$ . Since we consider utility to be equivalent to a certain monetary value, we can write the total utility as the sum of individual application utilities:

$$\hat{U} = \sum_i [U^i(x^1, \dots, x^i, \dots, x^n)] \quad (6.4)$$

and the optimization of surplus can be written as

$$max \sum_i [U^i(x^1, \dots, x^i, \dots, x^n) - C^i(x^i)]$$

$$\begin{aligned} \text{s. t. } & \sum_i C^i(x^i) \leq b \\ & x_{min}^i \leq x^i \leq x_{max}^i, \end{aligned} \tag{6.5}$$

where  $x_{min}^i$  and  $x_{max}^i$  represent the minimum and maximum transmission requirements for stream  $i$ , and  $C^i$  is the cost of the type of service selected for stream  $i$  at requested transmission parameter  $x^i$ .

The general approach to solving this problem is to represent each utility  $U^i(\cdot)$  as a continuous function of the entire space of transmission parameters of all  $n$  applications, and solve the Kuhn-Tucker equations so as to maximize the total surplus.

A simpler method is to use a similar approach as to that for the single application case, along with some heuristics. In this case, we make the simplifying assumption that the individual application utility functions can be defined independently and is a function only of the transmission parameters of that application such that  $U^i(\cdot) = U^i(x^i)$ . This is a reasonable assumption since  $U^i(\cdot)$  would normally depend strongly mainly on the vector  $x^i$ . We can then write the total utility as:

$$\hat{U} = \sum_i [U^i(x^i(T_{\text{spec}}, R_{\text{spec}}))] \tag{6.6}$$

where  $x^i$  is the transmission ( $T_{\text{spec}}$ ) and quality of service parameter ( $R_{\text{spec}}$ ) tuple for the  $i_{th}$  application.

As earlier, we can decompose a single utility function  $U^i(x^i)$  into a set of service-specific utility functions which are functions only of bandwidth, each corresponding to a particular delay and loss provided by a particular service. Clearly, several combinations of services (and hence, service-specific utility functions) are possible. We first consider one particular combination of service-specific utility functions. Let

the utility of an application  $i$  be defined at  $L^i$  bandwidth levels. The utility at each level is  $u_l^i$  ( $l = 1, 2, \dots, L^i$ ), and the utility function is piece-wise linear. Segment  $l$  (the straight line between levels  $l$  and  $l + 1$ ) has a slope  $k_l^i$ . The optimal transmission parameter set for a particular combination of service-specific utility functions is then determined as follows:

1. From the utility function for each application  $i$ , determine the segment endpoint  $l_{opt}$  ( $l = 1, 2, \dots, L^i$ ), with bandwidth  $B_{opt}^i$ , at which the surplus (utility minus cost) is maximized for that application. Let the cost of the targeted bandwidth be  $C_{opt}^i(B_{opt}^i)$ .
2. If the total expenditure needed for the system,  $\sum_i C_{opt}^i(B_{opt}^i)$ , exceeds the total system budget, go to step 3, else stop.
3. From all the applications that receive service at level  $l_{opt} > l_{min}$ , find the application  $i_{victim}$  with the smallest slope in the surplus ( $u_l^i - C_l^i$ ) from level  $l_{opt}$  to  $l_{opt} - 1$  (this corresponds to the smallest sensitivity of application surplus to a reduction in bandwidth). Reduce the current bandwidth allocation for this application to the next lower bandwidth level ( $l_{opt} \leftarrow l_{opt} - 1$ ).
4. If the total system expenditure remains greater than the system budget, go back to step 3. If there is excess budget, allocate the excess budget to the current victim application (from step 3) to acquire as much bandwidth as permitted by the budget.

The above algorithm is repeated for each possible combination of service-specific utility functions; each time, an optimal transmission parameter set is obtained.

#### 6.2.4 Adaptation of Applications with Example Utility Functions

If the utilities of all the applications are represented in the format of equation 6.2, the optimization process for a system with multiple applications can be represented as:

$$\begin{aligned} & \max \sum_i [U_0^i + w^i \log \frac{x^i}{x_m^i} - k_d^i d - k_l^i l - p^i x^i] \\ \text{s. t. } & \sum_i p^i x^i \leq b, \quad x^i \geq x_m^i, \forall i, \quad d \leq D, \quad l \leq L \end{aligned} \quad (6.7)$$

where  $D$  and  $L$  are respectively the loss and delay bound of an application, above which the application no longer functions usefully.

The optimization involves assigning a service class and a bandwidth to each application  $i$ . For a particular assignment of service classes to applications, if the user can obtain the optimal bandwidth distribution according to equation 6.7 at a cost below his budget, then the bandwidth allocation that maximizes the perceived surplus for an application can be shown to be:

$$x^i = \frac{w^i}{p^i}. \quad (6.8)$$

Hence,  $w^i$  represents the money a user would spend based on its perceived value for an application. The above bandwidth distribution is considered for all possible service class assignments (constrained by application requirements and budget), and the one giving the highest total surplus is used.

If there is no set of service class assignments for which the optimal distribution of equation 6.8 can be obtained at a cost below the budget, the total budget is first distributed to the component applications according to their relative bandwidth sensitivity  $w^i$ . That is, each application receives a budget share  $b^i$  such that

$$b^i = b \frac{w^i}{\sum_k w^k} \quad (6.9)$$

Each application is then allocated a service  $i$  and bandwidth  $x^i = \frac{b^i}{p^i}$  which maximizes its individual surplus according to equation 6.2.

The discussion so far assumes that each price  $p^i$  is per unit average bandwidth. A price based on unit equivalent bandwidth [86] may be fairer since it takes into account the burstiness of user traffic. In this case, the adaptation of the source rate in response to a price change is not trivial. For the simple pricing scheme proposed in Section 5.2., a user could calculate a new average bandwidth when the price increases, or introduce additional buffering at the source to reduce its burstiness, at the cost of a higher delay.

### 6.3. Resource Allocation under CPA-AUC Policy

When the congestion price is derived from the auction-based CPA-AUC policy, the user adaptation is in fact a result of resource re-allocation by the network. If the transmission parameters such as loss and delay are fixed, the user samples its utility vs bandwidth characteristic at a fixed number of points, and submits the resulting (price,bandwidth) bid pairs. The network periodically determines the end-to-end price and bandwidth allocation for each user using the results of  $M$ -bid auctions as described in Chapter 5, and informs the user. The user application adjusts its sending rate to meet its target bandwidth.



### **6.3.1 Adaptation of Single Application over Multiple Transmission Parameters**

In this case, we assume as before that the application utility will take the form of a small set of utility versus transmission bandwidth functions, each for a different service with different loss and delay expectations. In this case, a user sends a different set of bids for each available service by sampling the corresponding utility versus bandwidth function. Each set of bids is submitted to the corresponding auction at a node or domain, and the user will eventually receive an end-to-end allocation decision for each set of bids. The user then selects the service for which the received allocation provides the highest benefit.

### **6.3.2 Auction for Multiple Applications corresponding to a Single Task**

We now consider the simultaneous adaptation of transmission parameters of a set of  $n$  applications performing a single task. A user will submit bids for each application individually. If the total of the allocated resources for all the applications is under its budget, the user will adjust the bandwidth for each application correspondingly. Otherwise, the total budget is first distributed to the component applications according to their relative bandwidth sensitivity. Then the resource request for each allocation is based on the budget.

### **6.3.3 Bidding over Multiple Hops**

A connection will in general span multiple hops and multiple network domains. We assume that each network domain, and possibly each node, allocates bandwidth locally based on a  $M$ -bid auction. The user utility function represents the willingness to pay of the user to receive different levels of bandwidth end-to-end, that is, from auctions at each hop. It is not obvious how the user should distribute this total

willingness to pay among bids at the different nodes. In this paper, we assume that the user's willingness to pay for a bandwidth is distributed equally among all the hops. If the user has traffic information from the network, it will probably try to allocate a higher bid to a bottlenecked node(s). This is a complex issue, and we would like to address this in future work.

## 6.4. System Stability and Network Dynamics

Application adaptation as well as applications entering and leaving the network lead to resource re-allocation and possibly adjustment of service prices. In this section, we first consider the stability of our pricing algorithm, and then the stability of the corresponding rate adaptation process.

### 6.4.1 Price Stability

In our proposed pricing strategy of Chapter 5, three price components are considered: holding price ( $p_h$ ), usage price ( $p_u$ ), and congestion price ( $p_c$ ). For a specific network provider, the holding price ( $p_h$ ), and usage price ( $p_u$ ) for a particular service are fixed, or change infrequently. Hence, only the stability of the congestion price needs to be considered.

The adaptation of the proposed congestion price follows the *tâtonnement* process for an equilibrium. The price will be quoted upward or downward, depending on whether or not demand exceeds supply, until the demand and supply reach equilibrium and a stable price  $p^e$  is located.

Since demand is a function of price, we can denote demand as  $D(p)$ . For a network service class, the targeted resource supply is fixed and is denoted as  $S$ . Suppose the rate of change of price moves directly with excess demand,  $E(p) = D(p) - S$  as follows:

$$p' = \frac{dp}{dt} = f(D(p) - S) = f(E(p)), \quad (6.10)$$

where  $f' \geq 0$ . The price change drives the demand and supply towards equilibrium. If the *tâtonnement* process is successful, the mechanism in equation 6.10 will generate a path of prices which will approach  $p^e$  as  $t$  increases:

$$\lim_{t \rightarrow \infty} p(t) = p^e \quad (6.11)$$

If equation 6.10 holds for any initial price  $p$  and  $p^e$  is unique, the system is called *globally stable*. If there is more than one equilibrium-price vector, then if  $p(t)$  reaches any of the  $p^e$ 's, the model is called *locally stable*. We only consider local stability in our system, where equation 6.11 holds for all prices  $p$  in some *neighborhood* of  $p^e$ . To prove that the local price stability exists, the function  $f(E(p))$  can be represented by a Taylor series expansion:

$$\frac{dp}{dt} = f(E(p^e)) + f'E'(p^e)(p - p^e) + \dots \quad (6.12)$$

The higher order terms are negligible in comparison with the first-order term in equation 6.12, as long as only *local* stability is considered. Since  $E(p^e) = 0$  by the definition of price, the equation 6.12 can be written as:

$$\frac{dp}{dt} = f'E'(p - p^e) \quad (6.13)$$

The solution of this equation is:

$$p(t) = p^e + (p^0 - p^e)e^{(f'E')t} \quad (6.14)$$

where  $p^0$  is any initial price.

The assertion of stability requires that the exponential term in equation 6.14

approaches zero as  $t \rightarrow \infty$ . Since  $f' > 0$ , so the stability assertion requires

$$E' = D_p(p) < 0 \tag{6.15}$$

In a reasonable network system, user demand will decrease as the price increases, so  $D_p(p) < 0$ . This proves that the proposed price will reach stability as times increases. However, the convergence speed of the system will depend on the convergence rate parameter  $\sigma(D, S)$ , or  $\sigma(p)$ . In our experiments, in order to obtain rapid but smooth convergence,  $\sigma(D, S)$  is large when the demand is much higher than supply, and is gradually reduced as the demand approaches supply.

Since the user demand will change as users join and leave, a new stable price may be reached as the total user demand changes. In the above process, the total demand and supply are assumed to be known instaneously. For a network with delay, this assumption may not be true. Since in our proposed model, the price is only updated periodically and in the time unit of minutes, the network delay has negligible influence on the stability.

#### 6.4.2 Stability of User Bandwidth Requests

Even though the network can reach stability for any fixed set of bandwidth requirements, the stability can be disturbed when new applications enter the network and existing applications leave the network. In addition, bandwidth adaptation by a number of users sharing the same link bandwidth can also lead to the oscillation of the system price and user requests, before the demand and supply reach equilibrium.

In the core network, oscillatory behavior can be minimized by aggregating RNAP requests, reducing the frequency with which the RNAP agent re-allocates resources and adjusts the price. The resource negotiated will be incremented or decremented with some minimum granularity. When the sum of per-flow requests approaches

the resources reserved for the aggregate (or earlier, at some pre-defined threshold), the client negotiator will reserve an additional block of resources. Similarly, the requested reservation is decremented in blocks as required as the requested bandwidth decreases. The larger the block, the less frequently the aggregate session needs to be re-negotiated, but a higher holding cost is incurred for resources which may be under-utilized.

As the network price changes, users will renegotiate resources to optimize their perceived benefit (surplus) from the service. The total user requests are hence oscillatory. The piece-wise linear utility function used to simplify the optimization can sometimes result in a severe oscillation between two adaptation points far apart in bandwidth, as will be seen in the experiments in Section 4.6.2. In our experiments, we used two measures to damp out the oscillations. The first measure was to use a proportional plus derivative (PD) controller. During each negotiation period, instead of letting the requirement jump to a new optimal bandwidth, the user shifts to a bandwidth between the current one and the optimal one, resulting in temporarily sub-optimal operation. The PD control law regulates the bandwidth request as follows:

$$r_{i+1} = \begin{cases} r_i - \alpha_0(r_i - r^*) - \alpha_1(r_i - r_{i-1}), & \text{if } \frac{|SP(r^*) - SP(r_i)|}{SP(r_i)} > \theta \\ r_i, & \text{otherwise} \end{cases} \quad (6.16)$$

where  $r^*$  is the desired optimal rate,  $r_i$  is the rate requested for negotiation period  $i$ , and  $SP(x)$  represents the surplus corresponding to bandwidth  $x$ . Quicker convergence is attained by making  $\alpha_0$  large, while the overshoot is minimized by making  $\alpha_1$  large.

In addition to the PD control, the bandwidth was allowed to be adjusted only if the new bandwidth led to an increase in surplus of at least  $\theta$  %. This prevented bandwidth adaptation which did not result in a significant improvement in the

perceived surplus.

## Chapter 7

### Simulation

In this chapter, we describe our simulation model for the CPA and FP policies. Depending on the service type and network infrastructure, the network may learn user resource requirements explicitly through a signaling protocol, or implicitly by traffic measurement. We first simulate explicit resource reservation and price signaling through RNAP. Unless otherwise specified, the policies are simulated at the call level, that is, we consider user resource contention due to the total user requested bandwidth exceeding the provisioned system bandwidth, rather than due to the burstiness of user traffic. The results are shown in Section 7.2. through Section 7.5..

We also perform simulations at packet level over a single DiffServ service domain, under which resources are not explicitly reserved for each flow. We simulate the service performance with or without admission control from the domain. User resource requirements are declared explicitly through RNAP, allowing admission control to be enforced if required in an experiment. The individual and total user resource demands are also obtained through measurement. Price and network statistics are signaled to users through RNAP.

## 7.1. Simulation Model

We used the *network simulator* [95] environment to simulate two network topologies, shown in Fig. 7-1 and Fig. 7-2. Topology 1 contains two backbone nodes, six access nodes, and twenty-four end nodes. Topology 2 is a more general network topology described and used in [96]. This topology contains five backbone nodes, fifteen access nodes, and sixty end nodes. All links are full duplex and point-to-point. The links connecting the backbone nodes are 3 Mb/s, the links connecting the access nodes to the backbone nodes are 2 Mb/s, and the links connecting the end nodes to the access nodes are 1 Mb/s. We use topology 1 in most of our simulations to allow congestion to be simulated at a single bottleneck node, and use topology 2 to illustrate the CPA performance under a more general network topology [97].

We modified the DiffServ module developed by Sean Murphy to support dynamic SLA negotiation and monitor the user traffic at ingress point. A Weighted-Round-Robin scheduler is modeled at each node, with weights distributed equally among EF, AF, and Best Effort (BE) classes. Although the DiffServ proposals mention 4 AF classes with three levels of drop precedence in each, we only simulated one AF class to make the simulations less resource-intensive, since this does not affect the general results in any way. Three different buffer management algorithms are used for different DiffServ classes - tail-dropping for EF, RED-with-In-Out [98] for AF, and Random Early Detection [99] for the BE traffic. The default queue length for EF, AF and BE are set respectively to 50, 100, 200 packets. Other parameters are set to the default values in the *network simulator* implementation.

A combination of exponential on-off and Pareto on-off traffic sources are used in the simulation. Unless otherwise specified, the traffic consists of 50% of each for all the service classes, and the on time and off time are both set to 0.5 seconds. The shape parameter for Pareto sources is set to 1.5. The packet size is uniformly



distributed and the mean is set to 200 bytes. The traffic conditioners are configured with one profile for each traffic source, with peak rate and bucket size set to the on-off source peak rate and maximum amount of traffic sent during an on period respectively for both EF and AF classes.

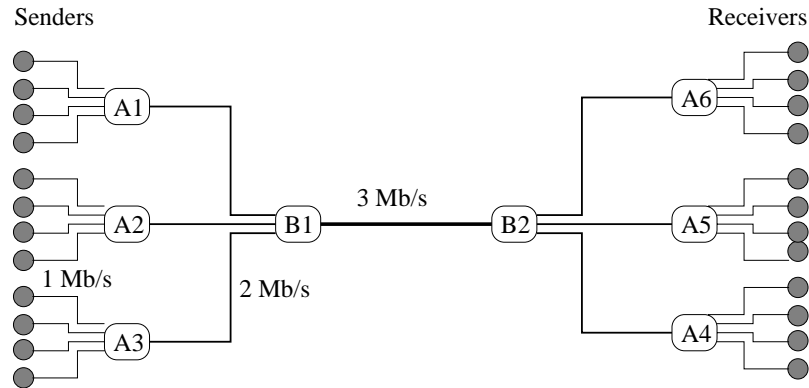


Figure 7-1: Simulation network topology 1

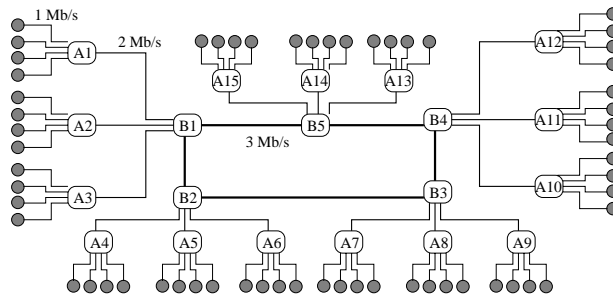


Figure 7-2: Simulation network topology 2

We also characterize the system load by *burst index* and *offered load*. The offered load for a service class is defined as the ratio between the total user resource requirement for a service type, and the configured class capacity at the bottleneck. Under the FP policy, the total user resource requirement is also the actual resource demand from all the users. Under the CPA policy, the total user resource requirement is what the total resource demand would be if there were no resource contention at the bottleneck and the network did not impose an additional congestion-dependent

price. The burst index is defined as  $OffTime/(OnTime + OffTime)$  for both types of On-Off sources.

User requests are generated according to a Poisson arrival process and the lifetime of each flow is exponentially distributed with an average length of 10 minutes. In topology 1, users from the sender side independently initialize unidirectional flows towards randomly selected receiver side end nodes. At each sender side end node, up to  $N_s$  flows can be initialized. At most  $12N_s$  flows (60 sessions with  $N_s$  set to 5) can run simultaneously in the whole network. In topology 2, all the users initialize unidirectional flows towards randomly selected end nodes. At most  $60N_s$  users (360 sessions with  $N_s$  set to 6) are allowed to run simultaneously in the whole network.

For ease of understanding, all prices in this section are given in terms of price per minute of a 64 kb/s transmission, currently equivalent to a telephone call. The basic price charged by the FP policy, and the basic usage price charged by CPA ( $p_{basic}$ ), are both set to \$0.08/min. We set the target average load of the EF class at 40%, the AF class at 60%, and the BE class at 90%. Therefore, based on the pricing strategy proposed in Chapter 5, the usage price for EF, AF and BE classes are set respectively as \$0.20/min, \$0.13/min, and \$0.089/min. When admission control is enforced, the holding price for the CPA policy is correspondingly set to \$0.067/min for EF class, and \$0.044/min for AF class.

Congestion pricing is applied at a node when instantaneous usage exceeds the target load threshold of each class or when the loss or delay exceeds the bounds associated with the class (delay bound of 2 ms, 5 ms, and 100 ms, respectively, for EF, AF, and BE, and loss bounds of  $10^{-6}$ ,  $10^{-4}$  and  $10^{-2}$  respectively). The price adjustment procedure is also controlled by a pair of parameters, the price adjustment step  $\sigma$  from equation 5.9 and the price adjustment threshold parameter  $\theta$ , defined in Chapter 5. Unless otherwise specified, values of  $\sigma = 0.06$  and  $\theta = 0.05$

are used.

The users are assumed to have the general form of the utility function shown in Section 6.1.5. At the beginning of each experiment, the user population is divided into users of the EF, AF and BE classes, although in some experiments they are allowed to adapt to price changes by switching to a different class.

For EF users, the elasticity factor  $w$  (which is also the user's willingness to pay), is uniformly distributed between \$0.13/min and \$0.40/min for a 64 kb/s bandwidth. For AF and BE users, it is uniformly distributed between \$0.09/min and \$0.26/min, and \$0.06/min and \$0.18/min, respectively. The minimum delay and loss requirements for each type of users are set to be the same as the expected performance bound of the corresponding service class. The opportunity cost parameter  $U_0$  is set to the amount a user is willing to pay for its minimum bandwidth requirement, and is hence given by  $U_0 = p_{high} \cdot x_{min}$ , where  $p_{high}$  is the maximum price the user will pay before terminating his connection altogether. Also, the  $M$  bids under CPA-AUC are obtained by sampling the utility function at five equally spaced points - therefore, each  $M$  bid is a set of five bids. Users re-negotiate their resource requirements with a period of 30 seconds in all the experiments. The total simulation time for a call level experiment is 60,000 seconds, and the simulation time for a packet level experiment is 20,000 seconds due to the CPU memory constraint.

We use a number of engineering and economic metrics to evaluate our experiments. The engineering metrics include the average reserved bandwidth (expressed as a ratio of the link capacity) over all negotiation periods, the average traffic arrival rate at the bottleneck, the average packet delay, the average packet loss rate, and the user request blocking probability. The averages are computed as exponentially weighted moving averages. The economic performance metrics include the average and total user benefit (the perceived value obtained by users based on their utility

functions), the end-to-end price for each service class, the charge a user incurs, and the network revenue a network provider can earn from all the admitted requests during a simulation.

We group the results of our simulations as follows. In Section 7.2., we present simulation results of the FP policy and the two versions of the CPA policy under the default conditions and parameter values as given earlier in this section, and compare their performance under these conditions. In section 7.3., we vary the number of customers sharing a system and evaluate the effect of the increased multiplexing of session requests under the different policies as the number of sessions is increased. In Section 7.4., we look at the effect of different features of user behavior on the relative performance of CPA and FP: user demand elasticity, adaptive and non-adaptive user populations, and adaptation during session initiation only as compared to adaptation during the life of the session. In Section 7.5., we discuss the effect of some network control parameters specific to the CPA-TAT policy. Finally, we present results from simulations of DiffServ services, performed at the packet level, under the CPA and FP policies.

We show results for the CPA-AUC policy only in Sections 7.2. and 7.3.. In subsequent sections we generally present results only for the FP and CPA-TAT policies. The results obtained under CPA-TAT and CPA-AUC are qualitatively similar, and the performance benefits observed for the CPA-TAT policy are applicable to CPA-AUC as well.

## **7.2. Performance Comparison of CPA-TAT, CPA-AUC and FP policies**

All of the simulations in this section are performed using topology 1 (Figure 7-1), except the simulations in Section 7.2.6, which use topology 2 (Figure 7-2) and

therefore allow multiple bottlenecks.

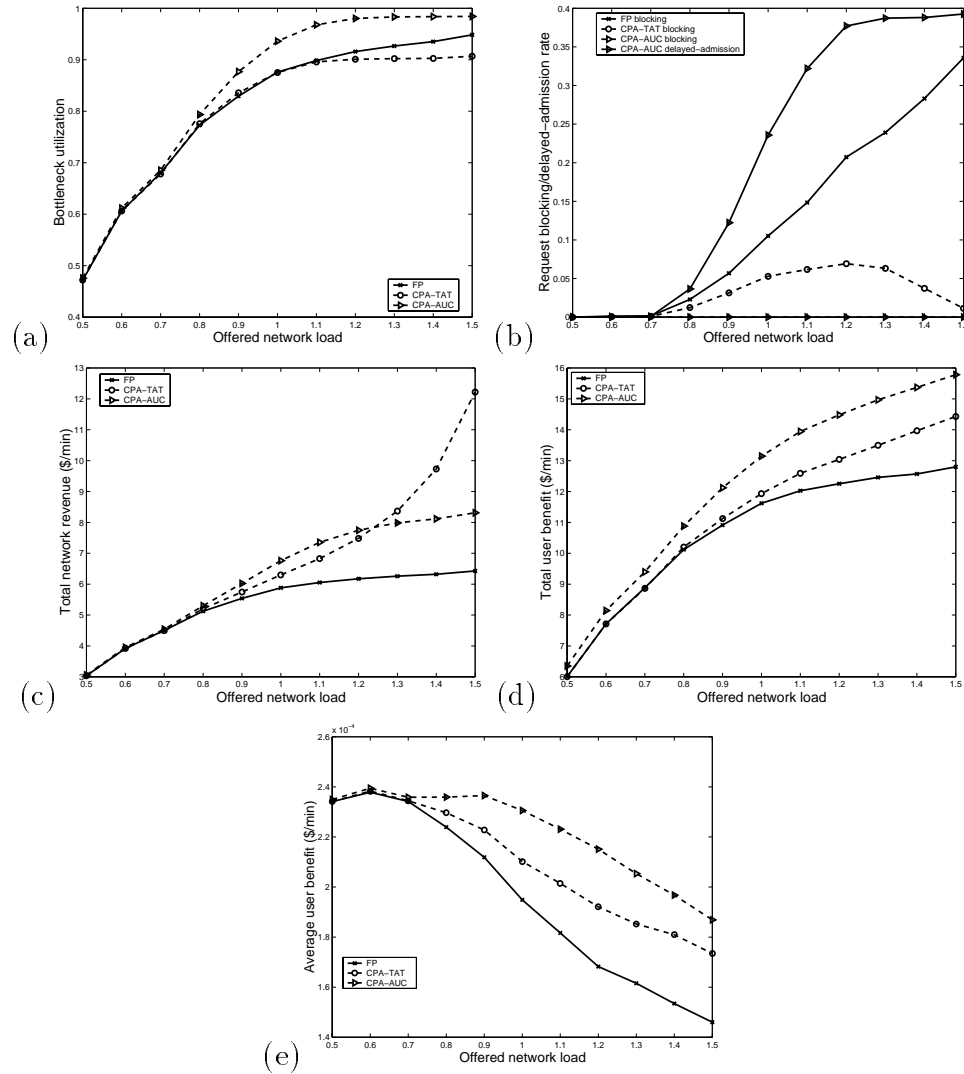


Figure 7-3: Performance metrics of CPA-TAT, CPA-AUC, and FP policies as a function of offered load: (a) bottleneck utilization; (b) blocking probability; (c) total network revenue; (d) total user benefit; (e) average user benefit.

### 7.2.1 Bottleneck Bandwidth Utilization

Fig. 7-3 (a) shows the variation of the bottleneck utilization as a function of the offered load, expressed as a fraction of the link capacity. The network utilization under FP policy increases continuously with the increase of offered load. The CPA-

TAT coupled with user adaptation is seen to maintain the total traffic load at the targeted level, 0.9 in this experiment, while the network load of both FP policy and CPA-AUC policy increases continuously with the increase in offered load. This is because the CPA-TAT policy tries to restrict user demand to a target level by driving up the congestion price, while the FP and CPA-AUC policy try to sell (or auction) all of the link capacity.

### 7.2.2 User Request Blocking Rate

All the policies admit connections until the total link capacity is saturated. Fig. 7-3 (b) indicates that the blocking rate of the FP scheme increases almost linearly as the offered load increases beyond 0.9, while both CPA policies are able to keep the blocking rate low even at very high loads. Both policies essentially drive users with elastic bandwidth requirements to use their reduced bandwidth requirement at high loads, rather than their optimal bandwidth requirements (which result in maximum user satisfaction). Hence, in period of congestion, the network can admit more users. The blocking rate of CPA-TAT increases initially and actually starts to decrease after reaching a maximum at offered load 1.2. This is because the price adjustment step is proportional to the excess bandwidth above the targeted utilization and increases progressively faster with offered load at higher loads, and the user bandwidth request decreases proportionally with the price according to the general utility function of Section 6.1.5. The blocking rate of CPA-TAT is up to 30 times smaller than that of FP. With centralized control, CPA-AUC model allows the resources to be re-distributed among competing applications based on individual user's willingness to pay. Since the total resource requirement based on each user's minimum bandwidth is set to be less than the network capacity in the simulation, the blocking rate of the CPA-AUC policy is seen to be almost zero. The more efficient

bandwidth allocation of CPA-AUC contributes to its increased bandwidth utilization in Fig 7-3 (a). The blocking rate with the immediate, “auctionless” admission policy actually indicates the probability of new users receiving delayed admission (until the next scheduled auction at the bottleneck node) instead of immediate admission. Up to 40% of users are delayed at high loads.

### 7.2.3 Network Revenue

Fig. 7-3 (c) shows that the revenue of FP flattens out after the onset of request-blocking, since the average number of accepted connections increases very slowly beyond this point. Though the total admitted load under CPA-AUC is close to that of FP, the revenue under CPA-AUC increases continuously as a result of the increase of the market price with offered load during congestion. The revenue of CPA-TAT policy is seen to increase faster than linearly after the network utilization saturates at the targeted level. CPA-TAT obtains more revenue than CPA-AUC at very high loads due to its lower blocking rate, as well as higher market price. The loss of revenue due to the scaling down of individual bandwidth requests of CPA policies are more than offset by gains due to the admission of more connections and the increase in the congestion price.

### 7.2.4 Average and Total User Benefit

Fig. 7-3 (d) shows that the user benefit flattens out for all policies after the onset of request blocking. The total benefit gained under both CPA policies are higher than that under FP beyond this point, and the difference increases as the offered load increases. As illustrated in Section ??, there is a potential opportunity cost associated with a request being blocked. The decrease in perceived benefit per connection of CPA due to the reduction of bandwidth is offset by the increase in the

number of admitted connections, each of which receives an “opportunity”. In effect, the CPA policy allows the network bandwidth to be used more efficiently under high loads. Since the CPA-TAT controls the total bandwidth to a targeted level and the CPA-AUC does not, the CPA-AUC can allocate more total bandwidth, resulting in higher total user benefit.

Similarly, Fig. 7-3 (e) shows the average perceived benefit per user against offered load. The average user benefit of CPA-AUC is higher than that of CPA-TAT, and the benefit of both CPA policies are higher than under the FP policy. For the FP policy, individual user requests do not depend on the offered load, and consequently, the average benefit per *admitted* user is independent of offered load. However, a progressively smaller fraction of users is admitted by the FP policy due to blocking. Therefore, the average perceived benefit of FP across all users decreases sharply with the load.

