Distributed Algorithms and Protocols for Scalable Internet Telephony

Jonathan Rosenberg

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

ABSTRACT

Distributed Algorithms and Protocols for Scalable Internet Telephony

Jonathan Rosenberg

Internet telephony service is defined as the provision of real-time, interactive, multimedia telecommunications services between human users, using the public Internet.

The most difficult problem in providing Internet telephony is to overcome the increased jitter, delay, and loss (as compared to circuit-switched networks) suffered by voice. Past work has separately investigated Forward Error Correction (FEC) and playout buffer adaptation mechanisms to resolve these problems. We show that these mechanisms must be considered jointly. We propose and simulate a number of algorithms for integrating FEC into playout buffer adaptation schemes, and show that they are superior to non-integrated algorithms.

Receiving feedback about network transport quality is essential for supporting adaptive applications. We examine the issues surrounding scalability of transport feedback in large scale multicast groups. We present, analyze, and simulate a class of algorithms termed reconsideration, which support congestion controlled feedback in highly dynamic groups, and then show how the memory requirements of our algorithms can be reduced.

We consider signaling protocols for providing call establishment, management, features, and applications. After an analysis of existing Internet telephony signaling protocols, we propose a new protocol, the Session Initiation Protocol (SIP), which overcomes the limitations of existing protocols. We describe an implementation of this protocol in software, and discuss applications we have built with it.

We consider interconnection with the telephone network, and focus on the problem of discovery of telephony gateways. We show that this is a subset of a broader wide area service discovery problem. After reviewing existing protocols for resource discovery (and finding them lacking for wide area applications), we present a scalable protocol for wide area service discovery, which is ideal for discovery of gateways, amongst other resources.

Finally, we consider the problem of a service architecture for Internet telephony, which provides features and complex applications to users. We review the service architectures that have been presented in the literature. We then propose our architecture, the application component architecture, which combines the best aspects of existing work. We show how this architecture can be used to provide several complex applications.

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, Professor Henning Schulzrinne deserves recognition for his tremendous contributions to this work, in addition to recognition for the immeasurable impact he has had on my career and professional development. Professor Schulzrinne provided guidance, knowledge, insight and direction whenever it was needed. He introduced me to many of the pioneers in the Internet, giving me an opportunity to learn from them as well. He introduced me to the Internet Engineering Task Force (IETF), a professional organization through which much of this work has found a commercial outlet. I will be forever in his debt.

Several other individuals deserve special recognition for their impact on this thesis.

Lili Qiu deserves recognition as a key contributor of much of the work in transport chapter. She prepared many of the simulations and the plots used in the section on playout buffer integration. She was also responsible for the fine tuning of parameters that took place for many of the algorithms. She contributed the idea of the previous optimal algorithm, and helped refine the formulation for the analytical algorithm.

The taxonomy for feedback presented in the feedback section was the result of joint work with Joerg Nonnenmacher and Markus Hoffman from Bell Laboratories. Daniel Rubenstein deserves complete credit for the proof of the steady-state unconditional reconsideration rate. The initial concept of reconsideration (which we now call conditional reconsideration) was proposed by Henning Schulzrinne. The original concept of SSRC sampling was proposed by Steve Casner. Much of this work has been reviewed and commented on by members of the AVT working group in IETF, and in particular, Steve Casner, Colin Perkins and Bill Fenner.

The work on generating requirements for the general service discovery problem was done jointly with Erik Guttman from SUN Microsystems and Ryan Moats from AT&T Labs. Dave

Oran from Cisco deserves credit for first recognizing that the bilateral model is nearly identical to the BGP4 model. Erik Guttman provided many comments on the WASRV protocol which made their way into this thesis. Dina Katabi contributed many ideas and thoughts on performance for WASRV.

Mark Handley (ACIRI), Henning Schulzrinne and Eve Schooler (Caltech) deserve recognition for the initial work on SIP. Pete Mataga from dynamicsoft contributed to many of the ideas on the component architecture for SIP, along with assistance in defining some of the combined services presented here. Jon Peterson from Level(3) and Gonzalo Camarillo from Ericcson provided helpful and valuable input on third party call control.

The implementation of the ACA is ongoing work at dynamicsoft, where I am currently employed. Many software engineers deserve credit for the long nights involved in realizing the controller described in this dissertation, and development of applications using this architecture. They include Peter Mataga, Prasad Sripathi, Edgar Villanueva, John Eichelsdorfer, Srinivas Maganti, Srinivas Dharmaji, Andrew McGrath, Kevin Grey, Ajay Deo, Kelvin Porter, Ed Gokhman, Xin Feng, David Ladd, and Anders Kristensen. Additional thanks to Eric Burger from Snowshore Networks, and Ed Yackey from Voyant, for their comments and support of this architecture.

Last but not least, I would like to thank my wife Michelle, and son Joshua, for providing encouragement and understanding throughout my educational career.

This thesis is dedicated to Michelle and Joshua, for their support and understanding.

Chapter 1

Introduction

The problem of carrying voice on IP-based packet networks was first identified by Cohen et al. [1] in 1977. Much of Cohen's work, and the work that followed, focused on recovering from the lower quality offered by packet networks (asynchronous delivery, high packet loss rates, high latencies, substantial packet jitter) as compared to circuit-switched networks. Of particular interest was recovery from packet jitter through the use of receiver jitter buffers, and loss compensation techniques to handle packet loss. However, as usage of IP networks for multimedia delivery increased, the community began to realize that the delivery of multimedia communications services over IP networks was a much broader, and much more difficult problem.

The problem is more difficult because the actual transport of multimedia from point A to point B is only a small piece of an overall multimedia communications service. Signaling protocols are needed to establish and maintain calls. Features need to be defined and architected. Multiparty conferences, ranging from three people to millions of people, need to be considered. Interoperability with the legacy Public Switched Telephone Network (PSTN) needs to be considered. Quality must be provided, not just for the media itself, but for the service overall. All of these, taken together, are needed to provide a complete *Internet telephony service*. As a result, we define Internet telephony service as the provision of real-time, interactive, multimedia telecommunications services between human users, using the public Internet.

1.1 Components of an Internet Telephony Service

Many systems need to be designed and developed to provide a complete Internet telephony service, as defined above. We can identify at least five that have been considered to date:

Transport: The system that carries voice and video between two points on an IP network. The transport system is responsible for handling packet loss, packet jitter, and delay. On the Internet, voice and video transport are provided by the Real-Time Transport Protocol (RTP) [2].

Transport Control: The system that manages and controls the behavior of the transport algorithms and protocols. It provides feedback to senders (and third parties) on the loss, delay, and jitter being provided by the transport network. On the Internet, this is provided by the Real Time Control Protocol (RTCP) [2].

Call Signaling: The system that sets up, tears down, and manages the multimedia calls which make use of the underlying transport and transport control systems.

Applications: The system which provides Internet telephony features and applications to users. Examples of these include call forwarding, transfer, conferencing, personal assistant, and pre-paid calling cards. The application system makes extensive use of signaling protocols.

Resource Discovery: The system that allows for the discovery of network servers, such as gateways, feature servers, bridges, and media servers, which are used by the signaling and services system to provide call control and features.

Other components, such as management systems, are also important for Internet telephony. However, we have chosen to focus on these five above, since we feel they represent the core of an Internet telephony service.

In this thesis, we investigate the problems of providing a complete Internet telephony service, focusing on the differences between IP networks and circuit switched networks, and their implications on providing the service.

1.1.1 Organization

This dissertation is divided into five main chapters, with each focusing on problems in each of the five components we describe above. Rather than reviewing existing literature up front, we distribute the review in each chapter.

The first chapter, on transport, more clearly defines the implications of the Internet on voice quality through measurements we have taken and analyzed. Our analysis focuses on the voice quality observed by the user after recovery algorithms have been applied, and demonstrates that media-unaware Forward Error Correction (FEC) has advantages over media-aware FEC when used with low-rate codecs. After reviewing past work on addressing the quality problems, we identify a new problem introduced by an interaction between two existing mechanisms, namely playout buffer adaptation and FEC. We analyze the problem and propose new classes of playout buffer algorithms that take this interaction into account, and then demonstrate the improvement in performance they provide. We propose a new protocol for carrying FEC within RTP, and we describe a novel algorithm that allows a receiver to utilize a large class of FEC codes.

The second chapter considers transport control. It introduces three problems we have discovered in the existing RTCP control mechanisms, which are congestion, state storage, and delay. We examine the possible set of solutions proposed elsewhere in the literature, aided by a taxonomy we developed for this purpose. We conclude that the ideal solution is one that provides backwards compatible improvements to the existing RTCP mechanisms. We then propose a set of new RTCP control algorithms called *reconsideration,* which can eliminate the RTCP congestion problems. We develop analytical models for the reconsideration algorithms, and through these models, demonstrate the existence of the congestion problem and the performance of our solutions. We back up the analysis with simulations, using a simulator we constructed. To resolve the state storage problems, we propose an algorithm for dynamic sampling which can reduce the memory requirements of systems in large conferences, with little impact on performance. We demonstrate these claims with simulations and analysis.

The third chapter considers signaling protocols for providing call establishment, call management, features, and applications. After an analysis of existing Internet telephony signaling protocols, we propose a new protocol, the Session Initiation Protocol (SIP), which overcomes the limitations of existing protocols. We describe an implementation of this protocol in software, and discuss applications we have built with it.

The fourth chapter considers resource discovery. We define the problem of Internet telephony gateway discovery, required for interconnection with the telephone network. We show that this is a subset of a broader wide area service discovery problem. After reviewing existing protocols for resource discovery (and finding them lacking for wide area applications), we present a scalable protocol for wide area service discovery, called the *Wide Area Service Discovery Protocol* which is ideal for discovery of gateways, amongst other resources.

The fifth chapter considers a service architecture for Internet telephony, which provides features and complex applications to users. We define requirements of a service architecture for Internet telephony, and then review the service architectures that have been presented in the literature. We then propose our architecture, the Application Component Architecture (ACA), which combines the best aspects of existing work. We show how this architecture can be used to provide several complex applications.

The final chapter concludes and reviews our findings.

Chapter 2

Transport

2.1 Introduction

As we have mentioned in the introduction, the best effort delivery service offered by the Internet results in highly variable packet delays, loss, and jitter [3, 4]. The packet loss probabilities and packet delays are often beyond what is considered acceptable for good speech quality. The International Telecommunications Union (ITU) has recommended one-way delays no greater than 150 ms for most applications [5], with a limit of 400 ms for acceptable voice communication. Tolerable loss rates depend heavily on the speech codec in use [6], and can range from 0 to 10% [7]. The implication is that the best effort Internet will not always be sufficient for high quality voice.

There are two approaches that can be taken to combat this problem. The first is to provide improved network layer performance. The Internet Engineering Task Force (IETF) has proposed the Integrated Services architecture [8, 9] as one approach to the problem. *Intserv*, and its companion signaling protocol, the Resource Reservation Protocol (RSVP) [10, 11], allow hosts to request end-to-end QoS. Using the guaranteed service model, they can request a bounded delay with zero loss. The controlled load model allows hosts to request service identical to an unloaded network, without specific numerical guarantees. However, scaling concerns (among other difficulties) have led the IETF to consider a more lightweight approach to network QoS, called differentiated services [12, 13, 14, 15].

The second approach for reducing loss and delay is through end-to-end adaptive mechanisms. In this case, end systems measure the service being delivered by the network (using RTCP [2]), and send additional information, and/or run additional algorithms, to improve voice quality. These mechanisms do not rely on explicit support from the network beyond normal packet transport. It is for this reason they are considered end-to-end mechanisms.

Ideally, a well engineered, QoS-aware network would obviate the need for end-to-end adaptation. However, the heterogeneous nature of the Internet leads us to conclude that it is unlikely for any solution to be ubiquitously deployed any time soon. As such, end-to-end adaptive mechanisms are, and will remain, critical for high quality voice.

One of the primary mechanisms used for end-to-end compensation for loss is Forward Error Correction (FEC), both media-aware and media-unaware [16]. In this chapter, we consider numerous issues that arise in the usage of FEC. In Section 2.2, we motivate the need for it through measurements of Internet voice transport performance. Then, in Section 2.3, we review the existing schemes for forward error correction, and in Section 2.4 present a novel analysis which demonstrates the superiority of media-unaware FEC for low bitrate codecs. We then consider a number of critical issues that arise in deploying a system that uses FEC. First and foremost, we demonstrate an important interaction between FEC and adaptive playout buffers, and in Section 2.5, develop a set of new playout buffer adaptation algorithms which are *FEC-aware* and demonstrate their superior performance. Finally, we consider how to add FEC to the RTP [2] framework, paying particular attention to supporting sender adaptation.

2.2 Internet Measurements

In order to determine the most appropriate mechanism for end-to-end recovery from packet loss, it is necessary to understand the nature of that loss, and of packet delays.

2.2.1 Previous Work

There have been several studies undertaken to demonstrate the “performance” of the Internet, focusing on metrics such as loss, delay, re-ordering, burstiness and correlation of losses.

The largest study to date was conducted by Paxson [3, 4]. His work used TCP to determine network performance between 35 different pairs of hosts. His study, not surprisingly, revealed substantial variation in most metrics. He observed variations based on time of day, geography, and year. He observed loss probabilities that ranged between 0% and 65%. Paxson also found wide variability in the correlation of losses. From the set of traces collected during November to December of 1995, he examined the distribution of *outages*, which are the duration of time over which there is complete packet loss. He observed that 10% of the outages were less than a few milliseconds, while another 10% were more than a few seconds! Similar variability is observed for delays.

Mukherjee [17] found that end-to-end one way packet delays were well modeled using a shifted gamma distribution, but the parameters of the distribution depended on the path and time of day.

Several studies have been conducted to explore performance of real time media on the Internet. The study by Bolot et al. [18] focused on a single link from INRIA in France to the University of Maryland in the U.S. They found that there was substantial correlation in delays, and demonstrated that this was due to their rapid probe packets piling up behind a large packet in a congested buffer. This *spike* phenomenon was first observed by Mills [19] in 1983. Their study of loss demonstrated correlated losses for packets sent close together. However, for packets sent far apart (where far depends on the bandwidth of the bottleneck link), they found losses to be independent.

Sanghi et al. [20] used UDP to determine the connection properties between a number of hosts. They found that losses generally occurred one at a time, and they observed the spike phenomena later confirmed by Bolot [18].

Yajnik and Kurose [21] examine the spatial and temporal correlation of losses for multicast traffic in the Mbone. Their study consisted of 17 nodes on the Mbone, listening to a variety of sessions. Their results showed that spatial correlation (where multiple participants lose the same packet) was fairly low, and that backbone losses were low except for occasional periods of high loss. Temporally, they found the majority of losses to be isolated. However, there were outliers of very long bursts which were found to contribute heavily to the overall loss probability. These results agree with those by Bolot et al. [18].

Yajnik et al. [22] examine the temporal dependence of packet loss for unicast traffic collected over 128 hours. They find that for packets spaced greater than 1 second apart, losses are uncorrelated. They also find that measuring packet loss over time by using sliding windows over some past number of packets gives much better results than exponential weighted averages of non-overlapping windows.

Handley [23] investigates Mbone performance through RTCP reports. His findings show temporal variations in loss, with rates typically between 0 and 10%.

Maxemchuk and Lo [24] examine network performance for supporting Internet telephony. They use UDP to collect data between several sites, and then apply a static playout buffer and loss compensation to determine the loss rates seen by the application. They also define an objective metric for quality based on the fraction of time the connection has no loss.

2.2.2 Measurement Approach

Our aim is to add to existing work through two main contributions. First, It is useful to take occasional "snapshots" of Internet performance, obtaining new data between new sites. Our new traces add to the overall amount of data available on Internet performance. Secondly, most of the past work on measurements has not considered statistics that reflect end-to-end performance. The focus has been on characterizing network behavior, and not on how the network behavior fits into the overall application performance. It is our aim to fill this gap by considering the impact on an important Internet telephony component, the adaptive playout buffer, on loss and delay.

Our system for measurements is similar to the one used by Yajnik et al. [22], and is depicted in Figure 2.1. We developed two pieces of software - the *station* and the *controller*. The station is based on the tracing tool used by Sanghi [20]. It is a daemon process capable of sending UDP packets of arbitrary size, at arbitrary intervals, to a specified address and port. The packets include an origination timestamp and sequence number. The station is also capable of receiving packets on a specified port, logging the reception time and the origination timestamp and sequence number of the packet. The station can optionally reflect the packet back to the originator. The originator can log the reception time of the reflected packet, the timestamps and

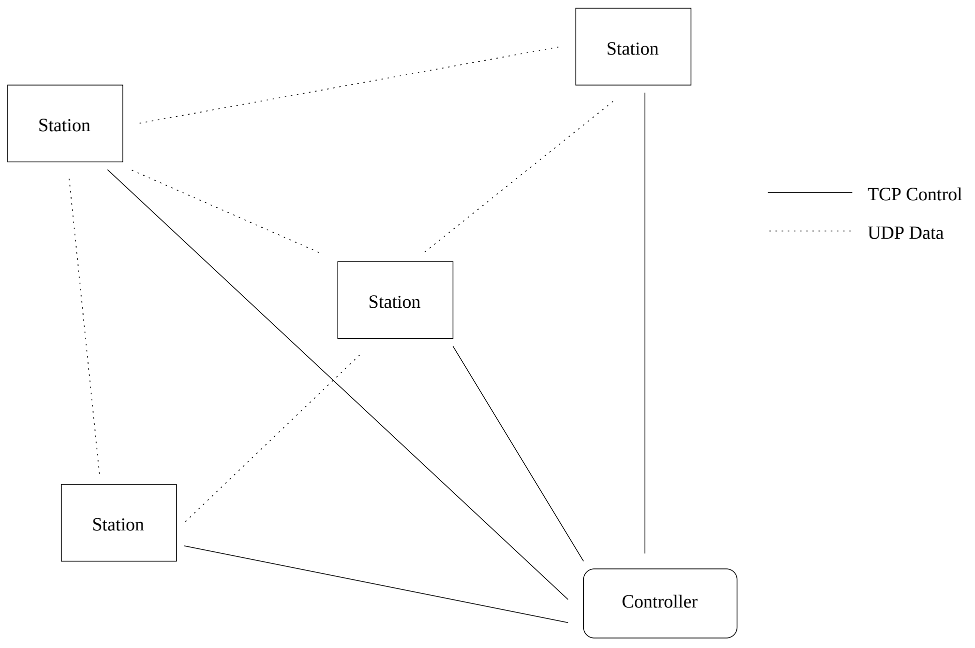


Figure 2.1: Measurement setup

Station Number	Station Location	IP Address
1	GMD Fokus, Germany	193.175.132.184
2	Columbia University, NY, USA	128.59.19.141
3	U. Mass. Amherst, Amherst, MA, USA	128.119.40.203
4	U. California at Santa Cruz, USA	128.114.134.117

Table 2.1: Locations of stations

sequence number within the packet, to disk.

The controller is capable of configuring and starting the stations. It specifies the address, port, packet size, packet interval, and measurement epoch used by the stations. One station is configured to act as the sender, and the other as the receiver. The control is exercised over permanent TCP connections established with each station. Once the station has been instructed to start, it sends periodic keepalives to the controller, updating it with the total number of packets sent and received to date. The controller has a simple shell, which allows commands to be entered manually or through a script.

We placed stations at several sites throughout the world, shown in Table 2.1.

Trace #	Sender	Receiver	Day	Recv. Data?	Sender Data?
1	Germany	Columbia Univ.	9/23/97	Y	N
2	USC	Columbia Univ.	9/22/97	Y	N
3	USC	UMASS	9/23/97	Y	N
4	Columbia Univ.	UMass	9/19/97	N	Y
5	Columbia Univ.	USC	9/18/97	N	Y
6	Columbia Univ.	Germany	9/19/97	N	Y

Table 2.2: Statistics of Traces

We collected data between six pairings of the above four sites. All of the data was collected during the end of September in 1997. The stations were configured to send packets as if they were generated from a G.723.1 [25] speech codec running at 6.3 kb/s over RTP [2], with a single 30 ms frame is placed in each packet. The result is 24 bits of payload in addition to a 40 byte IP/UDP/RTP header. A packet was sent every 30 ms for a total duration of two hours. Due to unfortunate limitations in computational access, we were not always able to obtain the trace files generated at the sender and the receiver. Table 2.2 lists the traces that were collected. The columns *Recv. Data* and *Sender Data* indicate whether the trace files were collected from the receiver and sender, respectively.

The measurements we obtained reflect the performance of the network in delivering packets. To consider the actual performance that an Internet telephony application can expect to see, we simulated the effects of a playout buffer. The playout buffer smooths out network jitter, at the expense of additional delays. Packets which arrive too late are considered lost. The implication is that a playout buffer increases both the loss and the delay seen by an application. In our simulation, the second adaptive algorithm described by Ramjee et al. [26] was used. This algorithm adjusts the playout delay at the beginning of each talkspurt, so that the buffer depth represents the 4 times the standard deviation about the mean packet delays. The algorithm also handles spikes of delay, by increasing the playout delays more rapidly as network delays increase, and reducing the playout delays slowly as they decrease. Since our trace data did not contain silence periods, we adjusted the playout delays based on simulated talkspurts. We used the on-off Markov model for speech described by Brady [27] to generate a sample path of talkspurts and silence periods.

When the beginning of a talkspurt was encountered in the sample path, the playout delay was adjusted according to the algorithm.

The playout buffer can effectively be seen as a filter on the raw trace data. Packets which arrived after their scheduled playout time are removed from the trace, and the receive times are modified to instead reflect their playout times.

We only considered the addition of a playout buffer on those traces where data was collected at the receiver (traces 1, 2, and 3). This is because playout buffer adaptation is normally performed at the receiver of a media stream.

2.2.3 Results for Receivers

For traces 1, 2 and 3, we computed a number of traditional loss metrics, both on the raw data, and on the data filtered with the playout buffer. The metrics we computed were:

Windowed Loss Probability We broke the trace into N non-overlapping windows, and computed the fraction of packets lost in the window. The result is an estimate of the loss probability.

Conditional Loss Probability (CLP) We computed the probability the i^{th} packet is lost, given that the $(i-n)^{th}$ packet was lost.

Burst Length Distribution We computed a histogram of the number of consecutive packets lost.

The use of playout buffers also increases the overall delays seen by the application. For each packet that arrived in time for playout, we computed the difference between its playout time and arrival time. We then computed the distribution of this difference.