### 7.2.5 Dynamics of the System Price and User Bandwidth

Figs. 7-4 (a)(b) and (d)(e) show the variation of the system price and user bandwidth at three different levels of offered load under CPA-TAT and CPA-AUC policies. The bandwidth demand is shown for an “average” user, that is, one whose minimum and maximum bandwidth requirements are averages of the corresponding requirements of the user population. The price and bandwidth are nearly static at a load of 0.8, and are adjusted more frequently at higher offered loads, due to the more frequent arrival and departure of users. The price variation of CPA-AUC is smaller than that of CPA-TAT. Both policies have stable prices.

Figs. 7-4 (c) and (f) show the average and standard deviations of the system price and user bandwidth demand as a function of the offered load. The standard deviation in both figures shows the same trend, an increase to a certain level and



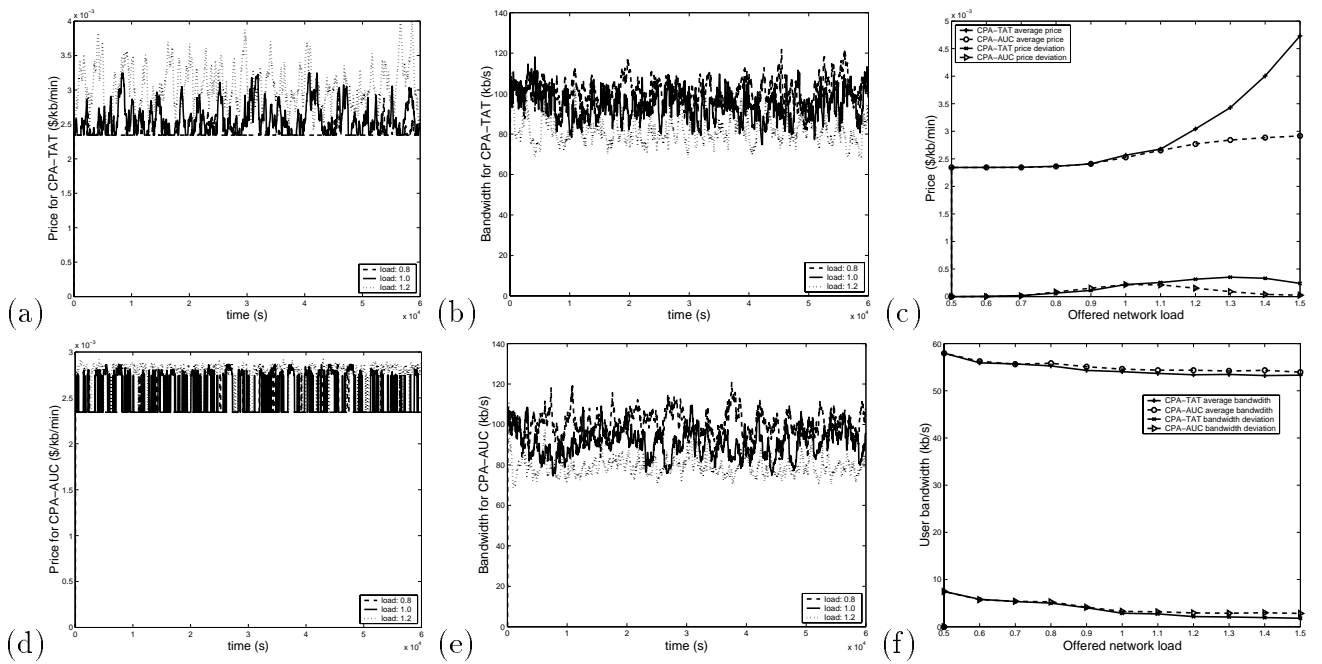


Figure 7-4: System dynamics under CPA-TAT and CPA-AUC: variation over time of system price (a), user demand (b) of CPA-TAT; time-average and standard deviation of system price of CPA-TAT and CPA-AUC at offered load 1.2 (c); variation over time of system price (d), user demand (e) of CPA-AUC; and average user demand and standard deviation of CPA-TAT and CPA-AUC at an offered load of 1.2 (f).

then a decrease. Initially, the price variations increase with the offered load due to the more aggressive congestion control. At heavy loads, the increased multiplexing of user demand smooths the total demand, and therefore reduces fluctuations in the price.

The average price is higher under CPA-TAT than under CPA-AUC during congestion. When the offered load exceeds the congestion threshold, CPA-TAT attempts to limit the utilization to the threshold value. This results in a more aggressive increase in the congestion price than under CPA-AUC. In a sense, the resource allocation under CPA-AUC truly reflects the demand for installed bandwidth relative to the supply. Since CPA-TAT is pro-active in preventing congestion, it may impose a higher price than the actual demand price. This also leads to a higher revenue for CPA-TAT as compared with CPA-AUC as indicated earlier, but a lower utilization, and lower total and average bandwidth usage and perceived benefit for the users. The total variation in price over a range of loads also depends on the basic usage price and holding price values, which should be set to reflect the long-term user demand for different service classes, so that demand fluctuations above the congestion threshold are short-term and infrequent, and congestion pricing is only occasionally employed to smooth out traffic peaks.

### **7.2.6 A General Network Topology**

In the experiments above, we studied the performance of CPA when the traffic shares a common bottleneck. In this section, we assume network topology 2 in Fig. 7-2, with the potential for multiple bottlenecks to exist, and for these bottlenecks to interact.

In the simulation, traffic is generated symmetrically from all users, as described in Section 5. The five backbone links are the potential bottleneck links. Note that in

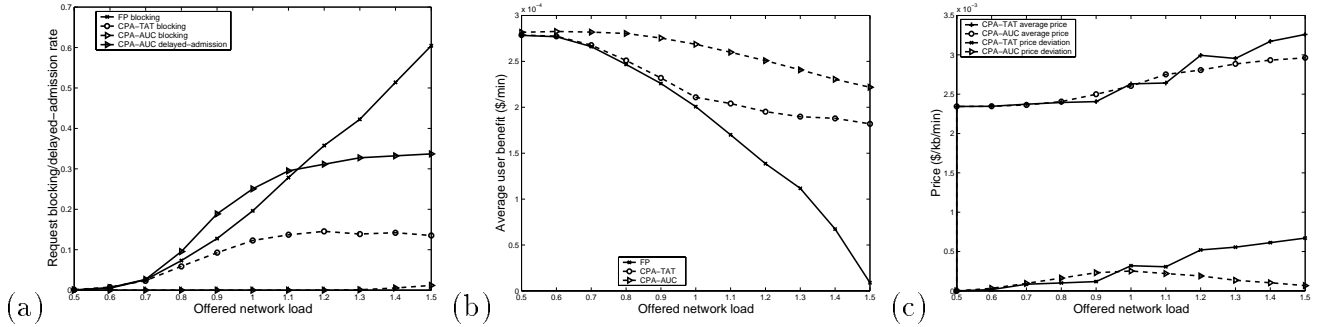


Figure 7-5: Performance metrics of CPA-TAT, CPA-AUC, and FP policies as a function of offered load with topology 2: (a) blocking probability; (b) average user benefit; (c) Time-average and standard deviation of system price of CPA-TAT and CPA-AUC at offered load 1.2.

reality, the backbone links are normally over-provisioned. We target the backbone links to be bottlenecks only for the convenience of simulation. We monitor the utilization at one of the backbone links, and calculate all the other parameters across the whole network. Fig. 7-5 shows the blocking probability and average perceived user benefit as a function of offered load, and the variation of the system price with time, for the three policies. All three metrics show trends similar to those for a single bottleneck, though the overall request blocking rate is higher than with a single bottleneck for all the policies. The variation with time of the average price under CPA-TAT is less smooth than the single bottleneck case due to the coupling of the traffic between different paths.

### 7.2.7 Result Summary

The results in this section indicate that the CPA policy takes advantage of application adaptivity for significant gains in network availability, revenue, and perceived user benefit, relative to the fixed-price policy. The congestion-based pricing is stable and effective. If the nominal (un-congested) price is set to correctly reflect long-term user demand, the congestion-based pricing should effectively limit short-term fluctuations in load.

Both CPA-TAT and CPA-AUC can effectively perform the role of congestion

control and form the congestion price. The congestion price calculated using auction tends to better reflect the actual usage, and allows higher network bandwidth utilization. This is at cost of its implementation complexity, and longer set-up delay for new connections (or alternatively, higher initial blocking rate if inter-auction admissions are allowed).

### 7.3. Effect of Session Multiplexing

In this simulation, we vary the number of customers sharing a system and evaluate the effect of the increased multiplexing of session requests under both CPA-TAT policy and CPA-AUC policy. We keep the network topology and user utility distributions unchanged, but scale the link capacity proportionally with the maximum number of flows.

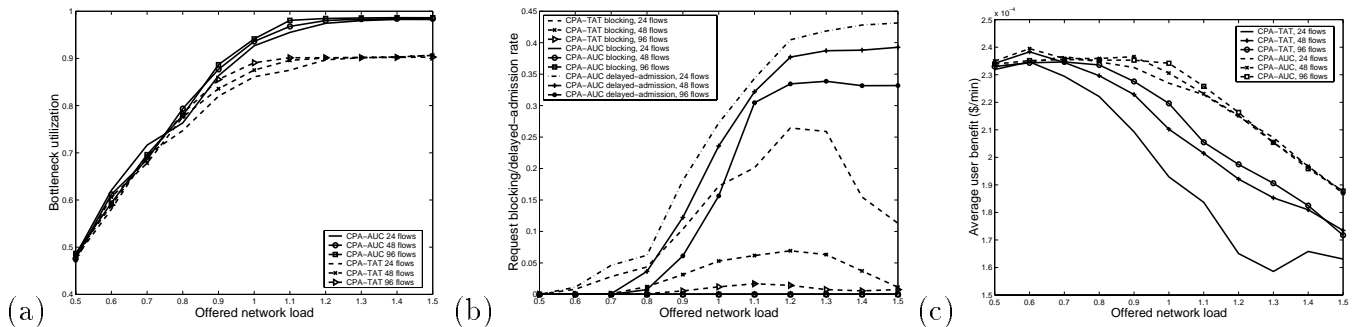


Figure 7-6: Performance of CPA and FP with different number of customers sharing the system: (a) bottleneck utilization; (b) blocking probability; (c) average user benefit

Fig. 7-6 (a) shows that the overall link utilization under CPA-AUC increases as the number of connections increases, at a given offered load. The link utilization under CPA-TAT also increases with the number of flows at moderate to high loads, but the utilization is eventually limited to the targeted level. Fig. 7-6 (b) shows that, as the number of connections increases, both the blocking rate under CPA-TAT and delayed-admission rate under CPA-AUC decrease. This is because the larger

number of connections leads to better traffic multiplexing and hence more efficient use of network bandwidth. Under CPA-TAT, the blocking rate with 96 connections is up to 30 times smaller than that with 24 connections. Under CPA-AUC, the delayed-admission rate with 96 connections is up to 1.3 times smaller than that with 24 connections. Correspondingly, the average user benefit for both CPA-TAT and CPA-AUC increases as the number of connections increases (Fig. 7-6 c).

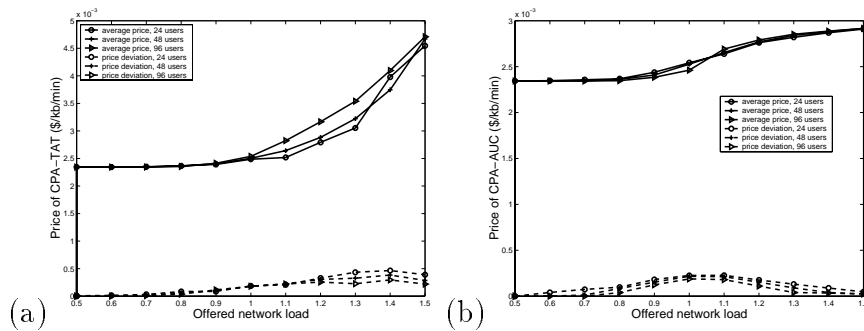


Figure 7-7: Time-average and standard deviation of system price of CPA-TAT (a) and CPA-AUC (b), plotted against offered load.

Fig. 7-7 depicts the average price of CPA-TAT and CPA-AUC as the network scales. As expected, both price become smoother as more users share the network, which is indicated by the smaller standard deviations. The average price of CPA-TAT increases slightly as the number of connections increases, due to the slightly higher network utilization. The average price of CPA-AUC does not change appreciably as network scales.

The results in this section indicate that the performance of the CPA policy further improves as the network scales and more connections share the resources.

## 7.4. User Behavior

We now look at effects of different aspects of user behavior on the performance of the CPA and FP policies. In Section 7.4.1, we evaluate the effect of user demand

elasticity on the system performance. In section 7.4.2, we study the system performance when different fractions of the user population are bandwidth-adaptive, and the remaining users maintain fixed bandwidth requests independent of the network price. In Section 7.4.3, we compare the system performance when users only adapt at session set-up, with the default adaptive behavior during the session.

#### 7.4.1 Effect of User Demand Elasticity

In the previous simulations, as mentioned in section 5, user utility functions of the form of equation 6.2 were used, with the elasticity factor  $w$  and the minimum and maximum bandwidth requirements uniformly distributed. In this section, we study the effect of the user demand elasticity factor  $w$  on the system performance. A smaller value of  $w$  corresponds to a more elastic demand, since the bandwidth-dependent component of the utility is smaller, and the user can reduce its bandwidth request in response to a price increase with only a small decrease in utility. (As explained in Section 6.2.4,  $w$  also represents a user's willingness to pay for bandwidth). We study the bandwidth and charge sharing among users with different utility functions, with  $w$  randomly set to the average default value of \$0.25/min. times 0.8, 1.0, or 1.2, keeping the other utility function parameters constant.

1. Fig. 7-8 (a) and (b) shows the instantaneous bandwidth sharing among users when all users have utility functions with the same elasticity, and three different elasticities, respectively. The offered load is 1.2. In both cases, we show the maximum, minimum as well as average bandwidth for all the active sessions in the system. Fig. 7-8 (a) indicates that when all the users have the same utility function, they share the bandwidth fairly at any time, and the maximum, minimum and average bandwidth demands coincide. Fig. 7-8 (b) indicates that when the users have different demand elasticities ( $w = 0.20, 0.25,$  and

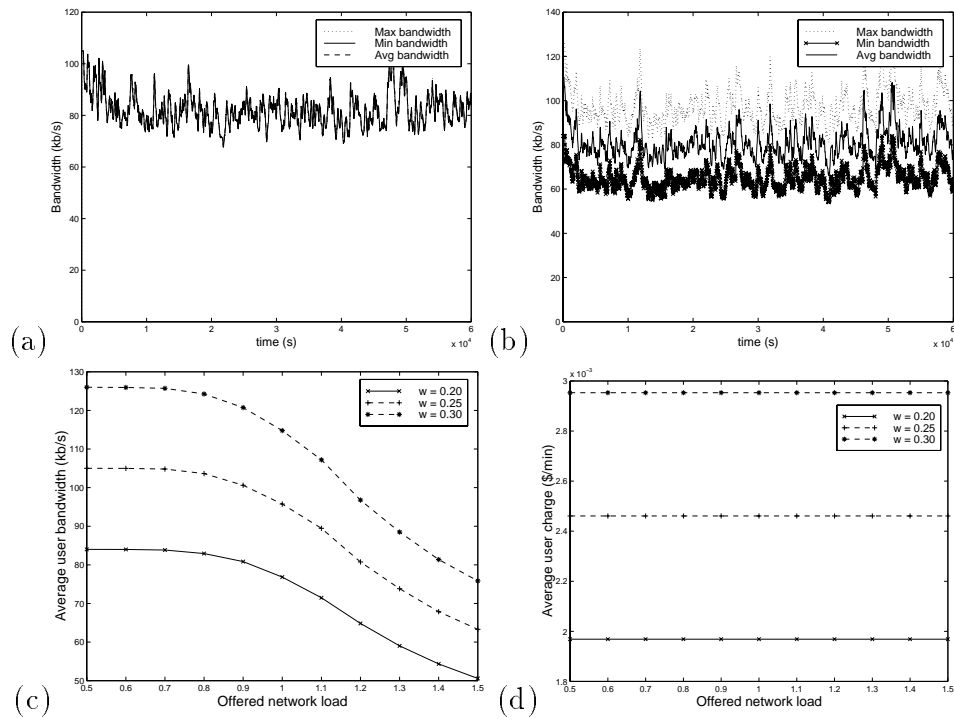


Figure 7-8: Effect of the elasticity factor  $w$  on bandwidth allocation and user expenditure: (a) minimum, maximum and average requested bandwidth when users have the same utility function; (b) minimum, maximum and average requested bandwidth when users have three different utility functions, with  $w$  set to 0.20, 0.25, and 0.30 \$/min. respectively; (c) average bandwidth reserved by users with the three different values of  $w$ ; (d) average expenditure of users with the three different values of  $w$

0.30 \$/min), their bandwidth shares span a range of approximately 50 kb/s.

2. Fig. 7-8 (c) shows the average bandwidth allocated to users with the three different utility functions. At a given offered load, the bandwidth users obtain is proportional to their individual demand elasticity factor or willingness to pay (that is, inversely proportional to the bandwidth demand elasticity). Fig. 7-8 (d) shows that average user expenditure is also proportional to  $w$ , and independent of the offered load.

Evidently, users with more elastic requirements are more sensitive to price changes and reduce their resource requirements faster as price increases, therefore receiving a smaller share of the bandwidth. In effect, users with more stringent bandwidth requirements choose to pay a higher charge and “borrow” bandwidth from users with more elastic requirements when the network is congested.

#### 7.4.2 Adaptive and Non-adaptive Users

In this section, we consider the environment where some users adapt their bandwidth requests under the CPA policy, while others maintain fixed service requests even when the congestion price is imposed. The latter group represents users with a willingness to pay that is high enough to maintain their maximum bandwidth requirements even at the highest price charged by the network. Under the CPA policy, the congestion-dependent price component increases until usage is driven down to the target utilization. We restrict the maximum price to 0.49 cents/kb/min for this set of simulations, so that the price does not increase indefinitely when 100% of the users are non-adaptive.

1. Fig. 7-9 (a) shows that when some users do not adapt, the network utilization can no longer be kept at the targeted level when the offered load exceeds a



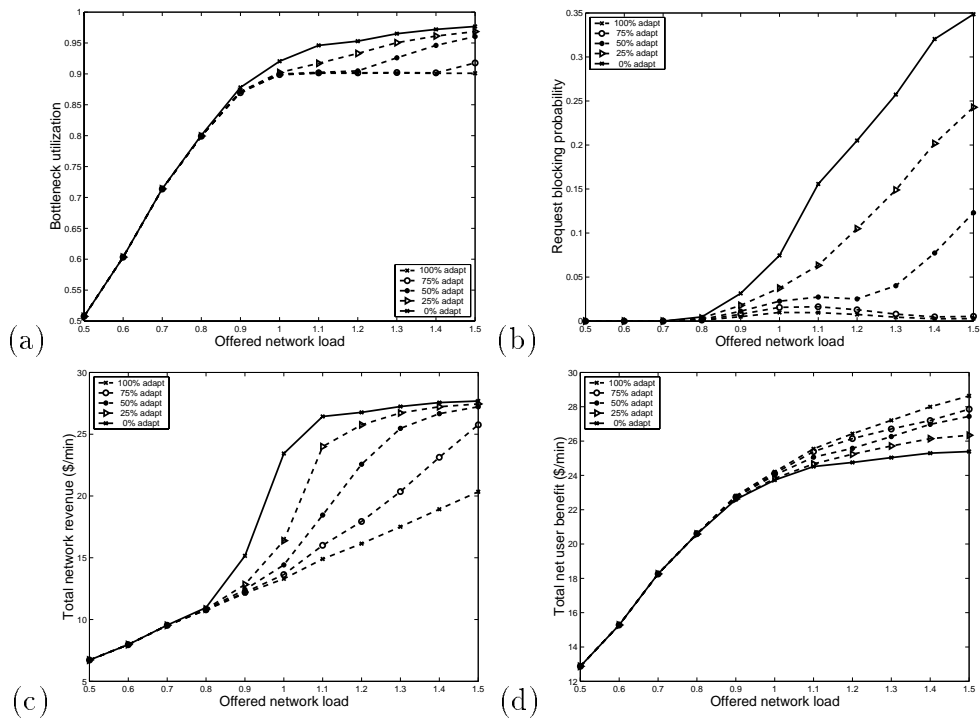


Figure 7-9: Performance of CPA when only some of the users adapt their bandwidth requests: (a) bottleneck utilization; (b) blocking probability; (c) network revenue; (d) total user benefit, plotted against offered load.

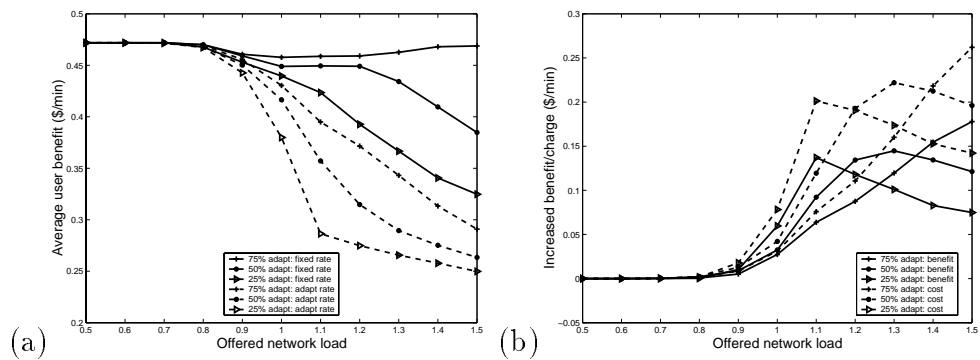


Figure 7-10: Relative benefits and expenditures of adaptive and non-adaptive users, under three different proportions of non-adaptive to adaptive users: (a) average user benefit for the two groups of users; (b) average benefits and average expenditures of non-adaptive users relative to average benefits and expenditures of non-adaptive users

certain threshold. This threshold decreases as the ratio of the adaptive requests decreases. Fig. 7-9 (b) shows that corresponding to the sharp increase in utilization beyond a certain load threshold, the blocking rate also increases sharply. We can also see that the network blocking probability reduces significantly even when some of the flows are adaptive, compared to when none of the flows is adaptive. Therefore, even if some of the users are adaptive, all the users can receive improved performance, particularly up to a certain threshold load.

2. Fig. 7-9 (c) shows that the total revenue increases as the proportion of non-adaptive users increases, since more users maintain their service request at high loads, and pay the higher price. Fig. 7-9 (d) shows that at a given load, the total user benefit increases with the number of adaptive users, an outcome of the lower blocking probability.
3. In CPA policy, an adaptive user selects a request  $x^*$  to maximize its net benefit, i.e.,  $U(x^*) - C(x^*) \geq U(x') - C(x')$ . Therefore,  $U(x^*) - U(x') \geq C(x^*) - C(x')$ . At equilibrium, any improvement in service quality is offset by the increased cost, and any decrease in cost obtained is offset by the resulting decrease in service quality. In this simulation, we assume that all the users (adaptive and non-adaptive) have the same utility function, and hence their perceived benefits and surplus can be compared.

Fig. 7-10 (a) shows that a non-adaptive user receives a higher perceived benefit (corresponding to a higher quality of service) compared to a non-adaptive user, although the absolute benefit decreases with increasing load, and also with a decrease in the proportion of adaptive users. Fig. 7-10 (b) shows that for adaptive users, the lower received benefit relative to non-adaptive

users is more than compensated by a lower cost. The cost paid by users decreases proportionally with bandwidth, while the received benefit decreases less sharply because of the opportunity benefit associated with simply holding a connection. Consequently, adaptive users receive a higher net than non-adaptive users. This may be regarded as a higher “value for money” and is an economic reward for user adaptation.

The above results seem to indicate that the performance benefits of CPA decrease as fewer users adapt, which is to be expected. The results do show that even a small proportion of adaptive users may result in a significant performance benefit and better service for the entire user population.

We should also expect CPA to have an additional inherent advantage over the FP policy even when most of the users are non-adaptive. In reality, the usage price shown in Section 5.2.2 would reflect the estimated long-term network load. The congestion price would be only used to smooth out temporary peaks, and the general usage pattern would result in optimal utilization at the offered usage price. However, a vendor charging a static price (FP) would need to charge a certain premium above this optimal price, as a risk premium, while the CPA policy allows the vendor to operate around the optimal price and use congestion pricing to protect against demand peaks.

### **7.4.3 Initial Adaptation versus Continuous Adaptation**

For a given service price and quality, an application can initially determine its optimal bandwidth requirement to maximize its perceived benefit. Some applications, such as multimedia adaptive applications [19], can also adapt their sending rate during an ongoing multimedia session. Under the RNAP framework, as mentioned in Section 4.2., users can negotiate and re-negotiate services at any time. In this

section, we compare the performance under the two scenarios: bandwidth selection only at session set-up, and ongoing bandwidth adaptation during a session. Clearly, the first scenario is sub-optimal, since the users as a group become less adaptive. We are interested in how this sub-optimality affects the performance metrics under the simulation conditions.

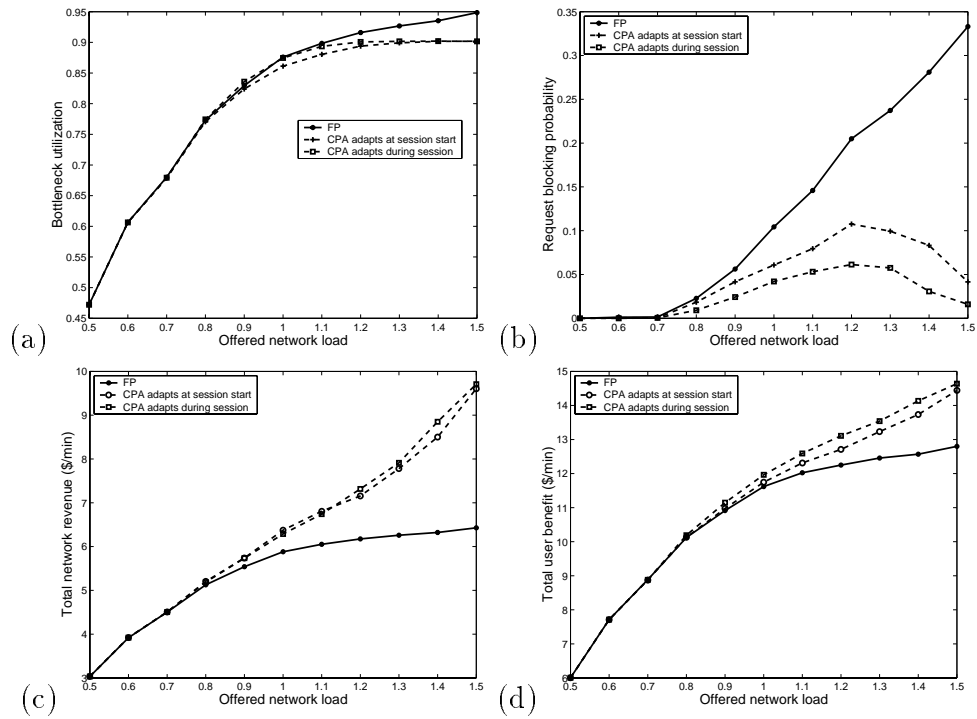


Figure 7-11: Performance when CPA users select bandwidth only at session set-up, compared with performance when they continue to adapt during the session (a) bottleneck utilization; (b) blocking probability; (c) network revenue; (d) total user benefit.

Fig. 7-11 (a) shows that initial adaptation results in a slightly lower network utilization at moderate-to-high loads, about 3-5% smaller than the utilization under ongoing adaptation. This is because if a session arrives during a traffic peak, it will request a smaller bandwidth, which will not be scaled back after the demand is driven down. Fig. 7-11 (b) shows that as expected, adaptation during a session allows for more efficient bandwidth usage and the blocking probability is reduced by half. Fig. 7-11 (c) shows that the total network revenue increases slightly with

initial adaptation, and Fig. 7-11 (d) shows a decrease in the total user benefit, arising from the higher blocking rate.

## 7.5. CPA-TAT Control Parameters

In this section, we study the impact of certain network control parameters on the network and user metrics. The parameters are: the congestion control threshold (or targeted link utilization) beyond which the congestion-dependent price component is imposed; the price adjustment factor  $\sigma$  in equation 5.9, used to control the rate at which a congested link is brought back to the targeted utilization; and the price adjustment threshold parameter  $\theta$ , defined in Section 5.2.3. The parameters are varied one at a time in the first three subsections, with the other two parameters set to the default values defined in Section 7.1.. In the final subsection, we study the effect of a *partial admission* policy in which the connection is admitted by the network even if the requested bandwidth cannot be allocated, provided the available bandwidth is greater than the minimum bandwidth requirement of the user.

### 7.5.1 Effect of Congestion Control Threshold

As shown in Figure 7-12 (a), for three different values of the threshold parameter  $\rho$ , the CPA policy limits the average link utilization to  $\rho$ .

Figure 7-12 (b) shows that the blocking rate depends strongly on the targeted utilization, decreasing by a factor of 140 when  $\rho$  decreases from 0.95 to 0.85. In general, the total network revenue does not depend strongly on the target utilization (Figure 7-12 (c)). A lower target utilization causes a lower user demand (Figure 7-13 d), but also a correspondingly higher system price (Figure 7-13 c) at the same offered load.

Figure 7-12(c) shows that at moderate loads, maximum user benefit is obtained

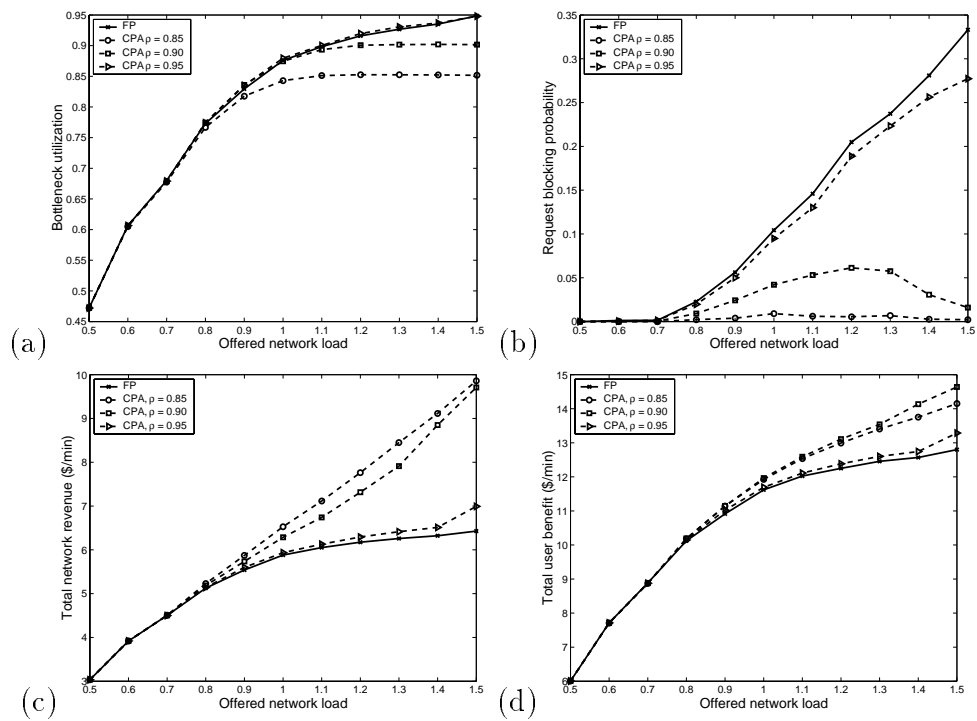


Figure 7-12: Performance of CPA and FP policies at different values of target congestion control threshold  $\rho$ : (a) bottleneck utilization; (b) blocking probability; (c) total network revenue; (d) total user benefit.