Figures 2.2, 2.3, and 2.4 show the mean loss probability for traces 1, 2, and 3, respectively. Each figure contains two lines, one representing the loss probability of the raw trace, and the other representing the loss probability of the trace after the playout buffer has been applied. All traces show substantial variability. Trace 2 is particularly interesting. It shows three distinct regions of loss, the region from sequence numbers 0 to 100,000, which show a loss probability of around 8%, a small region around sequence number 20,000 which shows a complete outage where all

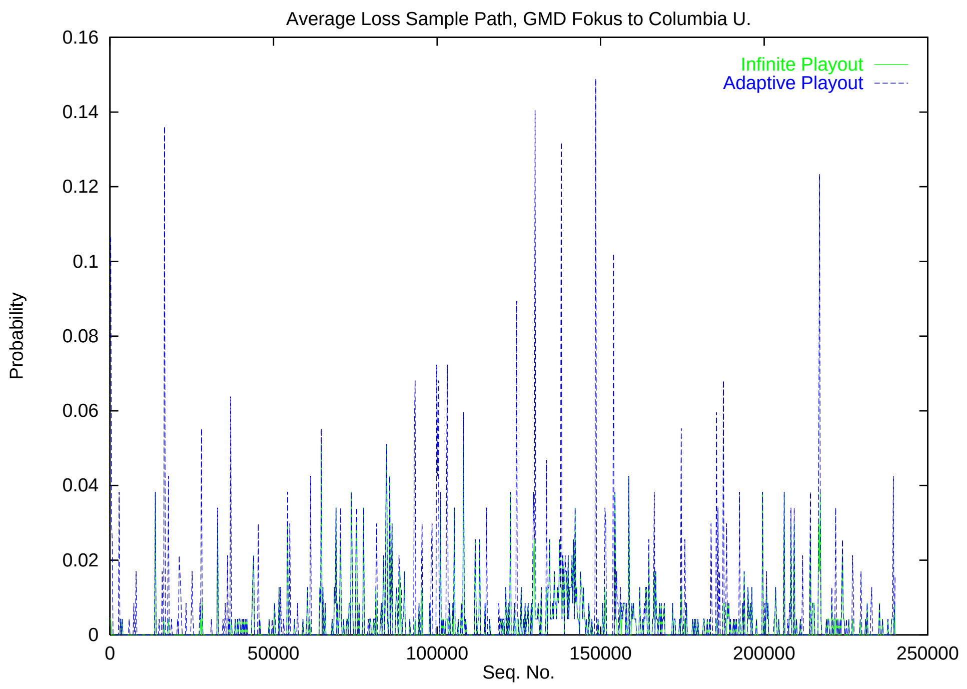


Figure 2.2: Mean loss probability for trace 1

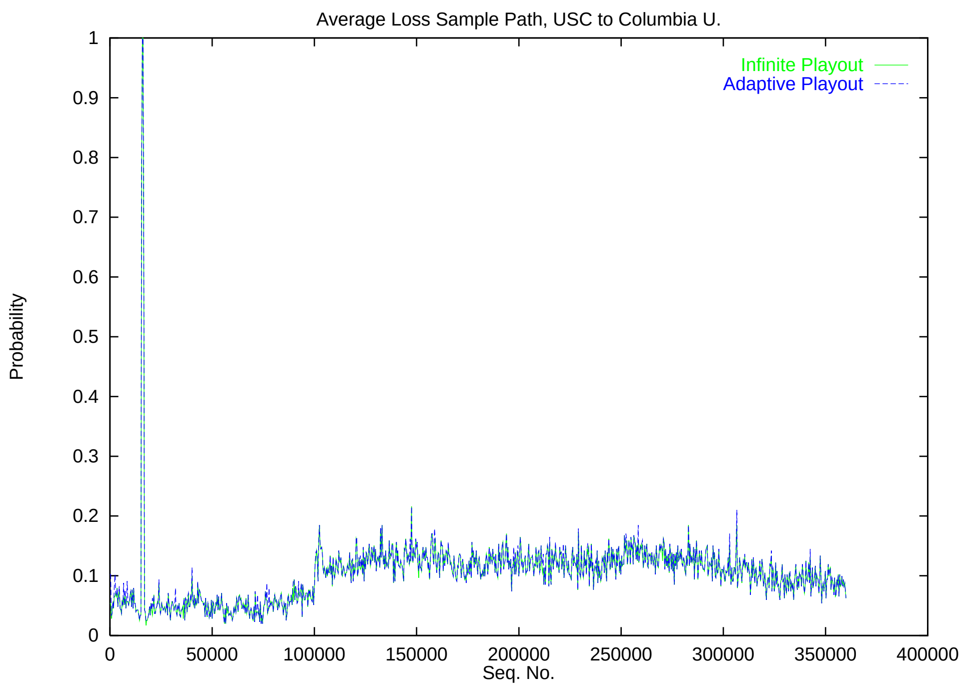


Figure 2.3: Mean loss probability for trace 2

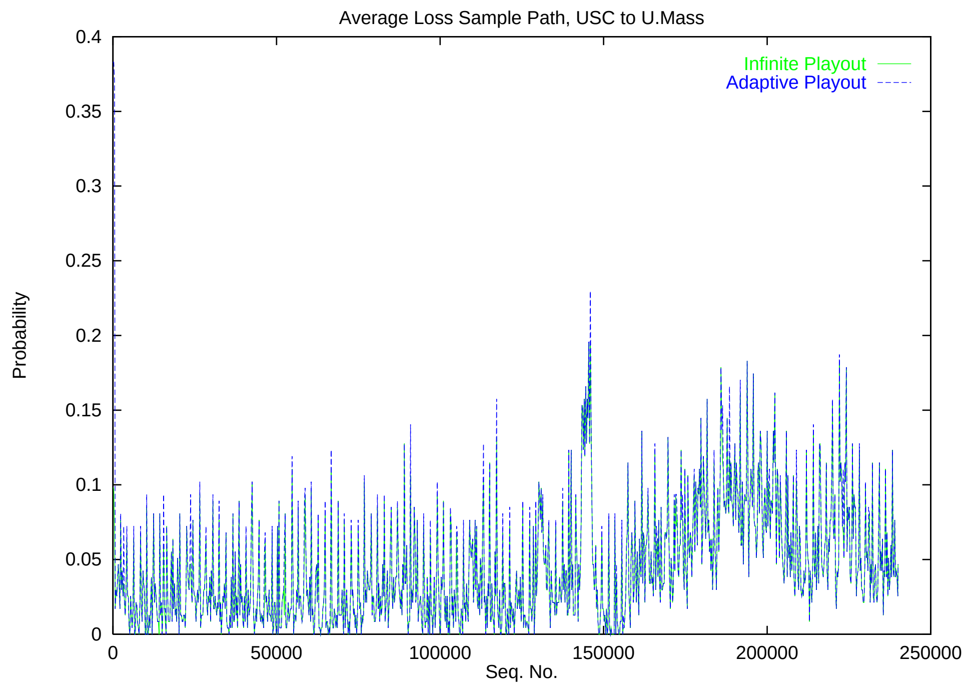


Figure 2.4: Mean loss probability for trace 3

packets are lost, and the region from sequence number 100,000 until the end of the trace, with a loss probability of around 12%. In all three figures, the curves for the raw and playout buffered traces are very close. This means that the playout buffer algorithm is fairly conservative, resulting in little additional packet loss due to playout buffer underrun.

Figures 2.5, 2.6 and 2.7 show the conditional loss probability vs. the lag k for traces 1, 2, and 3, respectively. Each plot shows the conditional loss probability for the raw and "filtered" data (by filtered, we mean that the playout buffer has been applied). They also show the mean loss probability for the raw and unfiltered data. These plots consistently reveal several interesting properties. First, the conditional loss probability for small lags is extremely high. The CLP eventually trails off, approaching the mean loss probability only for substantially long lags. This clearly indicates that packet losses are not independent, as postulated by Bolot [18]. Figures 2.5 and 2.6 also show another interesting feature. The mean loss probabilities between the raw and playout filtered traces are quite close (the flat lines on the bottom which represent the two mean loss probabilities are nearly on top of each other). However, the CLP for small lags is significantly higher as a result of playout buffer adaptation. The effect is present in trace 3, but is

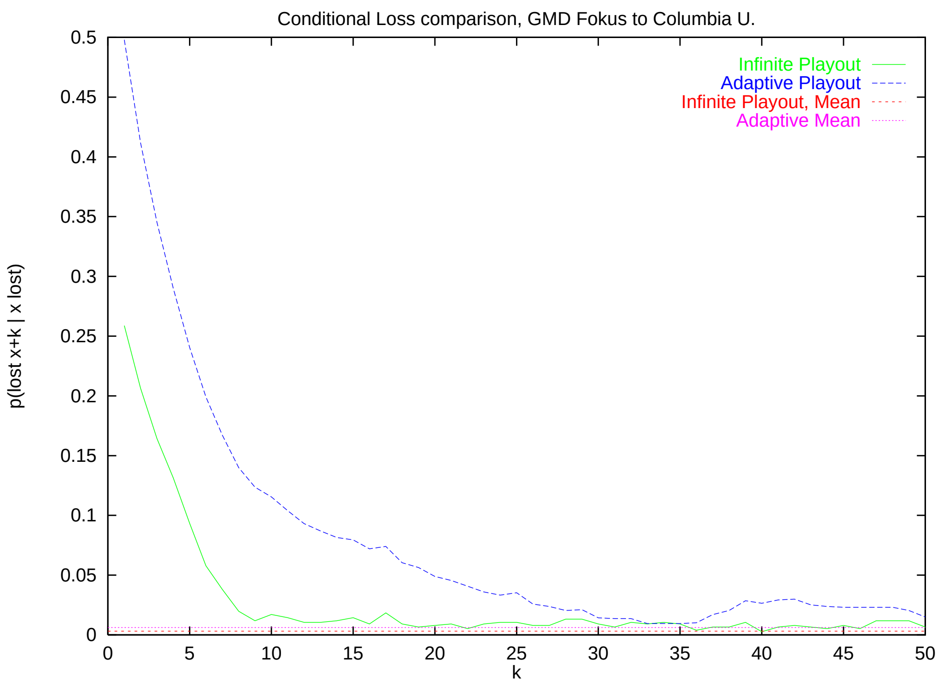


Figure 2.5: Conditional loss probability for trace 1

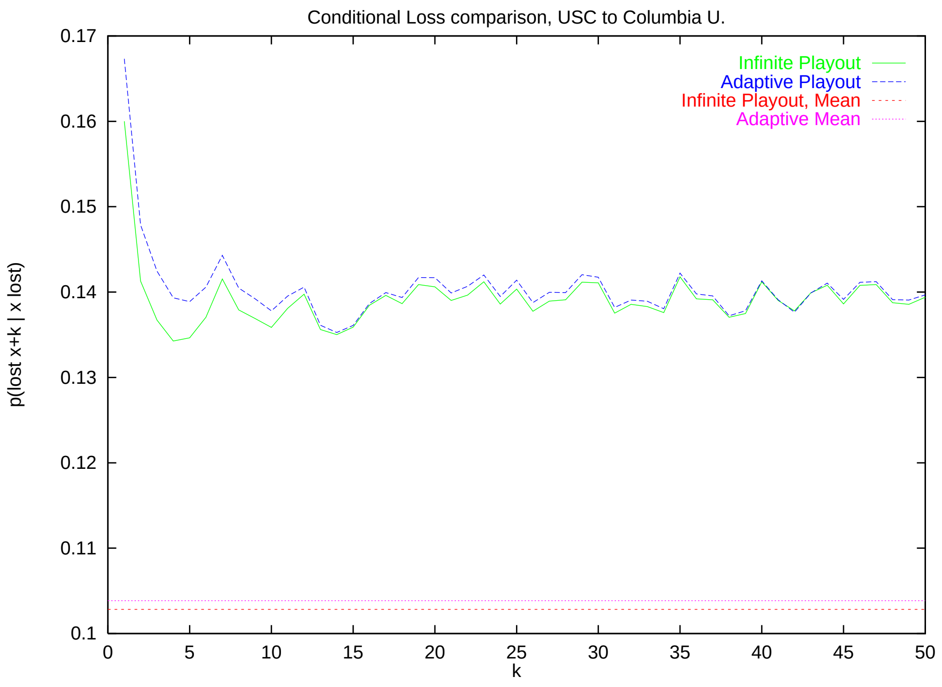


Figure 2.6: Conditional loss probability for trace 2

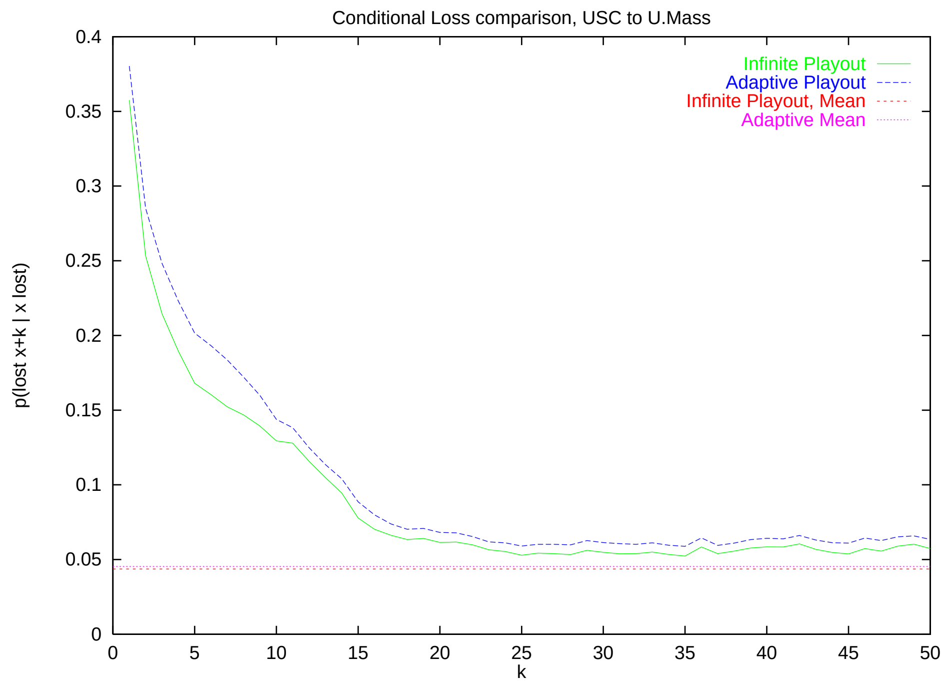


Figure 2.7: Conditional loss probability for trace 3

much less pronounced. We believe this is attributable to the much higher jitter in the trace, which causes overly conservative playout buffer sizing. The conclusion, however, is that the playout buffer algorithm is not affecting the mean loss probabilities, but is having a significant effect on its correlation structure.

The effect can be further examined through the burst length distributions, which are shown in Figures 2.8, 2.9 and 2.10. The probability of each burst length is shown on a logarithmic scale. Each figure shows two curves, one for the raw data, and the other for the data filtered through the playout buffer algorithm. The plots show a linear decrease (on a logarithmic scale) of burst length probability as the length increases to around five or ten. This would indicate an exponential decrease in actual probability. However, the rate of decrease slows down for very large burst lengths, of which there are a significant number. This indicates that packet losses are *usually* independent, with the exception of occasional very long bursts of consecutive losses. These long bursts account for the abnormally high conditional loss probabilities. These figures also illustrate more clearly the effect of the playout buffers. The playout buffers have little impact on the frequency of small burst lengths, but seem to significantly increase the frequency of

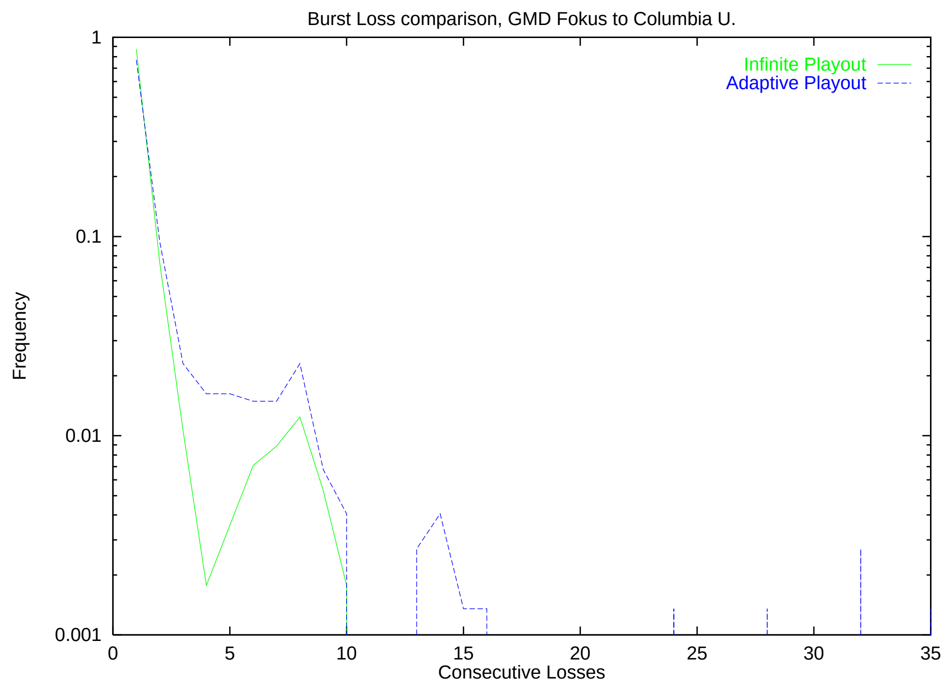


Figure 2.8: Burst loss length distribution for trace 1

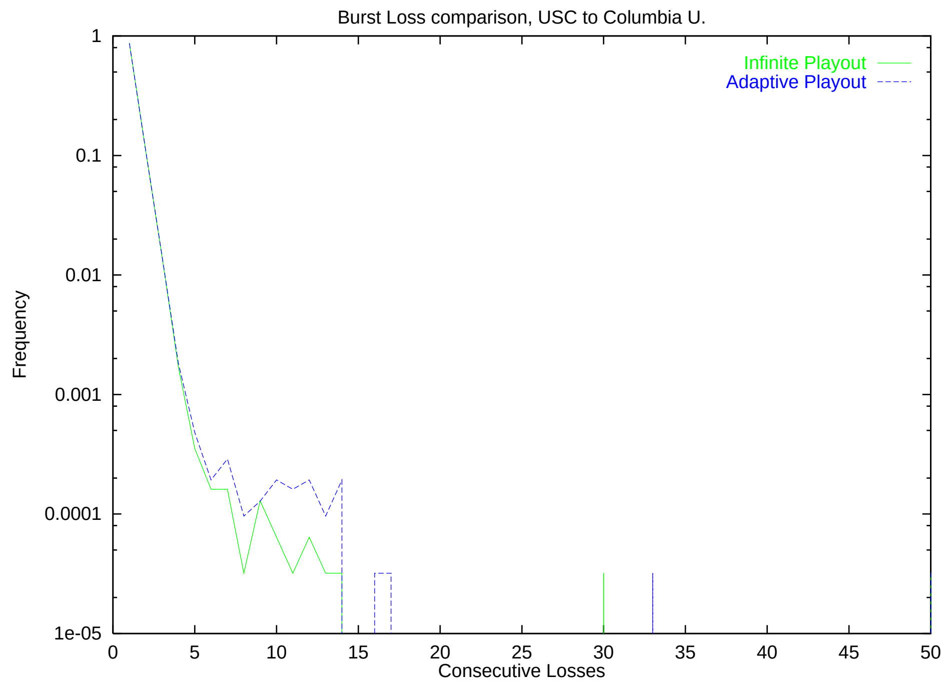


Figure 2.9: Burst loss length distribution for trace 2

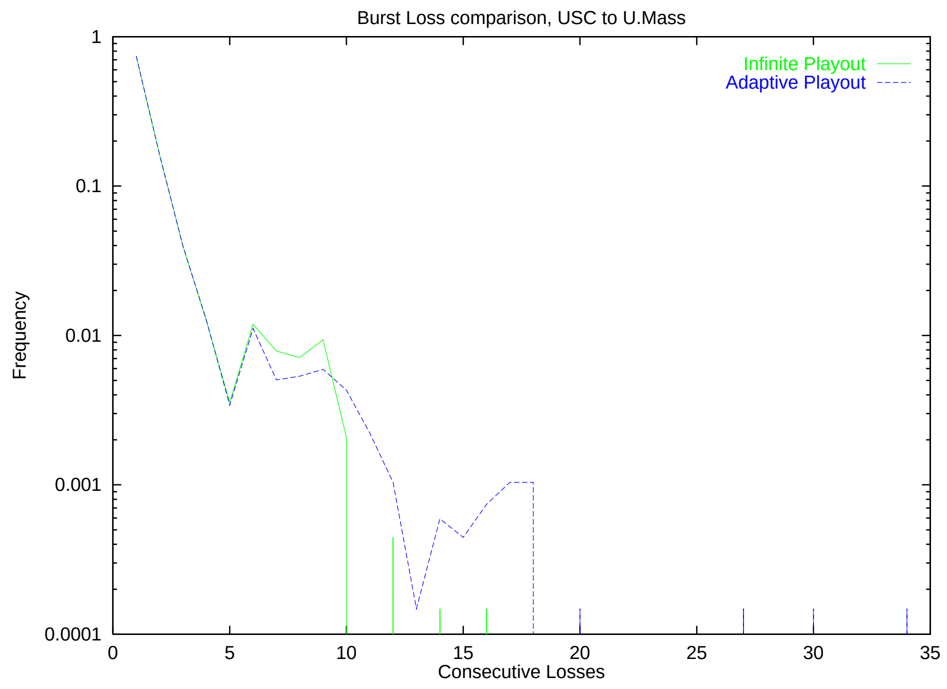


Figure 2.10: Burst loss length distribution for trace 3

very long burst lengths. This means that the playout buffers seldom result in the loss of isolated packets; rather, they cause long bursts of consistently late packets to be lost.

The playout buffer increases both the packet loss rates, and the packet delays. This increase is show in Figures 2.11, 2.12 and 2.13 for traces 1, 2 and 3, respectively. The figures show the cumulative distribution of the increase in packet delay as a result of the playout buffer algorithms. The results show a nice, smooth distribution of delay increase. The mean increase for trace 1 is around 60 ms, for trace 2 around 70 ms, and for trace 3, 300 ms. The large increase in delay for trace 3 explains the smaller effect the playout buffer has on the loss probabilities.

2.2.4 Results for Senders

For traces 4, 5 and 6, only round trip information was obtained. The round trip time (RTT) measurements include losses from both sender to receiver, and receiver back to sender. Paxson's study [4] found loss probabilities to be asymmetric. The result is that it is difficult to glean useful network loss information from the round trip measurements. Since we do not have one way delay

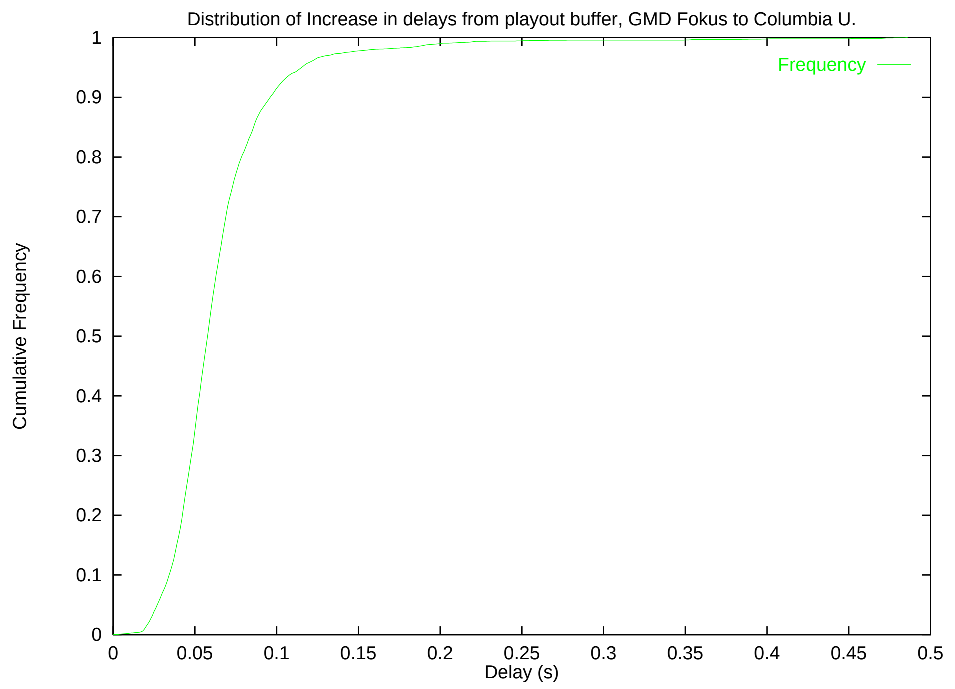


Figure 2.11: Cumulative distribution of delay increase from playout buffers for trace 1

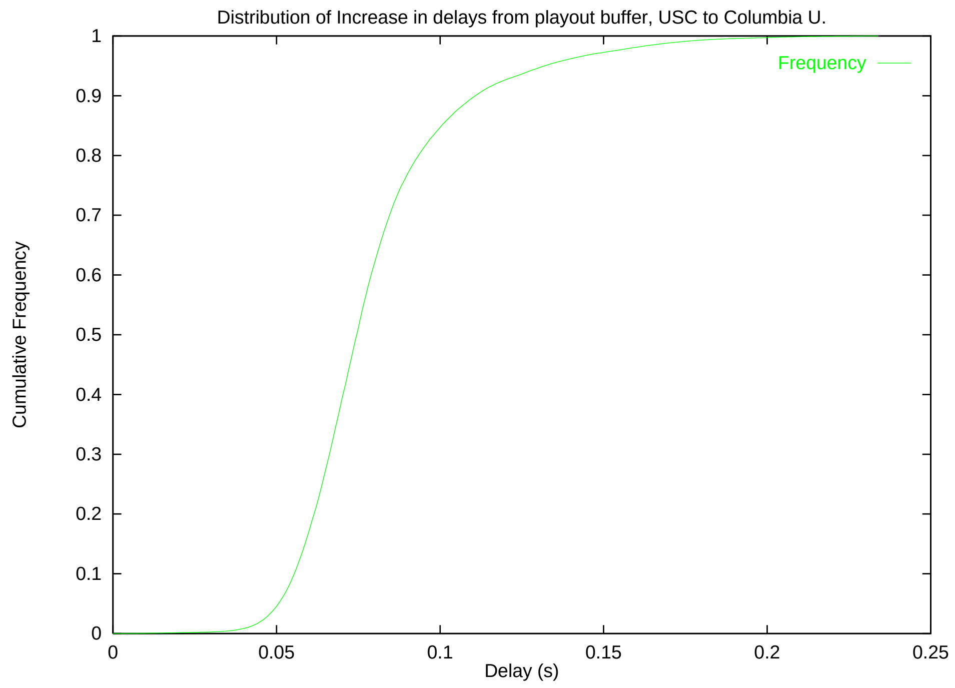


Figure 2.12: Cumulative distribution of delay increase from playout buffers for trace 2

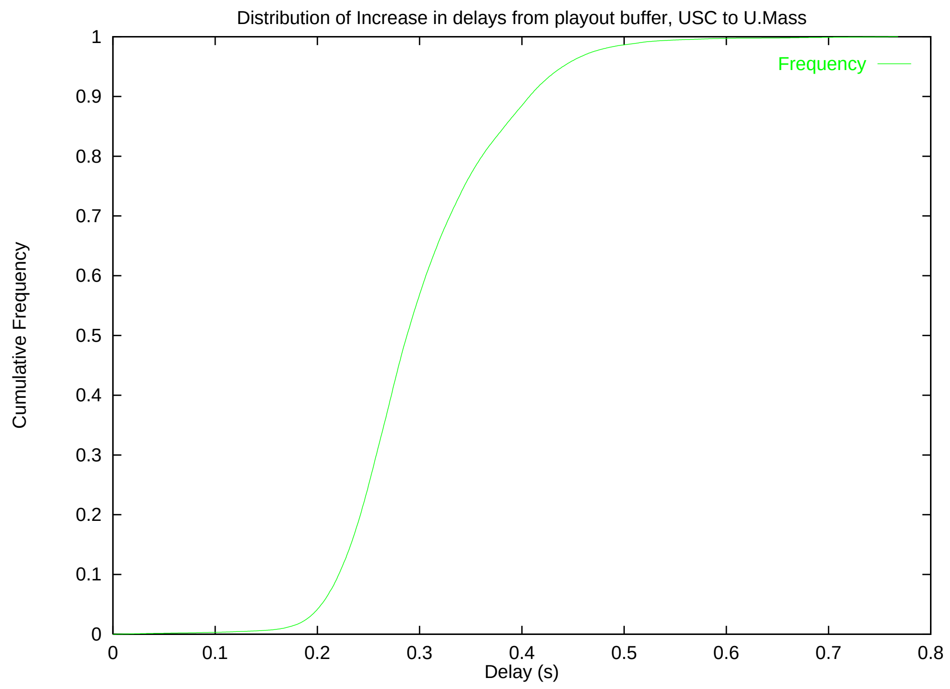


Figure 2.13: Cumulative distribution of delay increase from playout buffers for trace 3

information, the impact of playout buffers on application performance cannot be determined either. However, the round trip information is very useful for examining round trip delay estimates.

Traces 2.14, 2.15 and 2.16 show the cumulative distribution of the round trip time for Columbia U. to U. Mass, Columbia U. to USC, and Columbia U. to GMD Fokus, respectively. The traces within the United States show a fairly smooth distribution over a wide range, with an almost linear increase in the middle of the range of values. This is in sharp contrast to the distribution from Columbia to Germany, which indicates that the RTT's were within a narrow window of values, roughly 120 to 135 ms.

The plots of the mean RTT over time (measured in 1024 non-overlapping windows) show similar trends. These plots are show in Figures 2.17, 2.18 and 2.19 for traces 4, 5 and 6, respectively. The RTT for the traces within the United States shows a fairly random variation. However, the RTT for trace 6 shows distinct regions of behavior. From time 0 to 20,000, the mean RTT stays around 123 ms, and then jumps up slightly to around 127 ms, where it stays until around time 325,000, where it seems to drop once more. During this time, there appear to be occasional bursts of extremely large RTT's. One at time 140,000 shows an average RTT of a little over

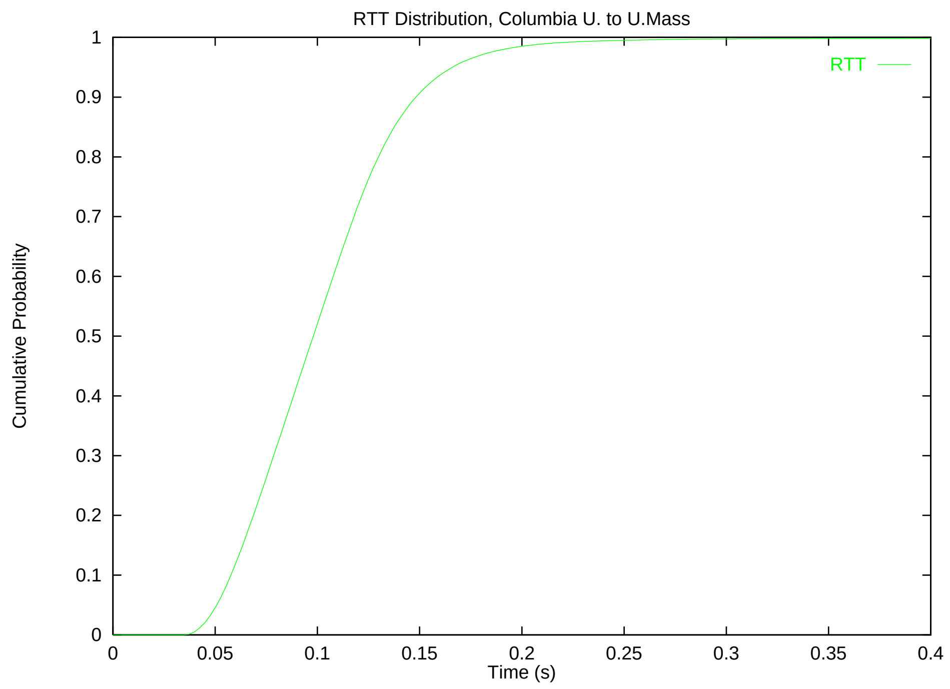


Figure 2.14: RTT delay distribution, Columbia to U.Mass

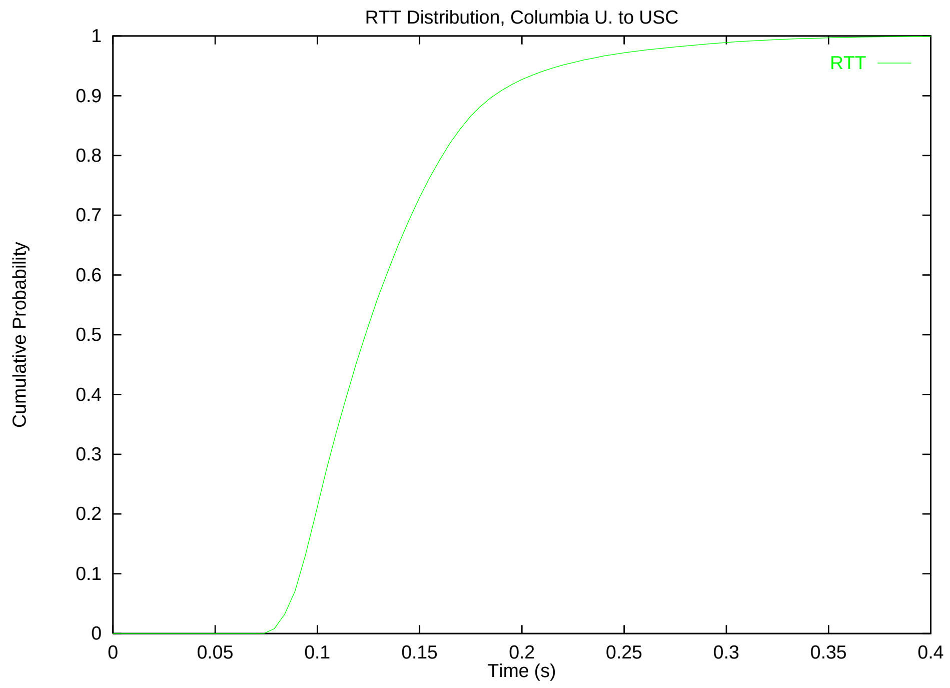


Figure 2.15: RTT delay distribution, Columbia to USC

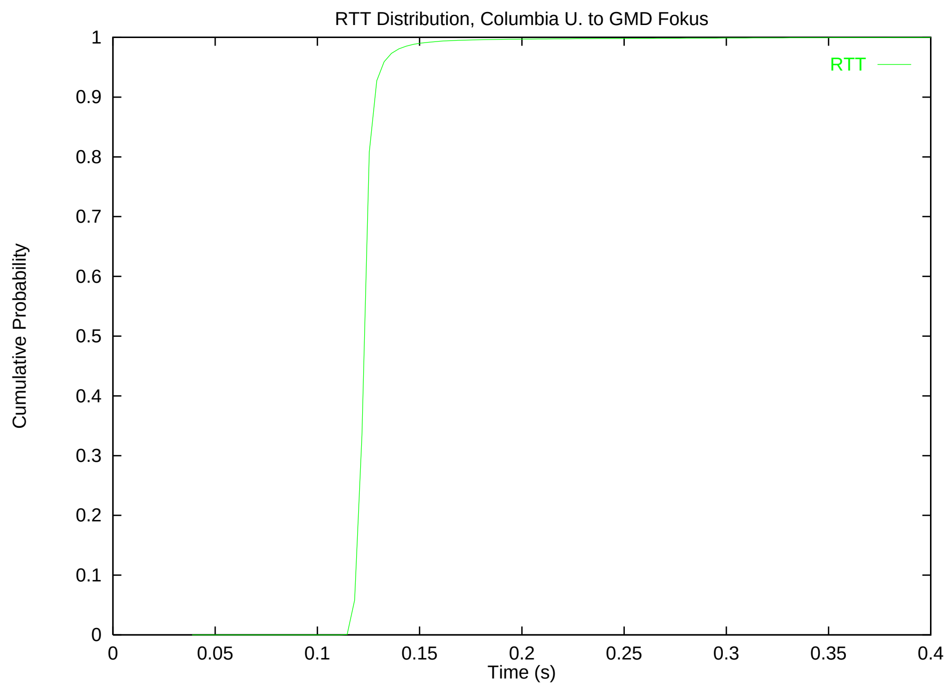


Figure 2.16: RTT delay distribution, Columbia to Germany

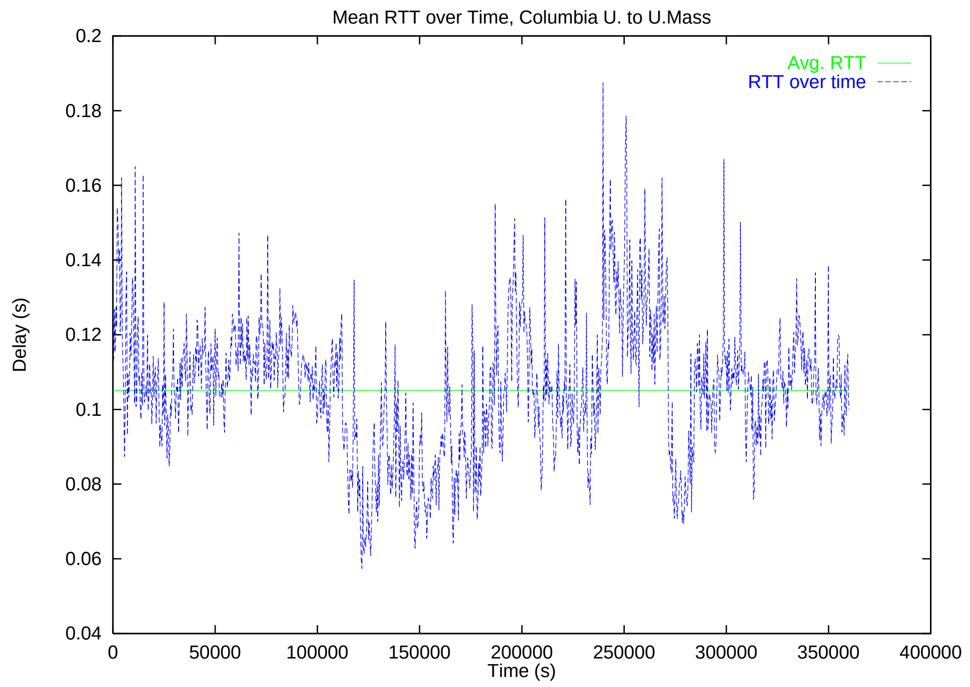


Figure 2.17: Evolution of RTT, Columbia to U. Mass

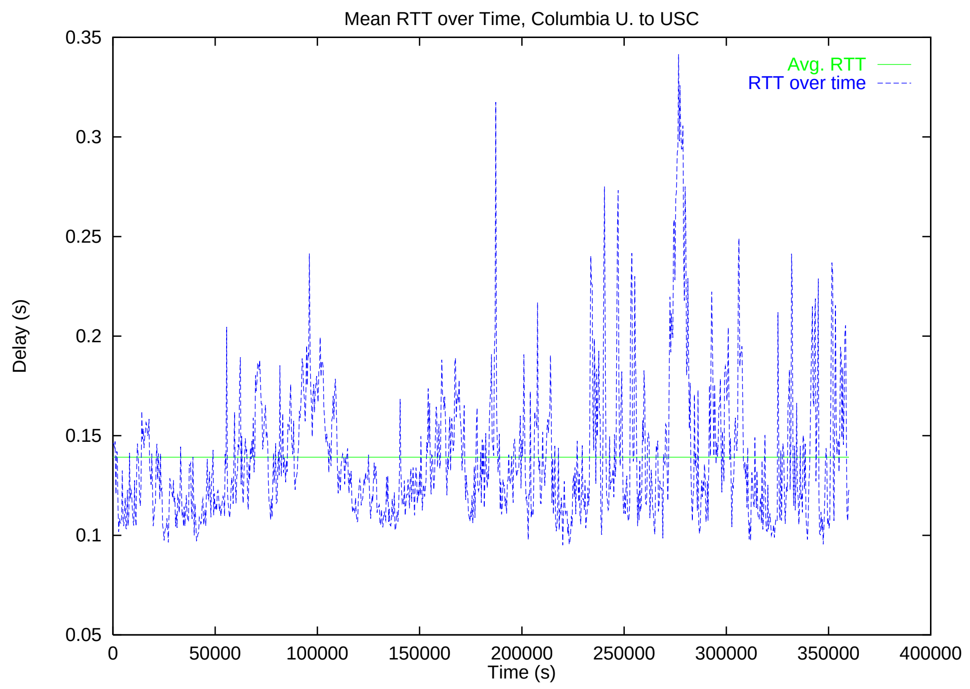


Figure 2.18: Evolution of RTT, Columbia to USC

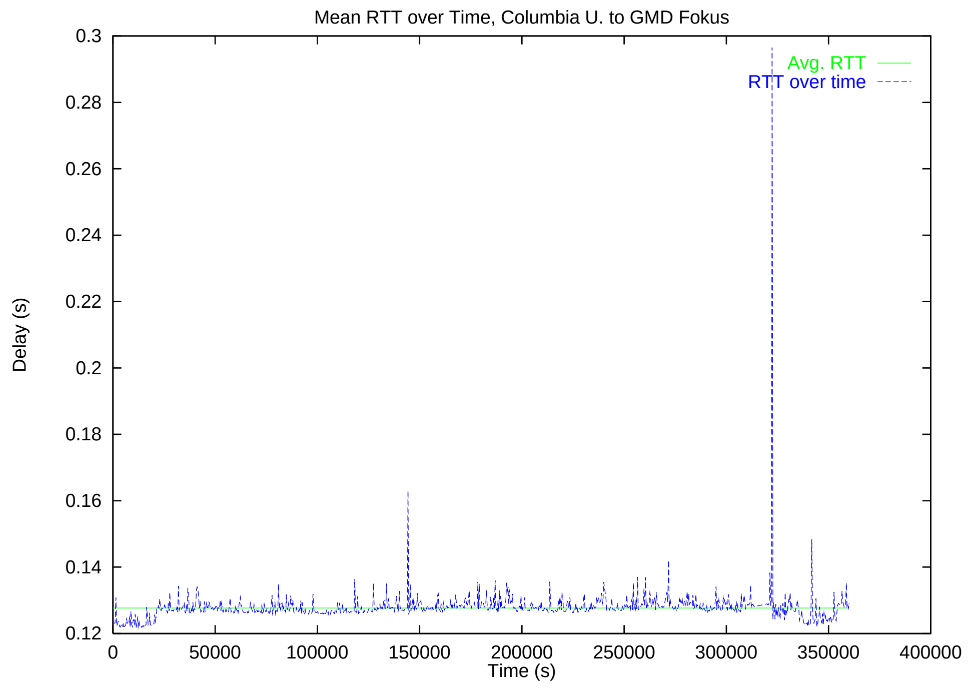


Figure 2.19: Evolution of RTT, Columbia to Germany

160 ms, and another at time 320,000 shows an average RTT of almost 300 ms, almost three times the average RTT for the entire trace!

2.2.5 Conclusions

Care must be taken in drawing conclusions from Internet packet traces; the only certainty is that the network characteristics vary tremendously from hour to hour, day to day, and from link to link. However, our measurements did provide some insight into the kind of behaviors we might see.

The receiver results seem to indicate that correlated losses can, and do, occur regularly. However, the traces consistently indicate that the correlations arise due to large bursts of consecutively lost packets. Furthermore, the impact of the playout buffers seem to be to increase the number and length of these bursts as seen by the application. Unfortunately, it is nearly impossible to compensate for such long bursts of length in a way that meets the real time requirements of Internet telephony. However, handling the large number of uncorrelated losses is possible through FEC techniques, and it is these techniques that we explore in the remainder of this chapter.

2.3 Review of Existing Recovery Mechanisms

A wide variety of tools have been used to compensate for network losses. Excellent surveys can be found by Perkins and Carle [28, 29, 16]. The basic approaches these surveys describe include:

Local Repair: When a packet is lost, the receiver attempts to fill in the missing information through some kind of interpolation. This does not depend on any additional information being sent from the transmitter. Many speech codecs include specifications on how to accomplish this [30, 25]. However, studies have shown that these mechanisms do not work well for unvoiced speech [6], and they still result in de-synchronization of the encoder and decoder for frame-based codecs, as we demonstrate in Section 2.4.1.

Redundant Encodings: In this approach, each packet contains a lower-fidelity encoding of previous packets. This is shown in Figure 2.24. If a packet is lost, the receiver can use the lower fidelity version in a later packet to fill in the missing voice data [31, 32, 33, 34]. This lower fidelity version requires fewer bits to represent than the original, thus reducing bandwidth consumption. The approach is dependent on detailed knowledge of the media being transported. It is this “layer violation” which allows the redundant encodings approach to be efficient. This mechanism is part of a general class of FEC mechanisms which are *media-aware*. Another example of a media-aware mechanism is the transmission of redundant information only during speech segments where local repair will not work well. This is the approach adopted with Speech Property based FEC (SP-FEC), proposed by Sanneck [6].

Packet FEC: In this approach, traditional channel codes (such as parity and Reed-Solomon codes) are applied across packets. The result are additional FEC packets which can be used to exactly recover missing data packets. This approach has been proposed for voice recovery and for reliable multicast [35, 36, 37, 38, 39, 40]. It has the advantage of perfectly recovering the missing data, independent of whether the media was speech, video, or interactive chat. It is for this reason that packet FEC is also known as *media-unaware* FEC.

Interleaving: In this approach, the bursty nature of packet loss is mitigated by spreading the codec data over multiple packets. The technique has been known for a long time [41], and has been recently applied for media on the Internet [42]. Interleaving, while simple to implement, results in significant increases in packetization delay at the sender.

Retransmission: Packet losses can be mitigated by requesting retransmission of missing packets. The latencies this approach introduces often make it prohibitive for real time media, but some studies have shown it feasible when the jitter is substantial compared to the network latency, because of the delay introduced by playout buffers [43, 44]. However, its limited applicability makes it unsuitable as a general solution.

Both the media-aware and media unaware recovery approaches increase delay. Since the redundant information is sent after the original, the receiver must wait for it to arrive. The amount of time to wait depends on many factors, which we consider in detail in Section 2.5.

2.4 Media Aware vs. Media Unaware Recovery

Media aware FEC is advantageous in that it can be very efficient. Since it uses knowledge about the data being protected, it can use few bits for the redundant information. However, it has numerous drawbacks.

The first drawback is computational. To support redundant codecs, the sender must effectively support multiple, parallel speech codecs. The codecs used for the redundant data are typically low bitrate. Unfortunately, as a general rule, the lower the bitrate, the more computationally complex the encoder. The result is that the media-aware mechanisms can be computationally complex. Media-unaware mechanisms are generally simpler; the use of simple XOR-based parity codes requires almost no computation.

Secondly, when lost packets are recovered with the redundant encodings scheme, the reconstructed data is at a lower fidelity. When packet losses are moderate, this means the recovered audio stream will be alternately constructed from a high and low quality codec. This alternation itself can be distracting.

Finally, the redundant encodings approach is able to successfully recover the content of the speech corresponding to a missing packet. However, the quality of reconstructed speech for subsequent frames (even though the packets are received) may be adversely affected. This is because almost all modern speech codecs (including G.729 and G.723, which are widely used for Internet telephony) maintain state in the decoder. The information received from the encoder causes speech to be generated, and also updates the decoder state machine. The encoder maintains the same state machine; this allows it to send data that can be decoded given the current state of the decoder state machine. The redundant encodings mechanism allows the speech content of a lost packet to be recovered, but it does nothing to help update the state of the decoder state machine. As far as the decoder state machine is concerned, the packet has still been lost. The resulting de-synchronization of the encoder and decoder will cause speech quality for subsequent frames to worsen. The state machines will eventually reconverge; these codecs are generally engineered with finite memories, so that old data becomes irrelevant for correctly reconstructing the current frame. Additionally, many of these codecs have built-in mechanisms for estimating the missing input when a packet is lost. This helps mitigate the effect on the decoder state machine, but does not prevent it. This is in contrast to media-unaware FEC. Since the contents of the lost packet are reconstructed verbatim, they can still be used to drive the decoder state machine. As a result, there is no loss of synchronization, and subsequent speech is reconstructed correctly.

This last point, however, is rather subjective. The difference in recovery performance between media aware and media unaware FEC will depend on a number of factors. In particular, it depends on the amount of time the decoder requires to resynchronize, and on the impact of the loss of synchronization on the reconstructed speech quality. These factors, in turn, depend on the type of loss (isolated vs. bursty), the specific speech codec, and the content of the speech itself. It is our aim in this section to quantify the difference between media-aware and media-unaware recovery.

2.4.1 Resynchronization Time

The first step in quantifying the difference in performance is to consider the amount of time required for the encoder and decoder state machines to resynchronize. If this time is negligible, then the synchronization effect is not significant.

Determining the resynchronization time is nontrivial. A number of possible definitions exist:

- Compute the difference between the decoded speech with no frame erasures, and the decoded speech with erasures. After an erasure occurs, compute the amount of time until the energy in the difference falls below some threshold. The threshold can either be static or adaptive according to the strength of the error signal. This approach is the simplest, requiring no interaction with the encoder or decoder except through the decoder output. Since Mean Square Error (MSE) is a poor estimate of speech quality except when it is very small or very large, it likely defines an upper bound on the real convergence time, whatever that may be.

- Use the same approach as above, but instead of a plain difference signal, use a perceptually weighted error signal. G.729 defines a perceptual weighting filter which is derived from the LP synthesis filter, and this can be used directly. This approach requires interaction with the encoder (or decoder) to obtain the LP filter parameters. However, it yields a realistic measure of the human perception of the convergence time.

- Instead of computing the error between the decoded speech, the decoder state can be consider as a multidimensional vector, and a distance measure can be defined (perhaps perceptually) to compute the difference between the encoder and decoder state over time. This approach has the advantage of actually looking at the state, and not the output of that state (i.e., the decoded speech). However, it requires interaction between encoder and decoder, and it necessitates the definition of a complex distance metric.
- Use either the first or second option, but with the postfilter turned off. Since the postfilter does not represent decoder state (at least, state that must be synchronized with the encoder), it may get in the way of measurements based on the speech itself.

The approach used here is the first one above. It is the simplest, and it gives an upper bound on the real convergence time. We used an adaptive threshold to determine convergence. A simple, one-pass algorithm was used to compute the convergence time. The MSE in each frame (the MSE is measured between the decoded version of the unerrored speech bitstream and the decoded version of the errored bitstream. This eliminates coding loss from the computation.) is computed. The starting time of the convergence period is defined as the first good frame received after a burst of erased frames. As subsequent frames are received, the maximum MSE so far is noted, and the threshold is set at 1% of this quantity. The first frame with an MSE below this threshold is considered the last frame of the convergence period. The difference in frame numbers represents the convergence time, measured in units of frames.

The specifics of the experiment are as follows. The process starts out with an original speech segment. The G.729 encoder is then run on the speech to generate the bitstream, with no errors. A program then erases n consecutive frames from this bitstream, waits m frames, and then erases n more. The process continues through the entire bitstream. In all our experiments, m was set to a large number (50), corresponding to 500 ms, to avoid interactions between bursts. The errored bitstream and the original, uncorrupted bitstream are then decoded. The result are two speech segments. The MSE signal between the two is then computed, and the convergence estimation technique above is applied.

The experiments were conducted using some of the speech files in the ITU corpus. In particular, the speech segments f15.d, f17.d, f26.d, f34.d, f43.d, f44.d, m19.d, m27.d, m31.d, m33.d, m41.d, and m51.d were used. The files beginning with f are female speakers, and the ones beginning with m are male speakers. Each speech segment is approximately 2 to 8 seconds in duration, with varying content and background noise. Each of these files contains 16 bit signed linear Pulse Code Modulated (PCM) speech samples at 8 kHz.

Figure 2.20 shows the cumulative distribution of convergence times across all speech segments. As the figure shows, this distribution does not appear to depend strongly on the burst length of the losses.