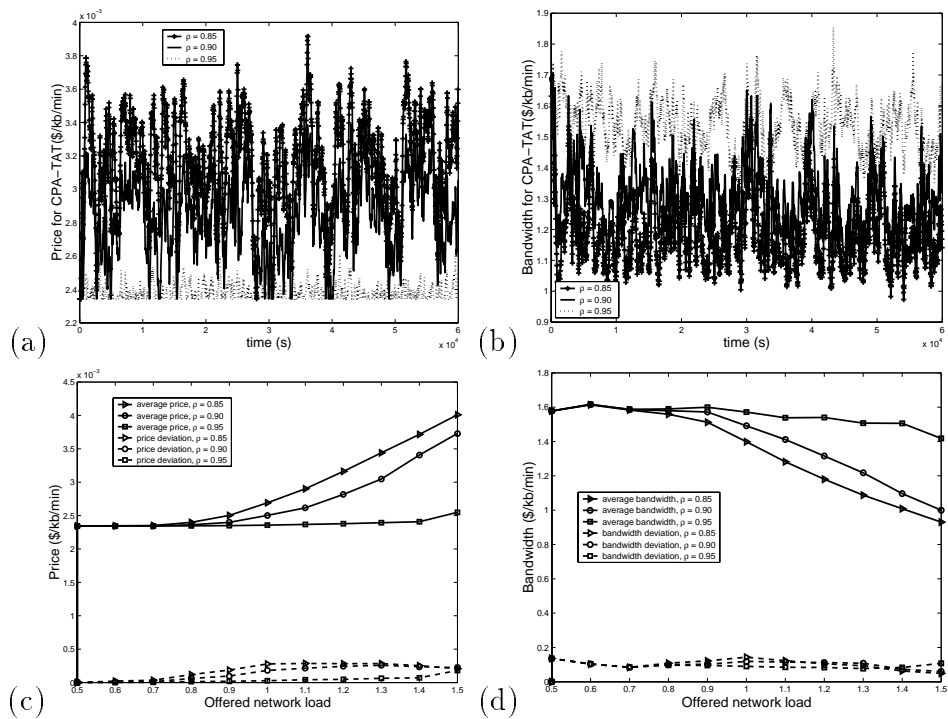


Figure 7-13: System dynamics at different values of the congestion control threshold: variation over time of system price (a), and average user demand (b), at an offered load of 1.2; time-average and standard deviation of system price (c) and average user demand (d), plotted against offered load.

for the middle value of the target utilization, 0.90. At a lower congestion threshold, the user bandwidth demand is driven too low by the congestion pricing mechanism, and at a higher congestion threshold, the blocking rate increases, and fewer users are admitted. At very high offered loads, the effect of the reduced user demand dominates, and the user benefit is highest at the highest target utilization.

As expected, Figs. 7-13 (a) and (b) indicate that both the price and user demand are adjusted more frequently with the decrease of the target utilization. Figs. 7-13 (c) and (d) show that a lower target utilization also results in a larger standard deviation of the system price and bandwidth demand due to the more aggressive congestion control.

The results in this section indicate that an appropriate target utilization should be selected - under the simulated conditions, 0.90 appears to be a reasonable value. The user benefit decreases if the target utilization is either too low or too high. Also, with too low a target, demand fluctuations are higher, too high a targeted level, results in a high blocking rate.

### 7.5.2 Effect of Price Adjustment Step

In this section, we investigate the influence of the price scaling parameter  $\sigma$  on the network performance, for  $\sigma = 0.012$ , 0.06 and 0.30.

Fig. 7-14 (a) indicates that all three values are effective in controlling the network load to the targeted level at heavy load. However, at the highest value, 0.30, the network is significantly under-utilized at moderate to high loads, indicating that the pricing algorithm is too aggressive in driving down demand below the congestion threshold. This is evident in Fig. 7-15 (c) which shows the network price to be significantly higher in the same load range for  $\sigma = 0.30$ . Fig. 7-14 (b) shows that as expected, the blocking probability decreases with increasing  $\sigma$ . The small blocking



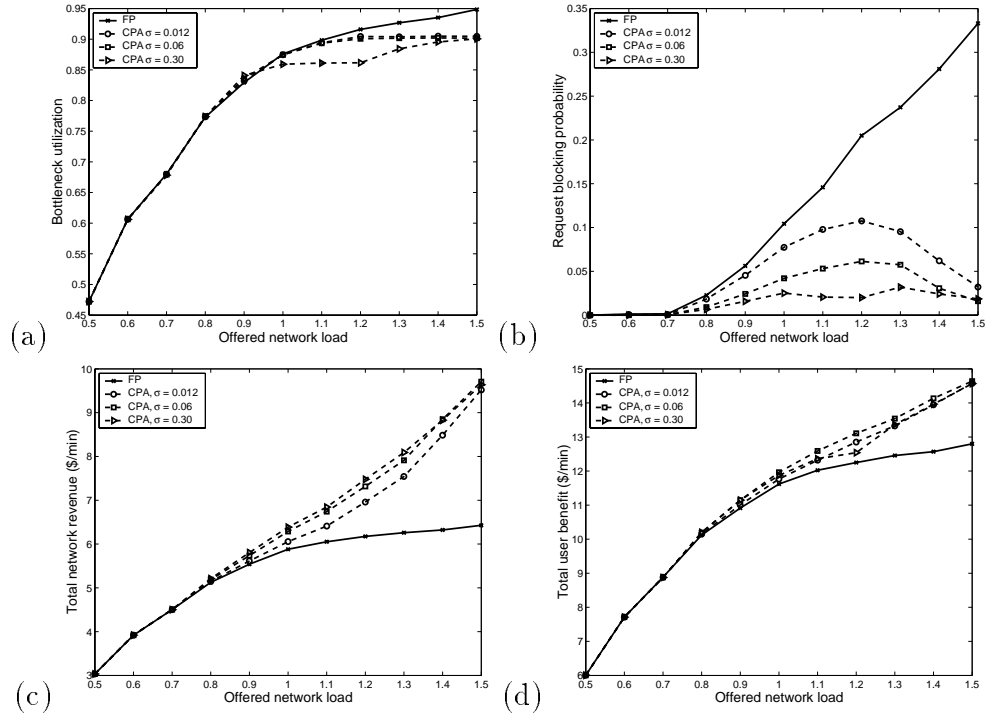


Figure 7-14: Performance of CPA and FP at different values of  $\sigma$ : (a) bottleneck utilization; (b) blocking probability; (c) total network revenue; (d) total user benefit.

probability at  $\sigma = 0.30$  cannot compensate for the under-utilization in determining the total user benefit, however, and the total benefit is significantly smaller at  $\sigma = 0.30$  than at the two lower values (Fig. 7-14 (d)). The opposing effects of a low utilization and high network price (Fig. 7-15 (c)) at moderate to high loads for  $\sigma = 0.30$  results in a slightly higher revenue compared to the two smaller values of  $\sigma$ , as shown in Fig. 7-14 (c).

Fig. 7-15 (a) and (b) shows that the price and user bandwidth are adjusted more frequently with a larger  $\sigma$ . The standard deviations of the price and average user bandwidth demand also increase progressively with a larger  $\sigma$  (Figs. 7-15(c) and (d)).

At low loads (no congestion) and very high loads, all three values of  $\sigma$  result in similar average levels of price and user demand, but at intermediate loads, the

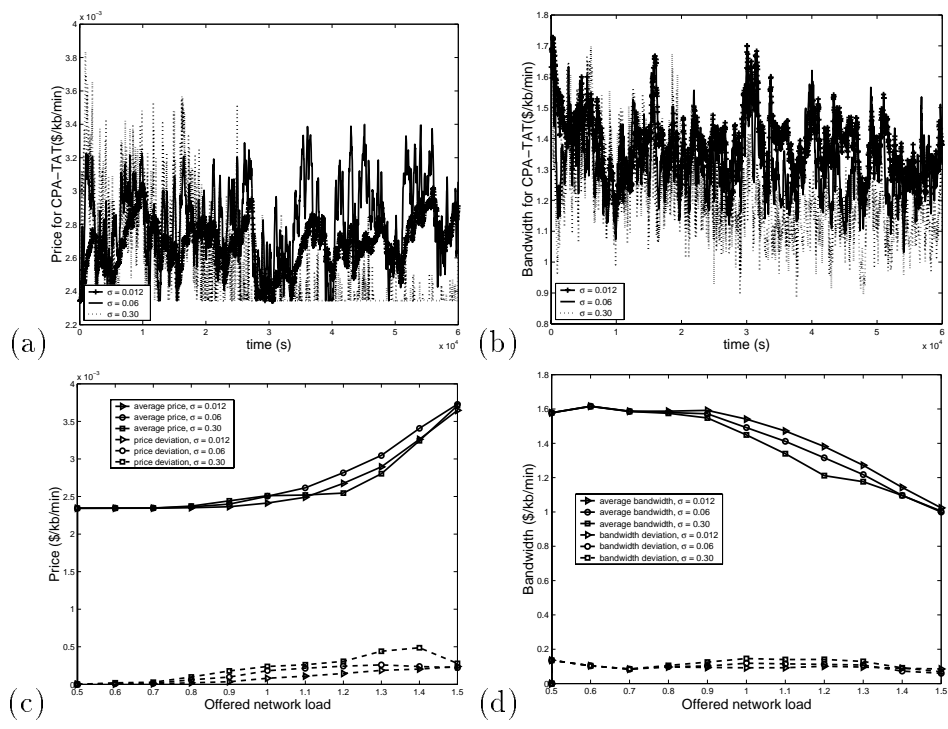


Figure 7-15: System dynamics at different values of  $\sigma$ : variation over time of system price (a), and average user demand (b), at on offered load of 1.2; time-average and standard deviation of system price (c) and average user demand (d), plotted against offered load.

highest value of  $\sigma$  results in a much higher price and lower user bandwidth demand, corresponding also to the lower utilization at these loads (Fig. 7-14 (a)).

From the results above, we see that increasing  $\sigma$  significantly reduces the blocking probability. Too large a value of  $\sigma$  results in network under-utilization at offered loads close to the target utilization, and also results in large network dynamics. Under our simulations,  $\sigma = 0.06$  appears to be roughly optimal, and  $\sigma = 0.30$  is clearly too high.

### 7.5.3 Effect of Price Adjustment Threshold

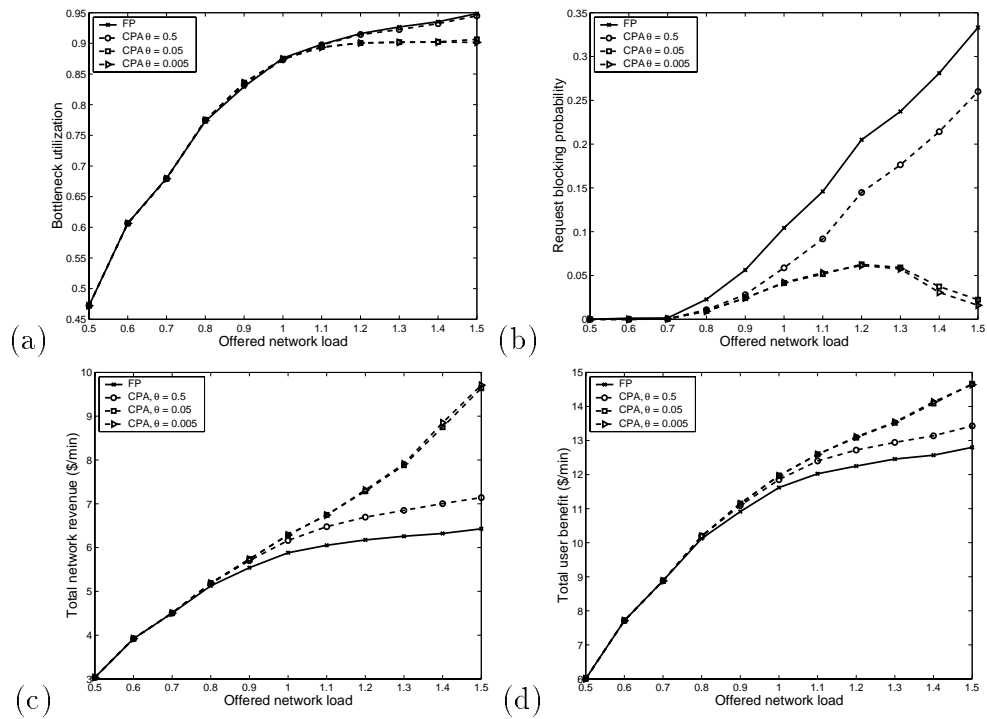


Figure 7-16: Performance of CPA and FP at different values of  $\theta$ : a) bottleneck utilization; b) blocking probability; c) average net user benefit; (d) total net user benefit.

Figs. 7-16 (a)-(d) show user and network metrics against offered load with  $\theta$  set to 0.5, 0.05, and 0.005, corresponding to progressively smaller excess demand thresholds before congestion control is activated. In all four figures, the two smaller

values of  $\theta$  correspond to very similar characteristics, except that  $\theta = 0.005$  gives a slightly more well-controlled utilization at very high loads. In general, reducing  $\theta$  to 0.005 does not result in significantly different performance compared to the default value of  $\theta = 0.05$  used in earlier experiments. With  $\theta = 0.5$ , congestion pricing and user demand adaptation are barely initiated and utilization cannot be limited to the target value (Fig. 7-16 (a)). Therefore, the blocking probability, revenue, and total user benefit are all somewhat better than the performance obtained with FP, and much worse than that obtained with the lower values of  $\theta$ .

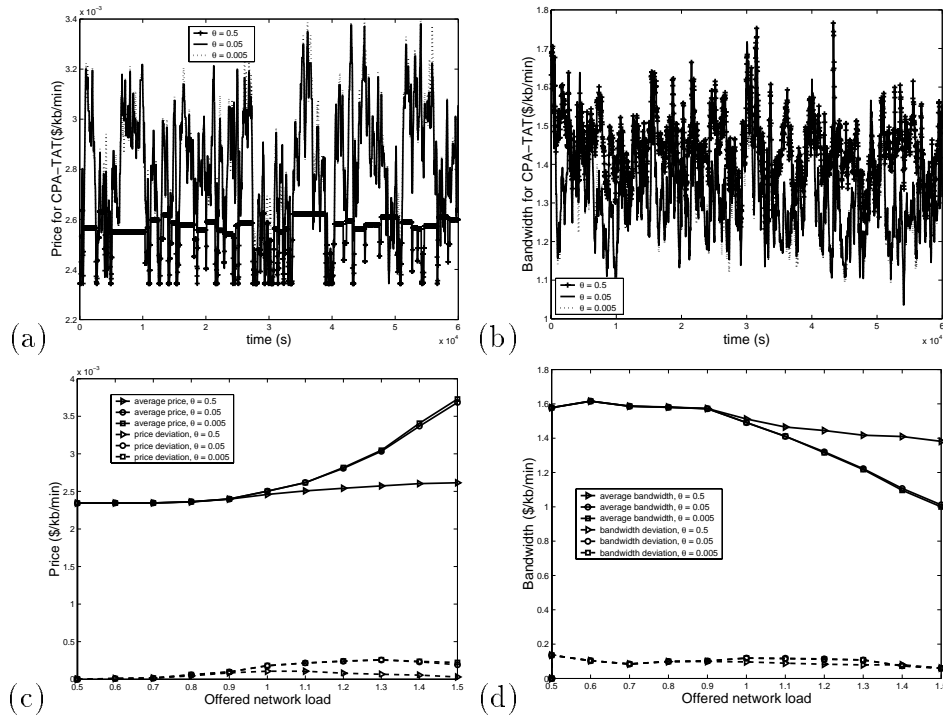


Figure 7-17: System dynamics at different values of  $\theta$ : variation over time of system price (a), and average user demand (b), at on offered load of 0.9;

#### 7.5.4 CPA-TAT with Partial Admission

In the CPA-TAT policy framework, a new user requests a certain bandwidth depending on its utility function and the current network price. The network either

admits or denies the request depending on the availability of bandwidth at each link. As described in Section 4.6., RNAP also allows the users to transmit data rate information. With respect to the sender HRN, the data rates represent the minimum and maximum sending rates the sender is willing and able to transmit. With respect to the receiver HRN, these rates indicate the minimum and maximum data rates the receiver is willing and able to receive. The minimum and maximum data rate from a sender indicate its demand scalability. If the user allows the network to intercept this information, the network can admit the user reservation request when the available bandwidth is less than the current required bandwidth, but is greater than the user's minimum requirement. We call this kind of admission *partial admission*. Since the users adapt to changes in network price continuously, after a user has received a partial admission, it is likely to obtain its fair bandwidth share (based on its utility function) as part of the adaptation process.

Figs. 7-18 (a)-(d) show that the CPA policy with partial admission does result in a significant decrease in the blocking probability (by about 30%) relative to the default CPA policy, indicating that the available bandwidth is now used more efficiently. Consequently, there is a small but noticeable improvement in utilization, network revenue and total user benefit after the onset of congestion.

## 7.6. Packet-level Simulation of DiffServ Classes

In this section, we present the results of DiffServ service simulations at the packet level, as discussed in section 7.1.. A single traffic parameter for the AF class was varied in each experiment, and its effect on CPA and FP policy performance was studied. In the first and second experiments, we vary the load burstiness and average load respectively of the AF class, and evaluate the improvements given by CPA over FP. In the third experiment, incentive driven traffic migration between classes is

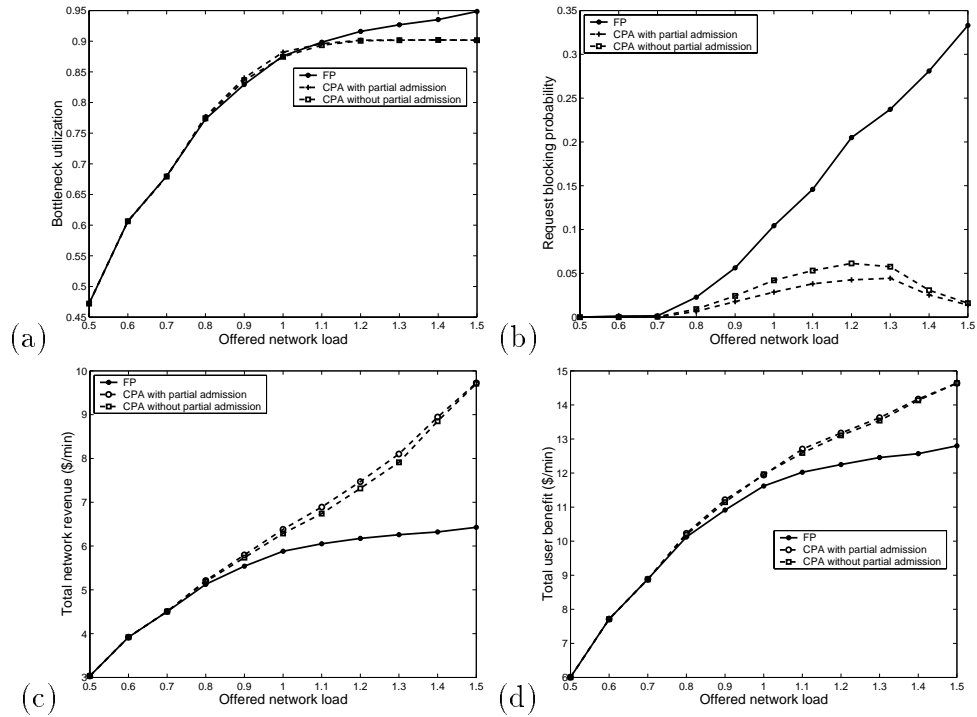


Figure 7-18: Performance of CPA with and without partial admission: (a) bottleneck utilization; (b) blocking probability; (c) network revenue; (d) total user benefit.]

shown to improve the overall system performance. In the last experiment, we show that access control to a service class is critical in maintaining expected performance levels. Combining access control with user service adaptation effectively reduces the request blocking rate.

### 7.6.1 Effect of Traffic Burstiness

We first compare the performance of FP and CPA policies as the burst index of AF class increases, at a constant average offered load of 60%.

Fig. 7-19 (a) shows that the average AF price increases under CPA due to the increasing congestion price as the burst index exceeds 0.4. In response, the AF traffic backs off. Fig. 7-19 (a) also shows that the standard deviation in the AF price increases with the burst index, indicating greater fluctuations in the price.

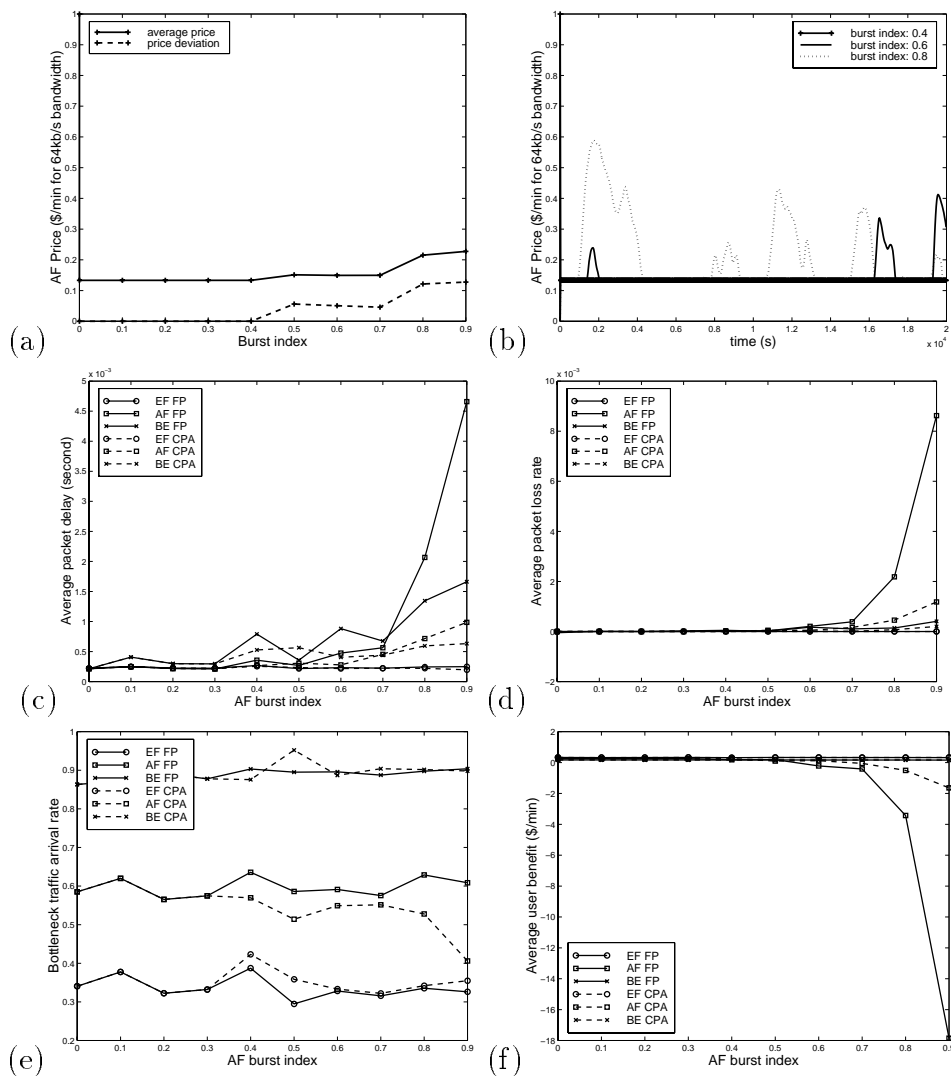


Figure 7-19: System dynamics under CPA with increase in AF traffic burst index: (a) price average and standard deviation of AF class; (b) variation over time of AF. Performance metrics of CPA and FP policies as a function of burst index of AF class: (c) average packet delay; (d) average packet loss; (e) average traffic arrival rate; (f) average user benefit.

Fig. 7-19 (b) shows the dynamic variation of the AF class price at three different levels of burstiness, confirming this trend.

Fig. 7-19 (c) and (d) show that under FP policy the average packet delay and loss of the AF class increase sharply as the burst index exceeds 0.4. As a result of the user traffic back-off under CPA the delay and loss of AF class are well controlled below the respective performance bounds of 5 ms and  $10^{-4}$  up to a burst index of 0.8. The average user benefit for CPA (Fig. 7-19 f) decreases due to the reduction of bandwidth, but remains higher than that of the FP policy. There is also a smaller degradation in the performance of the BE class at high burst indices. This appears to be because the BE class operates under a relatively high load, and therefore borrows bandwidth from the AF class when the AF class is lightly loaded. It can no longer do so when the AF traffic burstiness increases.

The results in this section indicate that the CPA policy takes advantage of application adaptivity for significant gains in network performance, and perceived user benefit, relative to the fixed-price policy. The congestion-based pricing is stable and effective.

### 7.6.2 Effect of Traffic Load

In this simulation, we keep the load and burstiness of EF class and BE class and the burst index of the AF class at their default values, and vary the offered load of AF class. The average AF price under CPA is seen to increase with offered load (Fig. 7-20 (a)). The standard deviation of the price shows an increase to a certain level and then a decrease. Initially, the price deviation increases due to the more aggressive congestion control. At heavy loads, the increased multiplexing of user demand smooths the total demand, and therefore reduces fluctuations in the price. Fig. 7-20 (e) shows that the actual arrival rate of AF under CPA backs off as users



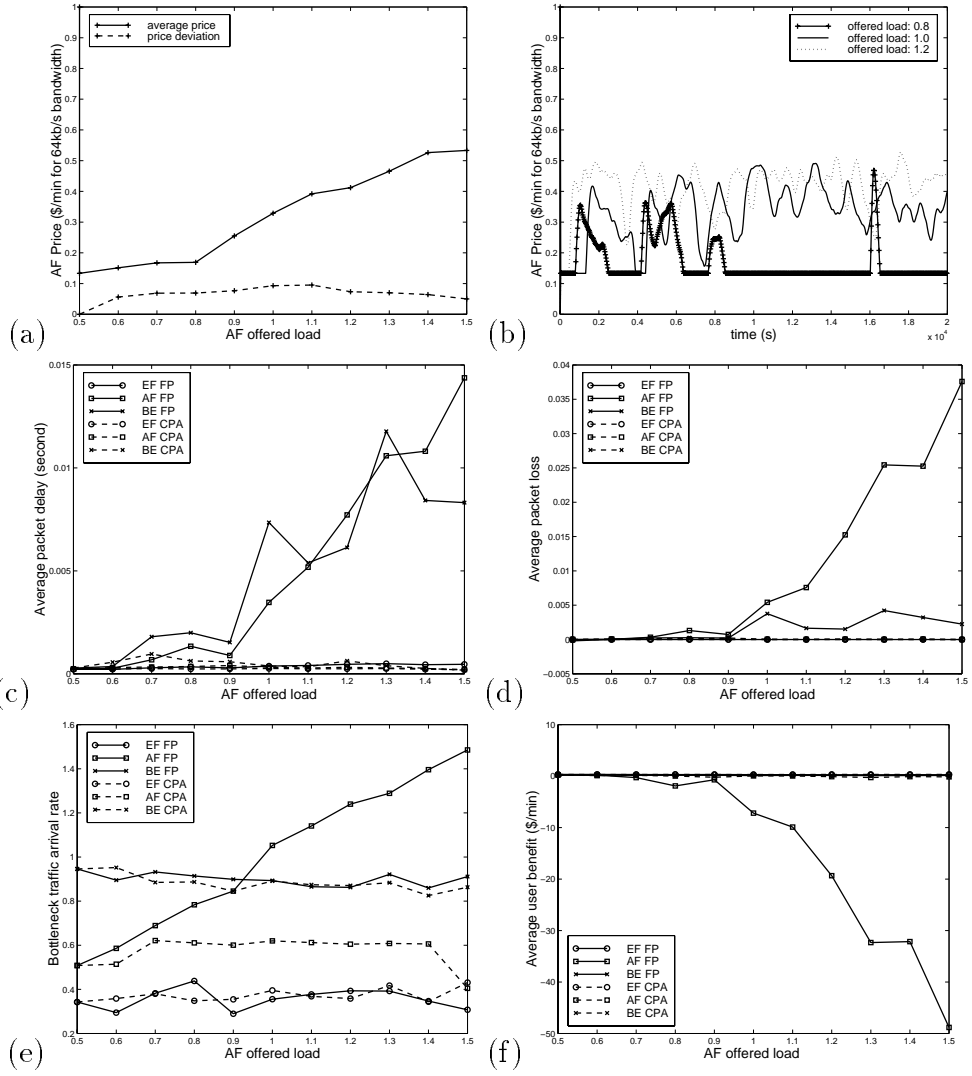


Figure 7-20: System dynamics under CPA with increase in AF offered load: (a) average and standard deviation of AF class price; (b) variation over time of AF class price. Performance metrics of CPA and FP policies as a function of AF offered load: (c) average packet delay; (d) average packet loss; (e) average bottleneck traffic arrival rate; (f) average user benefit.

adapt to the higher price.

Figs. 7-20 (c) and (d) show that the delay and loss of AF class under FP quickly increases after the offered load increases above 0.6 and approaches the provisioned capacity. As a result, the performance bounds for AF class can no longer be met. The high AF load also degrades BE performance. This is apparently because BE operates at a high load (0.9) and tends to borrow bandwidth from AF and EF when the latter classes are lightly loaded.

Figs. 7-20 (c), (d), and (e) show that CPA coupled with user adaptation is able to control congestion and maintain the total traffic load of a service class at the targeted level, and hence allows the service class to meet the expected performance bounds. Similar to our observation in Section 7.6.1, if the nominal price of the system correctly reflect long-term user demand, dynamic pricing driven service re-negotiation can effectively limits short-term fluctuations in load. Usage price of a class should be adjusted if persistent high user demand exist for a service.

### 7.6.3 Load Balance between Classes

As seen from the previous section, the performance of a class will suffer if the load into that class is too high. In general, a user under CPA policy will select a service class which provides it the highest benefit based on the price and performance parameters of a class as announced by the providers. The performance parameters are generally based on long-term statistics. In this section, we assume that a user can learn from network performance data received over a short period, and select the class that would provide the highest benefit based on the user utility function, network performance statistics and service price, as discussed in Section ??.