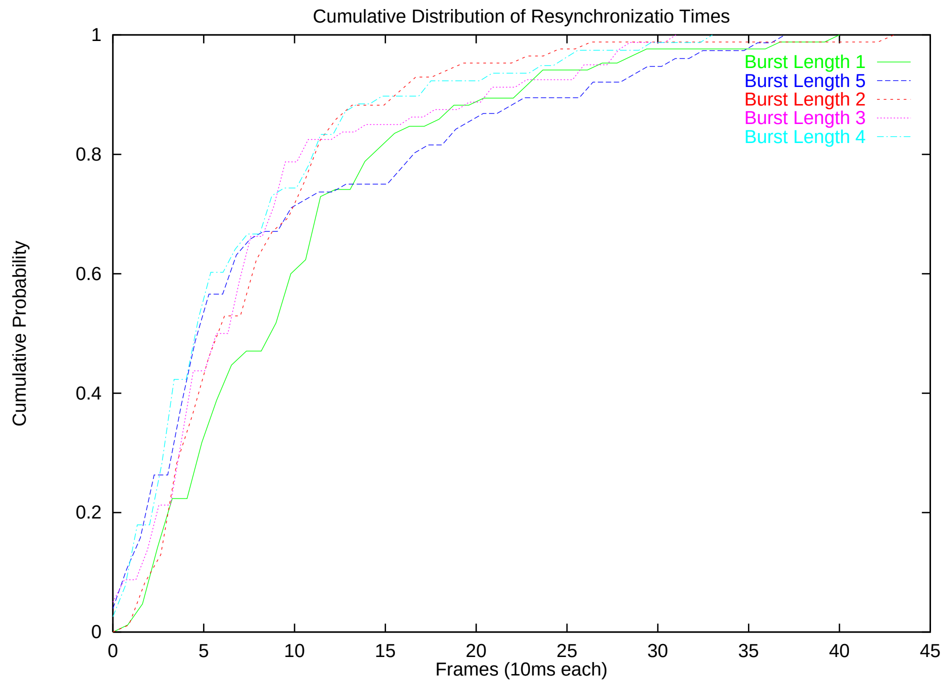


Figure 2.20: Cumulative distribution of resynchronization times

Table 2.3 summarizes this figure, showing the mean and standard deviation of convergence times.

The substantial variation in the convergence times attests to the fact that they are extremely speech dependent. Given that frame sizes are 10 ms, the average convergence time is between 70 ms and 100 ms, but they can be as large as 200 to 300 ms. This is not an insignificant

Burst Size	Mean Resync. Time	Std. Dev. of Resync. Time
1	10.26	7.92
2	8.21	6.51
3	8.22	7.39
4	7.27	6.93
5	9.25	9.39

Table 2.3: Resynchronization Time vs. Burst Length

duration, and therefore the resynchronization effect may be significant in comparing the performance of media-aware and media-unaware FEC. Recent results by Sanneck have confirmed our findings on the typical durations for the codec resynchronization [45].

2.4.2 Magnitude of Error

The previous subsection examined the *duration* of the de-synchronization. Equally important in gauging its impact on quality is the *magnitude* of the effect. In this section, we consider, using both objective and subjective metrics, the degradation of speech quality due to the loss of a burst of n frames.

2.4.2.1 Objective Measurements

In this study, the speech quality is measured using the average MSE over the convergence period, as defined above. It is well known that the MSE is *not* a true measure of speech quality. However, it can serve as a useful bound, and coupled with the subjective measurements below, can help give a reasonable estimate of speech quality.

Figure 2.21 plots the cumulative distribution of the MSE in each frame, for burst lengths of 1 to 5. Note that in this case, there is a noticeable increase in the error energies with increasing burst sizes.

These observations are verified by the computation of the average and standard deviation of the energy in the entire error signal for the five different burst sizes. The results are depicted in Table 2.4.

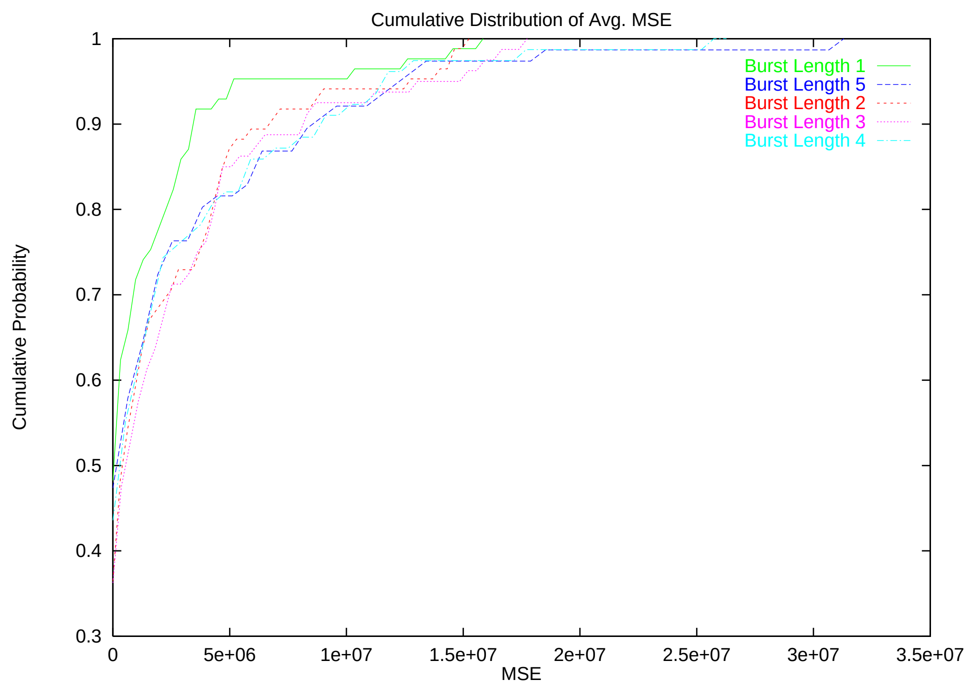


Figure 2.21: Cumulative distribution of avg. MSE

Burst Size	Avg. MSE	Std. Dev. MSE
1	$1.4 \cdot 10^6$	$2.9 \cdot 10^6$
2	$2.3 \cdot 10^6$	$3.6 \cdot 10^6$
3	$2.6 \cdot 10^6$	$4.1 \cdot 10^6$
4	$2.6 \cdot 10^6$	$4.5 \cdot 10^6$
5	$2.7 \cdot 10^6$	$5.0 \cdot 10^6$

Table 2.4: MSE vs. Burst Length

The results indicate a sharp jump in the error energy when the burst size increases from 1 to 2. Subsequent increases in the burst sizes cause further increases in the MSE, but by less severe amounts. The variability in the error also appears to increase with increasing burst size. The conclusion would appear to be that the codecs concealment algorithm can do a good job at recovery for isolated packet losses, but does worse as the size of the burst increases.

Burst Size	1	2	3	4	5
Mean MOS	4.208333	3.275	2.425	2.0833	1.7
S.D. in MOS	0.660124	0.916629	1.198697	1.426437	1.585875

Table 2.5: Subjective Evaluation of Speech Quality

2.4.2.2 Subjective Measurements

The numerical results of the previous section were backed up by subjective testing. To perform the test, 12 non-experts listened to a set of speech samples. The speech samples consisted of six versions of the same speech: the decoded speech with no errors in the bitstream, and the decoded speech with instances of 1, 2, 3, 4, and 5 consecutive frame erasures, spaced 500 ms apart. Using the error-free decoded speech as an ideal point, the subjects were asked to rate the five other samples on a scale of 1 to 5, with 5 being as good as the error free decoded signal, and 1 being much worse. This scale is known as the Mean Opinion Score (MOS). The test was run using the sequence f15.d, which is a female voice speaking a sentence roughly seven seconds in duration. The subject evaluated each pair of speech signals (unerrored and errored) independently. Each subject wore headphones to eliminate room noise. The pairs were evaluated in random order. The results of the analysis are depicted in Table 2.5.

The table shows trends similar to those in the objective measures. With a single frame erasure, the speech quality is slightly worse, but still very good. However, there is a sharp drop in the speech quality as soon as two consecutive frame erasures occur. Subsequent increases in the burst size reduce the speech quality more, but by decreasing amounts. Of course, there are two phenomenon taking effect here. The first is quality degradation due to increased resynchronization times of encoder and decoder. The second is reduced speech quality due to the loss of an increasing number of consecutive packets.

The conclusion from the objective and subjective measurements is that the speech quality in the absence of FEC of any sort (just using the G.729 concealment algorithm) is acceptable for recovery from a single loss. However, the quality worsens with multiple losses. Our measurements above have shown that multiple consecutive losses are quite common; this would imply that FEC is useful in many cases.

2.4.3 Contribution of Resynchronization

The measurements of the previous section reveal that speech quality is adversely affected by loss, particularly with two of more consecutively lost packets. However, the results say nothing about whether this quality degradation is due to poor performance of the G.729 loss concealment algorithm applied for the missing frames, or whether it is due to incorrect speech generated for subsequent frames due to the state resynchronization problem. In this section, we perform an experiment to determine which is the case.

To make such a determination, two speech signals were generated. The first one is the output of the decoder with no frame erasures. The second is the output of the decoder with instances of consecutively erased frames. Let's say frames N and $N + 1$ only were erased. Two new speech signals, called *mix1* and *mix2*, are then generated. The signal mix1 is formed by replacing frames N and $N + 1$ in the error-free decoder output with frames N and $N + 1$ from the errored decoder output. The signal mix2 is formed by replacing frames N and $N + 1$ in the errored output signal with the correct N^{th} and $N+1^{th}$ frames from the error-free decoder output. In this fashion, mix1 emulates the decoder operation during the erased frames, but as soon as the next good frame arrives, its output is perfect again, as it would be if its state were restored. On the other hand, mix2 has correct speech output during the erased frames, as if the speech were replaced by the correct version. However, the state is not restored, since the next good frame will be generated from state updated from the concealment algorithm. The nature of these two speech signals is depicted graphically in Figure 2.22.

Mix1 is representative of a system that allows for recovery of decoder state, at the expense of perfectly recovering the lost speech frames. Mix2 is representative of a system that allows for recovery of the lost speech frames, at the expense of losing decoder and encoder state synchronization. Media-unaware FEC is similar to the system represented by mix1, except that it will typically *also* recover the lost speech frames. Media-aware FEC is similar to the system represented by mix2, except it usually does not recover the lost speech frames perfectly, since a lower fidelity codec is used.

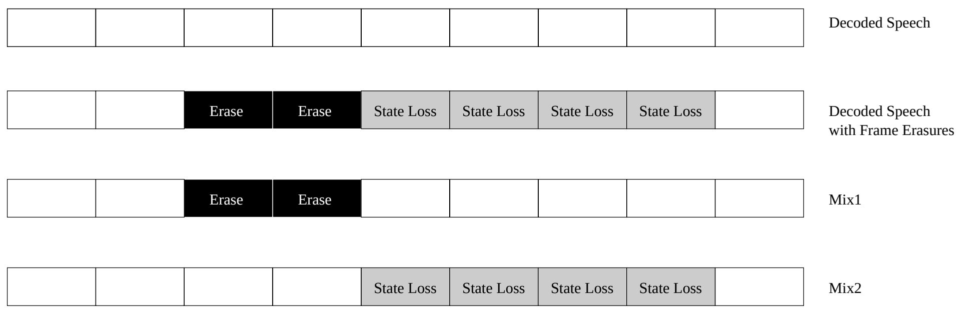


Figure 2.22: Mix1 and mix2

Burst Size	Mix1		Mix2	
	Avg. MSE	SD of MSE	Avg. MSE	SD of MSE
1	$1.8 \cdot 10^6$	$4.5 \cdot 10^6$	$1.1 \cdot 10^6$	$2.4 \cdot 10^6$
2	$2.8 \cdot 10^6$	$4.9 \cdot 10^6$	$1.6 \cdot 10^6$	$2.9 \cdot 10^6$
3	$3.0 \cdot 10^6$	$4.8 \cdot 10^6$	$1.6 \cdot 10^6$	$3.1 \cdot 10^6$
4	$3.5 \cdot 10^6$	$6.3 \cdot 10^6$	$1.2 \cdot 10^6$	$2.2 \cdot 10^6$
5	$3.5 \cdot 10^6$	$7.2 \cdot 10^6$	$1.0 \cdot 10^5$	$1.6 \cdot 10^6$

Table 2.6: Avg. and SD of MSE for mix1 and mix2

By comparing the speech quality of mix1 and mix2, we can gauge the impact of state resynchronization on speech quality.

2.4.3.1 Objective Comparison

We first perform an objective comparison of the two speech signals. We do this using the average MSE over the duration of the convergence time. For mix1, this duration is exactly equal to the number of consecutive frames lost. For mix2, it is the amount of time required to resynchronize, minus the duration of the loss itself.

The results are shown in tabular form in Table 2.6. Mix1 has a larger energy in its error signal, although the duration of the error is much reduced. Mix2 has less energy in its error signal, but the error persists for longer.

To determine which is actually better, we performed some subjective tests.

2.4.3.2 Subjective Tests

In the subjective evaluation, each of the 12 subjects listened to a number of pairs of speech signals. Each pair was the same content, but one was the mix1 version (state restoration), and the other was the mix2 version (speech restoration). Thirteen pairs of speech signals were presented; 1 pair (f15.d) contained instances of single frame erasures. The remaining twelve consisted of the speech signals f15.d, f26.d, and m29.d, with 2, 3, 4 and 5 frame erasures each, with 500 ms between each episode of erasures. For each pair, the subject selected which one sounded better, or chose neither if they sounded the same.

The results are depicted in Figure 2.23. For each speech sample used, the number of subjects who preferred mix1, mix2, or neither is indicated. The area below the mix1 line indicates the number of subjects preferring mix1. The area between the mix2 line and the mix1 line indicates the number of subjects preferring mix2. The area between the top of the chart and the mix2 line indicates the number of subjects with no preference. As the results show, mix1 was preferred more than mix2 for all but two of the twelve speech samples. For the f15.d speech segment (the longest of the three segments), the subjects almost unanimously chose mix1.

The conclusion is that the resynchronization effect is substantial. Users generally preferred the scheme which restored the state of the encoder and decoder, at the expense of reduced speech quality during packet losses. This implies that media-unaware FEC is preferable for these low rate codecs over media-aware FEC.

2.5 Integrating FEC with Playout Buffers

The results of the previous sections have demonstrated the importance of playout buffers on application performance, and also have demonstrated the need for media-unaware FEC to combat loss. In this section, we consider the interplay between adaptive playout buffer algorithms and FEC.

The study of FEC for loss recovery, and playout buffer adaptation for jitter compensation, have proceeded independently. However, we have observed that there is a coupling between the two. All of the FEC mechanisms send some redundant information which is based on previously

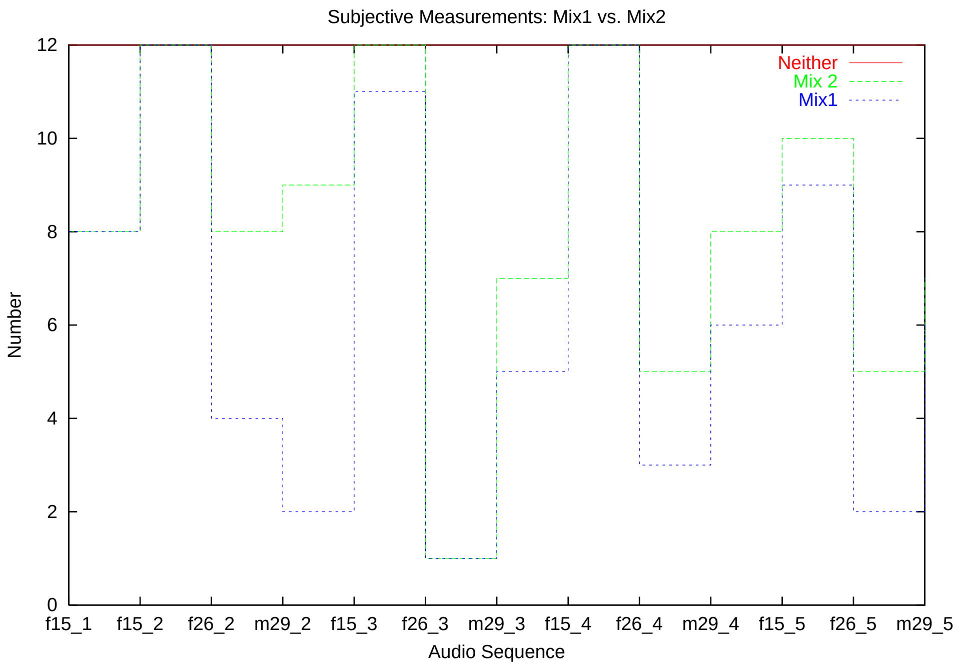


Figure 2.23: Preferences for mix1, mix2, or neither

transmitted packets. Waiting for the redundant information results in a delay penalty, and consequently an increase in size of the playout buffers. When network loss rates are high, accepting the delay penalty for increased recovery capabilities is appropriate. However, when network loss rates are low, the FEC may not provide useful information, and increasing the playout buffer sizing to wait for it is not appropriate. The result is that playout buffer adaptation should depend on *both* FEC and network loss conditions *and* network jitter. However, existing tools that utilize FEC (such as *rat* from UCL and *freephone* from INRIA) use *decoupled* adaptation algorithms. These algorithms compute a playout delay, using traditional algorithms, as if FEC were absent, take the fixed delay required to reconstruct missing packets at the decoder using FEC, and then combine these two delays together. These decoupled algorithms may insert insufficient delay when network loss probabilities are substantial, and too much delay when they are not, resulting in poor performance [46]. We further observe that existing playout buffer adaptation algorithms generally aim to minimize the network losses at the expense of delay. However, speech quality can still be good in the face of moderate packet loss. Based on the subjective results in Section 2.4.2.2, MOS scores were still excellent for the case of isolated packet losses. In that sample, the loss rate was 2%. As such, it would be beneficial to be able to tune these algorithms to achieve a target loss (possibly non-zero) to meet the operating range of the codecs, allowing for a reduction in delays.

In this section, we explore these problems in more detail. In Section 2.5.1 we demonstrate, through simple analysis, the need for coupling packet loss and jitter into playout buffer adaptation. In Section 2.5.2 we briefly review some existing playout buffer adaptation algorithms. In Section 2.5.3, we present several new algorithms which achieve the desired coupling and tunability objectives. Section 2.5.4 shows the improvements offered by our algorithms through simulations. Section 2.7 summarizes our work and discusses future directions.

2.5.1 The Coupling Effect

The main role of playout buffer adaptation algorithms is to trade loss for delay. These algorithms choose a playout delay D for each talkspurt, where D is defined as the difference between playout time and generation time for all packets in a talkspurt (although some algorithms adjust the delay mid-talkspurt to compensate for errors in buffer sizing). Increasing D results in a larger end-to-end delay, but decreases the fraction of late packets. The algorithms work by choosing a target “optimal” operating point of loss and delay, and adjusting the playout delay to come as close to the optimal point as possible. If an algorithm chooses some delay D for a talkspurt, the probability receiving it on time and playing it out to the application (which we denote the playout probability p_R) is:

$$p_R = (1 - p)P[n_i < D] \tag{2.1}$$

where p is the loss probability for a packet, and n_i is the delay of the packet (measured as the difference between arrival time and generation time), given the packet arrives. We use the expression $P[X]$ to denote the probability of event X. This expression assumes that the network losses and delays are independent random variables. Of course, network losses are not independent. However, as we noted in Section 2.2.5, we observed that packet losses are frequently independent, with occasional bursts of highly correlated loss. Handling the latter is extremely difficult, so we focus on dealing with the uncorrelated losses using FEC. As such, it is sufficient for our analysis to focus on models that apply to the phenomena we are trying to correct - independent packet loss.

The dependency of the playout probability p_R on the network loss probabilities and the network delay distribution is a simple multiplicative one. The result is that many playout adaptation strategies can completely ignore the loss rate p. In particular, any algorithm which tries to choose the minimum D which achieves the highest possible p_R need not consider p at all. These algorithms need only find the smallest D for which $P[n_i < D] = 1$. All of the algorithms described by Ramjee and Moon [26, 47] fall into this category.

The dependency of p_R on the delay distribution and loss probabilities becomes more complex when FEC is introduced. The result is that the choice of D is more strongly influenced by p. In the subsections which follow, we derive expressions for the relationship of p_R to the delay and loss, and show how the more complex interrelationships increase the importance of considering loss in adaptation algorithms.

2.5.1.1 Redundant Codecs

Consider the FEC mechanisms described by Bolot and Garcia [32]. These algorithms use multiple low-bitrate versions of a packet, each version being piggybacked on a subsequent packet. A packet is played out so long as a receiver gets the original or any one of the $k - 1$ redundant versions of the packet on time. If packets are lost independently with probability p, the probability of playing out a packet (p_R), given that the receiver is willing to wait for $l \leq k$ of the packets, is $1 - p^l$.

This says nothing, however, about how long a receiver must wait for these l packets. Computing the probability under a delay constraint will require us to factor in the probability that the l packets arrive in time. Assume that packets are generated at the sender at intervals of Δ. A playout delay of D at the receiver means that a packet must be present D seconds after its generation at the sender in order to be used. A packet is played out if the original packet arrives on time, or if a redundant packet arrives in time. To be in time, the redundant packet's network delay, plus the amount of time between its generation and the original packet generation, must be less than the playout delay. More formally, if we assume network delays are independent from packet to packet, the probability of playing out a packet when the playout delay is D is:

$$p_R = 1 - \prod_{j=0}^{k-1}(1 - (1-p)P[n_i + j\Delta < D]) \tag{2.2}$$

Note the more complex dependency of p_R on the network loss probability p and the network delay distribution n_i. The result is that the choice of D is more complex, and will most likely depend strongly on p. Consider once more the basic playout adaptation target: to choose the minimum D which results in the maximum possible p_R. When p is zero, D need only be set to the largest network delay possible. However, as p increases, the value of D rapidly increases up to a maximum value of $(k-1)\Delta$ plus the largest delay possible. Note that p now plays a crucial role in choosing D once FEC is enabled. A decoupled algorithm might choose D to always be $(k-1)\Delta + \max n_i$, which results in overly large playout delays when network losses are low. A different decoupled algorithm might choose D to always be some quantile of n_i. This algorithm will result in overly small playout delays when network losses are high. As such, coupling loss and delay into buffer adaptation is critical for proper loss and delay tradeoffs over a wide range of network conditions.

2.5.1.2 Reed-Solomon FEC

The dependency of playout probability on the packet loss probabilities and packet delay distributions is even more complex for Reed Solomon coding. A Reed Solomon code causes an extra $n-k$ packets to be sent for every k data packets. So long as any k of the total n packets transmitted are received, the original k data packets can be recovered [39]. A code which sends n packets for every k data packets is referred to as an (n, k) code. We assume the $n-k$ FEC packets are sent piggybacked on subsequent data packets, as shown in Figure 2.24. The first 3 data packets are protected by 2 FEC packets. Rather then being sent separately, the contents of the 2 FEC packets are attached to the end of the next two data packets. Recent studies by Bolot [34] have shown that spacing FEC in this fashion results in better performance than sending the FEC packets immediately after the final data packet.

When a packet is lost, the receiver must wait until it obtains a total of k of the n packets in the block before recovery can take place. The amount of time needed depends on which of the

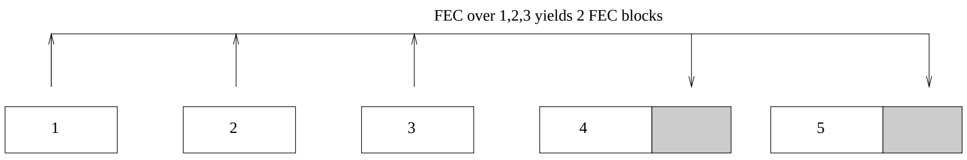


Figure 2.24: Piggybacking FEC packets for a $(5, 3)$ Reed Solomon code

data packets is to be recovered. If the last data packet (the k^{th}) was lost and is to be recovered, and the other $k - 1$ data packets were received, the receiver need only wait for one additional FEC packet beyond the last data packet.

However, if the first packet among the k data packets is to be recovered, the receiver must wait for at least $k - 1$ additional packets to arrive, and possibly more, depending on the number of those that are lost. The probability of playing out a packet is therefore a weighted average over all k data packets in the block.

$$p_R = \frac{1}{k} \sum_{i=1}^{k} p_R^i \tag{2.3}$$

$$p_R = \frac{1}{k} \sum_{i=1}^{k} \sum_{\vec{S_i}, |\vec{S_i}| \geq k \vee \vec{S_i}[i]=1} P[\vec{S_i}] \tag{2.4}$$

$\vec{S_i}$ is a binary vector with n components, representing an arrival pattern. The j^{th} component is 1 if the j^{th} packet in the block of n is received before the playout of the i^{th} packet, 0 otherwise. The sum is over those arrival patterns where the i^{th} packet itself arrives in time, or where at least k packets arrive in time.

Computing $P[\vec{S_i}]$ is fairly straightforward. Assuming independent packet delays and losses, it is the product of the timely arrival probability (or one minus it) across each component:

$$P[\vec{S_i}] = \prod_{j=0}^{n} F(i, j, \vec{S_i(j)})$$

where:

$$F(i, j, z) = \begin{cases} (1-p)P[n_j + (i-j)\Delta < D] & z = 1 \\ 1 - (1-p)P[n_j + (i-j)\Delta < D] & z = 0 \end{cases} \tag{2.5}$$

Here, too, there exists a more complex dependency of packet loss and delay on the playout probability. As in Eq. 2.2, when $p = 0$, D need only be set to the largest delay experienced by any packet. As p increases, this delay rapidly increases to its maximum of $(k-1)\Delta + max_j(n_j)$.

2.5.1.3 Conditions for Dependency

The complex dependencies of Eqs. 2.2 and 2.5 actually reduce to simpler ones under certain network conditions. In particular, the dependencies are simplified when the network jitter is significantly higher than the delay introduced by the FEC mechanism. For the redundant codecs and Reed-Solomon FEC, this delay is $(n-1)\Delta$.

This claim is very intuitive. The amount of delay in the playout buffer is on the same order of magnitude as the network jitter. If this delay is very large, there will always be enough delay in the buffer to make use of the FEC, whether its needed or not. As a result, the FEC does not need to be factored into the playout buffer sizing, and traditional adaptive playout buffer algorithms will perform adequately. The claim can also be justified analytically. Consider Eq. 2.5, and in particular the term $P(n_j + (i-j)\Delta < D)$. The algorithms described by Ramjee et al. [26] choose a playout delay D which is roughly the average packet delay d_i plus some multiple of the standard deviation of the delay v_i, $D = \hat{d}_i + \mu\hat{v}_i$. Thus:

$$\begin{aligned} P[n_i + (i-j)\Delta < D] &= P[n_i + (i-j)\Delta < \hat{d}_i + \mu\hat{v}_i] \\ &= P[n_i < \hat{d}_i + \mu\hat{v}_i + (i-j)\Delta] \end{aligned}$$

Since $|i-j| < n$, if $n\Delta \ll \hat{v}_i$ (FEC delay less than the jitter), the third term in the sum is insignificant compared to the second, so:

$$P[n_i + (i-j)\Delta < D] \approx P[n_i < \hat{d}_i + \mu\hat{v}_i]$$

Assuming stationarity of delays, this eliminates the dependency of $F(i,j,z)$ on both i and j. Consider once more the common desired behavior: choosing the minimal D which results in the largest p_R. Now, since the i and j terms are no longer significant in $F(i,j,z)$, the choice is independent of p, as it is in the absence of FEC.

We note, however, that the delay measurements described above indicate that this will often not be the case. In our trace of the delays from Columbia to U. Mass, we found the RTT to vary from 50 ms to 200 ms. If one way delays are roughly half this, the delays vary from 25 ms to 100 ms. Typical packetization delays Δ are 20 to 60 ms. With a typical block size n of 3 to 6, $(n-1)\Delta$ is between 40 to 600 ms, which is larger than the network jitter. The implication is that playout buffer algorithms will need to take the packet loss probabilities into consideration in order to operate well.

2.5.1.4 A Note on Applicability

The astute reader might observe that if loss rates are low, the sender should not be sending FEC in the first place. Such sender adaptation might eliminate the need for coupling FEC to playout buffer adaptation at receivers. First off, this is possible only for predictable loss rates (that is, the loss rate at time T can be correctly predicted from reported loss rates at time T-RTT). This is often not the case. In addition, the coupling is very important for multicast. With multicast, it is likely that some receivers will experience more loss than others. The sender may choose to include FEC in order to improve reception quality for lossy receivers. In this case, receivers experiencing low loss will want to adapt in order to reduce their playout delays, since the FEC is not needed. Even in the unicast case, receiver adaptation is useful when the sender is incapable or unwilling of performing adaptation. Receiver adaptation is also useful when feedback is not sent frequently.

2.5.2 Existing Playout Buffer Algorithms

We briefly review the existing algorithms for playout buffer adaptation that have been proposed in the literature.

The problem of playout buffer adaptation was first identified by Cohen [1] in 1977. Barberis and Pazzaglia [48] propose delaying the first packet in a talkspurt by a fixed delay T (known as the Null Timing Information algorithm, NTI), or delaying the first packet by a fixed delay T minus the transit time of the first packet (the Complete Timing Information algorithm, CTI), the latter requiring synchronized time at sender and receiver. They derive results for the expected end-to-end delay of the system based on these algorithms. However, they provide no means for computing T when the network delay distributions are unknown. Gopal, Wong, and Majithia present similar algorithms [49] with different assumptions in the analysis of the loss and delay. Suda, Miyahara, and Hasegawa [50] propose some hybrid algorithms based on the work of Barberis [48]. Montgomery [51] proposes mechanisms to accurately compute the delay of the first packet, needed for the NTI algorithm. Alvares-Cuevas, Bertran, Oller, and Selga [52] propose mechanisms for measuring the delay of the first packet in a talkspurt for the CTI approach. These early algorithms all assume the delay distributions are known, and do not address adaptive algorithms for measuring this delay.

The first work in adaptive algorithms is by Naylor and Kleinrock [53], which proposes to use as a playout delay T in the NTI (Null Timing Information [48]) algorithm the largest difference in delays among the previous m packets. Recent work in addressing the problem specifically for the Internet is found in the work of Ramjee et al. [26] and Moon et al. [47]. Campbell [54] describes a QoS architecture which includes components for jitter adaptation. Their jitter compensation algorithm builds on that of Ramjee [26] by using a slightly different formulation and less conservative parameters.

There is a large amount of work on the overall problem of continuous media synchronization, of which playout buffer adaptation is only one component. The Adaptive Stream Synchronization Protocol (ASP) [55] allows synchronization of multiple streams across multiple hosts, using one stream as a master. The Lancaster Orchestration Service [56] allows multiple streams to be coordinated. Escobar et al. [57] propose a synchronization protocol for multi-site conferences.

Since our work builds on the work of Ramjee et al. and Moon et al. [26, 47], we review it briefly. Ramjee et al. [26] propose four adaptation algorithms. Each algorithm computes, in some fashion, an estimate of the mean network delay seen up to the arrival of the i^{th} packet, $\hat{d}_i$, and a variation measure in this delay, $\hat{v}_i$. The playout delay is adjusted at the beginning of each talkspurt. If t_i is the generation time of a packet which is the first in a talkspurt, the playout time p_i of the first packet is computed as $p_i = t_i + \hat{d}_i + \mu \hat{v}_i$, with $\mu = 4$. For a subsequent packet j in the same talkspurt, its playout time is computed as $p_j = p_i + t_j - t_i$.

All four algorithms maintain a running estimate of $\hat{d}_i$ and $\hat{v}_i$, updated for each packet. All four compute $\hat{v}_i$ in the same fashion:

$$\hat{v}_i = \alpha \hat{v}_{i-1} + (1 - \alpha) \left| \hat{d}_i - n_i \right|$$

where n_i is the network delay of the i^{th} packet. The four algorithms thus differ only in their computation of $\hat{d}_i$:

Exp-Avg: This algorithm estimates the mean delay through an exponentially weighted average, much like the estimation of the variance above. In particular, $\hat{d}_i = \alpha\hat{d}_{i-1} + (1-\alpha)n_i$, with $\alpha = 0.998002$.

Fast Exp-Avg: This algorithm is similar to the first, except it adapts more quickly to increases in delays by using a smaller weighting factor as delays increase:

$$\hat{d}_i = \begin{cases} \beta\hat{d}_{i-1} + (1-\beta)n_i & : \quad n_i > \hat{d}_{i-1} \\ \alpha\hat{d}_{i-1} + (1-\alpha)n_i & : \quad n_i \leq \hat{d}_{i-1} \end{cases}$$

For the algorithm to respond quickly to increasing delays, $\beta < \alpha$. Ramjee et al. [26] used $\beta = 0.75$.

Min-Delays: This algorithm attempts to be more aggressive in minimizing delays. It uses the minimum delay of all packets received in the current talkspurt (let $\mathcal{S}_i$ be this set of delays) as the average delay, i.e. $\hat{d}_i = \min_{j\in\mathcal{S}_i}(n_j)$

Spk-Delay: It has been observed by Bolot [18] that network delays often exhibit *spikes*, which are sharp increases in delay followed by nearly simultaneous reception of a large number of packets. *Spk-Delay* contains a spike detection algorithm which finds these spikes. During a spike, the delay estimate tracks the delays closely, and after a spike, an exponential weighted average is used. We avoid a detailed description here, referring the reader to the work of Ramjee et al. [26] for details.

Moon et al. [47] propose an algorithm similar to *Spk-Delay*, which we call *Window*. This algorithm also looks for spikes. During spike mode, the delay of the first packet in a talkspurt is used as the playout delay. In normal (non-spike) mode, the playout delay is chosen by finding the delay which represents the q^{th} quantile among the last w packets received by the receiver. This determination is easily made by incrementally updating a histogram of the delays among the last w packets. In their simulations, a value of $10,000$ is used for w.

2.5.3 New Playout Buffer Algorithms

In this section, we present our new algorithms which are capable of coupling and meeting non-zero loss targets.

2.5.3.1 Virtual Delay Algorithms

Our first contribution are a class of algorithms we call *virtual delay* playout adaptation algorithms. These algorithms are all modifications of existing algorithms to allow them to compensate for FEC. They are based on the following simple observation. Without FEC, the probability of playing out a particular packet is given in Equation 2.1. With FEC, we can generalize this simple formulation. A packet is played out if it either arrives before its playout time, or is recovered before its playout time. If p_N is the probability that a packet neither arrives nor is recovered, and d_V is the difference between the time a packet either arrives or is recovered (given it arrives or is recovered) and the time it was generated, the playout probability is

$$p_R = (1 - p_N)P[d_V < D].$$

This formulation is identical to the one in Equation 2.1, but with the simple random variables (p and n_i) replaced by the more complex ones (p_N and d_V). We can therefore use any existing playout adaptation algorithms which compute the playout delay D as some function of the packet delays by substituting d_V for n_i in the computation. Formally:

Definition 1 *The virtual delay d_V of a packet is the difference in time between the earlier of the arrival and recovery times, and the generation time. If a packet neither arrives nor can be recovered, the virtual delay is undefined.*

Any existing playout buffer adaptation algorithm is then *virtualized* by using the virtual delays to drive it instead of network delays. Deriving an expression for the virtual delay depends on the FEC mechanism in use.

2.5.3.1.1 Formulation for Redundant Codecs For the redundant encodings mechanism, deriving an expression for the virtual delay is straightforward. It is simply the difference in time between the arrival of a packet or any of its redundant versions, and the generation of the original packet. If a_i is the arrival time of the i^{th} packet (undefined if it never arrives), t_i is the generation time of the i^{th} packet, and each packet contains redundant versions of the previous $K-1$ packets, d_V^i (the virtual delay of the i^{th} packet) is defined as:

$$d_V^i = \min_{j=0..K-1}(a_{i+j}) - t_i \qquad (2.6)$$

2.5.3.1.2 Formulation for Reed Solomon FEC For Reed Solomon FEC, deriving an expression for the virtual delay is also straightforward. We assume an (n, k) code. A packet can be recovered when k packets in the block arrive. Thus, the virtual delay is the minimum of the k^{th} arrival in the block and the actual arrival time of the packet (if it arrives), minus the generation time of the packet. For convenience of notation, assume that $i = 0$ is the first packet in the block. Thus:

$$d_V^i = \min(a_i, r_i) - t_i,$$

where r_i is the the smallest arrival time of the j^{th} packet, a_j, for $j = 1..n$ which satisfies

$$\sum_{j=1}^{n} I(a_j \leq r_i) \geq k.$$

Note that $I(x)$ is the indicator function, equal to 1 if its argument is true, zero otherwise. This equation basically says that r_i is the arrival time of the k^{th} packet to arrive.

2.5.3.1.3 Implementation In most cases, it is not necessary to derive an expression for d_V to do this. A receiver just implements the FEC algorithm as it normally would. The instant it recovers a packet, the virtual delay of the packet is computed as the difference in time between transmission of the packet and the current time. If a packet is correctly received, the virtual delay is the network delay of the packet, unless it is recovered before it is received. The virtual delays are then used in the adaptation algorithm instead of the network delays.

It is important to note that implementing the virtualized version of the algorithms is a natural consequence of a layered implementation of FEC and playout buffer adaptation, where the recovery using FEC is done before the playout buffer adaptation. The playout buffer adaptation component cannot differentiate recovered packets from received packets. In this case, it will compute the arrival time of a packet as either its recovery time or arrival time, whichever is lower. The result is that the playout buffer adaptation algorithm will unknowingly compute virtual delays instead of real delays, and thus become virtual.

2.5.3.1.4 Proof of Correctness While virtualization seems a natural approach to adding FEC awareness, the question is: is it mathematically correct? Does the algorithm, in an ideal setting, achieve its goal?

In this section, we prove, in the case of redundant audio codecs, that virtualization produces correct behavior. First, we need a definition of correctness. To do this, we look to the behavior of regular playout buffers in the absence of FEC. These algorithms choose some playout delay D. The playout probability is simply

$$p_R = (1-p)P[n_i < D].$$

To be correct, the virtual algorithms should exhibit the same behavior. Choosing a playout delay of D should result in an playout probability p_R such that

$$p_R = (1-p_N)P[d_V < D].$$

We have already derived, in equation 2.2, an expression for the probability a packet is played out with the redundant codec mechanisms. We can therefore prove correctness by showing that these two are equivalent, that

$$(1-p_N)P[d_V < D] = 1 - \prod_{j=0}^{K-1} 1 - (1-p)P[n_i + j\Delta < D].$$

We start with the definition of d_V for redundant codecs, given in Equation 2.6. Note that this equation is only over those packets which arrive, and is defined only if at least one packet arrives. In order to perform an analysis with this variable, we must define a closely related one which is defined under all conditions. We therefore define d'_V, which is infinite if none of the packets arrive. We also presume, without loss of generality, that the arrival time of a packet which never arrives is infinite. The first step is to derive an expression for $P[d'_V < D]$:

$$\begin{aligned}
P[d'_V < D] &= 1 - P[d'_V \geq D] \\
&= 1 - P[\min_{j=0..K-1}(a_{i+j}) - t_i \geq D] \\
&= 1 - \prod_{j=0}^{K-1} P[a_{i+j} - t_i \geq D]
\end{aligned}$$

Computing an expression for $P[a_{i+j} - t_i \geq D]$:

$$\begin{aligned}
P[a_{i+j} - t_i \geq D] &= 1 - P[a_{i+j} - t_i < D] \\
&= 1 - P[a_{i+j} - t_{i+j} + (t_{i+j} - t_i) < D] \\
&= 1 - P[a_{i+j} - t_{i+j} + i\Delta < D] \\
&= 1 - (1-p)P[n_{i+j} + i\Delta < D]
\end{aligned}$$

The last step causes the addition of the $(1 - p)$ term since the packet must arrive for its arrival time to be finite. Finally, assuming stationarity:

$$P[a_{i+j} - t_i \geq D] = 1 - (1-p)P[n_i + i\Delta < D] \tag{2.7}$$

The result is an expression for $P[d'_V < D]$:

$$P[d'_V < D] = 1 - \prod_{j=0}^{K-1} (1 - (1-p)P[n_i + i\Delta < D]) \tag{2.8}$$

Next, we must relate d_V to d'_V. The relationship is fairly straightforward. Since d_V is defined only when at least one packet arrives, it is equivalent to d'_V conditioned on the arrival of at least one packet. Let event a represent the arrival of at least one packet. Then:

$$\begin{aligned}
P[d_V < D] &= P[d'_V < D|a] \\
&= \frac{P[d'_V < D \wedge a]}{P[a]}
\end{aligned}$$

For d'_V to be less than D, it must be finite, which means that at least one packet must arrive. Thus, the event $d'_V < D$ includes the event a. $P[a] = 1 - p^K$, so:

$$\begin{aligned}
P[d_V < D] &= \frac{P[d'_V < D]}{1 - p^K} \\
&= \frac{1}{1 - p^k} 1 - \prod_{j=0}^{K-1} (1 - (1-p)P[n_i + i\Delta < D])
\end{aligned}$$

Finally, p_N is the probability that the packet is neither receiver nor recovered. This is just the probability that all the packets in the block are lost, so $p_N = p^K$. Combining this with Eq. 2.9 and inserting into Eq. 2.5.3.1.4 yields the desired result:

$$(1 - p_N)P[d_V < D] = 1 - \prod_{j=0}^{K-1} 1 - (1 - p)P[n_i + i\Delta < D]$$

This verifies correctness of the virtualization for the redundant codecs. Unfortunately, the proof of correctness for Reed Solomon FEC is less straightforward, and omitted here.

2.5.3.1.5 Supporting Target Loss Probabilities It is the job of the playout adaptation algorithm to trade loss for delay. For the algorithms by Ramjee et al. [26], this tradeoff is controlled by the variation multiplier μ, set to a large static value (here, 4). The result are algorithms which target near zero loss. To be able to target non-zero losses, we must use a different parameter. If the delay distribution of the packets were known ahead of time, the parameter could easily be chosen as the desired quantile of the delays. However, the delay distributions are not known ahead of time. As a result, we propose to make this parameter adaptive.

Our algorithm assumes a target value for the application loss probability p_R. Let this target be p_T. Unfortunately, you can't always get what you want, and this target may be unachievable given the current network loss rate. Therefore, we compute a current target loss rate p_C:

$$p_C = \max(p_T, f(p)) \tag{2.9}$$

where $f(p)$ is the achievable minimum application loss probability given a network loss rate of p.

The minimum achievable application loss probability is the application loss probability given infinitely large buffers. This represents a lower bound on the probability for any adaptive playout algorithm. $f(p)$ is readily computed:

$$f(p) = lim_{D \to \infty} 1 - p_R$$

In the case of redundant codecs, we take the limit of one minus Eq. 2.2:

$$f(p) = lim_{D\to\infty} \prod_{j=0}^{K-1} 1 - (1-p)P[n_i + j\Delta < D] \tag{2.10}$$

The limiting case implies that $P[x < D] = 1$ for any finite r.v. x, including $n_i + j\Delta$. So:

$$\begin{aligned} f(p) &= \prod_{j=0}^{K-1} 1 - (1-p) && (2.11) \\ &= p^K && (2.12) \end{aligned}$$

For Reed Solomon FEC, it is the limit of one minus Eq. 2.3. As $D \to \infty$, $F(i, j, z)$ simplifies to:

$$F(i,j,z) = \begin{cases} 1-p & z = 1 \\ p & z = 0 \end{cases}$$

From this:

$$P[\vec{S_i}] = (1-p)^l p^{n-l}$$

where there are l ones in the vector $\vec{S_i}$. p_R is a sum of $P[\vec{S_i}]$ over those $\vec{S_i}$ which have at least k ones, or for which the i^th entry is a one. The probability of a vector with the i^{th} component a one is just $1-p$. Given the i^{th} component is not a one, the probability is over those vectors with at least k of the remaining $n-1$ positions equal to one. As a result:

$$f(p) = p\left(1 - \sum_{l=k}^{n-1} \binom{n-1}{l} (1-p)^l p^{n-1-l}\right) \tag{2.13}$$

To make use of these equations, we maintain a running estimate of the network loss rate $\hat{p}$. At the end of each talkspurt, $\hat{p}$ is used to compute p_C via Eq. 2.9 and the desired target loss rate p_T. Then, the actual application loss rate p_L is measured. The variation multiplier μ defined by Ramjee et al. [26] is computed for the next talkspurt based on the following algorithm:

```
if $(p_C < p_L - \theta) \wedge (\mu \leq \mu_{\max} - \delta_{inc})$
  $\mu \leftarrow \mu + \delta_{inc}$;
else if $(p_C > p_L + \theta) \wedge (\mu \geq \mu_{\min} + \delta_{dec})$
```

```
    $\mu \leftarrow \mu - \delta_{dec}$;
else
    $\mu \leftarrow \mu$
```

θ is a threshold which introduces hysteresis into the algorithm. μ_{min} and μ_{max} are the minimum and maximum allowed values for μ. δ_{inc} and δ_{dec} represent the step size for the coefficient. In our simulations, we used $\mu_{min} = 0$, $\mu_{max} = 8$, $\theta = .05$, $\delta_{inc} = 0.4$ and $\delta_{dec} = 0.2$. We found performance was best when the rate of increase in μ (δ_{inc}) was larger than the rate of decrease (δ_{dec}). We believe this is because it allows the algorithm to act more conservatively, rapidly increasing delays (and thus reducing losses) but gradually reducing them. As such, short lived decreases in jitter or loss won't cause the algorithm to undershoot the needed delay.

This small enhancement can be applied to all of the algorithms proposed by Ramjee et al. [26]. When coupled with virtualization, we call the resulting algorithms *adaptively virtual*.

2.5.3.2 "Previous Optimal" Algorithm

Ideally, an optimal playout delay algorithm would work as follows[1]. It would work offline, and know the packet delays and losses ahead of time. It would choose a playout delay which would meet an arbitrarily chosen criteria. Unfortunately, this algorithm is non-causal. As with many non-causal filters, this algorithm can be made causal by delaying the output of the algorithm for a talkspurt. In other words, the optimal playout delay for the previous talkspurt can be used as a playout delay for the next talkspurt. More generally, we can use any function of the optimal playout delays from previous talkspurts as the playout delay of the current talkspurt. For reasons of simplicity, we considered exponentially weighted averages of previous talkspurts.

At the end of each talkspurt, we compute the optimal playout delay D_{opt} for that talkspurt. We then choose the playout delay D_w for the next talkspurt as

$$D_w = \rho D_{w-1} + (1 - \rho) D_{opt}$$

We chose $\rho = .25$, which allowed the algorithm to adapt quickly. We also observed that in cases where the target loss is quite low (less than 2%), we needed to add a variation multiplier, so that

[1]Note that the original idea for this algorithm was contributed by Lili Qiu

the actual playout delay D_{act} is $D_{act} = D_w + \mu\hat{v}_w$ where $\hat{v}_w$ is computed in the same fashion as described by Ramjee et al. [26], except that it is driven by D_w. We used the adaptive algorithm described in Section 2.5.3.1.5 to compute μ, but with $\mu_{MAX} = 6$, which we found to give slightly better results.

We define the optimal playout delay as the one which maximizes $F(D, p_R)$, where $F()$ is a user satisfaction function. Though to our knowledge there is no widely adopted function quantitatively characterizing how playout delay and loss determines the audio quality, we can use any reasonable function that can capture the relationship. For example, a reasonable choice of F as used in our simulations is to find the minimum playout delay needed to achieve a specified application loss rate. Our algorithm is general enough that it is not bound to any particular choice of F, though its success does depend on a sensible F.

For a particular choice of playout delay D, the number of packets played out are all of those with $d_V^i \leq D$. Interestingly, for any F which decreases with increasing delays, the optimal playout delay can only be equal to one of the values of d_V^i. For any delays between two adjacent d_V^i, the number of packets for which $d_V^i > D$ remains the same, but the delay increases. As a result, the user satisfaction function decreases. This means we are trying to maximize:

$$F(d_V^j, \frac{1}{N}\sum_i I(d_V^i \leq d_V^j))$$

over j, where N is the number of packets in the talkspurt, and I() is the indicator function.

In our explorations, we define $F(x, y)$ such that optimal operating point is the minimal delay which comes closest to the target operating playout probability p_R, but does not fall below it. The result is that we are seeking D_{opt} as the minimum d_V^j which satisfies:

$$\frac{1}{N}\sum_{i=1}^{N} I(d_V^i < d_V^j)) > p_R$$

Computing D_{opt} is O(N). To reduce the complexity, we quantize the virtual delays using a linear quantizer with L steps. The quantizer uses a step size of 5 ms, and sets the $\frac{L}{2}^{th}$ quantized value to the virtual delay of the first packet received for the talkspurt. Any packet with a delay more than 400 ms larger or smaller than the first packet is quantized to $L - 1$ and 0, respectively. We maintain a probability mass function of virtual delays in an array. At the end of the talkspurt, the probability distribution function is computed by summing this array. We then perform a search over the L entries in the distribution array to compute D_w. This search is O(1) (with a constant factor L).

2.5.3.3 Model-Based “Analytical” Playout Adaptation Algorithm

The analytical algorithm works by using Eq. 2.3 and Eq. 2.2 directly. The equations represent the resulting playout probability, given the network loss probability p and delay distribution, as a function of the playout delay D. Our approach is to measure these parameters in real time during operation, and then use them to choose a value of D which meets some specific criteria.

Our simulations focus on Reed Solomon FEC, in part because the results of Section 2.4 demonstrate its greater utility for low bitrate voice, compared to redundant codecs. Unfortunately, Eq. 2.3 is not sufficiently simplified to be of practical use in implementing a playout buffer algorithm.