In this simulation, the EF and BE classes are loaded at 30% and 80% respectively. When the load of AF class increases, the performance of AF class degrades and

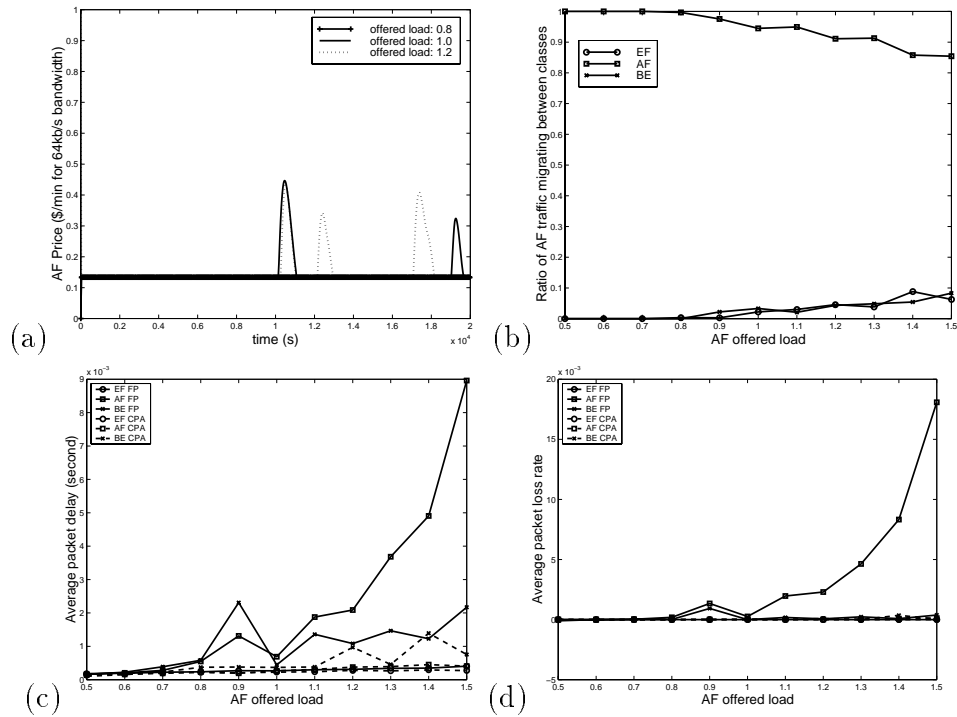


Figure 7-21: Performance metrics of CPA and FP policies with traffic migration between classes: (a) variation over time of AF class price; (b) ratio of AF class traffic migrating through class re-selection; (c) average packet delay of all classes; (d) average packet loss of all classes;

congestion price is invoked. In response, some applications switch from the AF class to the EF class, which provides better performance guarantee, or BE class, which allows it more bandwidth at a cheaper price. As the result of this re-selection, the load is better balanced across classes, and overall performance of the system improves (Fig. 7-21 (c) and (d)). Fig. 7-21 (a) shows that with load balancing in combination with adaptation within a single class, the congestion price needs to be invoked much less often than with adaptation within a class only, as in Fig. 7-20 (b). The proportion of migrating traffic is shown in Fig. 7-21 (b). We see that even when a small portion of users select other service classes, the performance of the over-loaded class is greatly improved.

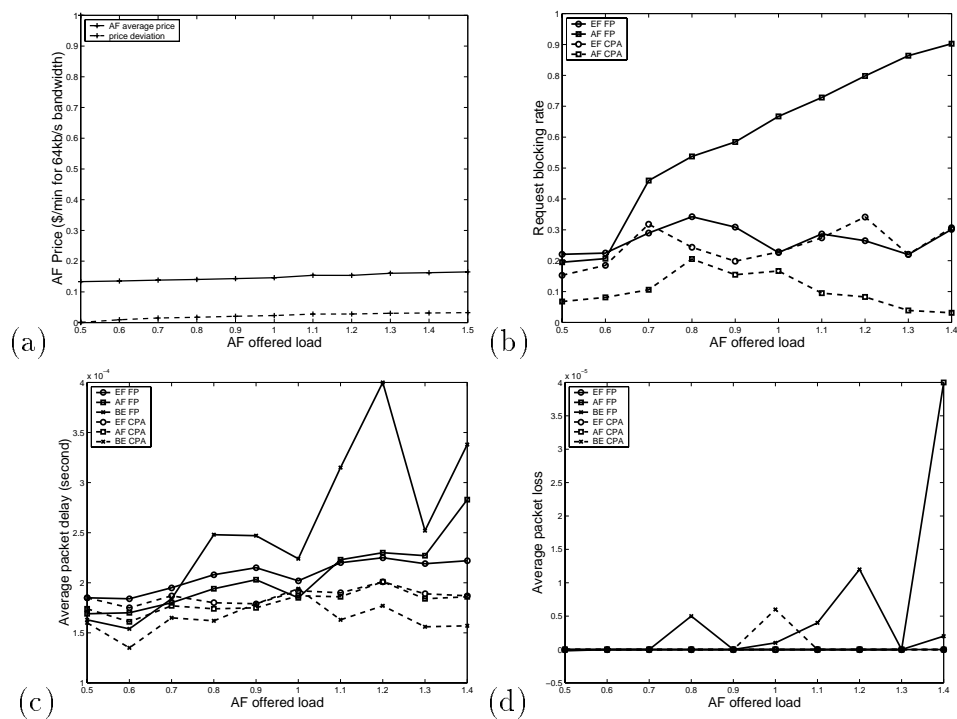


Figure 7-22: System dynamics under CPA with access control CPA as AF offered load increases: (a) average and standard deviation of AF class price. Performance metrics of CPA and FP policies with access control as a function of AF offered load: (b) user requests blocking rate; (c) average packet delay; (d) average packet loss.

#### 7.6.4 Effect of Admission Control

We have seen that the performance of a class can not be expected without any access control. In this section, we compare the performance of FP and CPA for a network with admission control for EF and AF class. The admission threshold for each class is set to 1.5 times the target load to increase the efficiency of the network.

With admission control, the performance of EF and AF classes are well controlled (Fig. 7-22 c and d). However, due to the burstiness of the traffic, the blocking rate under FP is high even at a very small offered load (Fig. 7-22 b), and increases almost linearly as the offered load increases beyond 0.6. With congestion control and service contract re-negotiation, the blocking rate of CPA is seen to be up to 30 times smaller than that under the FP policy, and actually starts to decrease after reaching a maximum at offered load 0.8. This is because the price adjustment step is proportional to the excess bandwidth above the targeted utilization and increases progressively faster with offered load at higher loads, and the user bandwidth request decreases proportionally with the price according to the general utility function of Section 6.1.5. Compared to Section 7.6.2, the average price under CPA (Fig. 7-22 a) is bounded to a smaller value at high offered loads, and has a smaller fluctuation.

The results indicate that access control is important in maintaining the expected performance of a class. However, admission control by itself may lead to a high blocking rate due to the network dynamics. By combining admission control with user traffic adaptation, the network is more efficiently used. With admission control, the dynamics of the network price can also be better controlled, so that users have a more reliable expectation of the price.

## 7.7. Other Mechanisms to Reduce Network Dynamics

Other than network parameter settings, user adaptation behavior too has an effect on the traffic dynamics seen by the user, and by the network. A user can set a minimum bandwidth adaptation increment, and communicate a new bandwidth request to the network only when the new calculated bandwidth requirement changes the existing bandwidth requirement by more than the minimum increment. This reduces the frequency of bandwidth adjustment at the cost of a sub-optimal bandwidth (in terms of perceived value). The extreme case is when the adaptation only occurs at the beginning, as is shown in Section 7.4.3.

A somewhat similar scenario can be envisioned in a core network, in which bandwidth reservation is carried out by other network providers rather than by individual users. In this case, the customer providers can change their bandwidth requests in multiples of a large block of bandwidth, only when the user flow-level demand to the customer providers changes by a certain increment. This can reduce both network dynamics and signaling overhead in the core network, and has been discussed in greater detail in [100].

## Chapter 8

### Implementation

In this chapter, we present work from two prototype implementation projects. We first describe preliminary experimental results from a simplified implementation of RNAP, based on an extension of the RSVP signaling protocol. This enables us to demonstrate many of the important features of our work. We then describe a stand-alone implementation of RNAP. This project was mainly concerned with demonstrating a complete RNAP-based negotiation framework, including user interfaces for interacting with the host and network RNAP agents.

#### 8.1. RNAP over RSVP

RNAP-over-RSVP was implemented out on a test-bed consisting of two nodes connected by a single 10 Mb/s link. An RNAP agent (LRN) was implemented at each node. Two types of service were implemented - the traditional best-effort service, and the IntServ Controlled Load service.

Although our implementation was simplified, it allowed us to demonstrate several features: the periodic RNAP negotiation process including resource negotiation and pricing and charging; the stability of the usage-sensitive pricing algorithm and its effectiveness in controlling congestion; the adaptation of user applications in

response to changes in network conditions and hence in the service price; and the effect of user utility functions on user adaptation and resource allocation.

The protocol implementation and test-bed setup are discussed in Sections 8.1.1 and 8.1.1.1, and results are presented and analyzed in Section 8.1.2.

### 8.1.1 Protocol Implementation

The RNAP *Quotation*, *Reserve* and *Commit* messages were implemented as embedded messages in the RSVP *Path*, *Resv* and *ResvErr* messages. The RNAP *Query* message was not implemented; this was not critical, since only a single service was available to the user. RNAP *Quotation*, *Reserve* and *Commit* information were embedded in RSVP *Path*, *Resv* and *ResvErr* messages. Since *Commit* messages could not easily be sent periodically in this implementation framework, the *Quotation* message carried periodic charging information (in the *Price* field) instead of the *Commit* message. The RNAP negotiation period was set to be the same as the RSVP refresh period, 30 seconds.

The sequence of messages was as follows:

1. RSVP *Path* messages, with embedded RNAP *Quotation* information are sent periodically from the sender-LRN towards the receiver-LRN. As a *Path* message passes each node, the *Price* field is updated to add the price computed at the local node and the incremental charge for the previous period.
2. The HRN at the receiver receives the *Path* message and sends a RSVP *Resv* request, with embedded RNAP *Reserve* information. The *Price* received from *Path* is copied into the *Price* field of the *Resv* message, with the *Price:HRN Data* field updated to indicate receiver information.
3. When a RSVP *Resv* request is rejected, an RSVP *ResvErr* message is sent



to the receiver HRN, with embedded *Commit* information. This information includes “bandwidth available” information in the *Price:HRN Data → Maximum Rate* field.

RSVP daemon version 4 from ISI [101] was extended to support RNAP. Resource reservation on a link was performed using Class-Based Queueing (CBQ) [102], as part of the ALTQ package [103].

Pricing was done as follows. A RSVP *Policy Element*, called the *Price Element*, was defined to hold the RNAP *Price* structure. As with other *Policy Elements*, the *Price Element* was opaque to RSVP and only understood by policy peers. The *Price Element* was embedded within the *POLICY\_DATA* objects [104][21] of *Path* messages, *Resv* messages and *ResvErr* messages.

The LRN at a node was implemented as part of the Local Policy Decision Point (LPDP) proposed in the COPS architecture [24][21]. The RNAP agent periodically computed a set of prices (for the CL service) based on traffic through a link, by monitoring the CBQ states. It also maintained state information for each RNAP session at the node. Congestion charge was levied based on the total link usage relative to the total link bandwidth.

Since the system offers only a single class of service, namely CBQ, we assume that the utility depends only on the bandwidth. In that class, delay depends on the allocated bandwidth and there is no congestion-induced packet loss.

### 8.1.1.1 Experimental Setup and Parameters

The test setup consisted of 2 routers (Ra and Rb) connected by a 10 Mb/s link, schematically represented in Fig. 8-1.

Three RNAP sessions were established end to end, and shared the same output interface of the link. To create different levels of network load, a simple data source

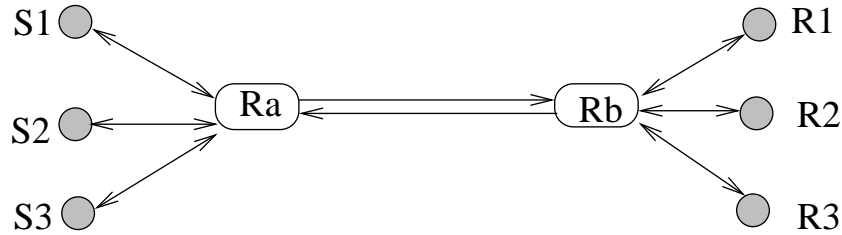


Figure 8-1: Testbed setup

model was used in each session to continuously send UDP packets. The packet generation rate was tunable to allow user adaptation.

Out of the total capacity of 10 Mb/s, 4 Mb/s was configured for CL service, The congestion threshold was set to 70% of the CL capacity (2.8 Mb/s). Background traffic was also sent using best effort service.

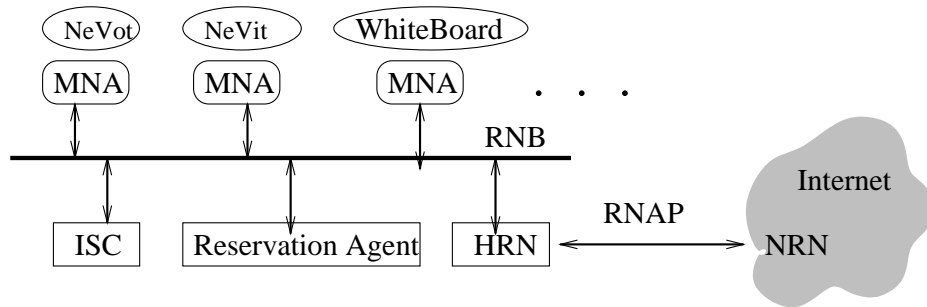


Figure 8-2: The architecture of the extended MInT system

In addition to experiments using the simple source model to generate traffic, one set of experiments was performed using traffic generated by a multimedia application - the Multimedia Internet Terminal (MInT) [105] system. The audio and video application components of MInT, NeVoT and NeViT, support rate adaptation. We extended the MInT system to couple the rate adaptation process to RNAP negotiation. Each application was represented by a Media Negotiation Agent (MNA). The MNA communicated application requirements and changes in requirements to the HRN over a Resource Negotiation Bus (RNB). The HRN was responsible for RNAP negotiation with the LRN, as well as allocation of resources (sending rates)

to the MNAs using the adaptation algorithm discussed in Section 6.2.3.

### 8.1.2 Experimental Results

We now describe a set of experiments which address the following issues: (i) the sharing of bandwidth between competing adaptive applications with identical utility functions; (ii) the sharing of bandwidth between competing applications with utility functions reflecting different amounts of elasticity in bandwidth requirements; (iii) distribution of bandwidth among applications belonging to a single-user multimedia system so as to maximize mission-wide value; (iv) the influence of specific changes in the utility function on the bandwidth adaptation; (v) adaptive behavior of audio and video applications belonging to the MInT system. In each experiment, we study the behavior of the price in response to bandwidth demand, the influence of the price in driving adaptation of user bandwidth requirements, and the “benefit” gained by the applications in terms of the surplus (or perceived value of the service relative to its cost). We ascertain that a stable and equitable distribution of bandwidth is reached in each case.

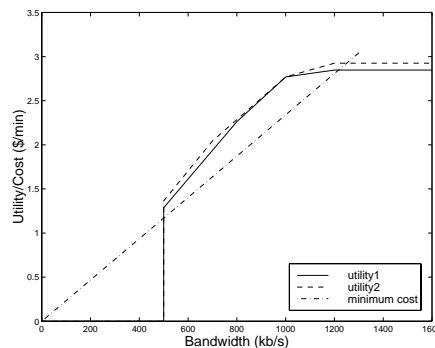


Figure 8-3: Utility functions used in the experiments of section 4.6.2 and 8.1.2.3

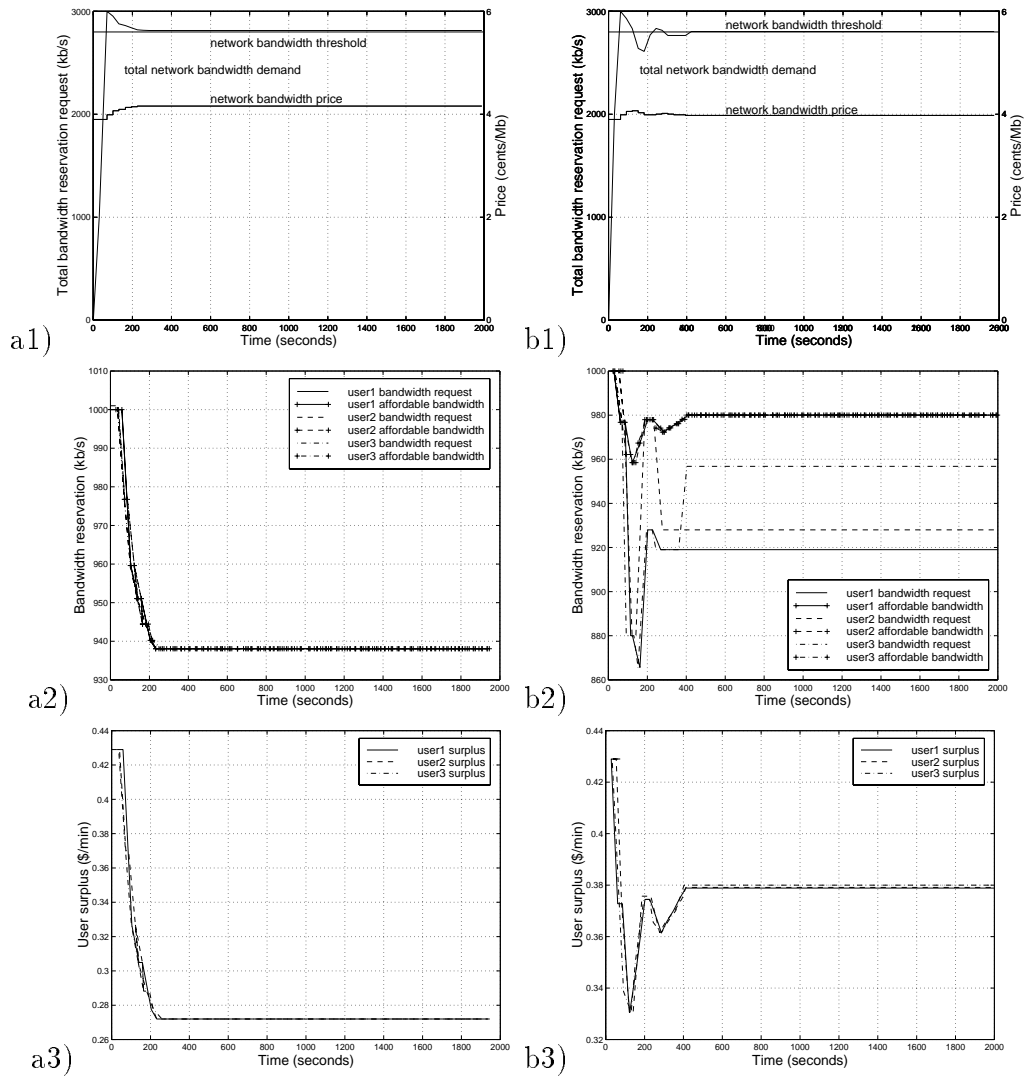


Figure 8-4: Allocation of bandwidth and surplus for three competing users sharing a link. a1, a2, and a3 show the results when the users all have the Utility 1 function from Fig. 8-3, and b1, b2, and b3 show corresponding results when the users have the Utility 2 function from the same figure

### 8.1.2.1 Bandwidth Sharing between Users

In the first experiment, we study the adaptive behavior when applications having the same utility function and belonging to different users compete for network resources. The same experiment is performed with two different utility functions, Utility1 and Utility2, shown in Fig. 8-3.

Fig. 8-4-a1, 8-4-a2, and 8-4-a3 show different aspects of adaptive behavior when Utility1 is used. Initially, in response to the initial price, each user determines that the optimal bandwidth (giving the maximum surplus) is 1000 kb/s. Since the total reservation of 3000 kb/s made by the three users is higher than the congestion threshold of 2800 kb/s, the network imposes an additional congestion price, resulting in a gradual increase in the price.

Fig. 8-4-a1 shows the initial increase in price, from 3.9 cents/Mb, until it stabilizes at 4.2 cents/Mb after about 150 seconds (corresponding to 5 negotiation periods). Fig. 8-4-a1 also shows the variation with time of the total bandwidth reservation, and Fig. 8-4-a2 shows the variation with time of the individual bandwidth reservations, and the maximum per-user bandwidth that the user budget permits. As the price increases, each user is constrained by its budget to decrease its sending rate in response. As a result, the reserved bandwidth decreases smoothly, until the link becomes un-congested, and the price stabilizes. Fig. 8-4 a3 shows a gradual decrease in the surplus obtained by each user until the price stabilizes. All users are observed to have nearly identical adaptation traces.

The second experiment uses Utility2 in Fig. 8-3. Utility2 differs from Utility1 in that the optimal bandwidth (at the initial un-congested link price) of 1000 kb/s differs only slightly from the next sub-optimal bandwidth of 700 kb/s with respect to the perceived surplus.

In Fig. 8-4b, the adaptation traces are observed to be different from that shown

in Fig. 8-4a. When the price increases, the applications are constrained by their budget to reduce their bandwidths initially. When the price increases to a certain value, the optimal bandwidth requirement for all the users (calculated at slightly different times) shifts to 700 kb/s, since the increase in cost for a larger bandwidth is higher than for a smaller bandwidth. Since the two optimal points in our example are very far apart in bandwidth, and the perceived surplus of the two bandwidths are very close, an oscillation between 2100 kb/s and 3000 kb/s was observed in the total bandwidth when this experiment was performed.

To avoid this problem, a proportional plus derivative (PD) controller [106] was used to reduce the oscillation. During each negotiation period, instead of letting the requirement jump to a new optimal bandwidth, the user shifts to a bandwidth between the current one and the optimal one, resulting in temporarily sub-optimal operation. The PD control law regulates the bandwidth request as follows:

$$r_{i+1} = \begin{cases} r_i - \alpha_0(r_i - r^*) - \alpha_1(r_i - r_{i-1}), & \text{if } \frac{|SP(r^*) - SP(r_i)|}{SP(r_i)} > \theta \\ r_i, & \text{otherwise} \end{cases} \quad (8.1)$$

where  $r^*$  is the desired optimal rate,  $r_i$  is the rate requested for negotiation period  $i$ , and  $SP(x)$  represents the surplus obtained by obtaining bandwidth  $x$ . Quicker convergence is attained by making  $\alpha_0$  large, while the overshoot is minimized by making  $\alpha_1$  large. In addition to the PD control, the bandwidth was allowed to be adjusted only if the new bandwidth led to an increase in surplus of at least  $\theta$  %. In the experiment,  $\alpha_0$ ,  $\alpha_1$  and  $\theta$  are set separately as 0.4, 0.6, and 2%, which led to the quick convergence without large overshoot.

Fig. 8-4b-2 shows that the bandwidth requirement of all three users stabilized within seven negotiation periods, with different users having different bandwidth shares. This is partly because of the asynchronous user negotiation behaviors, and partly because of the possible sub-optimal bandwidth (within  $\theta$  % of optimal) choice

of some users. All three users end up with final surplus values very close to each other (within 2 %). This is important since we consider the perceived surplus, rather than the bandwidth, as a measure of the user satisfaction.

### 8.1.2.2 Bandwidth Sensitivity and Demand Elasticity

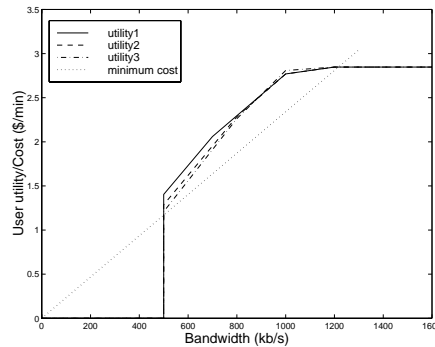


Figure 8-5: Utility functions with different bandwidth sensitivities

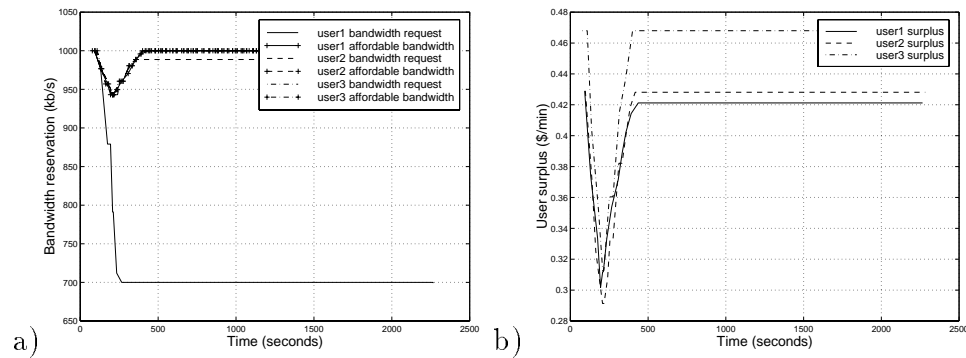


Figure 8-6: Bandwidth reservation a) and perceived surplus b) when the users have different demand elasticities

In this experiment, we study the effect of different elasticities in user demand on user bandwidth sharing and adaptation, using different utility functions (Fig. 8-5) for different users. An utility function with a smaller slope reflects a higher elasticity in the bandwidth requirement of the user. Fig. 8-6-a shows that the user with the more elastic requirement is more sensitive to price changes and reduces his resource requirement faster when the network price increases. Correspondingly, Fig.

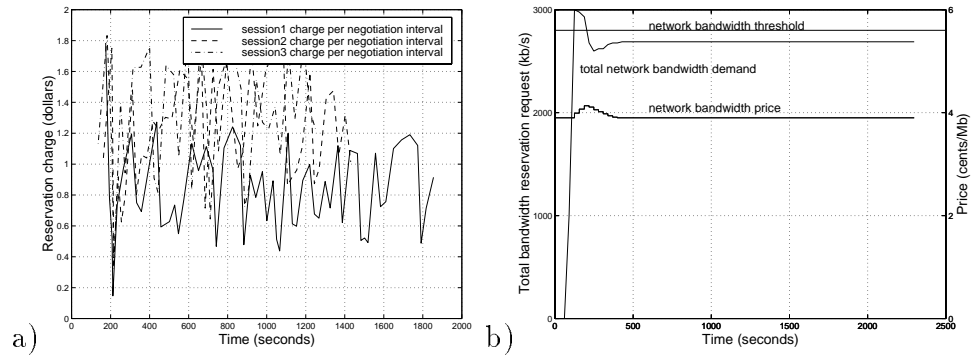


Figure 8-7: Network charges for different users a) and the total network bandwidth demand and price b) when the users have different demand elasticities

8-7-a shows that as a reward for elastic behavior, the average network charge for the more elastic user is lower, while the three users have similar perceived surplus (Fig. 8-6-b).

Thus, users with less stringent bandwidth requirements express this flexibility through a less bandwidth-sensitive utility function, and bear a greater share of reductions in bandwidth for congestion-control. Users with more bandwidth-sensitive requirements have to pay a higher charge during congestion to maintain their bandwidths at current levels.

### 8.1.2.3 Adaptation Across Media

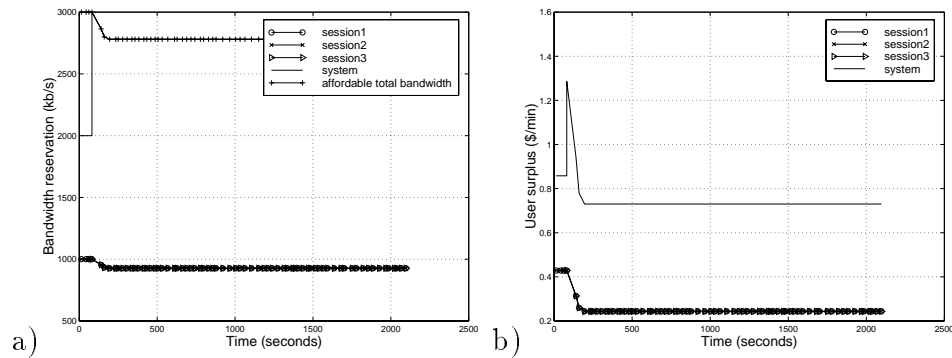


Figure 8-8: Bandwidth reservation a) and perceived surplus value b) for adaptation across media sessions in a system, all sessions having the same utility



In this section, we look at how utility functions guide the distribution of bandwidth across different media which are part of a multimedia system belonging to a single user. The results of two experiments are presented.

In the first experiment, the system consists of three media sessions, all of which have the same utility function, Utility1 shown in Fig. 8-3. When the system budget is exceeded due to congestion, the HRN adjusts the application bandwidths downwards according to the adaptation algorithm described in Chapter ?? . Since all the applications have identical utilities, the total system bandwidth is equally distributed between them at all times, as seen in Fig. 8-8a.

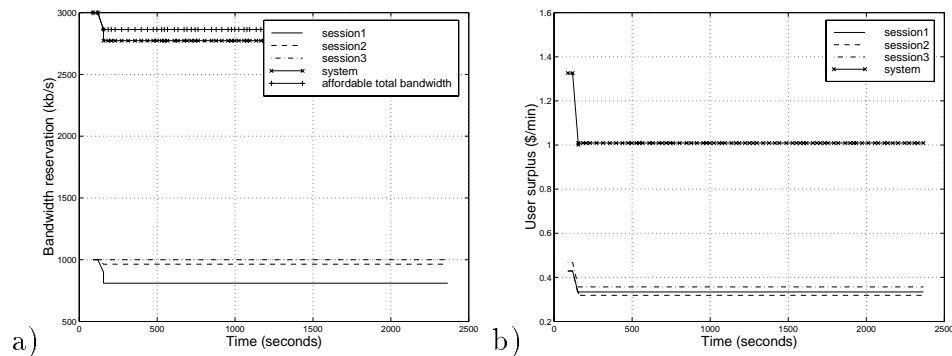


Figure 8-9: Resource reservation a) and perceived surplus value b) among sessions of a system with different bandwidth sensitivity

The second experiment is similar except that the three media sessions have different utility functions shown in Fig. 8-5. Fig. 8-9a shows that when the total optimal bandwidth requirement for all the media sessions in the system exceeds the system budget, the media session with the more elastic resource demand will be assigned relatively less bandwidth so as to maximize the overall perceived value. This is a similar result to that obtained in section 8.1.2.2 for multiple competing user applications. In effect, the system regards a media session with more elastic requirements as being more able to absorb bandwidth reductions, and “borrows” bandwidth from this session to give to other sessions.