To develop an alternate formulation, we further assume that within a block of n packets, the packets arrive in order. We observe that Eq. 2.3 basically says that the probability of playing out a packet is the probability it arrives or is recovered in time. If p_A is the probability it arrives in time, and p_F the probability it is recovered in time given it did not arrive in time, we can express p_R as:

$$p_R = p_A + (1 - p_A)p_F$$

Computing p_A is straightforward; its the probability a packet arrives $(1 - p)$ times the probability its on time given it arrives ($P[n_i \leq D]$):

$$p_A = (1 - p)P[n_i \leq D]$$

Now let us compute p_F. This quantity depends on the position of the lost data packet in the block of k packets. If the loss occurs in the later part of the group, it has a higher chance of recovery. This is because the likelihood is high that the FEC packets will arrive in time to make use of them for recovery. However, for a packet lost in the beginning of the block, the recovery probability is lower. This is because the FEC packets will not be sent for some time, and are unlikely to arrive in time. p_F is therefore computed by averaging over all k data packets:

$$p_F = \frac{1}{k}\sum_{i=1}^{k} p_F^i$$

Computation of p_F^i is somewhat complex. We define the following terms. Let event $X_1(i,j)$ be the event where the last packet in the group that could possibly be received prior to the playout time of the i^{th} packet is the $i+j^{th}$. Let event $X_2(i+j)$ be the event that at least $k-1$ packets of those sent before and including packet $i+j$ arrive. We then have:

$$p_F^i = \sum_{j=k+1-i}^{n-i} P[X_1(i,j)]P[X_2(i+j)]$$

Derivation of $P[X_1(i,j)]$ is fairly easy. Since all packets arrive in order, $i+j$ would be the last to arrive before playout if it would arrive before playout ($n_j + j\Delta \leq D$) and the next packet (if it arrives) would arrive after playout ($n_{j+1} + (j+1)\Delta > D$). To simplify computation, we assume that the two delays n_j and n_{j+1} are nearly identical, so we can assume they are a single random variable d:

$$P[X_1(i,j)] = \begin{cases} P[D-(j+1)\Delta < d \leq D - j\Delta] & \text{if } i+j<n \\ P[d \leq D - j\Delta] & \text{if } i+j=n \end{cases}$$

Derivation of $P[X_2(i+j)]$ is also straightforward. It is probability that at least k packets arrive, among the $i+j-1$ candidate packets (packet i is lost, so it is not a candidate):

$$P[X_2(i+j)] = \sum_{r=k}^{i+j-1} \binom{i+j-1}{r} (1-p)^r p^{i+j-1-r} \tag{2.14}$$

For simplicity, we define $S(j) = P[d \leq D - j\Delta]$ so that:

$$P[X_1(i,j)] = \begin{cases} S(j) - S(j+1) & \text{if } i+j<n \\ S(j) & \text{if } i+j=n \end{cases}$$

The final expression is:

$$\begin{aligned} p_R \quad &= \quad (1-p)P[d \leq D] \\ &\quad\;\; (1-(1-p)P[d \leq D])\frac{1}{k}\sum_{i=1}^{k}\sum_{j=k+1-i}^{n-i} P[X_1(i,j)]P[X_2(i+j)] \end{aligned}$$

To speed up computation of this function, we attempt to simplify the outer sum. We first expand $P[X_1(i,j)]$, splitting off the term where $j = n$:

$$\begin{aligned}
&\sum_{i=1}^{k}\sum_{j=k+1-i}^{n-i} P[X_1(i,j)]P[X_2(i+j)] \\
= &\sum_{i=1}^{k}\sum_{j=k+1-i}^{n-i-1} (S(j)-S(j+1))P[X_2(i+j)] + \sum_{i=1}^{k} S(n-i)P[X_2(i+n-i)]
\end{aligned}$$

We then add and subtract $S(n-i+1)P[X_2(i+n-i)]$ so that we can pull the term back into the summation:

$$\begin{aligned}
= &\sum_{i=1}^{k}\sum_{j=k+1-i}^{n-i-1} (S(j)-S(j+1))P[X_2(i+j)] + \sum_{i=1}^{k}(S(n-i)-S(n-i+1)) * P[X_2(i+n-i)] \\
&+\sum_{i=1}^{k} S(n-i+1)P[X_2(i+n-i)] \\
= &\sum_{i=1}^{k}\sum_{j=k+1-i}^{n-i} (S(j)-S(j+1))P[X_2(i+j)] + \sum_{i=1}^{k} S(n-i+1) * P[X_2(i+n-i)]
\end{aligned}$$

Now, changing variables to $g = j + i$, we can invert the orders of summation:

$$\begin{aligned}
= &\sum_{i=1}^{k}\sum_{g=k+1}^{n} (S(g-i)-S(g-i+1))P[X_2(g)] + \sum_{i=1}^{k} S(n-i+1)P[X_2(n)] \\
= &\sum_{g=k+1}^{n}\sum_{i=1}^{k}(S(g-i)-S(g-i+1)) * P[X_2(g)] + \sum_{i=1}^{k} S(n-i+1) * P[X_2(n)]
\end{aligned}$$

And finally, we can collapse the innermost sum in the first expression, since most of the $S(j)$ terms cancel each other out:

$$= \sum_{g=k+1}^{n} [(S(g-k)-S(g))P[X_2(g)]] + \sum_{i=1}^{k} S(n-i+1)P[X_2(n)]$$

The final expression is then

$$\begin{aligned}
p_R \;=\; &(1-p) * P[d \le D] + \\
&(1-(1-p)P[d \le D])\frac{1}{k}
\end{aligned}$$

$$\left(\sum_{g=k+1}^{n} (S(g-k) - S(g))P[X_2(g)] \right.$$
$$\left. + \sum_{i=1}^{k} S(n-i+1)P[X_2(n)] \right) \tag{2.15}$$

where

$$P[X_2(g)] = \sum_{r=k}^{g-1} \binom{g-1}{r} (1-p)^r p^{g-1-r}$$
$$S(j) = P[n_i \leq D - j\Delta]. \tag{2.16}$$

What we desire for the actual algorithm, in fact, is the inverse of this function: to compute D as a function of p_R and the delay and loss. Since the function cannot be inverted in closed form, we obtain D by trying various values, computing p_R based on those values using Eq. 2.15, and comparing p_R against the desired value. This is effectively a search. In fact, since the function is monotonically increasing with D, we perform a binary search. At the end of each talkspurt, the current target loss rate p_C is computed, using equation 2.9. We start with an arbitrary value for D, and compute p_R from above. If the result is below p_C by more than some threshold (we used 5%), a higher value of D is tried, and if the result is above p_C, a lower value is tried. The problem can also be posed as a zero-finding problem. We seek the value of x such that

$$p_R(x) - p_C = 0.$$

Standard numerical techniques can also be applied to obtain the value of x, assuming p_R can be differentiated (which is a challenge as well, and one that we did not bother to pursue).

The computation of p_R from Eq. 2.15 requires an expression for the network delay distribution $P(n_i < X)$ for a range of X, and an estimate of p. We measure the loss probability p by computing the percentage of lost packets in each talkspurt ($\tilde{p}$), and applying an exponential filter to average this value, yielding $\hat{p} = 0.25\tilde{p} + 0.75\hat{p}$.

To compute the network delay distribution, we maintain the delays of the last 1000 packets in a queue. Each delay is first quantized, using a linear quantizer with a step size of 5 ms and upper limit of 5 s. The frequency of each delay is maintained in a histogram. When a new packet arrives, the delay of the oldest packet is removed from the histogram, and the delay of the newest is added. The delay distribution is computed using a cumulative sum of the frequencies, and is done only at the end of each talkspurt.

2.5.4 Simulations

The objective of our simulations is two-fold: first, two demonstrate that our new algorithms outperform the decoupled ones; second, to determine the performance of the new algorithms compared to each other.

2.5.4.1 Simulation Model

In our simulation model, the sender generates speech packets every 20 ms. Each packet consists of an IP/UDP/RTP [2] header, totaling 40 bytes, in addition to 24 bytes of speech. This is equivalent to a 9.6 kb/s speech codec. The speech is protected with a (5,3) Reed-Solomon code covering non-overlapping groups of three packets. This means that every three packets, two additional FEC packets are generated. As long as the receiver gets any three of the five packets, the three media packets are recovered. These two FEC packets are piggybacked on the first two data packets in the next block, so that the FEC is sent spaced apart. Recent results [34] have shown that spacing the FEC in this manner yields better performance.

To model packet loss, we used both Bernoulli and Gilbert processes; studies have shown these to be reasonable models [34]. However, delay models, particularly those that capture correlation, are not easy to find. As a result, we based our simulations on a network model which combines real measured traces on the Internet with simulated losses. Our real traces are those described in Section 2.2. We used traces 1, 2 and 3. To decouple the effect of clock skew and our algorithms, we used the algorithm described by Moon [58] to remove clock skew from these traces. The resulting average one way delays were 2 ms, 6.4 ms and 13.3 ms, respectively. The jitter was 2.64 ms, 3.46 ms, and 8.18 ms, respectively, computed as the standard deviation in the delay.

To support a wider range of simulations, we salted these traces with additional losses. Of the packets which did arrive in the actual trace, each was subject to a simulated loss with probability p drawn from a Bernoulli process. By varying p, we were able to adjust the network loss rates from the actual value in the trace ($p = 0$) to 1 ($p = 1$). We also ran simulations with Gilbert loss, and observed similar results.

At the receiver, the playout buffer algorithms were implemented, and the FEC was used to recover lost packets when possible.

2.5.4.2 Coupled vs. Uncoupled

In this section, we compare uncoupled versions of the Exp-Avg, Spk-Det and Window algorithms with our virtually adaptive versions of these algorithms. For brevity, we omit the simulation results of the fast exponential average and min-delay algorithms, since according to our simulation results, these two algorithms usually do not outperform the others. For the window algorithm, we use a 1000-packet window instead of the 10,000-packet widow used by Moon et al. [47], which we found to be too large to adapt adequately. The adaptively virtual versions of the algorithms operate with a loss target of zero.

We use two uncoupled versions of these algorithms. The first version is the original algorithm. No extra delay is added to compensate for FEC. In the second version, we added 80 ms of delay to the original output of each algorithm. This corresponds to $N - 1$ packet intervals, enough to receive all the FEC for any packet in the absence of jitter.

Figures 2.25, 2.26 and 2.27 each contain six graphs. The two at the top compare the performance of the uncoupled *Exp-avg* algorithm with its adaptively virtual extension; the two in the middle compare the uncoupled *Spk-det* algorithm with its extension; the two at the bottom compare uncoupled *window* algorithm with its extension. Each graph in each figure has three curves, corresponding to the two uncoupled versions of the algorithm and the adaptively virtual extension. The left column in each figure shows the average application loss probability after FEC and playout buffer adaptation ($1 - p_R$) vs. the network loss probability p, varied by salting the traces. The right column shows the average D across all talkspurts vs. the network loss probability p. All plots use trace two.

Figure 2.25 corresponds to trace 2, Figure 2.26 corresponds to trace 1, and 2.27 corresponds to trace 3.

First let us consider Figure 2.26. In all three pairs of graphs, the results are the same. The

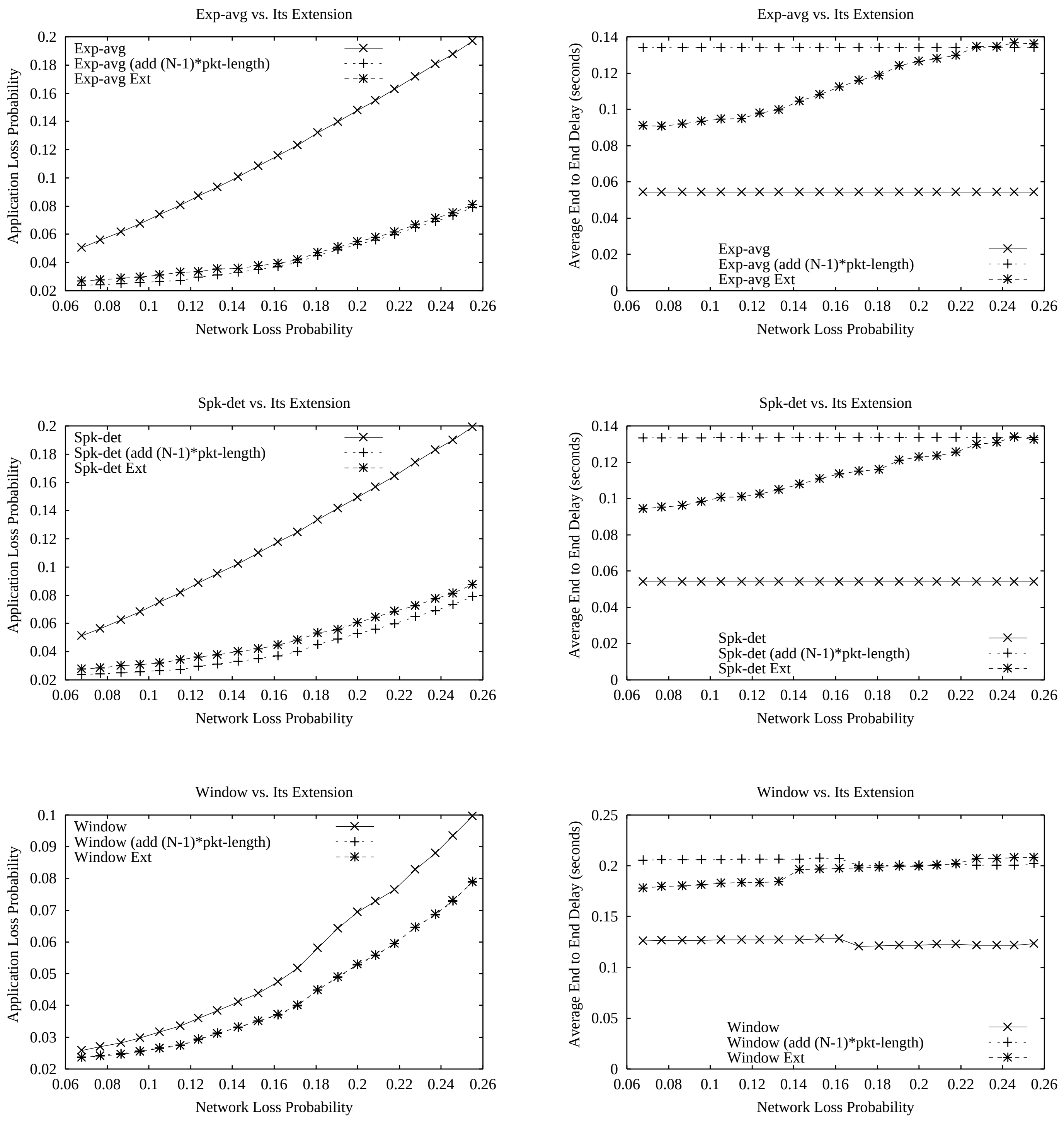


Figure 2.25: Performance of Adaptively Virtual Algorithms on Trace 2

original version of the algorithm shows increasing application loss probabilities with increasing network loss rate. This is because the playout delay computed by the algorithm is not large enough to make use of the FEC packets. The plots in the right column show this as well. The original version of each algorithm has a constant playout delay as the network loss varies. For the original versions of Exp-Avg and Spk-Det, the increase in application loss probability is linear with the network loss probability. This is because the playout delay is not sufficiently large to allow the FEC to be consistently used (80 ms is needed).

When we add 80 ms of delay to the output of the original algorithms, the loss rates drop substantially. Now, there is sufficient delay to make full use of FEC. In fact, the loss rates track those of our adaptively virtual extensions. However, the decoupled algorithm now tends to have consistently large playout delays, even when network loss rates are small and the delay is not needed.

Our adaptively virtual extensions perform much better. The right graphs show that in all cases, the network delays start low, and gradually increase as network loss rates increase. The result is that the end-to-end delay of our extension is generally lower than the uncoupled version which adds 80 ms, but with the same low application loss probabilities. This is exactly the desired behavior.

The results in Figure 2.26 show similar trends. However, in this trace, the network loss probabilities are very small to start with, and the jitter is quite low. As a result, the decoupled algorithm without the additional delay tends to significantly underestimate the playout delay needed to make use of the FEC, as the network loss rates increase. In fact, the playout delays are so small that the FEC is almost never used, and as a result, the application loss probabilities tend to increase linearly with the network loss probability, just as if the FEC was never sent at all. With the decoupled algorithm that adds an additional 80 ms, the playout delay is much larger than needed when network losses are low. Notice, however, that our adaptively virtual algorithm performs extremely well here. It increases the playout delays gradually as the network loss probability increases. The result is usually just enough buffering to make use of FEC. This can be seen in the graphs in the left column, where the application loss probabilities are quite close to the case where an additional 80 ms is always inserted.

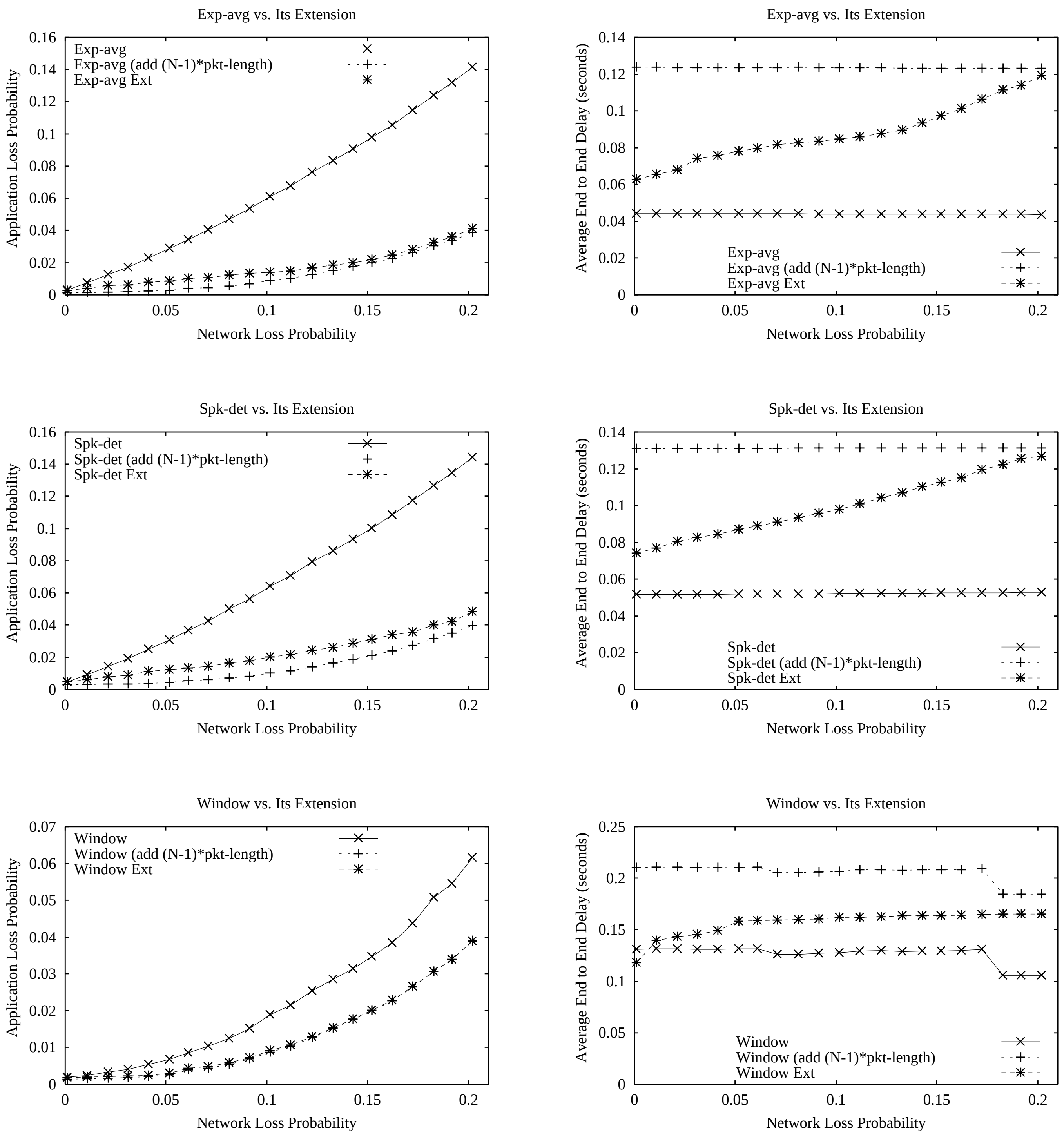


Figure 2.26: Performance of Adaptively Virtual Algorithms on Trace 1

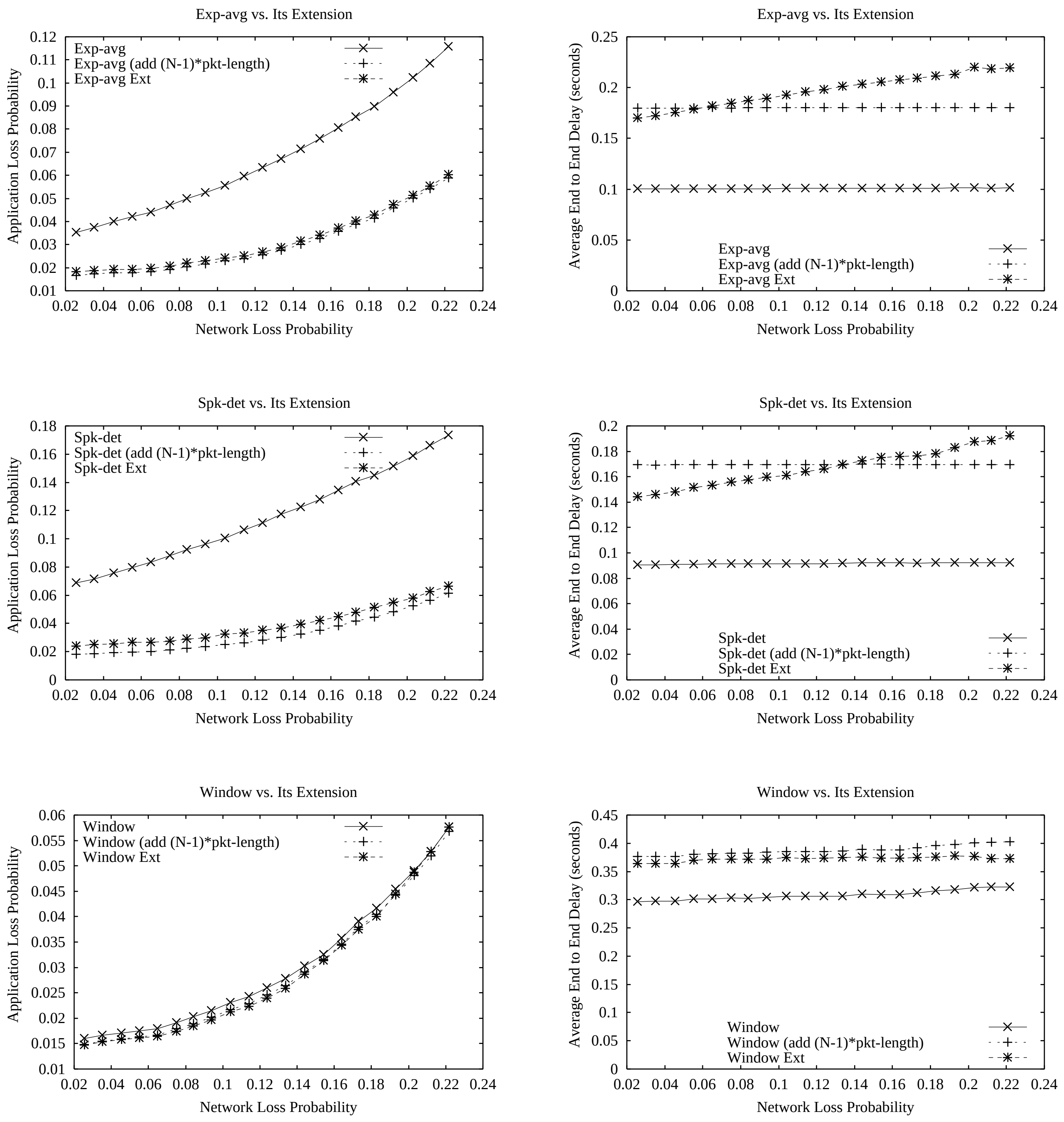


Figure 2.27: Performance of Adaptively Virtual Algorithms on Trace 3

Figure 2.27 shows results for trace 3. In this trace, there are substantial amounts of jitter and loss. The result is that our adaptively virtual algorithm adds enough delay to get all of the FEC data, even at low loss rates. This can be seen in the graphs in the right column, where the adaptively virtual playout delays track the playout delays of the decoupled algorithm with the additional delay. We believe this is because the substantial jitter is causing the algorithm to require long playout delays even to obtain small amounts of additional FEC.

2.5.4.3 Comparisons of New Algorithms

In this section, we compare the performance of our new algorithms (adaptively virtual extensions, previous optimal, and analytical) against each other, and against the optimal one. The simulation environment is identical to that described in Section 2.5.4.1. Our simulations cover two metrics: the ability to make good use of FEC with small delays, and the ability to achieve a target application loss rate with the minimal delay.

2.5.4.3.1 Using FEC with Minimal Delays Figure 2.28 depicts the application loss rate (left column) and average playout delay (right column) vs. network loss rate. Each graph shows the performance of the adaptively virtual Exp-Avg algorithm *Exp-Avg Ext*, the adaptively virtual Spk-Det algorithm *Spk-Det Ext*, the adaptively virtual window algorithm *Window Ext*, the previous optimal *Prev-opt (Bin)* algorithm and the analytical algorithm, all targeting zero loss. We also include a plot of an unrealizable non-causal "optimal" algorithm. This algorithm computes the playout delay for all talkspurts at the beginning of the trace (assuming knowledge of the packet delays and losses in the entire trace), based on minimizing the average delay for the entire trace for a given application loss probability. This optimal algorithm is actually the lower bound described by Moon [47], but with the virtual delays replacing the network delays in the computation.