### 8.1.2.4 Linear Operations on Utility Functions

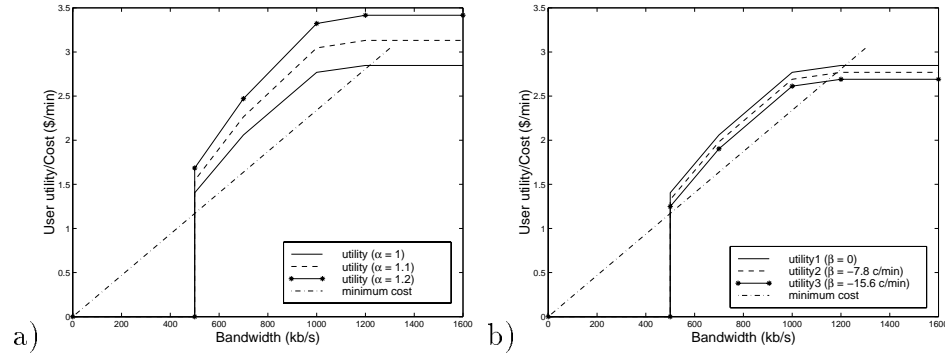


Figure 8-10: Equivalent utilities under multiplicative scaling a) and additive shifting b)

In Chapter 6, we qualitatively discussed how the shape of the user utility functions influences bandwidth selection and distribution. We now experimentally study the impact of two linear operations on the utility function, multiplicative scaling by a weight  $\alpha$ , and additive or subtractive shifting by an amount  $\beta$ . The experiment studies bandwidth distribution between multiple sessions in a system belonging to a single user, though similar results have also been observed with applications belonging to different users.

Consider three media sessions belonging to a system, all with the same basic (un-scaled) utility function (we use utility1 of Fig. 8-5). Sessions 1, 2, and 3 are assigned scaling factors of 1, 1.1, and 1.2 respectively. The resulting scaled utilities are shown in Fig. 8-10a.

Fig. 8-11 shows the variation of individual and system bandwidth allocations and perceived surpluses. Expectedly, when the adaptation is constrained by the system budget, an application with a higher  $\alpha$  gets a larger bandwidth share because of its lower elasticity of demand.

We now consider the effect of an offset applied uniformly to the utility over all bandwidths. In Fig. 8-12b, the utility1 function (which is the same as utility1 in Fig. 8-5a is shifted downwards and form utility2 and utility3. Three different

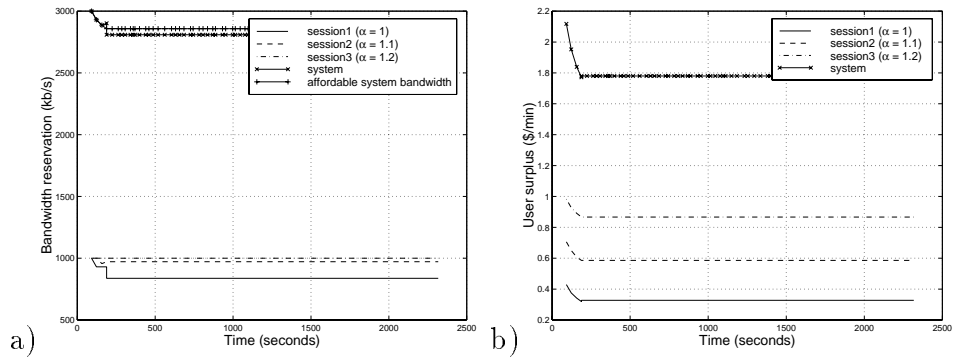


Figure 8-11: Bandwidth reservation and perceived surplus for utilities scaled multiplicatively by different amounts

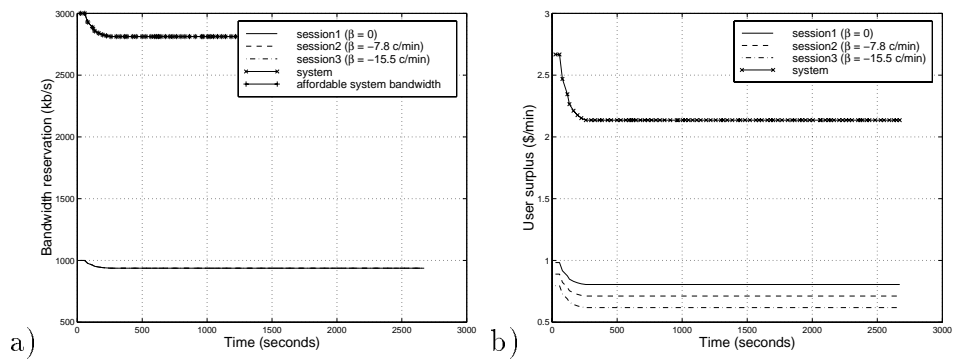


Figure 8-12: Bandwidth reservation and perceived surplus for utilities shifted additively by different amounts

sessions are assigned different utility functions.

The results shown on Fig. 8-12a shows that all three sessions are allocated the same bandwidth though Fig. 8-12b shows that the allocation results in different values of perceived surplus. This is because utility function represents the relative preference of the user for different bandwidths. The absolute value of the utility is not important - the adaptation algorithm only searches for the bandwidth with the maximum perceived value relative to its cost.

### 8.1.2.5 Adaptation in MInT

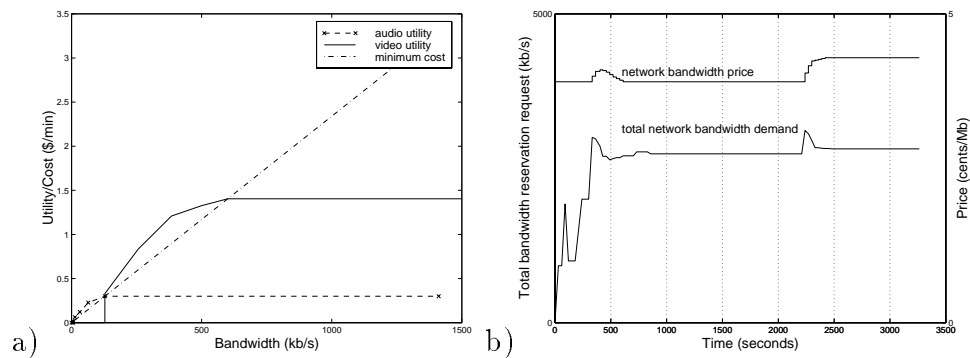


Figure 8-13: a) Audio and video utility functions used for adaptation by MInT b) Price and total bandwidth variation in the same experiment

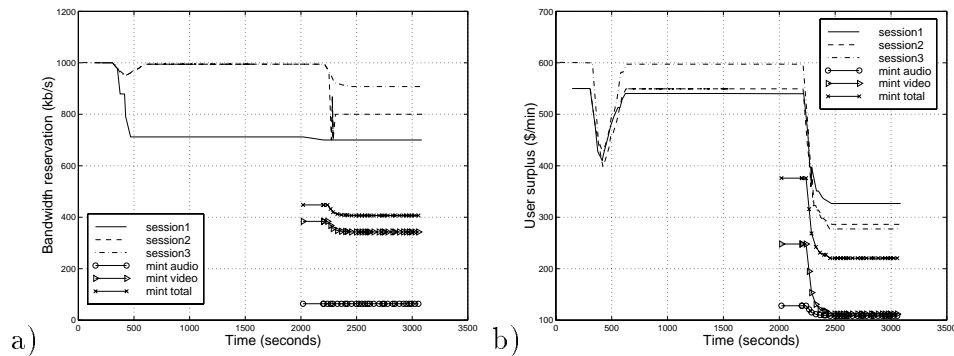


Figure 8-14: Individual bandwidth reservations and perceived surplus in the adaptation of MInT applications

Finally, we examine the adaptive behavior of the audio (NeVoT) and video

(NeViT) applications in the MInT video conference system. The utility functions for the audio and video applications are shown in Fig. 8-13a.

At the un-congested link bandwidth price, the optimal audio bandwidth for MInT is 64 kb/s, and the optimal video bandwidth is 384 kb/s. The MInT applications compete for bandwidth with three single media applications belonging to different users. The applications use the utility functions of Fig. 8-5. The three user applications are started first, and reach stability at time 630 seconds with bandwidth allocations of 712 kb/s, 994 kb/s, and 994 kb/s respectively.

At time 2000 seconds, the MInT video conference system is started, and it first requests optimal bandwidth allocation (64 kb/s + 384 kb/s). The total requested bandwidth exceeds the link congestion threshold, forcing the price up. It is observed the NeVoT bandwidth remains unchanged, and the NeViT bandwidth is reduced to 342 kb/s. The bandwidth share of the three competing user application drops to 700 kb/s, 800 kb/s and 907 kb/s respectively. User 1 has the most elastic bandwidth requirement between 700 kb/s and 1000 kb/s, and therefore initially gets a smaller share. But it is less elastic above 700 kb/s, and after the MInT applications are started, user 2, which has a relatively greater elasticity near its current allocation, reduces its requirement the most. The above experiment demonstrates the efficacy of the adaptation framework in allowing new sessions to join gracefully even when the network is highly loaded.

## 8.2. Stand-alone RNAP

In this section, we describe the prototyping of the negotiation framework by implementing the RNAP protocol stand-alone, instead of embedding it in another signaling protocol such as RSVP. The purpose of this prototyping is to demonstrate the functionality of a complete resource negotiation framework, including RNAP,

the proposed pricing model, and user adaptation model.

### 8.2.1 Architecture

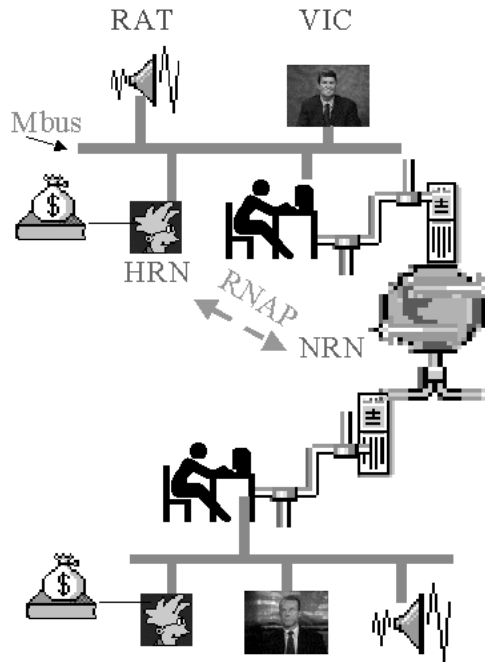


Figure 8-15: Testbed setup for stand-alone RNAP implementation

Fig. 8-15 represents the prototype set-up. A tele-conference system runs on each end user site, and a Host Resource Negotiator (HRN) negotiates with the network on its behalf using RNAP. The network supports DiffServ with EF and a few AF classes. A Local Resource Negotiator is implemented at each router. We now describe the system components in more detail.

#### 8.2.1.1 Host System Architecture

The host teleconference system consists of the Mbone tools VIC (Video Conferencing Tool, version 2.8) and RAT (Robust Audio Tool, version 4.2.6) [107]. The HRN communicates with VIC and RAT through a Message Bus (Mbus)[108]. It allocates

total the user bandwidth among the applications, and directs them to adapt sending rates based on allocated bandwidth.

RAT is an open-source audio conferencing and streaming application that allows users to participate in audio conferences over the Internet. These can be between two participants directly, or between a group of participants on a common multicast group. RAT is based on IETF standards, using RTP above UDP/IP as its transport protocol, and conforming to the profile for an audio and video conference with minimal control. RAT features a range of different rate and quality codecs, receiver based loss concealment to mask packet losses, and sender based channel coding in the form of redundant audio. The codecs supported by RAT include: G.711 PCM  $\mu$ -Law (64kb/s), G.711 PCM A-Law (64kb/s), Wide-Band ADPCM (64kb/s), G.726 ADPCM (16-40kb/s), DVI ADPCM (32kb/s), Variate Rate DVI ADPCM ( 32kb/s), Full Rate GSM (13kb/s), LPC (5.6kb/s), etc.

The UCB/LBNL video tool, VIC, is a real-time multimedia application for video conferencing over the Internet. VIC was designed with a flexible and extensible architecture to support heterogeneous environments and configurations. In high bandwidth settings, multi-megabit full-motion JPEG streams can be sourced using hardware assisted compression, while in low bandwidth environments like the Internet, aggressive low bit-rate coding can be carried out in software. Vic also runs on top of RTP, supporting both point-to-point and multiparty conferencing applications.

The Mbus is an enabling technology for modular system design. It is intended for efficient communication between multimedia conferencing software components and coordination of several system models.

### 8.2.1.2 Network Infrastructure

The network uses FreeBSD based routers, with ALTQ v2.2 [103] was used for traffic management, and CBQ extended for DiffServ support. An LRN is implemented at each router. It interfaces and controls the traffic control module of the ALTQ system. The RNAP protocol is used to establish flows through the network. RNAP messages are sent by the HRN with router alert[14][24] IP option turned on. Intermediate routers pick up the router alert packet, process them, and update the price information contained in RNAP packets.

In this section, we describe how network resources are managed by an LRN, and discuss resource partitioning, DiffServ support, admission control, and price and charge computation.

#### Resource Partitioning

Multiple service classes with different QoS specifications are supported by the network. Shared resources such as link bandwidth and router buffer space are partitioned among the multiple service classes. In the experiment, we only consider link bandwidth. For a router with  $N$  output interfaces supporting  $M$  service classes, we denote a service at an interface  $i$  with service class  $j$  as  $s^{ij}$ .

#### DiffServ Support

The testbed network supports DiffServ with EF and two AF classes. We implemented only two AF service class for simplicity, as the results are independent of the number of service classes. The LRN at the edge of a domain performs traffic monitoring and marking using the contracted service parameters. A multi-field classifier is used to distinguish between flows from different users. The Monitoring of flows and the marking of flow-packets are specific to the service class, and are described in Section 8.2.1.3.



Internal routers do not maintain per-flow states, and apply forwarding rules to behavior aggregates based on packet markings. Internal routers also measure aggregate traffic for admission control and price computation.

### Admission Control

To perform admission control for new arrivals, a LRN needs to estimate the amount of bandwidth being used by existing flow-aggregates. A time-window scheme is used to measure and predict the bandwidth usage. In an interval or a window of  $T$  seconds,  $L$  samples are taken for the actual bandwidth usage of the active flows, with sampling interval of  $q = T/L$  seconds. If the sampling interval is too small, the estimation may not be accurate when the traffic is too bursty. Traffic is estimated every  $T$  seconds based on the  $L$  samples of the corresponding interval and the bandwidth prediction of the previous interval. Let  $u_1^{ij}(k), u_2^{ij}(k), \dots, u_l^{ij}(k), \dots, u_L^{ij}(k)$  represent the  $L$  samples of the bandwidth usage of service  $s^{ij}$  at interval  $k$ , and  $bw_p^{ij}(k)$  represent the predicted bandwidth, the predicted bandwidth for interval  $k + 1$  can be decided using the following steps:

1. Look for the upper bound of the load during interval  $k$ , and find the moving average of the measured traffic upper bound and the bandwidth prediction of interval  $k$ .

$$bw_p^{ij}(k+1) = \begin{cases} \mu^{ij}bw_p^{ij}(k) + (1 - \mu^{ij}) \max[u_1^{ij}(k), u_2^{ij}(k), \dots, u_L^{ij}(k)], & 0 \leq \mu^{ij} \leq 1 \quad \text{if } k \geq 1 \\ 0, & \text{if } k = 0 \end{cases} \quad (8.2)$$

2. When the actual sampled bandwidth is greater than the predicted bandwidth at an interval, the predicted bandwidth will be corrected and set to the actual sampling bandwidth. During an interval  $k + 1$ , if any sample  $u_l^{ij}(k + 1) >$

$bw_p^{ij}(k+1), 1 \leq l \leq L$ , set  $bw_p^{ij}(k+1)$  to  $u_l^{ij}(k+1)$ .

Whenever a RNAP *Reserve* request arrives, the LRN performs an admissibility test to decide whether the flow can be admitted without affecting QoS guarantees provided to the existing flows. The bandwidth availability test is based on the predicted utilization, and the new flow's traffic profile. Various algorithms [4][23] exist to compute the bandwidth to be reserved for the new flow based on its traffic profile. We use the approach adopted by *measured sum*. If  $r$  is the resource request from a new arrival during interval  $k$ , the flow will be accepted if  $r + bw_p^{ij}(k) \leq bw_t^{ij}$ , with  $bw_t^{ij}$  representing the targeted bandwidth admission threshold for service  $s^{ij}$ . The target bandwidth can be as high as the total allocated bandwidth for service  $s^{ij}$ , but a lower target is usually used to protect against traffic bursts.

### Price and Charge Computation

The pricing strategy of Chapter 5 was implemented at each router. Price parameters can be set independently for different interfaces and routers. Assuming  $p_{basic}$  be the basic rate for fully used bandwidth, and  $\rho^{ij}$  be the expected load of a service  $s^{ij}$ , the usage price  $p_u^{ij}$  for service  $s^{ij}$  is given by  $p_{basic}/\rho^{ij}$ . The holding price  $p_h^{ij}$  can then be represented as:

$$p_h^{ij} = \alpha^{ij}(p_u^{ij} - p_u^{ij-1}) \quad (8.3)$$

With the targeted congestion control bandwidth for service  $s^{ij}$  set as  $bw_c^{ij}$ , congestion price is applied whenever the predicted bandwidth  $bw_p^{ij}$  exceeds  $bw_c^{ij}$ . The congestion price for an interval  $k$  is calculated as:

$$p_c^{ij}(k) = \min[\{p_c^{ij}(k-1) + \sigma^{ij}(bw_p^{ij}, bw_c^{ij}) * (bw_p^{ij} - bw_c^{ij})/bw_c^{ij}, 0\}^+, p_{max}^{ij}] \quad (8.4)$$

Users do not need to understand the pricing structure, and only the total price is communicated to the users. The total data volume transmitted during an interval is measured at the entry point to the network by the corresponding border LRN, and used to calculate the charge for the interval.

### 8.2.1.3 Testbed Setup

The testbed setup consists of two routers (Ra and Rb) connected by a 10 Mb/s link, schematically represented in Figure 8-15. The output bandwidth of routers Ra and Rb are partitioned into the three service classes, EF, AF1x, and AF2x. For different experiments, the network traffic consists of the output of the end host multimedia systems, and/or traffic consisting of multiple UDP packet-flows generated by multiple instances of a simple data source model. Each flow is established end-to-end by its own RNAP session, and share the same output interface of the link. The packet generation rate was tunable to allow user adaptation. Out of the total capacity of 10 Mb/s, 15% was set-aside for Expedited Forwarding, and 30% each for AF2x and AF1x. The congestion threshold was set to 60%, 65% and 70% for EF, AF2x and AF1x respectively. A two-rate token bucket meter was used to police the EF classes at edge routers. The marking action was to mark the EF packets with the EF code point for confirming flows, and dropping them for non-confirming flows. For the AF classes, a two-rate three-color maker was used for conditioning and policing. Three different code points were used to mark the AF traffic. Re-negotiation is carried out once every 60 seconds. Each flow is assumed to last for a period of 20 negotiation sessions. If a flow is rejected, the user waits for a random amount of time (uniformly distributed between 1 to 10 seconds), before retrying. Background best-effort traffic is also generated. An average of 30 flows for each service class is targeted over the entire experiment duration.

## 8.2.2 User Interface Design

User interfaces have been implemented at the user site and at a network site. The interface at user site allows a user to set up and modify its profile, and monitor the performance of its applications. The interface at the network site allows a provider to monitor the states of different routers. In this section, we show some of the interfaces in the prototyping.

### 8.2.2.1 GUI at User Site

#### Main Window

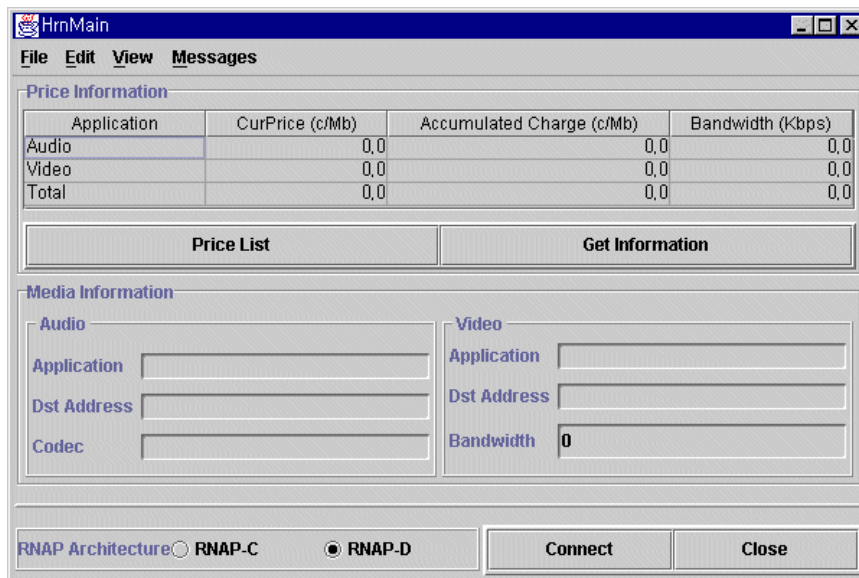


Figure 8-16: The main window for user profile configuration

The main window summarizes the state of all the user applications, VIC and RAT in these experiments. It indicates the current price, accumulated charge, and average bandwidth of all the applications.

#### User Profile Windows

The profile configuration windows allow a user to distribute its budget among all the applications in a system, configure the parameter individually for its applications. For an audio application, the rate adaption is through changing codecs, and hence the user indicates preferences by setting appropriate codecs. For a video application, the rate can be adjusted smoothly between user minimum required rate and maximum configured rate.

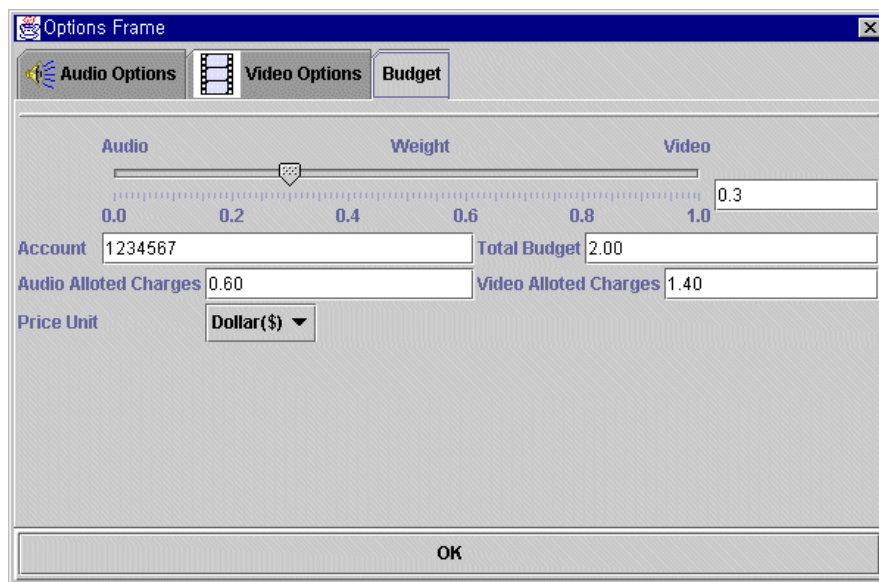


Figure 8-17: The user budget distribution window

## The Network and Application State Window

Figure 8-20 lists the price and statistics of difference services supported by the network, and the states for active user applications. The price and service statistics will facilitate a user to select the service that provide it the highest perceived value.

## User Application Running State Window

Figure 8-21 shows the variation of resource allocation, price, and user charges with time.

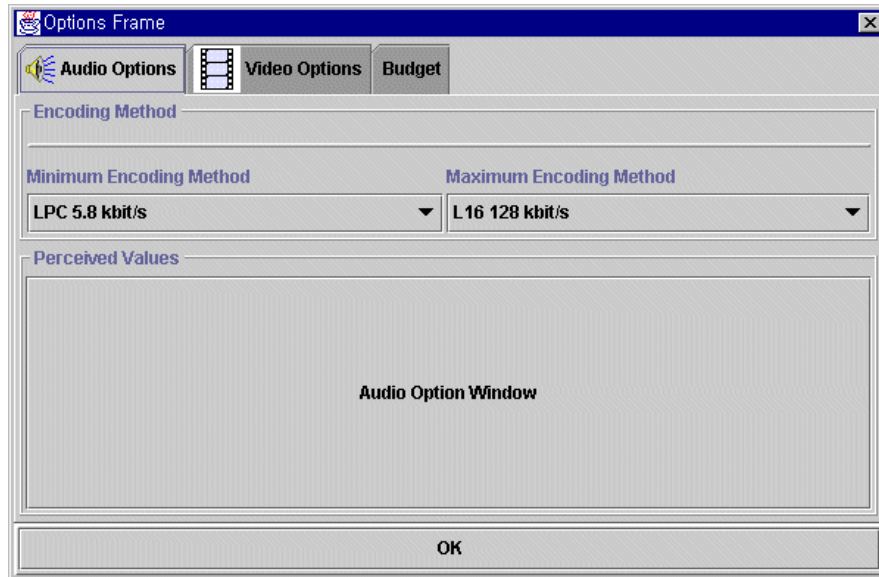


Figure 8-18: The audio profile window

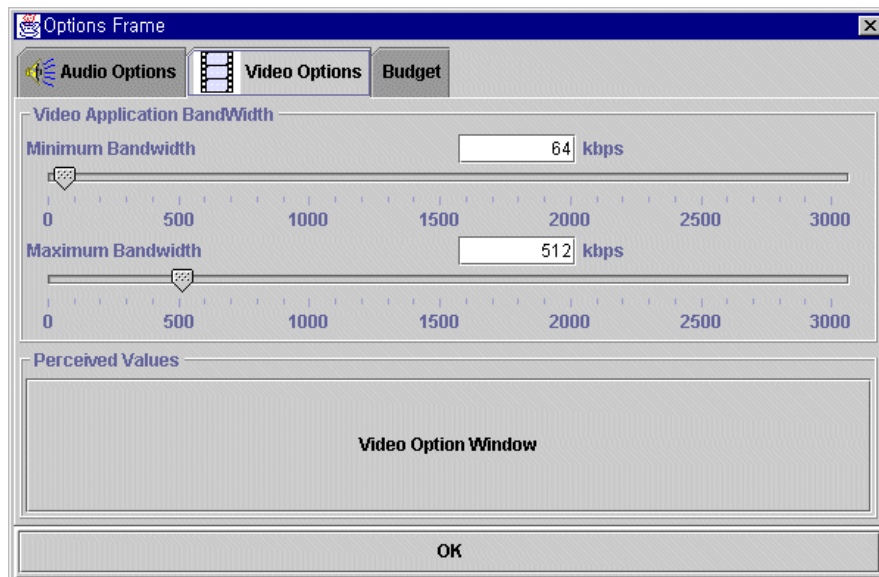


Figure 8-19: The video profile window

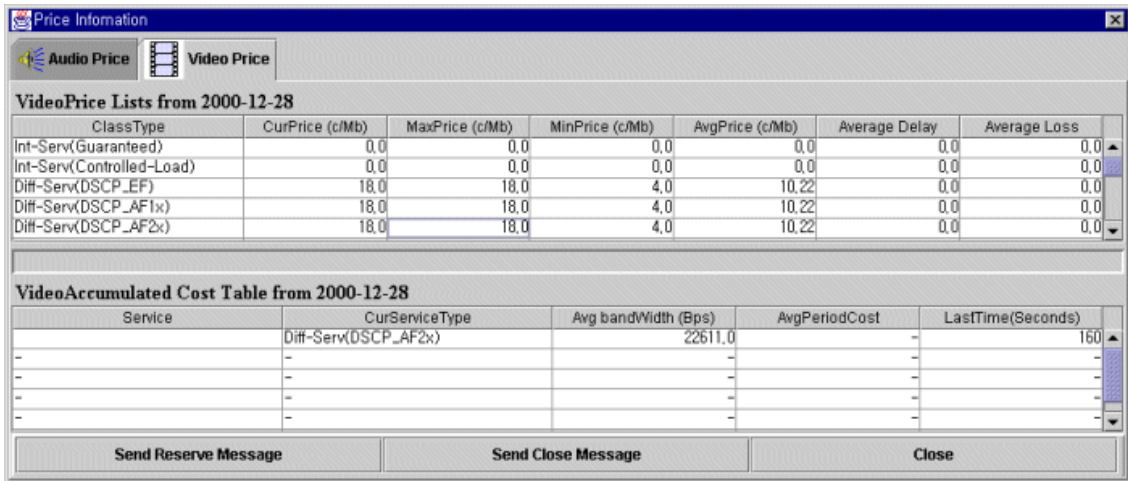


Figure 8-20: The service and application list window

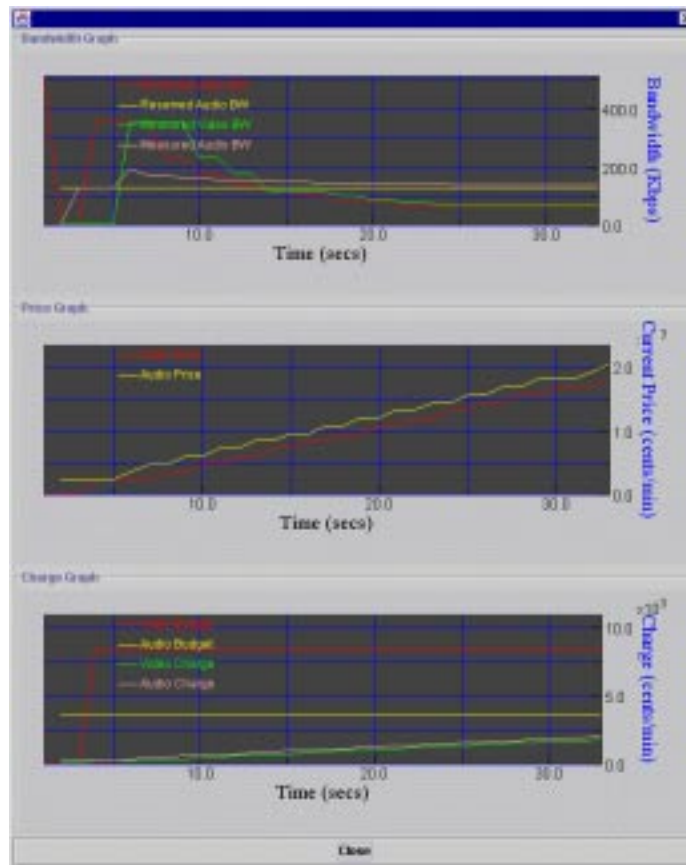


Figure 8-21: The application state window

### 8.2.2.2 GUI for the Network

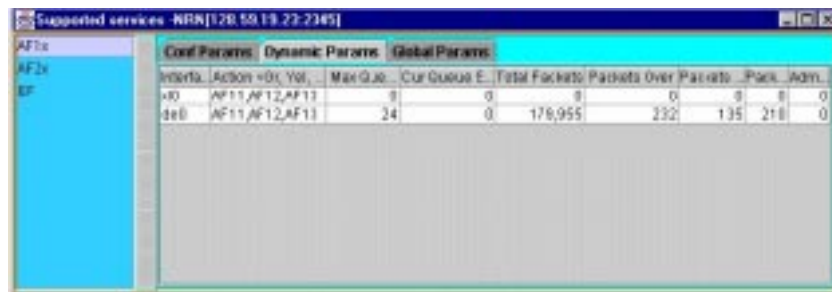
#### Information for Different Services



The screenshot shows a window titled 'Supported services - NRR[128.58.19.22.2345]'. It displays a table with three tabs: 'Conf Params', 'Dynamic Params', and 'Global Params'. The 'Conf Params' tab is active, showing a table with columns: 'Interface', 'Bandwidth', 'Targets', 'Unit Price', 'Holding Pk.', 'Congest. A.', and 'Pricing unit'. The data rows are:

Interface	Bandwidth	Targets	Unit Price	Holding Pk.	Congest. A.	Pricing unit
x0	1,000,000	85	3.524	0.523		2/green-yellow...
ge0	1,500,000	75	4	0.3		0.173/green-yellow...