The figure has three pairs of graphs; each pair is obtained from a different trace. The results indicate fairly consistent behaviors across traces. The adaptively virtual window algorithm tends to have the highest delays, and low loss rates, but not the lowest. The analytical and previous optimal algorithms generally have lower end-to-end delays than the adaptively virtual window algorithm (with the exception of the previous optimal algorithm in trace 1), with application loss

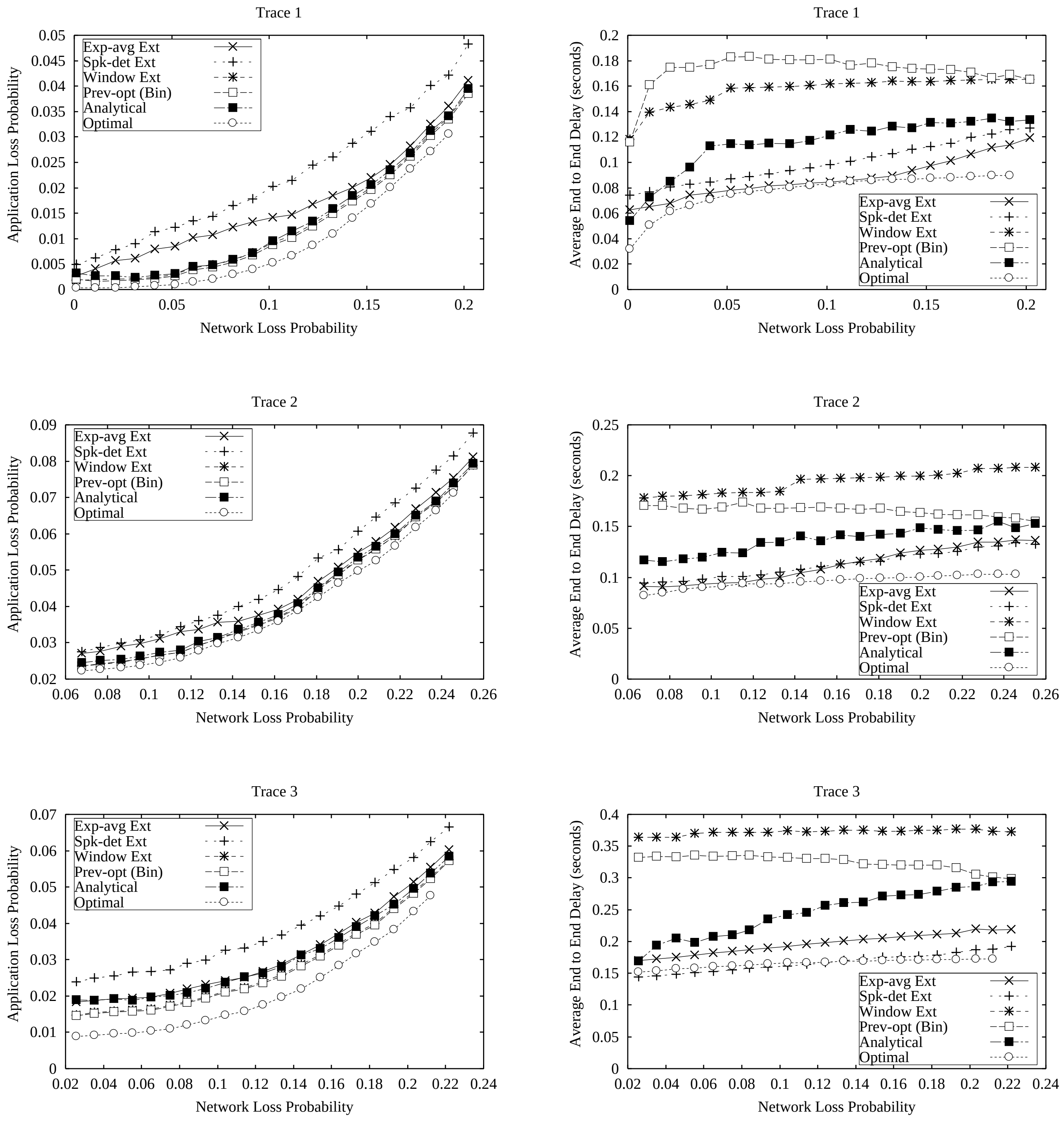


Figure 2.28: Comparison of Loss and Delay Performance across All Algorithms

probabilities that are generally equal to or less than the adaptively virtual window algorithm. This indicates that the analytical and previous-optimal algorithms are generally preferable to the adaptively virtual window algorithm.

The virtually adaptive Exp-Avg algorithm appears to perform quite well. Its delays are consistently closest to the optimal (only Spk-Det does noticeably better in trace 3), and its loss probabilities are only slightly higher than the previous optimal and analytical algorithms. The virtually adaptive Spk-Det algorithm also maintains low delays, but its application loss probabilities are generally the highest. This would indicate that the adaptively virtual Exp-Avg is generally preferable to the adaptively virtual Spk-Det. This result contradicts those of Ramjee et al. [26], where the more sophisticated Spk-Det outperforms Exp-Avg. We believe this is because Spk-Det attempts to track the network delays too closely and loses packets whenever its delay estimate is small. The results by Moon et al. [47] agree with ours.

2.5.4.3.2 Achieving a Specific Loss Target All of our algorithms are capable of achieving a specified loss target, input as a parameter to the algorithm. In Fig. 2.29, we consider the ability of the algorithms to meet a target loss probability of 0.07, as the network loss probability varies from 0 to 0.02. As with the other plots, the left column represents the application loss probability, and the right column, the end-to-end delay. Each pair of graphs is for a different trace.

The traces show that the adaptively virtual Exp-Avg and Spk-Det are the consistent top performers. They consistently come close to the target loss rate, and follow the optimal playout delay as well. The adaptively virtual window is consistently the worst performer. The analytical and previous optimal algorithms are somewhere between. We note that trace 1 presented a challenge for all of the algorithms. The small amounts of jitter in the trace meant that increasing the loss to the target with small network loss probabilities requires very fine tuning of the playout delay. As such, all the algorithms significantly undershoot the target until the network loss probability hits the target.

2.5.4.3.3 Achieving a Varying Loss Target In Fig. 2.30, we consider the ability of the algorithms to meet a wide range of targets under a fixed network condition. This fixed network condition corresponds to the unsalted traces 1, 2 and 3. The target loss rate is varied from 0 to

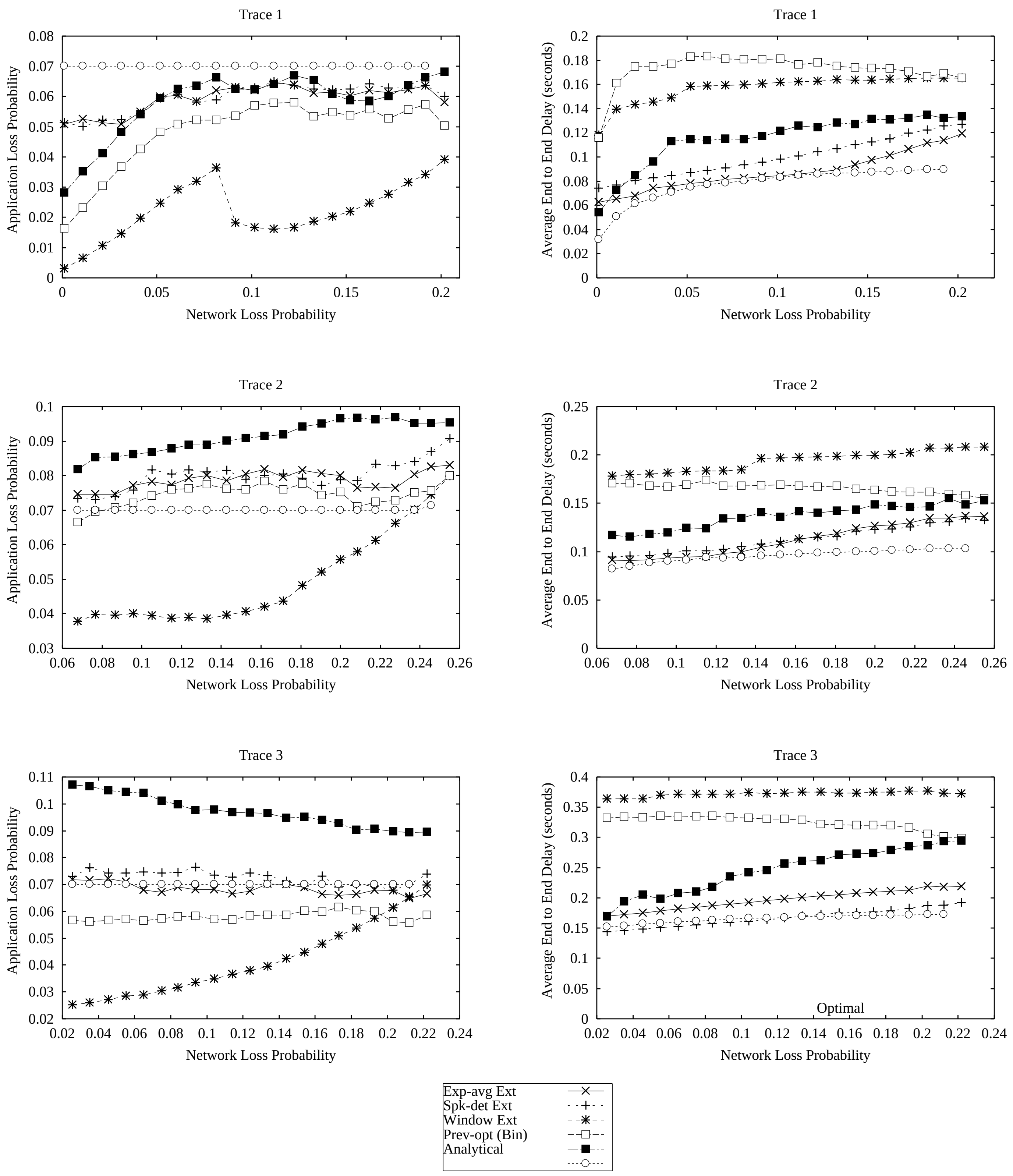


Figure 2.29: Performance of Algorithms in Achieving a Target Loss of 0.07

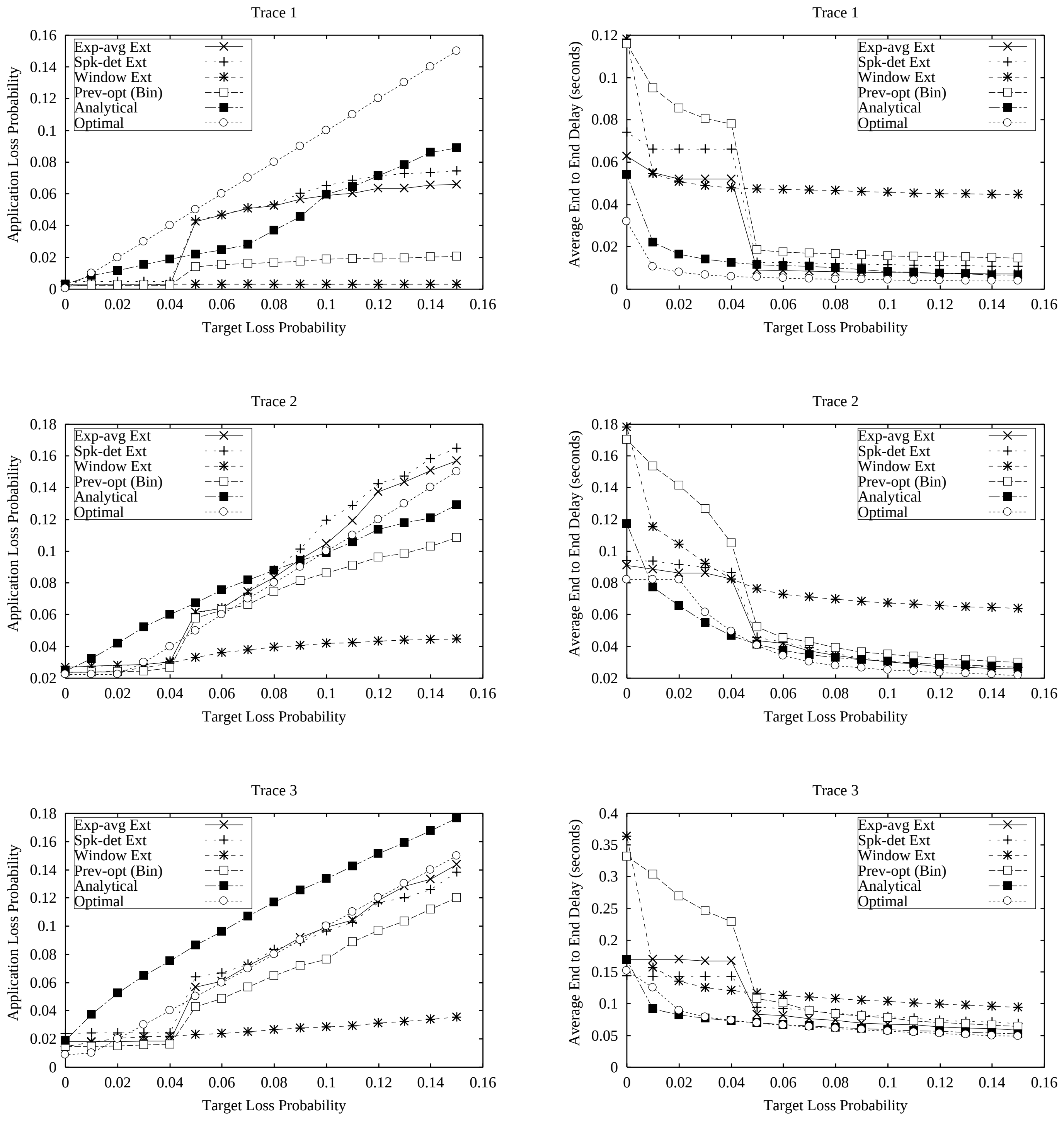


Figure 2.30: Performance of Algorithms in Achieving Varying Target Loss Probability

15%. As with the other plots, the application loss rate and playout delay are shown.

Ideally, the application loss rate should equal the target, corresponding to a straight line with slope one on the left graphs. The non-causal optimal algorithm achieves this goal, of course. The plots on the right show that the optimal algorithm decreases the end-to-end delay as the target increases, as expected.

The results here are consistent with those in the previous section. The adaptively virtual window algorithm is the worst performer, consistently overshooting the required delay and, as a result, undershooting the target. The previous optimal algorithm consistently undershoots the target loss, although it comes close in all traces but the first (Its worthwhile to note that trace 1 has little jitter or loss, making it difficult to achieve a large target loss rate). Not surprisingly, it also tends to overshoot the optimal delay. The analytical algorithm tends to have very good delay properties, coming close to the optimal delay. Its ability to meet the loss target varies, though. It consistently undershot it in trace 1 (although, in all fairness, all other algorithms undershot it in this trace as well), but overshot it in all the others, although not by too much. Performance is especially good in trace 2, where the analytical algorithm is the best performer. The adaptively virtual Exp-Avg and Spk-Det have similar performance. They both came consistently close to the target with reasonably good delays.

Overall, the simulations demonstrate that our algorithms were generally able to meet the goal of achieving a desired target loss rate. The Exp-Avg and Spk-Det appear to be the best performers overall.

2.6 Transport of Media-Unaware FEC

There are several practical issues to be resolved when using FEC for recovery of real time media. This includes its interaction with playout buffers, as we have discussed above, but it also includes the seemingly mundane task of pure transport. The issue is: how is the FEC data actually transported from the senders to receivers? In particular, if the FEC protects data transported by RTP, how is FEC incorporated into the RTP framework? In this section, we consider this problem for XOR-based parity in more detail. Extension of the concepts described here to Reed Solomon codes is under way [59].

2.6.1 Transport Requirements

A protocol for transport of FEC within the RTP framework poses the following problems:

- Each parity packet protects some set of the media packets. How is this information conveyed? Is it within the FEC packets, or signaled out of band? How flexible is it? What set of codes can be supported by it? Is it general purpose enough that the receiver does not need to know details of specific codes?
- RTP conveys the media, but also contains a number of header fields useful for playback of that media. This includes sequence numbers, timestamps, framing bits, and user identifiers. Is this information protected too? How can it be done efficiently?
- In multicast groups, it cannot be expected that all receivers will understand the FEC data. How can the FEC be transported in such a way that receivers which don't understand FEC can still get the media stream?
- RTP access links can use RTP compression [60] to reduce the overheads of the IP/UDP/RTP headers. This compression algorithm makes assumptions about the structure of the RTP header fields in order to work effectively. How can the FEC be transported within RTP without violating the assumptions made by RTP compressors?
- The overhead of FEC can be substantial. In order to reduce it, it is desirable to piggyback FEC information on subsequent media packets. How can this be accomplished within the RTP framework?

2.6.2 Previous Work

Work on transport of media-aware FEC began in 1995, and culminated in the RTP payload format for redundant codecs, standardized in RFC2198 [31]. Work continues on evolving this protocol based on implementation experience [61].

Media unaware FEC, using Reed Solomon codes in particular, has been used to support reliable multicast [39, 40, 62, 63]. However, these efforts have not focused on the transport component, and have not considered support of real time services. Nor have they considered the use of XOR-based parities.

The first work on transport of media-unaware FEC for real time services was in 1996, with an Internet draft by Budge et al., which has long since expired. They define a new RTP payload type which identifies the packet contents as XOR-based FEC-protected media. The RTP payload format in their proposal consists of two elements, the media-correction header and the payload. The media-correction header is 24 bits, and consists of three fields. The first is called the scheme, the second the mode, and the third, the length. The scheme identifies the particular error correction scheme in use. In particular, it defines the set of data packets over which the FEC is applied, and the order in which the packets (data and FEC) are sent. The mode identifies which packet in a group of data and FEC packets (typically called a block) this particular one corresponds to. For packets that contain just data (and not FEC), the length field contains the length of the payload. For packets which contain FEC, the length field contains the XOR of the length fields of the media packets which are covered by the FEC (by covered, we mean those packets used in the XOR that generates the FEC packet). Since packets must be padded out with zeroes (to be equal lengths) in order to perform the XOR operation, the length field allows recovery of the actual length of the pre-padded packets.

We find the work by Budge et al. to be lacking in several ways compared to the requirements:

- It does not indicate the media type of the actual data being protected. This is because the RTP PT field always indicates that the payload format is ”FEC-protected media”. Since many applications will need to change media payload types mid-stream (for example, sending Dual-Tone Multi-Frequency (DTMF) tones in-band), the presence of this field is important.

- The RTP timestamp field and marker bit are not covered by FEC. When a packet is lost and then reconstructed, the timestamp and marker bits are copied from the another packet. Correct recovery of these fields is important. Without it, playout buffer algorithms will obtain incorrect data on network latency, and silence durations may not be reconstructed properly.

- It defines four very specific schemes (one of which is no error correction), and assigns a value for the scheme field in the header to each. New schemes must be registered with IANA, the details written up, and receivers and senders alike must be upgraded to recognize and support them. This makes backwards compatibility difficult, requiring capabilities negotiation. It also means that transmitters are restricted to using the schemes defined thus far. The three non-null schemes defined by Budge use heavy forward error correction. These schemes are not appropriate for all loss conditions.
- The FEC is transmitted in the same RTP stream as the media. This means it will not work for multicast groups composed of FEC-aware and FEC-unaware receivers.

2.6.3 Our Approach

Our approach builds on Budge et al, but generalizes it substantially to meet the desired requirements. It is currently standardized in RFC 2733 [64].

2.6.3.1 Overview

First off, we specify that the FEC is sent in a separate RTP stream (that is, it is sent to a different port than the media it protects). We do allow, however, for the FEC to be sent piggybacked on the media RTP stream, using a redundant encoding [31]. The media packets are unaffected by the FEC, which is good for multicast. Rather than defining specific schemes ahead of time, we propose a generic approach. Each FEC packet contains a bitmask, called the offset mask, containing 24 bits. If bit i in the mask is set to 1, the media packet with sequence number $N + i$ was one of the media packets used to generate this FEC packet. N is called the sequence number base, and is sent in the FEC packet as well. The offset mask and payload type are sufficient to signal arbitrary parity-based forward error correction schemes with little overhead.

We also propose mechanisms to recover all of the RTP header fields. The value of certain fields in the RTP header of the FEC packet is set to the XOR of those fields in the RTP media packets being protected. This allows them to be recovered without any additional overhead. Unfortunately, all fields cannot be protected in this manner. The RTP version field must always be two to pass validity tests. The timestamp and sequence number must increase monotonically not to break RTP header compression, and the Synchronization Source (SSRC) field must be the same to determine which user is sending the data. Our approach takes all of these factors into account. Those fields which need to be recovered, but which cannot be set arbitrarily in the header, are included in the payload of the FEC packet.

2.6.3.2 Details

2.6.3.2.1 FEC Packet Structure An FEC packet is constructed by placing an FEC header and FEC payload in the RTP payload, as shown in Figure 2.31:

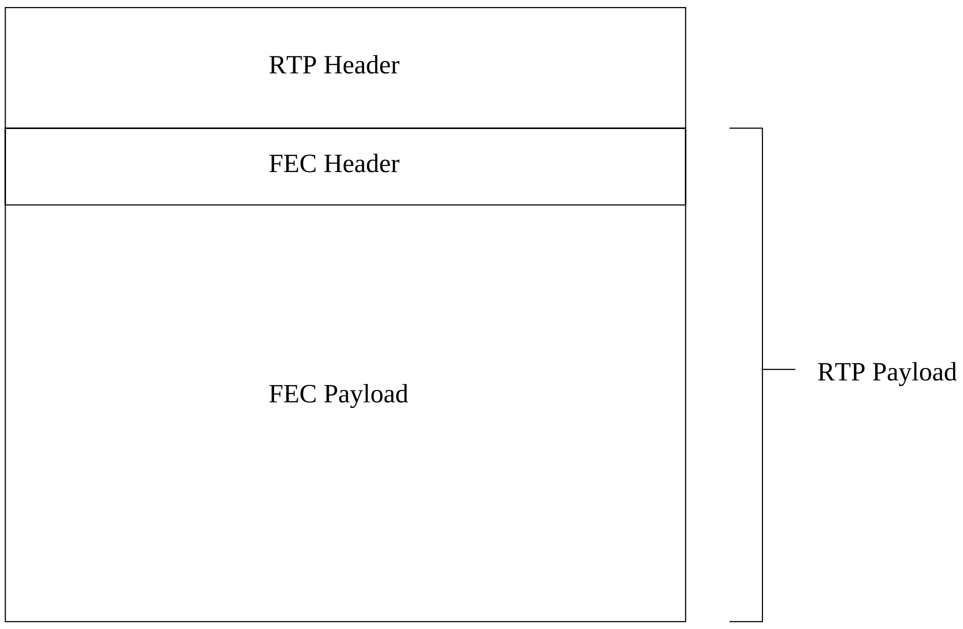


Figure 2.31: FEC packet structure

2.6.3.2.1.1 RTP Header of FEC Packets The version field is set to 2. The padding bit is computed via the protection operation, defined below. The extension bit is also computed via the protection operation. The SSRC value will generally be the same as the SSRC value of the media stream it protects. The CSRC Count (CC) and marker bits are computed via the protection operation. Neither the Contributing SSRC (CSRC) field or extension are present, independent of the values of the CC or Extension (X) bits.

The sequence number (SN) has the standard definition: it must be one higher than the sequence number in the previously transmitted FEC packet. The timestamp (TS) must be set to the value of the media RTP clock at the instant the FEC packet is transmitted. This results in the TS value in FEC packets to be monotonically increasing, independent of the FEC scheme.

The payload type for the FEC packet is determined through dynamic, out of band means. According to RFC 1889 [2], RTP participants which cannot recognize a payload type must discard it. This provides backwards compatibility. The FEC mechanisms can then be used in a multicast group with mixed FEC-capable and FEC-incapable receivers. Furthermore, if the FEC is sent as a separate RTP stream, using a different multicast group, the FEC-unaware hosts won't even be listening for that stream. This won't be the case if the FEC is piggy backed using RFC 2198 [31].

2.6.3.2.1.2 FEC Header This header is 12 bytes. The format of the header is shown in Figure 2.32, and consists of an SN base field, length recovery field, E field, PT recovery field, mask field and TS recovery field.

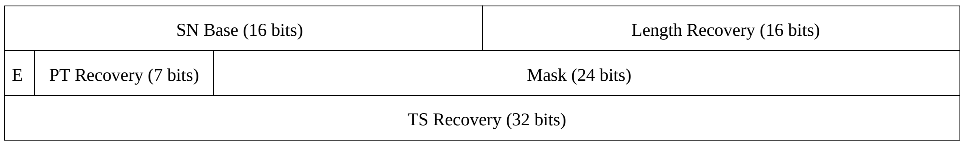


Figure 2.32: Parity header format

The length recovery field is used to determine the length of any recovered packets. It is computed via the protection operation applied to the unsigned 16 bit representation of the sums of the lengths (in bytes) of the media payload, CSRC list, extension and padding of media packets associated with this FEC packet (in other words, the CSRC list, extension, and padding, if present, are "counted" as part of the payload). This allows the FEC procedure to be applied even when the lengths of the media packets are not identical. It also allows for recovery of the CSRC list and extension, if present. For example, assume an FEC packet is being generated by XOR'ing two media packets together. The length of the two media packets are 3 (0b011) and 5 (0b101) bytes, respectively. The length recovery field is then encoded as 0b011 XOR 0b101 = 0b110.

The E bit indicates a header extension, allowing for future extensions of the format. Under normal operation, it is set to zero. Recent extensions for unequal error protection [65], where only portions of the packet are protected by FEC, have been defined as an extension to our format, and make use of the E bit.

The PT recovery field is obtained via the protection operation applied to the payload type values of the media packets associated with the FEC packet.

The mask field is 24 bits. If bit i in the mask is set to 1, then the media packet with sequence number $N + i$ is associated with this FEC packet, where N is the SN Base field in the FEC packet header. The least significant bit corresponds to $i = 0$, and the most significant to $i = 23$. The SN base field is set to the minimum sequence number of those media packets protected by FEC. This allows for the FEC operation to extend over any string of at most 24 packets.

The TS recovery field is computed via the protection operation applied to the timestamps of the media packets associated with this FEC packet. This allows the timestamp to be completely recovered.