Figure 8-22: Configuration of different services



The screenshot shows the same window as Figure 8-22, but with the 'Global Params' tab active. It displays a table with columns: 'Inertia', 'Action', 'Max QoS', 'Cur QoS', 'E.', 'Total Packets', 'Packets Over', 'Packets ...', and 'Pack. Adm.'. The data rows are:

Inertia	Action	Max QoS	Cur QoS	E.	Total Packets	Packets Over	Packets ...	Pack. Adm.
x0	AF11/AF12/AF13	8	0	8	0	8	8	0
ge0	AF11/AF12/AF13	24	0	176,955	232	135	218	0

Figure 8-23: Service statistics

Fig. 8-22 shows the configuration parameters and policing behavior for different services and Fig. 8-23 shows service states, for all the router interfaces of a domain.

#### Flow States

Figure 10-5,8-25,8-26 show the active flow states, including service contract, statistics, and charging information, at the edge of the domain.

#### States of Output Interfaces of a Router

Fig. 8-27 shows the resource provision and parameter configuration for different service classes at a router interface. The resource usage is monitored and the price is updated, as shown in Fig. 8-28.



The screenshot shows a window titled "Active flows - NRN[128.59.19.23:2345]". On the left, a list of IP addresses is shown, with "[192.168.3.11,40002]" selected. The main area has three tabs: "Flow Details", "Service Params", and "Dynamic Params". The "Flow Details" tab is active, displaying a table with two columns, A and B.

	A	B
Source		[192.168.3.11,40001]
Destination		[192.168.4.11,40001]
Protocol		UDP
In Interface		x10
Out Interface		de0
Service Class		FF
Policing type		in-out
Current State		Commt
Active Time (secs)		354

Figure 8-24: Flow profile

The screenshot shows the same window as Figure 8-24, but with the "Service Params" tab selected. It displays a table with two columns, A and B, showing various service parameters.

	A	B
Bucket Rate (bps)		128000.0
Bucket Size (bytes)		16000.0
Peak Rate(bps)		256000.0
Min Packet Size (bytes)		256
Max Packet Size (bytes)		256
Green Action		FF
Yellow Action		N/A
Red Action		DROP
Current Price (cents/MBit)		6.5

Figure 8-25: Service statistics of a flow

The screenshot shows the same window as Figure 8-24, but with the "Dynamic Params" tab selected. It displays a table with two columns, A and B, showing accumulated charging information.

	A	B
Accumulated Charge (cents)		879.043
Charge Current Cycle (cents)		0.832634
Green <Packets, Bytes, Current Demand (bps)>		13314,381676E,147456.0
Yellow <Packets, Bytes, Current Demand (bps)>		883,452096,0.0
Red <Packets, Bytes, Current Demand (bps)>		0,0,0.0

Figure 8-26: Accumulated charging information for a flow

Class	Bandwidth (bits)	Targeted Rate (%)	Unit Price (c/Mb)	Holding Pric...	Congestion...
AF1x	2,500,000	75	4	0.8	3.744
AF2x	2,500,000	65	4.615	0.185	4.714
EF	1,500,000	60	5	1.5	4.243

Figure 8-27: Service configuration at an interface

Class	Targeted bandwidth...	Estimated deman...	Measured d...	Congestion P...	Price (c/Mb)
AF1x	1,875,000	1,602,960	1,046,850	4.537	9.337
AF2x	1,625,000	1,255,140	1,076,360	4.702	9.502
EF	900,000	791,339	535,373	4.248	10.748

Figure 8-28: Service statistics at an interface

### 8.2.3 Results

In this section, we show some of the experimental results.

Fig. 8-29 shows the traffic utilization for AF1x. We see that the target utilization has been successfully achieved at various traffic loads.

Fig. 8-31 shows the behavior of non-adaptive applications with the variation of network load. As network load increases, the user request blocking rate is very high.

Fig. ?? shows several aspects of adaptive behavior. As more flows are added to the system, the measured usage increases. When the measured usage exceeds the targeted rate, the system applies the congestion price. Initially, the blocking rate is high until the user responds to the congestion. Eventually, the blocking rate reduces and can even be zero as users adapted their sending rate in response to increased prices. After the network reaches a stable state, the utilization of the system is maintained at the target level, and price is seen to be stable.



Figure 8-29: Measured traffic as compared with target utilization

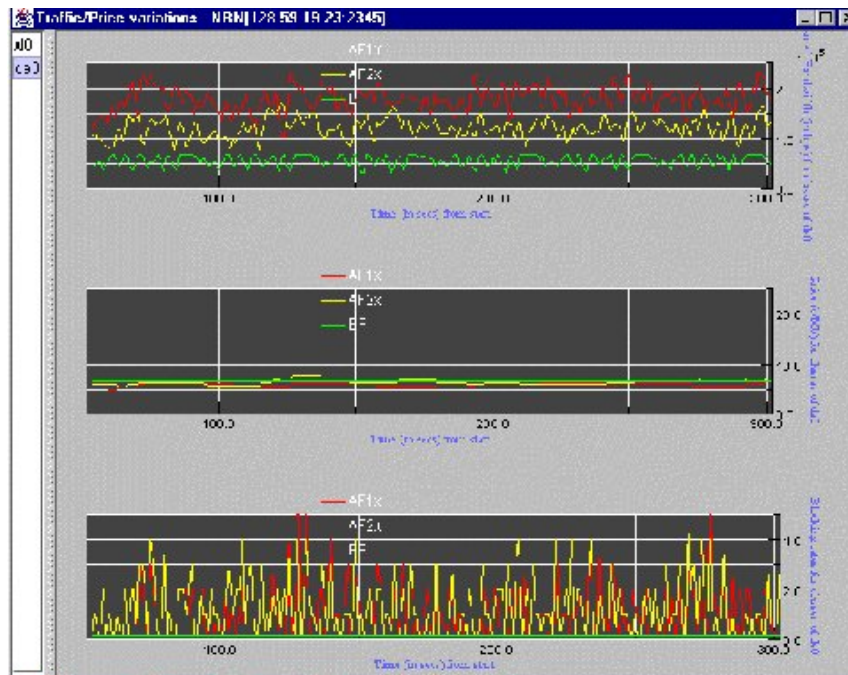


Figure 8-30: Service statistics when applications are not adaptive

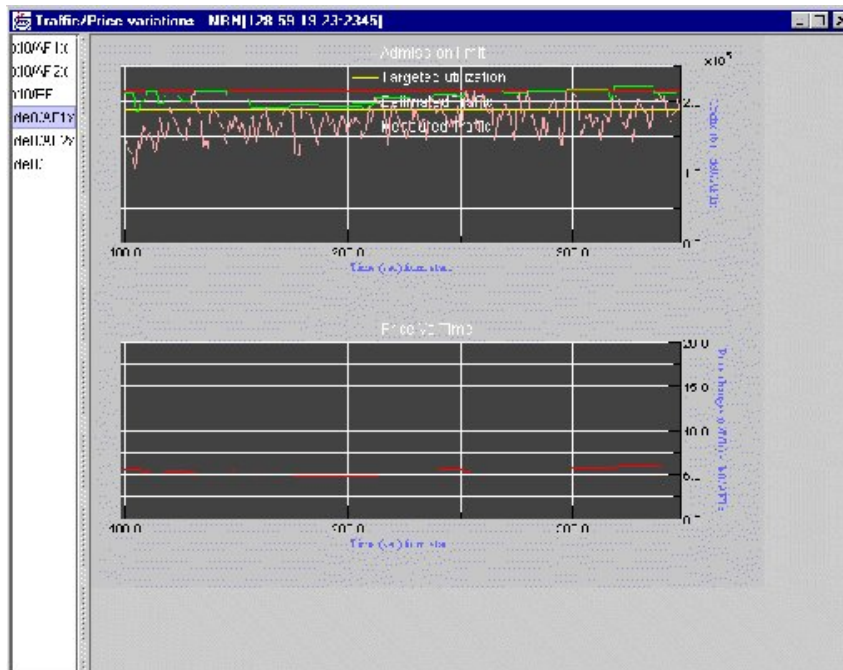


Figure 8-31: Service statistics when applications are adaptive

## Chapter 9

### Conclusion

We have proposed a dynamic resource negotiation framework incorporating a resource negotiation and pricing protocol RNAP, a rate and quality of service adaptation model for adaptive applications, and a pricing model that considers both the long-term user demand and short-term congestion. In the framework, users respond actively to changes in price signaled by the network by dynamically adjusting network resource usage by the application, so as to maximize the perceived utility relative to the price, subject to user budget and QoS constraints. At the same time, the framework provides sufficient flexibility to support users with limited negotiation capability, or with a requirement for very static and predictable service specifications.

We have demonstrated the functionality of the framework through two prototypes: RNAP is implemented stand-alone or embedded in RSVP. We have performed simulations to compare the performance within the framework of a congestion-price based adaptation model (CPA) and a fixed-price, non-adaptive model (FP). We have also used simulations to study the stability of the adaptation process, and nature of network dynamics.

We now summarize the different parts of the framework.

## 9.1. Resource Negotiation and Pricing Protocol - RNAP

RNAP has been designed to enable network service negotiation for multiple delivery services and environments. The RNAP protocol enables service negotiation between user applications and the access network, as well as between adjoining network domains. The protocol permits negotiation and communication of QoS specifications, user traffic profiles, admission of service requests, and pricing and charging information for requested services.

A pair of alternative protocol architectures has been described. The RNAP-D architecture is based on a distributed, per-node model, while the RNAP-C architecture concentrates the negotiation functionality at a centralized entity, the NRN. The first architecture is tailored to delivery services with relatively strict flow control and “hard”, or quantitative QoS specifications. The second architecture may be better suited for delivery service models dealing with service negotiations with a coarser granularity (multiple flows or flow-aggregates) and providing statistical or qualitative specifications. The two architectures use the same set of RNAP messages, and can co-exist and inter-operate across multiple administrative domains.

The protocol and architectures provide mechanisms for the collation of local prices end-to-end. Several price and charge collation mechanisms have been described for the distributed and centralized architectures, and end-to-end pricing and charging across several administrative domains has also been discussed. We now summarize some important features of RNAP.

**Flexibility in Service Negotiation:** RNAP supports service negotiation by the sender, receiver, or both. The protocol is generic and flexible to support multiple delivery services and environments (including IntServ, DiffServ, and best effort services), service negotiation at different levels of granularity (flow- and aggregate-based), negotiation by both sender and receiver, and “in-band”

and “out-of-band” resource reservation mechanisms. In addition, it can be embedded other existing QoS signaling protocol, or can be implemented stand alone.

**Dynamic Resource Re-negotiation Capability:** The periodic nature of the RNAP message sequence provides a natural way for the customer and service provider to re-negotiate services. This enables dynamic, usage sensitive adaptation of service parameters and pricing by the network, if required, and also enables user adaptation. Possible re-negotiation scenarios include periodic re-negotiation, in which the service contract expires after a period and is re-negotiated, and triggered re-negotiation initiated either by the customer or by the network provider.

**Service Predictability:** Predictability includes the quality expected from a service type, and the price charged for it. The periodic price and service quotation mechanism of RNAP provides predictability by keeping the price constant and committing the resources to a user during a negotiation period.

**Scalability:** To reduce the overhead of per-flow RNAP message processing and storage, we design a sink-tree based aggregation scheme. The RNAP messages and state information are aggregated in the core networks, allowing data measurement and charging to be at much larger granularity. While the aggregation was proposed for CPA-TAT, a similar scheme can be made to work with CPA-AUC if several user bids lie within a certain range.

**Stability:** Since the re-negotiation of network services will generally be driven by price changes, we have proved the stability of the negotiation process with a greater focus on pricing. The proposed CPA framework is intended for periods in the unit of minute or longer, and hence the transmission delay is

not an important factor in determining stability. To reduce the oscillation of the resource allocation due to frequent re-negotiation, resource negotiation between domains can occur with a larger granularity, at the cost of slightly lower bandwidth utilization. The end host can also reduce oscillations by re-negotiating only when the service price change is greater than a certain threshold.

## 9.2. Pricing Model

We have proposed a pricing structure for different service classes in DiffServ based on their relative performance, long-term demand, and short-term fluctuations in demand. The congestion price can be calculated either through two methods. The first method is a tâtonnement process in which the congestion price is proportional to the excess demand relative to the target utilization. The second method is based on a  $M$ -bid, second-price auction model. Compared to other auction models described in the literature, the  $M$ -bid auction model provides greater predictability and availability of service, by allowing users to express their willingness to pay for a number of levels of bandwidth in advance of the auction. Submitting multiple bids simultaneously also reduces signaling traffic during auctions by avoiding multiple or more frequent bid submissions. We have also proposed an intermediate admission scheme to reduce setup delay.

### CPA-TAT vs. CPA-AUC

The experiments in Section 7.2. indicate that both CPA-TAT and CPA-AUC can effectively perform the role of congestion control. Since CPA-TAT has to act proactively to limit network usage, its congestion pricing policy is more aggressive, and it cannot achieve as high a network utilization as CPA-TAT. Consequently, CPA-AUC performs better with respect to some performance metrics, particularly per-



ceived user benefit, and network utilization. The main drawback of CPA-AUC relative to CPA-TAT is the higher implementation complexity, particularly if auctioning is implemented per-node, instead of centralized auctioning per-domain. From the user perspective, the drawbacks are the potentially high set-up delay for new connections (even if immediate, inter-auction admission is allowed, the experiments show that up to 40% of the user requests received delayed admission), the need to reveal user preferences to the network, and the need to divide the user willingness-to-pay for a particular bandwidth among multiple nodes or domains, in order to obtain end-to-end resource reservation. In our experiments, we assume that the user bids the same price at all the nodes/domains, that is, it divides its willingness-to-pay equally. However, if users can determine the bottleneck node(s), they will tend to bid higher at these auctions, and correspondingly lower at the other auctions. This makes the resource allocation less predictable, and may result in unfair resource distributions.

### **9.3. User Adaptation Model**

We have proposed mechanisms for rate and QoS adaptation by an application or multi-application system, based on the utility (defined as user-perceived value) of a given combination of transmission parameters, relative to the (congestion-sensitive) cost of obtaining the corresponding service from the network. The adaptation system takes into account constraints imposed by the minimum application requirements and the budget specified by the user, and responds actively to changes in price signaled by the network by dynamically adjusting network resource usage by the application. In a multi-application system such as a video-conference application, the framework allows the system budget to be distributed among the component media so as to maximize the overall perceived value relative to cost. Some heuristics are

discussed to simplify this process. The system budget is dynamically re-distributed among applications in response to changes in price, as well as changes in the relative utilities with time or under different application scenarios. We have also modeled the demand behavior of adaptive users based on a perceptually reasonable user utility function.

#### **9.4. Performance of the Proposed Negotiation Framework**

We have studied the performance of the proposed negotiation framework in detail through simulations. In general, the CPA policy takes advantage of application adaptivity for significant gains in network availability, revenue, and perceived user benefit (in terms of the user utility functions), relative to the fixed-price policy. The congestion-based pricing is stable and effective in limiting utilization to a targeted level. If the nominal (un-congested) price is set to correctly reflect long-term user demand, the congestion-based pricing should effectively limit short-term fluctuations in load.

Our simulation results also show that the different DiffServ classes provide different levels of service only when they operate at different target utilization. In the absence of explicit admission control, a service class loaded beyond its target utilization (under either sustained or bursty loads) no longer meets its expected performance levels. Under these conditions, a congestion-sensitive pricing policy (CPA) coupled with user rate adaptation is able to control congestion and allow a service class to meet its performance assurances under large or bursty offered loads. Users see a reasonably stable service price and are able to maintain a very stable expenditure. Allowing users to migrate between service classes in response to price increase and network performance further stabilizes the individual service prices while maintaining system performance.

When admission control is enforced beyond a threshold load for each class, performance bounds can be met with a fixed service price. However, in this case, the CPA policy provides a greatly reduced connection blocking rate at high loads by driving down individual bandwidth requests, resulting in a higher overall user satisfaction. Compared to the CPA policy without admission control, the service price is further stabilized in this case.

We have also investigated the impact of various network control parameters on the network and user metrics. The user benefit decreases if the target utilization is set either too low or too high. Also, with too low a target, demand fluctuations are higher, while too high a targeted level results in a high blocking rate. Increasing the price scaling factor  $\sigma$  (which affects the speed of reaction to congestion) significantly reduces the blocking probability. However, too large a value of  $\sigma$  results in network under-utilization at offered loads close to the target utilization, and also results in large network dynamics. If the price adjustment threshold parameter  $\theta$  is set too high, there is no meaningful price adjustment and adaptive action. Below a certain level, further reductions in  $\theta$  do not give performance benefits or disadvantages.

Users with different demand elasticity are seen to share bandwidth fairly, with each user having a bandwidth share proportional to its relative willingness to pay for bandwidth. The results also show that even a small proportion of adaptive users may result in a significant performance benefit and better service for the entire user population - both adaptive and non-adaptive users. The performance improvement given by the CPA policy further improves as the network scales and more connections share the resources.

## 9.5. Future Work

Many issues remain open for further work. Some of these issues involve the pricing of network resources. There is currently little work on understanding the distribution of costs in providing network services. Also, this thesis focuses on reducing resource contention primarily on a medium time scale. If the congestion lasts a long time, and occurs very often, network resources are under-provisioned. This is a problem involving optimal resource provisioning on a longer time scale. There are potentially interesting issues involved in the interaction of the medium-time scale congestion control and long-time scale resource provisioning, for instance in terms of response speed, stability, and the relaxation of a long-term provisioning change.

An effective user valuation model will allow user to obtain the best value from the network transmission. The optimal user rate is based on the demand elasticity, total budget, perception of QoS and monetary value, and possibly personal preferences. The host resource negotiator can be designed to be more intelligent, and learn the preferences of the user over time.

In this thesis, we restrict ourselves mainly to a particular path, and study the dynamics of pricing and user adaptation among competing users due to a bottleneck on this path. There are a couple of related issues that we do not include in the scope of the current work. One issue is that of pricing in the presence of competing networks with dynamic pricing, and also user adaptation when multiple paths with different prices (from competing networks) are available to the same user. In general, we believe that if a user receives a reasonably stable and satisfactory QoS and price, there will be little incentive for it to switch networks unless there is a large price advantage to be gained. The other issue is that congestion pricing can help to balance the network traffic, by raising the price along a path with heavy load, and making an alternate route more attractive. Pricing and service negotiation in the

presence of alternative data paths or competing networks can be applied on a longer timer scale, and co-exist with the proposed mechanism.

The proposed framework will be very efficient in supporting data transmission in a resource-scarce network environment, such as wireless networks. In order to extend the current work into wireless networks, the characteristics of the wireless channels must be taken into account.

**Part II**

**Measurement and Analysis of  
LDAP Performance**

## Chapter 10

# Measurement and Analysis of LDAP Performance

### 10.1. Introduction

The Lightweight Directory Access Protocol (LDAP) is being used for an increasing number of directory applications. Applications include personnel databases for administration, tracking schedules [109], address translation databases for IP telephony, network databases for storing network configuration information and service policy rules [110][111]qosschema, and storage of authentication rules [112],[113].

In many of these cases, such as using LDAP directories for storage of personnel information and authentication rules, the data is relatively static, so that caching can be used to improve performance. In some situations, the database information needs to be updated frequently. For example, in IP telephony, every time a subscriber uses a different terminal or is at a different location, his account information may need to be updated. Despite the growing importance of LDAP services, there has been little work on how LDAP servers behave under different workloads, and in different operating environments. In particular, the performance of LDAP in a dynamic environment with frequent searches has not been looked at closely.

In this paper, we report on the development of a tool to benchmark LDAP server performance, and analyze results derived using this tool. In addition, we

have instrumented detailed profiling at the server and LDAP client API codes. These results include the contribution of various system components to the overall performance in terms of latency and throughput, the scaling of performance with directory size, entry size, and session re-use, and the importance of various factors in determining scalability. We also investigate modifications and usage patterns which lead to an improvement in server performance.

Given the growing use of LDAP in applications, it is useful and convenient to carry out the performance experiments using data based on an existing LDAP schema proposed for a real directory application. In this work, we use a schema proposed in [111][114] for the administration of *Service Level Specifications* (SLSs), which are used to configure networks for supporting different levels of services. In this application, it is envisioned that the administrative policies embodied in the LDAP schema will be stored on directories and downloaded to devices such as hosts, routers, policy servers, proxies, etc. If the SLS is allowed to be dynamically negotiated [22], the LDAP service must deal with frequent directory queries. In these respects, this application is representative of many current or proposed LDAP applications [110][112]Bartz9710:LDAProle. The results reported in this work should be generally applicable to many of the applications cited earlier; aspects of the work that are specific to SLS administration will be pointed out where appropriate.

The rest of this paper is organized as follows. In Section 2, we first provide a general background on the LDAP directory service, and then provide a very brief introduction to differentiated service networks and service level Specifications, as well as the LDAP schema proposed for this application. The experimental set-up is discussed in Section 3, followed by a discussion of the test methodology in Section 4. Experiments are described, and the results are presented and analyzed in Section 5, and related work is presented in Section 6. Finally, we summarize our results and



present some conclusions in Section 7.

## 10.2. Background

In this section, we first provide a brief introduction to the LDAP directory service. We then provide a background on the use of LDAP in administration of differentiated services networks. In this context, we also describe the LDAP directory structure used in our experiments.

### 10.2.1 The LDAP Directory Service

A directory service is a simplified database. Typically, it does not have the database mechanisms to support roll-back of transactions. Directories allow both read and write operations, but are intended primarily for high-volume, efficient read operations by clients.

LDAP is a distributed directory service protocol. LDAP is based on a client-server model and runs over TCP/IP. It can be used to access stand-alone directory servers or X.500 directories. Today, LDAPv2 is an Internet standard as defined by the IETF standards process. The standards document, RFC 1777 [115], dates back to March 1995. A newer specification, LDAPv3 [116], is currently a draft in review that is expected to become a new standard soon.

Information is stored in a LDAP directory in the form of entries arranged in a hierarchical tree-like structure (Figure 10-1). An LDAP entry is a collection of *attributes*, for example, an entry corresponding to a person may have as its attributes the name of the person, organization, email-address. Each attribute has a *type*, which is an identifying mnemonic (for example, the email attribute may have type “mail”) and the attribute takes one or more *values* (the email attribute might have “foo@cs.columbia.edu” as a value).

LDAP defines operations for querying and updating the directory. Operations are provided for adding and deleting an entry from the directory, changing an existing entry, and changing the name of an entry. Most of the time, though, LDAP is used to search for information in the directory. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter. Information can be requested from each entry that matches the criteria.

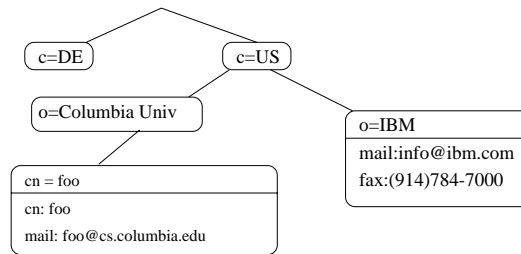


Figure 10-1: An example of organization of data in a LDAP directory

## 10.2.2 Using LDAP for SLS Administration

As mentioned earlier, although we assume a LDAP directory intended for storage of SLS policies, most of the experimental results presented in this work apply to LDAP services in general, and a detailed understanding of differentiated service networks and Service Level Specifications is not required to follow the rest of this paper. However, a brief background may be of interest to some readers.

### 10.2.2.1 Service Level Specifications

The current Internet operates on a best-effort basis, in which all packets are treated equally. Recently, there has been much interest in network service models with mechanisms to provide multiple service levels to users. The two main approaches under discussion are the integrated service model, which supports QoS levels by

allowing per-flow resource reservation using RSVP [18] signaling, and the differentiated service model [72][22], which provides multiple service classes which are served using different per-hop behaviors. In either model, the network provider negotiates a service level specification with a customer, defining aspects of network behavior such as the type of service user packets will receive, and the constraints the user traffic must adhere to. The SLS may be dynamically re-negotiated, based on changes in the customer requirements or network conditions.

The network access points and internal routers implement the classification, resource control, and administrative policies associated with SLSs. Researchers in the DiffServ community have proposed storing these policies in a central or distributed policy repository administered and accessed using a directory service such as LDAP [111][114][117]. In the proposed scenario, the policy repository is updated when the network provider negotiates new SLSs, or re-negotiates existing contracts, and also when the policies need to reflect changes in network topology or traffic levels. Network elements frequently access the policy database, and download the current set of rules according to which customer traffic is served.

In addition, the network provider provisions the network in order to provide the service contracted to customers. The provisioning is physical (adding or removing network elements) and logical (partitioning or configuring network elements). The network configuration information may be maintained in LDAP directories, and downloaded periodically by routers. This allows the network provider to adjust configurations (for example, buffer space, or packet drop precedences) with a finer granularity in response to network usage feedback.

### 10.2.2.2 Architecture of Network QoS Control Using LDAP

A preliminary schema using LDAP for configuration of DiffServ networks has been proposed in [111]. The various aspects of a service, such as the traffic profile the user traffic must conform to in order to receive the service, and the forwarding rules for conforming traffic, are captured in a set of policies. The generic architecture that is envisioned consists of a management tool, a policy repository, a policy decision entity, and a policy enforcement entity. Figure 10-2 shows the functional relations between these different entities, and it does not restrict where these functional entities should be located.

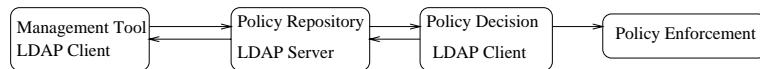


Figure 10-2: An architecture for network QoS control using LDAP

In the context of the service environment under consideration, the management tools are used by the network administrator to populate and maintain the LDAP directory with policies. Management tools may or may not reside on the same host as the directory server. Enforcement entities apply policy rules.

A decision entity and enforcement entity are usually assumed to reside at each edge device, or network access point. The edge device is referred to by its location and would most likely be placed at the access point between a local subnet and the backbone network, or at the boundary between backbone networks of two service providers. The decision entity downloads policy rules from the repository, through a LDAP client. The enforcement entity queries rules from the decision entity and carries out packet handling and monitoring functions. The decision entity may either download the entire policy repository all at once, or may query the directory when needed - for instance, when triggered by events such as an RSVP message or an IP packet bearing a TCP connect request.

A customer attaches to the network at one or more interfaces belonging to an edge device. Each interface is identified by an IP address. At each interface, one or more policies may be defined, and customer packets are monitored and processed according to these policies. Each policy is associated with a service level which defines actions on the part of network elements in handling customer packets. A policy may be applied on the basis of source/destination IP addresses, transport protocols, source/destination ports, and other parameters such as default port, URL's, etc.

Policy rules are stored in the LDAP directory as *SLSPolicyRules* objects (derived from the *Policy* class described in [111]). *SLSPolicyRules* objects may have attributes specifying the policy name, priority level of the rule, and the network interfaces to which the rule may be applied, as well as references to objects which specify the traffic profile, period of validity of the rule, type of RSVP service or DiffServ action, etc.

At initialization, the edge device identifies its interface addresses. It determines the set of policies required for these interfaces, and downloads the corresponding classification policy rules from the LDAP server, as well as the service specifications referred by the policies. Subsequently, the edge device may poll the server periodically to learn of modifications to the directory, and download its set of policy rules if the directory is modified. If asynchronous mode operations are supported by the directory service, the downloading of policy rules could also be triggered upon changes in the policy rules.

### 10.2.2.3 LDAP Directory Structure Used in the Experiments

The LDAP directory structure used in our tests is a simplified version of the directory used to develop the LDAP schema for supporting SLS [111][114][117], and is shown in Figure 10-3. Each *Customer* entry has a set of associated *Interface* en-

tries. The *Policy* entry directly under the *Customer* specifies policy rules common to multiple interfaces belonging to the customer, while the *Policy* entry for each *Interface* specifies the policy rules specific to customer traffic at that Interface. In general, the *Policy* entry refers to one or more of the *Service* entries in the directory to specify the service to be received by the corresponding traffic. The other entries shown in the LDAP directory include *Channel* and *Pacer* entries. A channel is a virtual pipe between an ingress edge-device and an egress edge-device. A pacer is the abstraction that limits the total amount of traffic that can be sent out into the backbone network at an access-point.

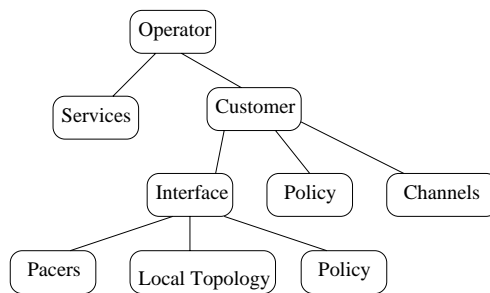


Figure 10-3: LDAP tree structure in tests

### 10.3. Experimental Setup

In this section we describe our experimental testbed, including the hardware we use, the LDAP server software structure, the LDAP client load generation and the benchmarking setup.

#### 10.3.1 Hardware

The LDAP server ran on a dual-processor Ultra-2 machine equipped with two 200 MHz Sun, UltraSPARC CPUs, 256 MB main memory. The LDAP server process was bound to one of the two CPUs. The LDAP clients ran on a couple of Sun

Ultra 1 models with 170 MHz CPU, 128 MB main memory, and one Sun ultra 10 machine with 299 MHz CPU and 256 MB main memory. The server and clients were connected via 10 Mb/s Ethernet.

### 10.3.2 LDAP Server

There are a number of commercial LDAP servers, including Netscape Directory Server, and Novell LDAP Services. We chose OpenLDAP 1.2 [118]. OpenLDAP is a complete open source suite of client and server applications derived from University of Michigan LDAP v3.3. The main reasons for our using OpenLDAP is its open source model, and its rapidly increasing user population. The open source model allowed us to perform detailed profiling of individual server modules and examine some modifications of the basic implementation instead of treating the server as a black box. The server is based on a stand-alone LDAP daemon (*slapd*) for directory service. Replicated service is also supported through a UNIX daemon *slurpd*. In this work, the goal is to study the performance and scalability of the server, and we restrict the LDAP clients to connect to one *slapd*. Slapd consists of two distinct parts: a front end that handles protocol communication with LDAP clients; and a backend that handles database operations. Slapd comes with three different backend databases to choose from. They are LDBM, a high-performance disk-based database; SHELL, a database interface to arbitrary UNIX commands or shell scripts; and PASSWD, a simple password file database. The LDBM backend relies on a low-level hash or B-tree package for its underlying database. In this work, we used an LDBM backend, namely the Berkeley DB version 2.4.14 package [119] hash database.

LDBM has two important configuration parameters: *cachesize*, the size in entries of an in-memory cache that holds LDAP directory entries, and *dbcachesize*, the

size in bytes of the in-memory cache associated with each open index file. In our experiments, *dbcachesize* was set equal to 10 MB, sufficiently large to keep all index files in-memory. The *cachesize* varied according to specific experiments.

### 10.3.3 LDAP Client

The overall client-server architecture used in our experiments is shown in Fig. 10-4. A collection of client machines are connected to a server machine. There can be more than one LDAP process running on a client or server machine. The client machines used had sufficient CPU capability that the delay at client sites could be ignored in measuring server performance.

A Bench Master process coordinates the client processes and generates an overall performance report. The setup parameters are defined in configuration files. The Bench Master constructs the command-line arguments for LDAP client processes based on the configuration files. The Bench Master is also responsible for spawning the LDAP clients remotely on the designated machines. Each of the LDAP clients reads the command line and starts up communication with the Bench Master. After all the LDAP clients have been initialized, the Bench Master instructs the LDAP clients to commence the benchmarking. It then waits for the end of the experiment when it receives all the statistics from all the LDAP clients. Each LDAP client queries the LDAP directory periodically, with the query starting time for each client randomized to prevent synchronization. In most of our experiments, the workload of the LDAP server was changed by varying the query interval of the LDAP clients. The Bench Master organizes the data from the clients into the benchmark report.



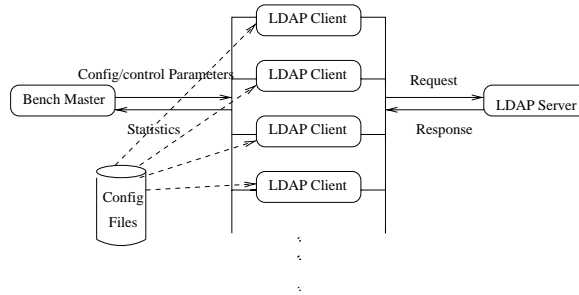


Figure 10-4: LDAP benchmarking testbed architecture

## 10.4. Test Methodology

Common LDAP operations are *modify*, *add*, *delete*, *compare* and *search*. In the directory application considered for our experiments, the service specifications should remain relatively static during normal operation, while the policies defined by customer-provider SLSs would be updated much more often, as customers negotiate new SLSs and re-negotiate old SLSs. *Search* operations are therefore likely to dominate the server load. In general, this is true for most LDAP applications. Accordingly, for most of our experiments the server workload consisted of *search* requests for downloading of policy rules (SLSPolicyRules objects) from the LDAP directory (Fig. 2).

The search filter for the search operation was constructed from the Interface address of interest, and the corresponding *Policy* object. The default entry size for most experiments was 488 bytes, and the default directory size was 10,000 entries.

A simple LDAP search involves a sequence of 4 operations: *ldap\_open*, *ldap\_bind*, one or more *ldap\_search* operations, and *ldap\_unbind* (Figure 10-5). *ldap\_open* initializes the LDAP library, opens a connection to the directory server, and returns a session handle for future use. The *ldap\_bind* operation is responsible for client authentication. The bind operation allows a client to identify itself to the directory server by using a Distinguished Name and some authentication credentials (a pass-

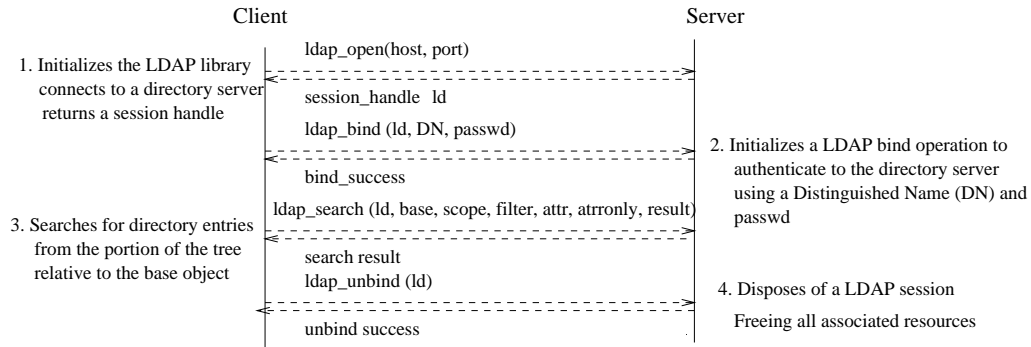


Figure 10-5: Sequence of steps in a simple LDAP search operation

word or other information). LDAP supports a variety of authentication methods. In our experiments, password authentication was used. When a bind operation is successfully completed, the directory server remembers the new identity until another bind is done or the LDAP session is terminated by calling *ldap\_unbind*. The identity is used by the server to make decisions about what kind of changes can be made to the directory. The *ldap\_search* operation initiates a LDAP search by specifying the criteria that entries fitting in the associated filter could be returned. The search filters were randomized in our experiments to avoid search results being cached. Finally, an LDAP session is disposed of by using *ldap\_unbind*.

Referring to Fig. 10-5, we define the following latency measures: the connect time is defined as the time from the sending of *ldap\_open* request until *ldap\_bind* operation is successful. The processing time is defined as the time required for the *ldap\_search* operation as well as the data transfer time for the retrieved results to be returned to the clients. The total time required for an LDAP search operation, from *ldap\_open* to *ldap\_unbind*, is defined as the response time. The measures that reflect the performance of the LDAP service are the connect, processing and response latencies, and the throughput of the server, represented by the average number of requests served per second.

In our experiments, each search operation involved all four of the above steps,

*ldap\_open*, *ldap\_bind*, *ldap\_search*, and *ldap\_unbind*. In a real application, a client performing multiple searches may prefer to leave the connection open and only do an unbind at the end. In this sense, the total response time data in the experiments represents a worst-case scenario. In Section 10.5.4, we consider the effect of leaving the connection open for multiple searches or for the duration of the experiment on performance.

In addition to the *search* experiments, the performance of the LDAP server for database updates is studied in Section 10.5.5, using a workload consisting of *ldap\_add* requests for adding SLSPolicyRules objects. In this case, the client must construct a new entry with a set of attributes before calling the *ldap\_add* routine.

## 10.5. Result Analysis

Our experiments have three main purposes: identify the contributions of various system components towards the overall LDAP performance; suggest measures to improve performance; and study the limits of LDAP performance, and what determines these limits. We organize the experimental results as follows. The overall performance with respect to throughput and latency is introduced in Section 10.5.1. The various components of the total search latency are studied in Section 10.5.2, followed by measures to improve LDAP performance. Some important limitations on LDAP performance are studied in Section 10.5.3. We then discuss the effect of session re-use on server performance in Section 10.5.4. Finally, performance of update operations is compared to the performance of search operations in Section 10.5.5.

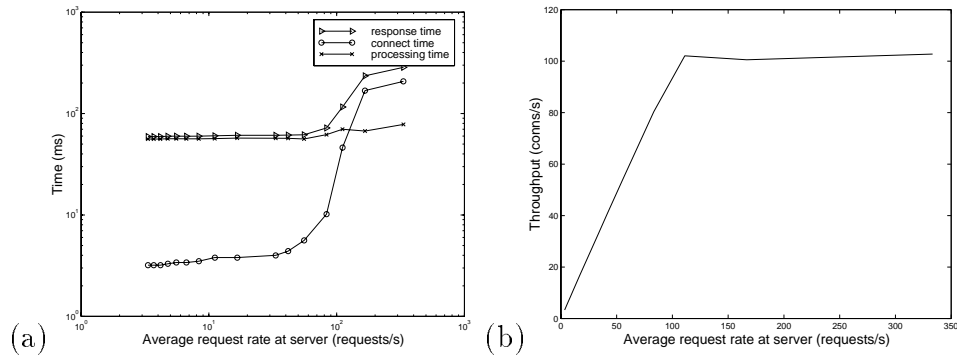


Figure 10-6: Average connection time, processing time and response time (a) and average server throughput (b) shown as a function of the average request rate at the server. The directory had 10,000 entries and each entry size was 488 bytes.

### 10.5.1 Overall LDAP Performance

The general variation in the LDAP latency and throughput as a function of server load are shown in Fig. 10-6. The load was generated by 1000 clients querying periodically, with each client request retrieving a 488 byte SLAPolicyRules entry from a database with 10,000 entries. Fig. 10-6 (a) shows the variation in connect, processing, and response latencies as a function of the average request rate at the LDAP server, and Fig. 10-6 (b) shows the variation in throughput, or number of queries served per second, as a function of the average request rate. Below a load threshold corresponding to a request rate of 105 per second, response latency remains fairly constant at approximately 64 ms, and is dominated by the processing time of 60 ms. Above this threshold, the response time increases rapidly with increasing load. In this region, the connect time is seen to be the main bottleneck; the processing time also increases with load, but its effect is less significant. Corresponding to the latency characteristics, Fig. 10-6 (b) shows that the server throughput saturates at a rate of approximately 105 requests per second. We now consider various aspects of this performance in more detail.

## 10.5.2 Improving the LDAP Search Performance

In this section, we first investigate the various components of the search and connect latency. We then consider two important measures to improve effective search latency and throughput: disabling the Nagle algorithm [120] implemented in TCP, and the caching of LDAP entries.

### 10.5.2.1 Components of LDAP Search Latency

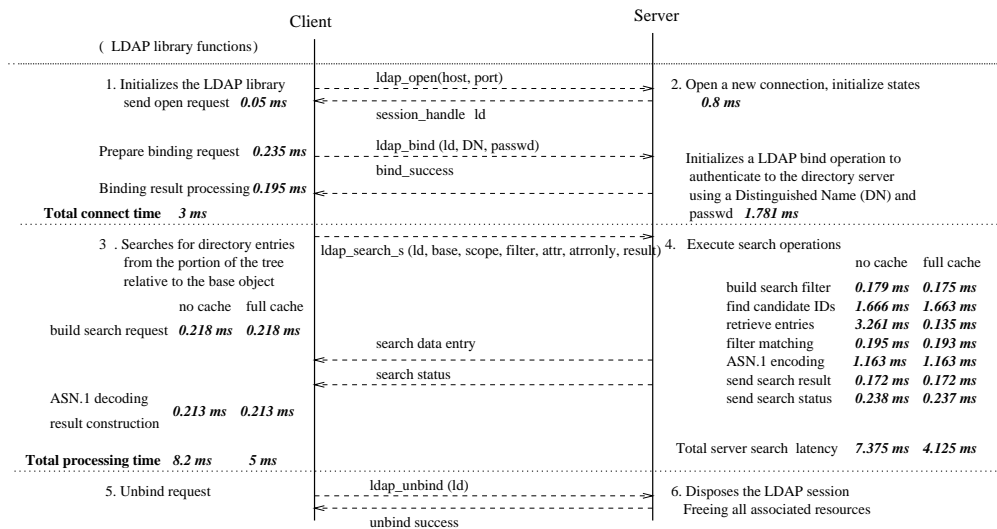


Figure 10-7: Latency associated with the various server and client process modules in a LDAP search operation

We now consider the components of the search latency in more detail. These results were obtained by adding monitoring code to the various process modules in the slapd daemon. Fig. 10-7 shows the major components that contribute to the server and client search latency, under a load of 105 search requests/second, at which the server is not yet saturated. Surprisingly, while the processing time as measured at the client is approximately 60 ms, results obtained from *tcpdump* and from the monitoring code in *slapd* show that out of 60 ms, approximately 50 ms is a waiting time arising from the Nagle algorithm implemented in TCP. We discuss

this in greater detail in Section 10.5.2.3, and consider the components of the actual search latency in this section.

At the server, the LDBM back-end uses an index mechanism to store and retrieve information. Each entry is assigned a unique ID, used to refer to the entry in the indexes. A search for entries first returns a list of IDs of entries that have the value being searched; the IDs are then used to retrieve the corresponding entries. The candidate ID lookup and the data entry retrieval are seen to take up around 5 ms, 60% of the total search latency of 8.2 ms.

The main front-end operations at the server are building the search filter, testing a search filter against an entry, ASN.1 encoding of the result entry, sending the search result, and sending the search status. The client-side LDAP library processing includes building the search request, ASN.1 decoding, and construction of the final search result. In all, the front-end operations take around 3.16 ms, 40% of the total response latency. 36.8% of the front-end latency is contributed by ASN.1 data encoding, followed by sending status information, 7.5%, building the search request, 7%, ASN.1 decoding, 6.7%, testing a search filter against an entry, 6.2%, forming the search filter, 5.7%, and sending the search result, 5.4%. The remaining operations, including the ASN.1 encoding and decoding of the query and other information, occupy the remaining 25% of the front-end latency. Profiling of the slapd daemon also showed that at heavy loads, the increase in the total response time is due to the CPU contention among competing threads.

### 10.5.2.2 Components of LDAP Connect Latency

The connect latency has two components, corresponding to the *ldap\_open* and *ldap\_bind* steps of the LDAP search operation shown in Fig. 10-5. *ldap\_open* initializes the LDAP library, opens a connection to the directory server, and returns

a session handle for future use. Thus, the open time mainly consists of the session set up time and the TCP connection time, as shown in Fig. 10-7.

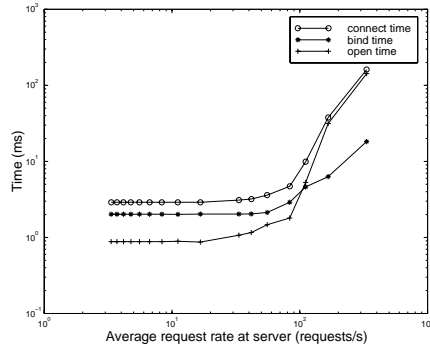


Figure 10-8: Variation of the *open* time and *bind* time components of the average connection time with average request rate at the server. The directory had 10,000 entries and each entry size was 488 bytes.

In version 2 of the LDAP protocol, the *ldap\_bind* step (client authentication) is a mandatory requirement. As mentioned previously, LDAP supports different authentication methods, and simple password-based authentication was used in the experiments reported here. Fig. 10-7 shows that the server takes 80% of the binding time, and the client takes 20% of the time.

Fig. 10-8 shows the open time and authentication time components of the connect time, as a function of the request rate. At small loads, the authentication time is more than twice the open time and dominate the connect latency, which is consistent with the profiling results. The increase in connect time beyond a request rate of 105 per second is largely dominated by the increase in open time.

### 10.5.2.3 Effect of the Nagle Algorithm on Search Latency

The Nagle algorithm restricts sending of packets when the segment available to send is less than a full MTU size, in order to reduce transmission of small packets and thus improve network utilization. The algorithm works as follows: if all outstanding data has been acknowledged, any segment is sent immediately. If there is unacknowledged

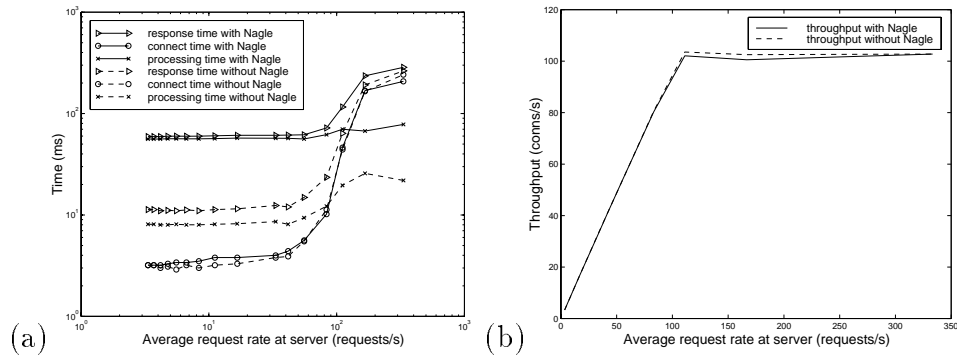


Figure 10-9: Comparison of the server performance with and without Nagle algorithm: (a) average connection time, processing time and response time; (b) average server throughput. The directory had 10,000 entries and each entry size was 488 bytes.

data, the segment is only transmitted if it is a full MTU size. Otherwise, it is queued in the hope that more data will soon be delivered to the TCP layer and then a full MTU can be sent. Fig. 10-9 shows that when the Nagle mechanism is disabled by enabling the `TCP_NODELAY` socket option, the LDAP search time is reduced from 60 ms to around 8 ms, while the throughput remains unchanged.

`slapd` responds to a search request from the client side ldap library functions in two steps: it first returns the data entries; it then sends the search and transmission status. The client side ldap library functions then construct the final results and send to the LDAP client. The results from `tcpdump` indicated that the sending of search status information (14 byte) was delayed about 50 ms until an acknowledgement message was received from client side.

Researchers have presented evidence [121][122] that the Nagle algorithm should be disabled in order to reduce latency and to protect against unforeseen interactions between TCP and HTTP with persistent connections. We believe Nagle algorithm should also be turned off in the LDAP application, since the delay of the last segment of the response was shown to be unnecessary from the `tcpdump` results. To avoid consequent extra packets on the network, functions such as `writenv()` can be used as have been used in WWW servers such as Flash and JAWS.



### 10.5.2.4 Effect of Caching LDAP Entries

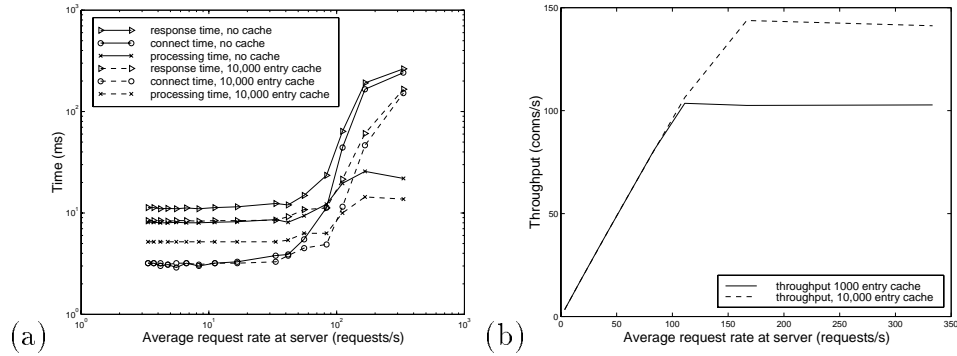


Figure 10-10: Comparison of the performance when the server is configured with 0 and 10,000 entry caches: (a) the response latency; (b) the throughput.

The LDBM backend can be configured to keep a cache for each index file (*db-cachesize*) and a cache of entries (*cachesize*) in memory. We compared the server performance with two different sizes of the index cache, 100 KB and 10 MB. The backend search time was almost identical for either case, although at the smaller cache size there was a large increase in the time required to populate the directory. This indicated that the index cache was large enough to hold the index files in either case.

We studied the impact of the entry cache size on the server performance by varying the cache size from 0 entries (out-of-memory directory without any cache) to 10,000 entries (in-memory directory with full cache) while keeping the index cache size at 10 MB. For a directory with 10,000 entries, a cache with 10,000 entries results in a reduction in the back-end entry retrieval latency (shown in Fig. 10-7) from 5 ms to 1.8 ms, with little change in the other latency components. Consequently, the total processing time reduces from 8.2 ms to 5 ms, and the contribution of the back-end processing time to the total latency reduces from 60% to 36%. When the latency is plotted as a function of load, we see a 40% reduction in search processing time due to caching over nearly the entire range of loads, as shown in Fig. 10-10 (a).

Correspondingly, the throughput increases 25% and reaches 140 requests/second, as shown in Fig. 10-10 (b). The relatively smaller improvement in throughput compared with the improvement in processing time is because at high loads, the connect latency is the dominant component in the total response latency. Memory usage was observed to increase 9% with the above increase in cache size.

Since we are primarily interested in the performance of the LDAP server, in the experiments in the remainder of this paper, the Nagle algorithm was disabled to eliminate the unnecessary wait before returning search results to the client. Unless otherwise specified, an LDBM index cache of size 10 MB, and entry cache equal to the size of the directory were used. Before an experiment, the cache was first filled up to avoid the extra overhead due to cache misses at the beginning of the experiment.

### 10.5.3 Performance Limitations

In this section, we study the limitations of the LDAP server performance in three important areas: server CPU capability, the scaling of the LDAP directory, and the scaling of LDAP entry size. An useful point to consider is that in some cases network connectivity may significantly influence the perceived server response. In our case, since clients were connected to the server over a high-speed LAN, this effect could be neglected.

#### 10.5.3.1 Server Processor Capability: Single vs Dual Processors

In this section, we consider the importance of processing capability in determining the server performance. As mentioned earlier, all our experiments were carried out on a dual processor server, but by binding the *slapd* process to one of the two CPU's, it was used in single-processor mode for the experiments in other sections.

To determine the influence of processor capability, we performed some experiments to compare the performance in single-processor mode with the performance in dual-processor mode.

Fig. 10-11 (a) shows the comparison of latency versus connection rate characteristics for the two cases, using a load generated by search operations. The dual processor server shows similar performance at low loads, and the advantage increases to give roughly 40% smaller latency at higher loads for the total response time. The reduction in latency is observed mainly due to the reduction in connect time. The processing time due to search actually increases slightly at heaviest load, which may be due to the memory contention between the two processors.

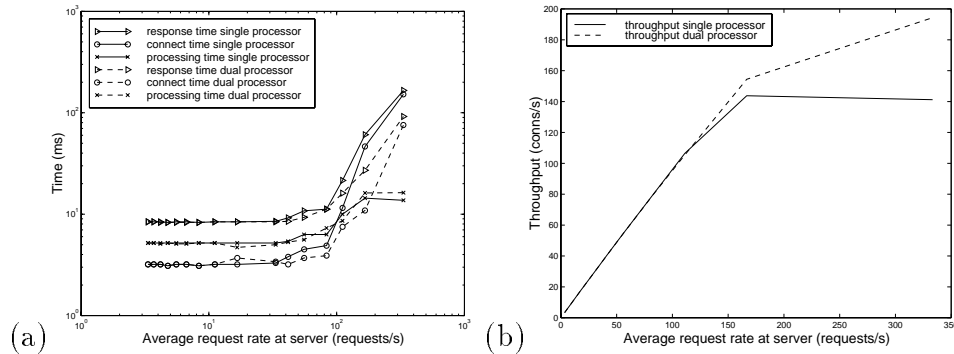


Figure 10-11: Effect of using a dual processor server on the response time and throughput, with a directory containing 10,000 entries, entry size 488 bytes: (a) connect, processing and response latencies versus request rate; (b) throughput versus request rate

Fig. 10-11 (b) shows the comparison of throughput characteristics for single and dual processors. As seen earlier, the throughput of the single processor server starts to saturate beyond a load of 105 requests per second, and saturates at 140 requests per second. The dual processor server starts to saturate beyond a load of 155 requests per second, and does not saturate completely even at a load of 194 requests per second. Also, at the highest load, CPU utilization in the single-processor server reached 98%, while CPU utilization in the dual processor remained less than 50%.

The above results suggest that the improvement given by the dual processor server would continue to increase at higher loads than those used in the experiment. Higher loads were obtained in an experiment with each client generating requests continuously (a new query upon completion of the previous query) instead of at periodic intervals. Read and write throughput characteristics were generated using LDAP *search* and *add* operations respectively. The results are shown in Fig. 10-12. Consistent with trends observed in Fig. 10-11 (a), at low loads the throughput characteristics are similar for dual processor and single processor servers. Beyond a threshold load of about 8-10 clients, the write throughput saturates at around 60 connections per second for the single processor server and 85 connections per second for dual processor operation, an improvement of roughly 40%. A similar effect is observed in the read throughput, which reaches saturation with just 4 clients and gives 150 connections per second for single processor server and 211 connections per second for dual processor server, an improvement of roughly 40%, the same improvement rate as the write operation.

There is a second load threshold in write throughput of about 36 clients for single processor and 48 clients for dual processors beyond which the throughput decrease with increase in load, while the read throughput remains constant within the load range of the experiments. The reduction in throughput of write operations may be due to the increasing contention of system resources among children processes and the increase in the network delay when the server loads increase. These experiments also show the throughput of *search* operations is roughly 2.5 times that of *add* operations in both the single processor and dual processor case.

Overall, processor capability plays a major role in limiting system performance for an in-memory directory, and using a dual processor server gives a significant performance benefit.

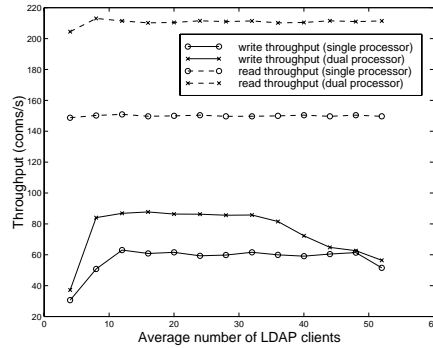


Figure 10-12: Throughput of search and add operations for single and dual processor servers, with a varying number of LDAP clients generating queries continuously

### 10.5.3.2 Scaling of Directory Size

In this section, we study the scaling of LDAP performance with the directory size, and discuss the limitations on performance at large directory sizes.

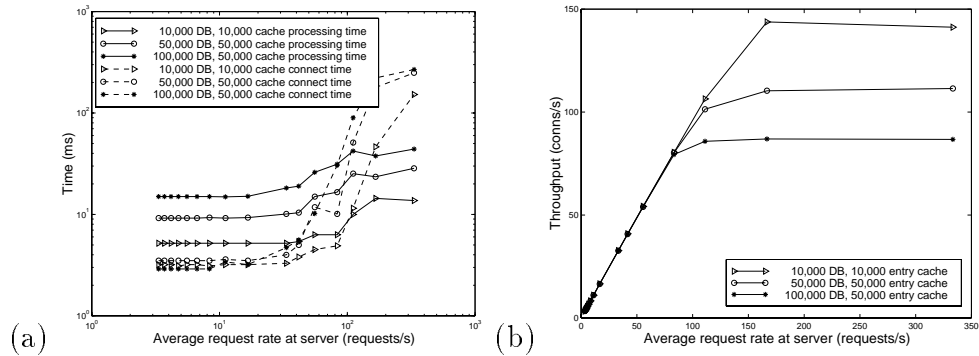


Figure 10-13: Effect of directory size on the server connect and processing times (a) and the server throughput (b).

We first consider the scaling up of the directory when the directory is in-memory. Fig. 10-13 (a) shows the comparison of response latency of a directory with 50,000 entries and 50,000 entry cache, with a directory with 10,000 entries and 10,000 entry cache. The increase in the total response with directory size is mainly due to the increase in the processing time, which increases by 60%, from 5 ms to 8 ms. Profiling at the server shows that the increase in the processing time is, as expected, due to increase in back-end processing. Specifically, it is mainly due to the increase in the

entry retrieval time (by 2.6 ms), and a slight increase in the ID search time (by 0.2 ms). Fig. 10-13 (b) shows that the throughput decreases 21% and saturates at 110 requests/second with the increase in directory size.

As the directory size increases, the database cache size is eventually limited by the available system RAM, and the directory can no longer be kept in-memory. In our system, when the cache size was increased beyond 50,000, performance degraded progressively due to lack of memory. When the directory is out-of-memory, performance scaling is limited both by the database search time, and by the disk access time. Fig. 10-13 shows that further increasing the number of directory entries from 50,000 to 100,000 while keeping the cache size at 50,000 entries causes the response time to increase another 7 ms (87.5%) and the total processing time reaches 15 ms. Correspondingly, the throughput reduces from 110 requests/second to 85 requests/second (23%). Since the connect latency dominates the response at high loads, the reduction in throughput is relatively small compared to the increase in processing time.

To summarize, there was a moderate decrease in latency and throughput with directory scaling up to 50,000 entries, due to an increase in database search time. Further scaling was constrained by system memory leading to an out-of-memory directory, and the deterioration in processing latency was significantly sharper, due to increasing disk access time and database search time.

### **10.5.3.3 Scaling of the Directory Entry Size**

In this section we study the scaling of performance with the size of the LDAP entry. In our experiments, we compared the performance for LDAP directories with 488 byte entries, and with 4880 byte entries. For the larger entry size, the availability of system RAM limited the maximum database cache size to 5000 entries, beyond

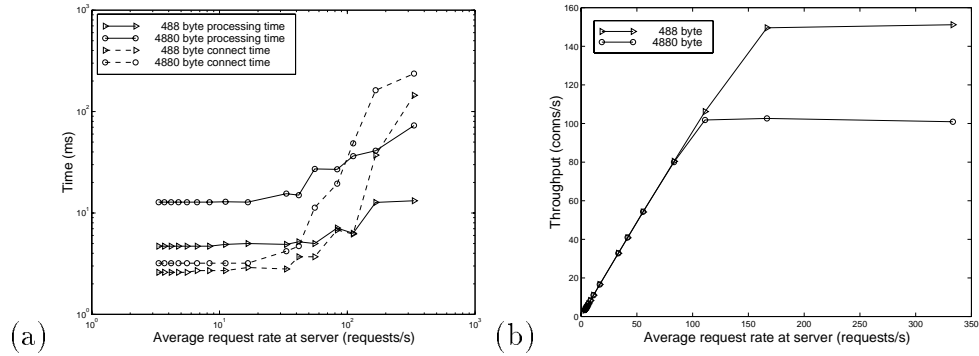


Figure 10-14: Effect of directory entry size on the server connect and processing times (a) and the server throughput (b) for 5000 entry directory and 5000 entry cache

which the performance degraded rapidly. We first performed the comparison for an in-memory directory, using a 5000 entry directory and 5000 entry cache for both entry sizes. Fig. 10-14 (a) shows the processing latency, and Fig. 10-14 (b) the throughput, as a function of load. At light and moderate loads, the total response latency increases by about 8 ms with increase in entry size, while the throughput remains the same. Profiling at the server shows that the increase in response time comes mainly from the increase in the ASN.1 encoding time, from 1.2 ms to 7 ms. In addition, the filter matching step took around 0.8 ms at the higher entry size, up from 0.2 ms at the lower entry size. These results are understandable, since both ANS.1 encoding and the filter matching process depend on the number of attributes in the LDAP entry, which increases as the entry size increases. The latency of the other front-end processing steps, and back-end processing steps increase by much smaller amounts. Under heavy loads, when the server saturates, Fig. 10-14 (b) shows the maximum throughput at 4880 bytes is 30% smaller.

The comparison between the two entry sizes was also performed with a directory of 10,000 entries, the cache size remaining at 5,000 entries. Fig. 10-15 (a) shows the processing latency, and Fig. 10-15 (b) the throughput, as a function of load. The increase in total response time with increase in entry size is now 40 ms at low and

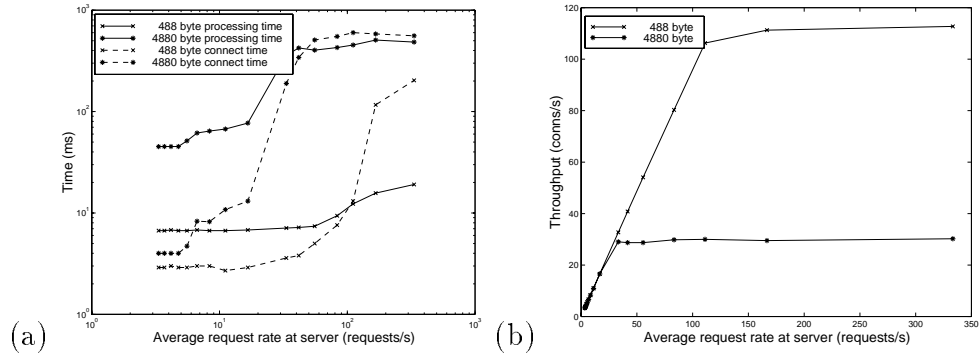


Figure 10-15: Effect of directory entry size on the server connect and processing times (a) and the server throughput (b) for a 10,000 entry directory, 5000 entry cache

moderate loads. The much larger deterioration in response time with entry size is due to the increased data transfer time from the disk in addition to the the increased front-end processing time. The increase in front-end processing time remains at about 8 ms, confirming that the front-end processing is not influenced greatly by the increase in directory size, and the increase in data handling time at the back-end is 32 ms. Also, for the larger entry size, the processing time is comparable with the connect latency even at high loads (unlike previous experiments in which the connect latency dominated the total response at high loads). Consequently the increase in processing time with entry size strongly influences the maximum throughput, which decreases sharply from 110 requests/second to 30 requests/second.

To summarize, for an in-memory directory the scaling of the total processing latency with entry-size is determined by the front-end processing, specifically, by the ASN.1 encoding and filter matching. For out-of-memory operation, the increase in processing latency is dominated by the increased back-end data retrieval time. The maximum throughput also decreases sharply in this case.



### 10.5.4 Session Reuse

In the experiments so far, the client requested a single search or add operation, and then closed its connection. We now study the change in performance when a client performs multiple searches in a LDAP session. We varied the degree of session reuse from 0 to 100%, with 0% corresponding to a new connection being established for every search, and 100% corresponding to the connection being left open during the duration of the experiment.

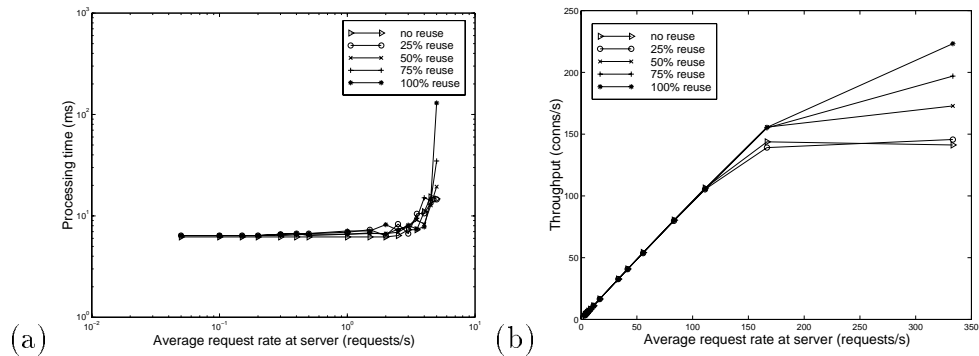


Figure 10-16: Effect of session reuse rate on the server processing times (a) and the server throughput (b).

Fig. 10-16 (a) shows that the search time increases at heavy loads as the session reuse rate increases from 0% to 100%. This is because the actual server load for searching operations increases with the increase in session reuse, while the total connect latency decreases because fewer connect requests are received. At the same time, the total response time decreases as can be deduced from the increase in the throughput shown in Fig. 10-16 (b) from 105 request/second to 155 requests/second at the onset of congestion., and from 140 requests/second to 223 requests/second under the heaviest load, an improvement of 60%.

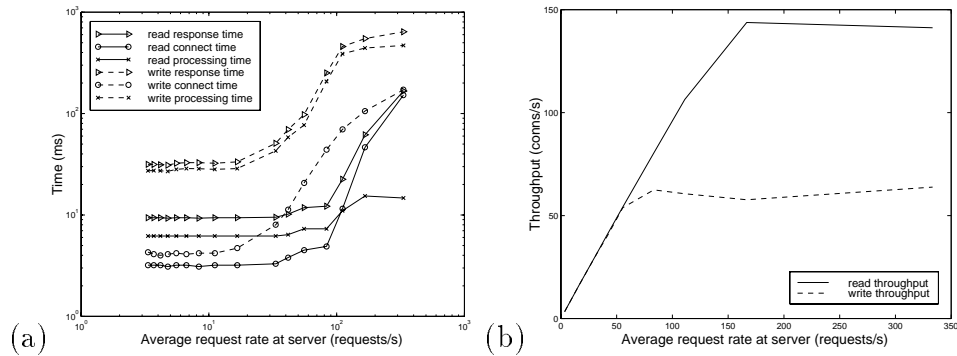


Figure 10-17: Comparison of latency and throughput characteristics for search and add operations: variation of connect, processing and response latency with server load (a), and variation of server throughput with server load (b). (Directory had 10,000 entries, entry size 488 bytes)

### 10.5.5 Performance under Update Load

The experiments described so far measured the performance under loads generated by *search* requests. In order to compare these results with the performance under update operation, a set of experiments was performed with clients each periodically querying the server with *add* requests. Fig. 10-17 shows that the processing time for *add* operations dominates the total response time not only at low loads, but also when the server saturates. This is unlike the search scenario, in which the connect time dominates the total response time when the server is under heavy load. This is probably because the latency due to locks generated by competing *write()* requests becomes significant under heavy load. At low loads, the *add* latency is about four times the search latency, and the difference between the two decreases at high loads, as the connect latency becomes more important. Fig. 10-17 (b) shows that the add throughput begins to saturate beyond a threshold of 60 requests per second and finally saturates at 65 requests per second, about 55% less than the search throughput.

The LDAP *modify* operation was not directly investigated. A *modify* operation involves searching and retrieving an existing entry, modifying it, and then writing it

back to the LDAP directory. Consequently, one would expect the processing latency in this case to be roughly the sum of the *search* and *add* processing latencies.

## 10.6. Related Work

Benchmarking of LDAP server performance has been reported recently by Mindcraft [123]. The purpose of Mindcraft's work is to compare the performance of three servers: Netscape Directory server 3.0 (NSDS3), Netscape Directory Server 1.0 (NSDS1) and Novell LDAP Services (NDS). In their work, the performance for a 10,000 entry personnel directory was measured on a 200 MHz Intel Pentium Pro with 512 MB RAM. The throughput for NSDS3, NSDS1, and NDS were found to be 183 requests/second, 38.4 requests/second and 0.8 requests/second respectively. The size of the LDAP entry was not specified. The directory was in-memory in all cases, and the performance with larger, out-of-memory directories (or large entry sizes) was not considered. Since the directory was in-memory, CPU capability was found to be the bottleneck.

In the above work, the LDAP server was generally treated as a black box. Our work differs significantly in our objectives and approach. We have determined the scalability of the performance particularly with respect to directory size and entry size, determined the contribution of different system components and parameters to the server performance and scalability, and provided suggestions for improving system performance. The detailed nature of this work also dictated the choice of OpenLDAP instead of a commercial server, as explained earlier.

## 10.7. Conclusions

In this paper, we have discussed the performance of the LDAP directory service in a dynamic, distributed environment, with frequent directory accesses. The LDAP directory structure used in the experiments is based on a proposed LDAP schema for administration of Service Level Specifications in differentiated service networks, and a brief explanation of the use of LDAP in this context has been provided. However, the experimental results are applicable to LDAP directory applications in general.

We have shown that under our normal operating conditions - a directory with 10,000 488 byte entries, and a cache size of 10,000 entries - the LDAP server has a response latency of 8 ms at loads up to 105 search requests per second, and a maximum throughput of 140 search requests per second. Out of the total response latency of 8 ms, 5 ms comes from the processing latency, 36% of which is contributed by back-end processing (entry retrieval from the database), and 64% by front-end processing. In general, at high loads, the connect latency increases sharply to dominate the overall response, and eventually limits the server throughput. Consequently, a change in the processing time due to changes in system parameters has a relatively smaller effect on the maximum throughput.

In addition to this basic performance specification, we have obtained a detailed profile of contributions of various system components to the overall performance; studied the scaling of performance with directory size, entry size, and session re-use; and determined the relative importance of various factors in determining scalability, namely front-end versus back-end processing, CPU capability, and available memory. We have also identified an important required modification to the basic OpenLDAP implementation in order to obtain the above performance. We now briefly summarize our important findings.

- *Disabling of Nagle algorithm.* The Nagle algorithm was observed to contribute

an additional wait time of roughly 50 ms to a search operation. Disabling the Nagle algorithm results in a reduction in response time by a factor of 7 under normal operation.

- *Entry caching.* For a directory with 10,000 entries, an in-memory directory (cache-size 10,000) has a 40% improvement in processing time and 25% improvement in throughput over a directory without a cache.
- *Scaling with directory size.* The scaling of performance with the number of entries in the directory is determined by the back-end processing. Up to 50,000 directory entries can be kept in-memory in our system, and the server processing time and throughput deteriorate by about 60% and 21% respectively when directory size increases from 10,000 to 50,000. Beyond this limit, the directory is out-of-memory due to system RAM constraints, and increasing the directory size from 50,000 to 100,000 entries results in a sharper increase in processing time of another 87.5%, and a decrease in throughput by 23%.
- *Scaling with entry size.* The scaling of performance with the entry size in the directory is determined by the front-end processing, mainly an increase in the time for ASN.1 encoding of the retrieved entry, as long as the directory is in-memory. An increase in entry size from 488 bytes to 4880 bytes for a 5,000 entry directory results in an increase in processing time of 8 ms at moderate load, 88% of which is due to the increased ASN.1 encoding time, and a throughput deterioration of about 30%. However, for a directory with 10,000 entries, the cache size is still limited to 5,000 by the system RAM, and a similar increase in entry size results in a much larger throughput deterioration of about 70%, mainly due to the increase in data transfer latency from the disk.

- *CPU capability.* For an in-memory directory, the CPU is a significant bottleneck. Using dual processors improves performance by 40%.
- *Session re-use.* In general, higher session re-use leads to improved performance. A 60% gain in performance is obtained when the session is left open during the duration of the experiment, relative to when a connection is opened and closed for each search request.

In conclusion, we believe that the results show that OpenLDAP slapd is a potential candidate for supporting policy administration in the differentiated service environment as well as in other network applications that need dynamic directory access support. In future, we plan to evaluate other LDAP servers based on the criteria developed in this paper, which may run on different platforms and with different implementations.

## References

- [1] I. Busse, B. Deffner, and H. Schulzrinne, “Dynamic QoS control of multimedia applications based on RTP,” *Computer Communications*, vol. 19, pp. 49–58, Jan 1996.
- [2] E. Amir, S. McCanne, and H. Zhang, “An application level video gateway,” in *Proc. of ACM Multimedia*, (San Francisco, California), Nov 1995.
- [3] I. Kouvelas, V. Hardman, and J. Crowcroft, “Network adaptive continuous-media applications through self organised transcoding,” in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Cambridge, England), pp. 241–255, Jul 1998.
- [4] P. Pan and H. Schulzrinne, “Do we need resource reservation?,” in *to be submitted*.
- [5] SWITCH, “Switchlan traffic statistics.” <http://www.switch.ch/lan/stat/>.
- [6] NORDUnet, “Nordunet network statistics.” <http://www.nordu.net/stats/>.
- [7] ABOVE.net, “Above.net’s real-time network status.” <http://stats.sjc.above.net/traffic/>.
- [8] BBC Internet Services, “Internet link usage.” <http://support.bbc.co.uk/support/mrtg/internet/>.

- [9] A. Odlyzko, “The history of communications and its implications for the Internet,” technical report, AT&T Labs, Florham Park, NJ, Jun 2000.
- [10] W. B. Norton, “Internet service providers and peering,” in *Proc. of NANOG 19*, (Albuquerque, New Mexico), Jun 2000.
- [11] R. Braden, D. Clark, and S. Shenker, “Integrated services in the internet architecture: an overview,” Request for Comments (Informational) 1633, Internet Engineering Task Force, Jun 1994.
- [12] K. Nichols, V. Jacobson, and L. Zhang, “A two-bit differentiated services architecture for the internet,” Request for Comments 2638, Internet Engineering Task Force, Jul 1999.
- [13] S. Shenker, C. Partridge, and R. Guerin, “Specification of guaranteed quality of service,” Request for Comments (Proposed Standard) 2212, Internet Engineering Task Force, Sept. 1997.
- [14] J. Wroclawski, “Specification of the controlled-load network element service,” Request for Comments (Proposed Standard) 2211, Internet Engineering Task Force, Sept. 1997.
- [15] S. Brim, B. Carpenter, and F. L. Faucheur, “Per hop behavior identification codes,” Request for Comments 2836, Internet Engineering Task Force, May 2000.
- [16] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, “Assured forwarding PHB group,” Request for Comments 2597, Internet Engineering Task Force, Jun 1999.
- [17] V. Jacobson, K. Nichols, and K. Poduri, “An expedited forwarding PHB,” Request for Comments 2598, Internet Engineering Task Force, Jun 1999.



- [18] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation protocol (RSVP) – version 1 functional specification,” Request for Comments (Proposed Standard) 2205, Internet Engineering Task Force, Sept. 1997.
- [19] X. Wang and H. Schulzrinne, “Comparison of adaptive internet multimedia applications,” *IEICE Transactions on Communications*, vol. 82, pp. 806–818, Jun 1999.
- [20] S. Jamin, S. J. Shenker, and P. B. Danzig, “Comparison of measurement-based admission control algorithms for controlled-load service,” in *Infocom*, (Kobe, Japan), p. 973, Apr 1997.
- [21] R. Yavatkar, D. Pendarakis, and R. Guerin, “A framework for policy-based admission control,” Request for Comments 2753, Internet Engineering Task Force, Jan 2000.
- [22] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An architecture for differentiated service,” Request for Comments 2475, Internet Engineering Task Force, Dec 1998.
- [23] I. . B. B. Information in <http://www.merit.edu/working.groups/i2-qbone-bb>.
- [24] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry, “The COPS (common open policy service) protocol,” Request for Comments 2748, Internet Engineering Task Force, Jan 2000.
- [25] P. Calhoun, G. Zorn, P. Pan, , and H. Akhtar, “DIAMETER framework document,” Internet Draft, Internet Engineering Task Force, Jun 2000. Work in progress.

- [26] J. Case, D. Harrington, R. Presuhn, and B. Wijnen, "Message processing and dispatching for the simple network management protocol (SNMP)," Request for Comments 2572, Internet Engineering Task Force, Apr 1999.
- [27] A. Hafid, G. V. Bochmann, and B. Kerherve, "A quality of service negotiation procedure for distributed multimedia presentational applications," in *Proceedings of the Fifth IEEE International Symposium On High Performance Distributed Computing (HPDC-5)*, (Syracuse, USA), 1996.
- [28] T. F. Abdelzaher, E. M. Atkins, and K. Shin, "Qos negotiation in real-time systems and its application to automated flight control," 1999.
- [29] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, "On quality of service optimization with discrete qos options," in *Proceedings of the IEEE Real-time Technology and Applications Symposium*, Jun 1999.
- [30] G. Bianchi, A. Campbell, and R.-F. Liao, "On utility-fair adaptive services in wireless networks," in *6th International Workshop on Quality of Service (IEEE/IFIP IWQOS'98)*, 1998.
- [31] T. Sakatani, "Congestion avoidance for video over IP networks," in *Proc. of First International COST 237 Workshop on Multimedia Transport and Tele-services*, (Vienna, Austria), pp. 256–273, Springer, Nov 1994.
- [32] K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith, "Adaptive, best-effort delivery of digital audio and video across packet switched networks," in *Third International Workshop on network and operating system support for digital audio and video*, (San Diego, California), pp. 1–12, IEEE Computer and Communications Societies, Nov 1992.

- [33] H. Kanakia, P. Mishra, and A. Reibman, "An adaptive congestion control scheme for real-time packet video transport," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (San Francisco, California), pp. 20–31, ACM/IEEE, Sept. 1993.
- [34] S. Jacobs and A. Eleftheriadis, "Real-time dynamic rate shaping and control for internet video applications," in *Proc. of First IEEE Workshop on Multimedia Signal Processing*, (Princeton, New Jersey), Jun 1997.
- [35] S. Jacobs and A. Eleftheriadis, "Streaming video using dynamic rate shaping and TCP congestion control," *Journal of Visual Communication and Image Representation*, vol. 9, pp. 211–222, Sept. 1998.
- [36] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, England), pp. 58–67, ACM, Aug 1994.
- [37] D. Sisalem and H. Schulzrinne, "The loss-delay adjustment algorithm: A TCP-friendly adaptation scheme," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Cambridge, England), pp. 215–226, Jul 1998.
- [38] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," *ACM Computer Communication Review*, vol. 26, pp. 117–130, Oct 1996.
- [39] L. Wu, R. Sharma, and B. Smith, "Thin streams: An architecture for multicasting layered video," in *Proc. International Workshop on Network and Op-*

- erating System Support for Digital Audio and Video (NOSSDAV)*, (St. Louis, Missouri), May 1997.
- [40] D. Sisalem and F. Emanuel, "QoS control using adaptive layered data transmission," in *Proc. of IEEE Multimedia Systems*, (Austin, Texas), Jun 1998.
- [41] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments (Proposed Standard) 1889, Internet Engineering Task Force, Jan 1996.
- [42] B. Vickers, C. Albuquerque, and T. Suda, "Adaptive multicast of multi-layered video: Rate-based and credit-based approaches," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), p. 1073, March/April 1998.
- [43] S. Kalyanaraman, *Traffic Management for the Available Bit Rate (ABR) Service in Asynchronous Transfer Mode (ATM) Networks*. PhD thesis, The Ohio State University, Columbus, Ohio, 1997.
- [44] J. F. MacKie-Mason and H. Varian, "Pricing congestible network resources," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 1141–1149, Sept. 1995.
- [45] H. Jiang and S. Jordan, "A pricing model for high speed networks with guaranteed quality of service," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), Mar 1996.
- [46] S. Low and P. Varaiya, "An algorithm for optimal service provisioning using resource pricing," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Toronto, Canada), Jun 1994.

- [47] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.
- [48] F. P. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, vol. 8, pp. 33–37, 1997.
- [49] D. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, pp. 1–14, Jun 1989.
- [50] R. J. Gibbens and F. P. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, pp. 1969–1985, 1999.
- [51] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," Request for Comments 2481, Internet Engineering Task Force, Jan 1999.
- [52] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, "Pricing in computer networks: Motivation, formulation, and example," *IEEE/ACM Transactions on Networking*, vol. 1, Dec 1993.
- [53] N. Anerousis and A. A. Lazar, "A framework for pricing virtual circuit and virtual path services in atm networks," in *ITC-15*, Dec 1997.
- [54] D. F. Ferguson, C. Nikolaou, and Y. Yemini, "An economy for flow control in computer networks," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Ottawa, Canada), pp. 110–118, IEEE, Apr 1989.
- [55] E. W. Fulp and D. S. Reeves, "Distributed network flow control based on dynamic competitive markets," in *Proceedings International Conference on Network Protocol (ICNP'98)*, Oct 1998.

- [56] J. Sairamesh, “Economic paradigms for information systems and networks,” in *PhD thesis, Columbia University*, (New York), 1997.
- [57] A. Odlyzko, “Paris metro pricing: The minimalist differentiated services solution,” in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Basking Ridge, New Jersey), Jun 1999.
- [58] K. Kumaran, M. Mandjes, D. Mitra, and I. Saniee, “Resource usage and charging in a multi-service multi-qos packet network,” Dec 1999.
- [59] A. Elwalid, D. Mitra, and R. H. Wentworth, “A new approach for allocating buffers and bandwidth to heterogeneous regulated traffic in an ATM node,” *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1115–1127, Aug 1995.
- [60] A. Elwalid and D. Mitra, “Design of generalized processor sharing schedulers which statistically multiplex heterogeneous QoS,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar 1999.
- [61] J. F. MacKie-Mason and H. Varian, “Pricing the internet,” in *Kahn and Keller (eds): Public Access to the Internet*, (Cambridge, MA), pp. 269–314, MIT Press, 1995.
- [62] H. Varian, *Microeconomic Analysis*. W.W. Norton & Co, 1993.
- [63] N. Semret and A. Lazar, “The progressive second price auction mechanism for network resource sharing,” in *8th International Symposium on Dynamic Games*, (Netherlands), Jul 1998.

- [64] G. Fankhauser, B. Stiller, C. Vogtli, and B. Plattner, “Reservation-based charging in an integrated services network,” in *Proceedings of the 4th Informatics Telecommunications Conference*, (Boca Raton), Mar 1999.
- [65] P. Reichl, G. Fankhauser, and B. Stiller, “Auction models for multi-provider internet connections,” in *Tagungsband zur 10. GI/ITG-Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen (MMB '99)*, Sept. 1999.
- [66] M. Karsten, J. Schmitt, L. Wolf, and R. Steinmetz, “An embedded charging approach for RSVP,” in *Proceedings of 6th IEEE/IFIP International Workshop on Quality of Service (IWQoS'98)*, (Napa, California), pp. 91–100, IEEE/IFIP, May 18–20 1998.
- [67] A. Gulbrandsen, P. Vixie, and L. Esibov, “A DNS RR for specifying the location of services (DNS SRV),” Request for Comments 2782, Internet Engineering Task Force, Feb 2000.
- [68] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, “Service location protocol,” Request for Comments 2165, Internet Engineering Task Force, Jun 1997.
- [69] RADB, “RADB, a distributed database for internet routing registry.” <http://www.radb.net>.
- [70] S. Shenker and J. Wroclawski, “General characterization parameters for integrated service network elements,” Request for Comments 2215, Internet Engineering Task Force, Sept. 1997.
- [71] J. Wroclawski, “The use of RSVP with IETF integrated services,” Request for Comments 2210, Internet Engineering Task Force, Sept. 1997.

- [72] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," Request for Comments 2474, Internet Engineering Task Force, Dec 1998.
- [73] O. Schelen and S. Pink, "Resource reservation agents in the internet," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Cambridge, England), pp. 153–156, Jul 1998.
- [74] P. Pan, E. Hahne, and H. Schulzrinne, "The border gateway reservation protocol (BGRP) for tree-based aggregation of inter-domain reservations," *Journal of Communications and Networks*, Jun 2000.
- [75] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," Request for Comments 1654, Internet Engineering Task Force, Jul 1994.
- [76] M. Hamilton and R. Wright, "Use of DNS aliases for network services," Request for Comments 2219, Internet Engineering Task Force, Oct 1997.
- [77] P. V. Mockapetris, "Domain names - implementation and specification," Request for Comments 1035, Internet Engineering Task Force, Nov 1987.
- [78] F. Baker, B. Lindell, and M. Talwar, "RSVP cryptographic authentication," Request for Comments 2747, Internet Engineering Task Force, Jan 2000.
- [79] L. Berger and T. O'Malley, "RSVP extensions for IPSEC data flows," Request for Comments 2207, Internet Engineering Task Force, Sept. 1997.
- [80] P. Reichl, S. Leinen, and B. Stiller, "A practical review of pricing and cost recovery for internet services," in *Proc. of the 2nd Internet Economics Workshop Berlin (IEW '99)*, (Berlin, Germany), May 1999.



- [81] J. Altmann, B. Rupp, and P. Varaiya, “Internet user reactions to usage-based pricing,” in *Proceedings of the 2nd Berlin Internet Economics Workshop (IEW ’99)*, (Berlin, Germany), May 1999.
- [82] P. Pan and H. Schulzrinne, “YESSIR: a simple reservation mechanism for the Internet,” *ACM Computer Communication Review*, vol. 29, pp. 89–101, Apr 1999.
- [83] X. Wang and H. Schulzrinne, “An integrated resource negotiation, pricing, and qos adaptation framework for multimedia applications,” in *IEEE Journal on Selected Areas in Communications*, vol. 18, 2000.
- [84] R. Guérin, H. Ahmadi, and M. Naghshineh, “Equivalent capacity and its application to bandwidth allocation in high-speed networks,” *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 968–981, Sept. 1991.
- [85] F. P. Kelly, S. Zachary, and I. Zeidins, “Notes on effective bandwidths,” *Stochastic Networks: Theory and Applications*, pp. 141–168, 1996.
- [86] C. Courcoubetis and V. Siris, “Managing and pricing service level agreements for differentiated services,” in *Proc. of 7th IEEE/IFIP International Workshop on Quality of Service (IWQoS’99)*, (London, UK), Jun 1999. GMD Report 95.
- [87] J. F. MacKie-Mason, “A smart market for resource reservation in a multiple quality of service information network,” technical report, University of Michigan, Sept. 1997.
- [88] International Telecommunication Union, “Methods for subjective determination of transmission quality,” Recommendation P.800, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Aug 1996.

- [89] C. Lambrecht and O. Verscheure, "Perceptual quality measure using a spatio-temporal model of human visual system," in *Proc. of IS&T/SPIE*, Feb 1996.
- [90] A. Watson and M. A. Sasse, "Evaluating audio and video quality in low-cost multimedia conferencing systems," in *Interacting with Computers*, vol. 8, pp. 255–275, 1996.
- [91] J. Janssen, D. D. Vleeschauwer, and G. H. Petit, "Delay and distortion bounds for packetized voice calls of traditional PSTN quality," in *Proceedings of the 1st IP Telephony Workshop (IPTel 2000)*, (Berlin, Germany), pp. 105–110, Apr 2000. GMD Report 95.
- [92] O. Ostberg, B. Lindstrom, and P.-O. Renhall, "Contribution of display size to speech intelligibility in video-phone systems," *International Journal of Human-Computer Interaction*, vol. 1, pp. 149–159, 1989.
- [93] A. H. Anderson, E. G. Bard, C. Sotillo, A. Newlands, and G. Doherty-Sneddon, "Limited visual control of the intelligibility of speech in face-to-face dialogue," in *Perception and Psychophysics*, vol. 59, pp. 580–592, 1997.
- [94] E. A. Isaacs and J. C. Tang, "What video can and cannot do for collaboration: A case study," *Multimedia Systems*, vol. 2, pp. 63–73, 1994.
- [95] Virtual InterNetwork Testbed, "The network simulator - ns (version 2)." <http://www.isi.edu/nsnam/ns/>.
- [96] M. Creis, "RSVP/ns: An implementation of rsvp for the network simulator ns-2," <http://www-student.informatik.uni-bonn.de/greis/rsvpn/index.html>.
- [97] X. Wang and H. Schulzrinne, "Performance study of congestion price based adaptive service," Technical Report CUCS-010-00, Columbia University, New York, Apr 2000.

- [98] D. D. Clark and W. Fang, “Explicit allocation of best-effort packet delivery service,” *IEEE/ACM Transactions on Networking*, vol. 6, pp. 362–373, Aug 1998.
- [99] S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, Aug 1993.
- [100] X. Wang and H. Schulzrinne, “RNAP: A resource negotiation and pricing protocol,” in *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV’99)*, (Basking Ridge, New Jersey), pp. 77–93, Jun 1999.
- [101] RSVP software release, “RSVP software release.”  
<ftp://ftp.isi.edu/rsvp/release>.
- [102] S. Floyd and V. Jacobson, “Link-sharing and resource management models for packet networks,” *IEEE/ACM Transactions on Networking*, vol. 3, Aug 1995.
- [103] K. Cho, “ALTQ: Alternate queueing for freebsd.”  
<http://www.csl.sony.co.jp/person/kjc/kjc/software.html>.
- [104] S. Herzog, “RSVP extensions for policy control,” Request for Comments 2750, Internet Engineering Task Force, Jan 2000.
- [105] D. Sisalem and H. Schulzrinne, “The multimedia internet terminal (MIInT),” *Telecommunications Systems*, vol. 9, pp. 423–444, Sept. 1998.
- [106] R. Vaccaro, *Digital control, a state space approach*. McGraw Hill, 1995.

- [107] University College Longon, “Mbone conferencing application.” <http://www-mice.cs.ucl.ac.uk/multimedia/software/>.
- [108] Mbus Technology, “Message bus.” <http://www.mbus.org/>.
- [109] T. Small, D. Hennessy, and F. Dawson, “Calendar attributes for vcard and LDAP,” Request for Comments 2739, Internet Engineering Task Force, Jan 2000.
- [110] B. Aboba, “Lightweight directory access protocol (v3): Schema for the routing policy specification language (rpsl),” internet draft, Internet Engineering Task Force, Nov 1997. Work in progress.
- [111] E. Ellesson, D. Verma, R. Rajan, and S. Kamat, “Schema for service level administration of differentiated services and integrated services in networks,” Internet Draft, Internet Engineering Task Force, Feb 1998. Work in progress.
- [112] B. Aboba, “Extension for PPP authentication,” Internet Draft, Internet Engineering Task Force, Nov 1997. Work in progress.
- [113] L. Bartz, “Ldap schema for role based access control,” internet draft (work in progress), Internet Engineering Task Force, Oct 1995.
- [114] R. Rajan, J. C. Martin, S. Kamat, M. See, R. Chaudhury, D. Verma, G. Powers, and R. Yavatkar, “Schema for differentiated services and integrated services in networks,” Internet Draft, Internet Engineering Task Force, Oct 1998. Work in progress.
- [115] W. Yeong, T. Howes, and S. Kille, “Lightweight directory access protocol,” Request for Comments 1777, Internet Engineering Task Force, Mar 1995.

- [116] M. Wahl, T. Howes, and S. Kille, “Lightweight directory access protocol (v3),” Request for Comments 2251, Internet Engineering Task Force, Dec 1997.
- [117] B. M., R. Jennings, S. Rao, and D. Verma, “Supporting service level agreements using differentiated services,” Internet Draft, Internet Engineering Task Force, Nov 1998. Work in progress.
- [118] OpenLdap, “The open source ldap suite.” <http://www.OpenLDAP.org>.
- [119] Sleepycat Software, “The berkeley database.” <http://www.sleepycat.com>.
- [120] J. Nagle, “Congestion control in IP/TCP internetworks,” *ACM Computer Communication Review*, vol. 14, pp. 11–17, Oct 1984.
- [121] J. Heidemann, “Performance interactions between P-HTTP and TCP implementations,” *ACM Computer Communication Review*, vol. 27, pp. 65–73, Apr 1997.
- [122] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud’hommeaux, H. W. Lie, and C. Lilley, “Network performance effects of HTTP/1.1, CSS1, and PNG,” Tech. Rep. 970914, W3C, Cannes, France, Sept. 1997.
- [123] Mindcraft, “Directory server certified performance reports.” <http://www.mindcraft.com/perfreports/ldap/>.