The payload of the FEC packet is the protection operation applied to the concatenation of the CSRC list, RTP extension, media payload, and padding of the media packets associated with the FEC packet.

2.6.3.2.2 Protection Operation The protection operation involves concatenating specific fields from the RTP header of the media packet, concatenating the payload, padding with zeroes, and then computing the XOR across the resulting bit strings. The resulting bit string is used to generate the FEC packet.

For each media packet to be protected, a bit string is generated by concatenating the following fields together in the order specified:

- Padding Bit (1 bit)
- Extension Bit (1 bit)
- CC bits (4 bits)
- Marker bit (1 bit)
- Payload Type (7 bits)

- Timestamp (32 bits)
- Unsigned 16 bit representation of the sum of the lengths of the CSRC List, length of the padding, length of the extension, and length of the media packet (16 bits)
- If CC is nonzero, the CSRC List (variable length)
- If X is 1, the Header Extension (variable length)
- The payload (variable length)
- Padding, if present (variable length)

If the lengths of the bit strings are not equal, each bit string that is shorter than the length of the longest, is padded to the length of the longest. The parity operation is then applied across the bit strings. The result is the bit string used to build the FEC packet. Call this the FEC bit string.

The first (most significant) bit in the FEC bit string is written into the Padding Bit of the FEC packet. The second bit in the FEC bit string is written into the Extension bit of the FEC packet. The next four bits of the FEC bit string are written into the CC field of the FEC packet. The next bit of the FEC bit string is written into the marker bit of the FEC packet. The next 7 bits of the FEC bit string are written into the PT recovery field in the FEC packet header. The next 32 bits of the FEC bit string are written into the TS recovery field in the packet header. The next 16 bits are written into the length recovery field in the FEC packet header. The remaining bits are set to be the payload of the FEC packet.

2.6.3.2.3 Reconstruction Let T be the list of packets (FEC and media) which can be combined to recover some media packet x_i. The procedure is as follows:

1. For the media packets in T, compute the bit string as described in the protection operation of the previous section.
2. For the FEC packet in T, compute the bit string in the same fashion, except always set the CSRC list, extension, and padding to null.

3. If any of the bit strings generated from the media packets are shorter than the bit string generated from the FEC packet, pad them to be the same length as bit string generated from the FEC. The padding MUST be added at the end of the bit string, and MAY be of any value.

4. Perform the exclusive or (parity) operation across the bit strings, resulting in a recovery bit string.

5. Create a new packet with the standard 12 byte RTP header and no payload.

6. Set the version of the new packet to 2.

7. Set the Padding bit in the new packet to the first bit in the recovery bit string.

8. Set the Extension bit in the new packet to the second bit in the recovery bit string.

9. Set the CC field to the next four bits in the recovery bit string.

10. Set the marker bit in the new packet to the next bit in the recovery bit string.

11. Set the payload type in the new packet to the next 7 bits in the recovery bit string.

12. Set the SN field in the new packet to x_i.

13. Set the TS field in the new packet to the next 32 bits in the recovery bit string.

14. Take the next 16 bits of the recovery bit string. Whatever unsigned integer this represents, take that many bytes from the recovery bit string and append them to the new packet. This represents the CSRC list, extension, payload, and padding.

15. Set the SSRC of the new packet to the SSRC of the media stream it's protecting.

This procedure will completely recover both the header and payload of an RTP packet.

2.6.4 Determination of the Set of Packets

Section 2.6.3.2.3 describes a procedure for recovering a missing packet. The first step in this process is to obtain a list T of FEC and media packets which can be used to reconstruct some media packet x_i, where T is some subset of the FEC and media packets received so far. The important question is: how is this done?

The process is best understood by viewing each packet (both media and FEC) as a vector. This vector has its i^{th} component 1 if media packet x_i is present in the XOR used to generate the packet, 0 otherwise. A media packet x_i can be represented as a vector with only one component equal to 1, in the i^{th} position. We furthermore consider the vectors elements to be over GF2; this has the effect of turning addition into exclusive or, and multiplication into logical and. The task, then, is to determine some subset T of the vectors such that:

$$\sum_{\vec{v_z} \in T} \vec{v_z} = \vec{1_i}$$

where $\vec{1_i}$ is the vector with a 1 in the i^{th} position, 0 everywhere else, and $\vec{v_z}$ is a vector representing a packet. When this equality holds, we can recover packet x_i.

The algorithm we use for this process proceeds in two steps:

1. Reduction of the available set of packets (vectors)

2. Computation of T

2.6.4.1 Reduction

The reduction step is fairly straightforward. As time passes, packets continually arrive. Not all of these packets can be used to recover a specific media packet. The aim of the reduction step is to eliminate from consideration those packets which will not be useful in the recovery of a specific media packet x_i, and arrive at a minimal set Z of FEC and media packets.

The most obvious way to perform such a reduction is by timing them out. Our approach is to presume that since the application is real time IP telephony, the FEC does not cover a span of time greater than the maximum tolerable end-to-end delay, which is around 250 ms. Thus, if the aim is to recover packet x_i, any packets, FEC or data, which are associated with media packets played out more than 250 ms before x_i, are not considered when recovering x_i.

We can further reduce the set of packets useful for recovering packet x_i by observing that the FEC is sometimes performed in blocks. There is no overlap in protection from one block to the next. Any FEC or media packets in a different block from x_i are therefore not useful for recovering x_i. Since the packet format does not convey a block size, it must be computed at the receivers. As it turns out, this computation is just a simple connected graph computation.

We construct the graph G in the following manner:

- We initialize the graph with N nodes, each representing a media packet within the 250 ms window of x_i. It doesn't matter whether those media packets arrived or not.
- For each FEC packet that has arrived, and is under consideration, add a node l to the graph. If the FEC packet is an XOR over media packet x_j, add an edge between l the node representing media packet x_j.

From the construction, it is clear that if a path exists from the node representing x_i to some other node l, the packet associated with l might be used in the recovery of x_i. From the contra-positive, a packet cannot be used if there is no path to it from x_i. Thus, we compute the connected component of G which includes the node representing packet x_i. The minimal Z is then constructed by including in it only those FEC packets associated with the nodes in the connected component. We also include any media packets that have arrived, and are covered by the FEC packets included in Z.

2.6.4.2 Computing T

Computing T occurs by performing a search over the subsets of the packets (their vector equivalents, actually) in Z. We convert Z into a matrix M. This is done by taking the vectors associated with each packet in Z, padding them with zeroes to be all the same lengths, and making each vector a row in M. If Z contained N equations, the matrix M has N rows. The number of columns in M is the same as the total number of media packets covered by all the FEC packets in Z.

The aim of the algorithm is to compute the submatrix of M, formed by taking a subset of the rows, such that the sum of those vectors (in GF2) in the submatrix is equal to the packet to be recovered. As there are N rows, there are 2^N possible subsets. One or more subset may result in a combination of packets such that when XOR'ed together, the result is the missing packet

x_i. For the vast majority of codes, an actual full search over this space is acceptable. The delay restrictions of IP telephony limit the useful block sizes to 5 to 10 packets. A $O(2^N)$ search for such small N poses no computational problems at all.

However, to improve performance, we have developed an improved algorithm that is very effective on sparse FEC. By sparse, we mean that there are many packets that can be used (large N), but each FEC packet is a XOR over a small number of media packets (small compared to N). This implies that the matrix M is sparse.

The algorithm works by realizing that recovery of some packet x_i requires that at least one of the vectors in the set T has a one in the i^{th} position. More precisely, there must be an odd number of vectors in the set with a one in the i^{th} position. This additional constraint allows us to reduce the set of combinations which must be checked. The reduction improves with the sparseness of the matrix.

The algorithm is defined recursively through a function CS(S, val). This function takes a set of vectors S, and a desired vector $\vec{val}$. The function attempts to find the subset of S so that the XOR of the subset equals $\vec{val}$. The function returns nothing if no subset exists, otherwise, it returns the subset. Initially, the function is called with Z as the set S, and$\vec{val}$ as $\vec{1^i}$ to recover the i^{th} packet.

The baseline case of the function is simple. If S contains a single vector, and this vector equals $\vec{val}$, the function returns that vector. Otherwise, it returns nothing.

Otherwise, the function proceeds by finding the column j in the matrix M (where M is formed from the vectors in S) such that the j^{th} component of $\vec{val}$ is a one, and the number of ones in the j^{th} column of M are smaller than any other column k for which the k^{th} component of $\vec{val}$ is a one. In other words, for each component of$\vec{val}$ that is a one, the number of ones in the corresponding column of M is checked, and the column with the fewest ones is computed. This is done by a simple linear search over M.

Lets say there are m rows of M that have a one in the j^{th} position, where j was computed from the previous paragraph. Since the sum over T must have a one in the j^{th} position, there must be an odd number of those m vectors present in T. The total number of combinations of those m vectors with an odd number is exactly 2^{m-1}. Each of those combinations yields some subset

R of vectors. One of these subsets must be in T. So, we try each subset, and see if some further subset of the remaining $N - m$ vectors, when XOR'ed together with the vectors in the subset, yields the desired value.

To do this, Those vectors in R are summed, and the result summed with $\vec{val}$, yielding $\vec{val}'$. The set R is then removed from S, and CS is called, this time with the reduced set of vectors $S - R$ and $\vec{val}'$ as its arguments. If the function returns no set, the loop continues, otherwise the function returns the set that was returned, unioned with R.

From this description, it can be readily observed that the worst case compute time of this function, denoted f, is dependent on the number of vectors n in the set S:

$$f(n) = 2^{m-1} f(n-m)$$

The value of m depends on the matrix, and will be different on each iteration above. Assuming the most desirable case, $m = 1$ for each iteration, we have the very nice result that $f(n) = O(1)$. Assuming m is some constant for each iteration:

$$f(n) = 2^{\frac{m-1}{m}n}$$

The result is that the actual run time of the algorithm can vary between $O(1)$ and exponential. However, for sparse matrices, where m is often 1 or 2, the compute time is much reduced.

2.7 Conclusion and Future Work

In this chapter, we have considered the problem of high quality, end-to-end transport of voice for IP telephony. To this end, we took some measurements of Internet performance, and considered its ability to deliver high quality voice to the application. We concluded that the addition of a jitter buffer tends to increase the burstiness of loss seen by the application, but that the majority of burst lengths were still small. The implication is that FEC is still a viable solution for recovery.

After reviewing existing recovery approaches, we considered whether recovery of the speech content, or recovery of the lost state, of a low bitrate speech codec was more important. Our subjective and objective metrics indicate that recovery of state of the codec is more important.

This implies that use of traditional channel coding techniques, such as parity and Reed Solomon codes, are appropriate.

While we did not attempt to propose any new channel coding approaches for packet protection, we considered several important problems that arise when it is actually used. First, we observed that there is an important interaction between the use of FEC and adaptive jitter buffers. We have demonstrated that there is a need to couple both loss and delay into adaptive playout buffer algorithms when FEC is used. We have presented a number of novel algorithms to perform this coupling. One such algorithm is in fact a class of algorithms called *adaptively virtual* algorithms that extend existing algorithms. Our algorithms also allow us to control the target application loss probabilities. Simulations reveal that our algorithms are effective, and that the more complex algorithms we developed don't perform the adaptively virtual ones.

Finally, we considered the problem of generic transport of FEC. The problem has been solved for media-aware FEC. For media-unaware FEC, we propose a mechanism for transport within the RTP framework. Our protocol is very general, allowing any parity code to be used, without prior negotiation or understanding between sender and receivers.

Chapter 3

QoS Feedback

3.1 Introduction

The basic service provided by the Internet is known as *best-effort*. This means that the network makes no guarantees about loss rates, delays, jitter, or rate provided to packets. Furthermore, there is no admission control, which means that congestion can arise at any time and at any point in the network.

In Chapter 2, we examined prior work on the performance delivered by the Internet, and present our own experiments to assess it. The only consistent result is that end-to-end loss and delay conditions are highly variable. They depend on factors such as time of year, day of the week, time of day, and location. They also tend to vary quite a lot even over short intervals.

In such an environment, multimedia applications need to be *adaptive* [32]. This means that they must adapt to network conditions, varying parameters such as codec type, bitrates, quantization parameters, and redundancy to operating points which yield good performance.

In order to support these adaptive applications, feedback on the Quality of Service (QoS) delivered by the network is critical. The feedback provides data senders information from data receivers about the reception quality. Typically this would include loss characteristics, round trip delays, and jitter. Unfortunately, supporting feedback becomes more problematic in a multicast group. There can be a large number of senders and an extremely large number of receivers, depending on the application. For example, shuttle launches and other sessions on the MBone have seen many hundreds of receivers [66, 67]. As multicast eventually becomes the distribution medium for television content, one can envision concerts or talk shows on the Internet where the number of receivers can be in the thousands and millions.

The problem is further complicated by the fact that the group sizes and nature of the applications can be highly dynamic. For example, a conference call might start with a few people, but could easily expand into hundreds as participants and listeners are invited in. As another example, a conference which starts out as private (such as a debate between two senators) might at some point become public, and grow to include thousands of listeners.

Because the size of multiparty, multicast-based groups can vary greatly in size, and the size can change dynamically, we believe it critical to support this kind of wide range of applications with a single feedback mechanism which scales well from two person to two million person groups, and to groups where the membership is dynamic. Such scalability allows the boundaries between traditionally separate applications (such as TV and conferencing) to blur, enabling new applications and services.

In this chapter, we consider mechanisms for scaling QoS level feedback in these environments. As QoS feedback is provided by the Real Time Transport Protocol (RTP) [2], we start with those feedback mechanisms as a baseline. We first describe the feedback mechanism in RTP, and point out several difficulties encountered when scaling the algorithm. We then outline the ideal behavior we would like to see in a solution for IP telephony. To explore the solution space, we present a taxonomy of solutions that characterize the ways in which feedback can be provided. Using our taxonomy, we point out past solutions to the scaling problem, and discuss how they are not appropriate for the requirements that have been outlined. We then propose our solutions. Our solution consists of two algorithms; one is a set of algorithms generally dubbed *reconsideration*. We present, analyze, and simulate reconsideration, and demonstrate its effectiveness as a partial solution to the scaling problems. We then present a novel sampling algorithm as an orthogonal solution that further improves scalability.

3.2 Overview of RTP

RTP provides transport services for real time applications, including IP telephony. It consists of two parts, which are the transport part itself (RTP), and a control and feedback component, called the Real Time Control Protocol (RTCP). Both RTP and RTCP are engineered for multicast multimedia conferences.

RTP is generally used in conjunction with the User Datagram Protocol (UDP), but can make use of any packet-based lower-layer protocol. When a host wishes to send a media packet, it takes the media, formats it for packetization, adds any media-specific packet headers, prepends the RTP header, and places it in a lower-layer payload. It is then sent into the network, either to a multicast group or unicast to another participant.

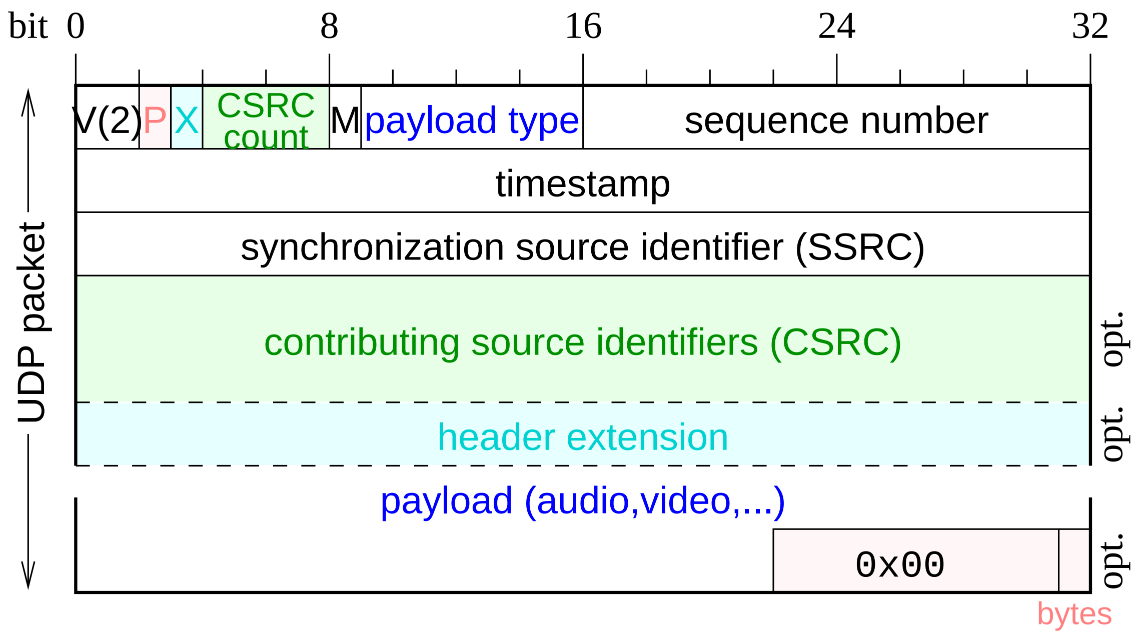


Figure 3.1: RTP fixed header format

The RTP header (Fig. 3.1) is 12 bytes long. The V field indicates the protocol version. The X flag signals the presence of a header extension between the fixed header and the payload. If the P bit is set, the payload is padded with zeroes to ensure proper alignment for encryption.

Participants (senders or listeners) in a multicast group are distinguished by a random 32-bit Synchronization Source (SSRC) identifier. Having an application-layer identifier allows to easily distinguish streams coming from the same translator or mixer and associate receiver reports with sources. In the rare event that two users happen to choose the same identifier, they redraw their SSRCs.

RTP supports the notion of media-dependent framing to assist in the reconstruction and playout process. The marker bit, M, provides information for this purpose. For audio, the first packet in a voice talkspurt can be scheduled for playout independently of those in the previous talkspurt. The bit is used in this case to indicate the first packet in a talkspurt. For video, a video frame can only be rendered when its last packet has arrived. Here, the marker bit is used to indicate the last packet in a video frame.

The payload type (PT) identifies the media encoding used in the packet. The sequence number (SN) increments sequentially from one packet to the next, and is used to detect losses and restore packet order. The timestamp (TS), incremented with the media sampling frequency, indicates when the media frame was generated.

RTP supports the notion of a *mixer*. A mixer is a device that takes RTP streams from many users, and combines them into a single stream containing a mix of the those streams. However, it is still desirable to indicate which participants have their audio mixed in a particular packet. This is the purpose of the Contributing SSRC or CSRC field. It contains a list of SSRC for those users mixed in a packet. The field is optional.

The payload itself may contain headers specific for the media.

3.2.1 RTCP: Control and Management

The Real Time Control Protocol (RTCP) is the companion control protocol for RTP. Media senders (sources) and receivers (sinks) periodically send RTCP packets to the same multicast group (but different ports) as is used to distribute RTP packets. Each RTCP packet contains a number of elements, usually a sender report (SR) or receiver report followed by source descriptions (SDES). Each serves a different function.

Sender Reports (SR) are generated by users who are also sending media (RTP sources). They describe the amount of data sent so far, as well as correlating the RTP sampling timestamp and absolute ("wall clock") time to allow synchronization between different media.

Receiver Reports (RR) are sent by RTP session participants which are receiving media (RTP sinks). Each such report contains one block for each RTP source in the group. Each block describes the instantaneous and cumulative loss rate and jitter from that source. The block also indicates the last timestamp and delay since receiving a sender report, allowing sources to estimate the round trip time to RTP sinks.

Source Descriptor (SDES) packets are used for session control. They contain the CNAME (Canonical Name), a globally unique identifier similar in format to an email address. The CNAME is used for resolving conflicts in the SSRC value and to associate different media streams generated by the same user. SDES packets also identify the participant through their name, email, and phone number. Client applications can display the name and email information in the user interface. This allows session participants to learn about the other participants in the session. It also allows them to obtain contact information (such as email and phone) to enable other forms of communication (such as initiation of a separate conference using SIP). This also makes it easier to contact a user should he, for example, have left his camera running.

If a user leaves an RTP session, they send a *BYE* RTCP message. Finally, *Application (APP)* elements can be used to add application-specific information to RTCP packets.

3.2.2 Scaling RTP

Scaling RTP to large groups requires scaling both RTP and RTCP. The use of silence suppression for audio enables scalability of the media transport. In normal conversations, only one or two people can be talking at a time, providing a natural scaling factor. For video, scalability can be provided by a similar means: only the person that is talking sends video updates. Alternatively, a mixer can be used, and the mixer distributes video streams only for those participants currently talking.

Scaling RTCP is more complex. Since it contains timely feedback information, it must be sent constantly. As such, the principal difficulty in achieving RTCP scalability to large group sizes is the rate of RTCP packet transmissions from a host. If each host sends packets at some fixed interval, the total packet rate sent to the multicast group increases linearly with the group size, N. This traffic would quickly congest the network, and be particularly problematic for hosts connected through low-speed dialup modems. To counter this, the RTP specification requires that end systems utilizing RTP listen to the multicast group, and count the number of distinct RTP participants which have sent an RTCP packet. This results in a group size estimate, L computed locally at each host. The interval between packet transmissions is then set to scale linearly with L. This has the effect of giving each group member (independent of group size) a fair share of some fixed RTCP packet rate in the multicast group.

Specifically, each user i in a multicast group using RTP maintains a single state variable, the learning curve, which we denote as $L(t)$. This variable represents the number of other users that have been heard from by i at time t. "Heard from" means validated, as specified in RTP[2]. A participant is validated once an RTCP packet from them is received, or once two RTP data packets have been received. The state is initialized to $L(0) = 1$ when the user i joins the group.

Each user multicasts RTCP reports periodically to the group. These reports contain information such as QoS feedback, user information, and messages for management of loosely controlled multimedia sessions. In order to avoid network congestion, the total amount of RTCP reports multicast to the group is set at 5% of the total multicast session bandwidth. The multicast session bandwidth is a fixed parameter, known to all participants. It is normally advertised through an announcement protocol, such as the Session Announcement Protocol (SAP) [68], or can be entered manually. The session bandwidth is chosen by the session creator. To meet this criteria, each user increases the period of their reporting as the group size increases. Specifically, let C be desired RTCP packet interarrival time between RTCP packets from any user, computed as the average RTCP packet size divided by 5% of the session bandwidth. Each user computes their *deterministic interval* as:

$$T_d = \max(T_{\min}, CL(t)) \tag{3.1}$$

where $T_{\min}$ is 2.5 s for the initial packet from the user, and 5 s for all other packets (this allows the initial packet to be sent quickly, to help speed up the validation process). The deterministic interval is the target average interval between RTCP transmissions from a specific user. To avoid synchronization, the actual interval is then computed as a random number uniformly distributed between 0.5 and 1.5 times T_d [69].

```
new_interval := C * current_group_size_estimate;
new_interval := max(new_interval, $T_{min}$);
new_interval := new_interval * random_factor;

send_packet();
schedule_timer(current_time + new_interval);
```

Figure 3.2: Current RTCP Algorithm

The algorithm for sending RTCP packets follows directly. Assume a user joins at time $t = 0$, and that C is smaller than T_{min}. The first packet from that user is scheduled at a time uniformly distributed between $1/2$ and $3/2$ of T_{min} (which is $2.5s$ for the first packet), putting the first packet transmission time between 1.25 and 3.75 seconds. We denote this time as t_0. All subsequent packets are sent at a time t_n equal to

$$t_n = t_{n-1} + R(\alpha)\max(5, CL(t_{n-1})), \tag{3.2}$$

where we have defined $R(\alpha)$ as a random variable uniformly distributed between $(1 - \alpha)$ and $(1 + \alpha)$, where α equals $1/2$ in the current algorithm (we generalize because α has a strong impact on transient behavior). A pseudo-code algorithm describing the behavior upon expiration of the interval timer is given in Figure 3.2.

3.3 Problems with RTCP Feedback

The above scaling mechanism works well for small to medium sized groups (up to perhaps a few hundred members). However, it suffers from problems when applied to larger groups, particularly ones whose group membership is dynamic. These problems can be classified as congestion, state storage, and delay. We note that the problems with RTP scalability are common to any application that uses distributed feedback. Similar problems have been observed for generating feedback for reliable multicast [70], and for counting participants in video conferencing sessions [71].

3.3.1 Congestion

In many cases, the access bandwidths for users will be small compared to link bandwidths within the network (28.8 kb/s modems, for example, can now handle multimedia RTP sessions when RTP header compression [60] is used). We also anticipate that many multicast RTP sessions will exhibit rapid increases in group membership at certain points in time. This can happen for a number of reasons. Many sessions have precise start times. Multimedia tools such as vat and vic can be automatically started by session directory tools, such as sd [72] and sdr [73], at the start of a session. Even without automation, users are likely to fire up their applications around the time the session is scheduled to begin. Such phenomena are common in current cable networks, where people change channels when shows begin and end. Studies have been performed to look at the group membership over time of some of the popular sessions on the MBone [66, 67]. These studies show exactly this kind of "step-join" behavior. The result of these step joins are inaccuracies in the group size estimates obtained by listening to the group. Each newly joined member believes that they are the only member, at least initially. They send RTCP packets at their fair share of the RTCP bandwidth (which each believes is all of it). Combined with slow access links, the result is a flood of RTCP reports, causing access link congestion and loss.

For example, consider an RTP session where the total RTCP rate is to be limited to 1 kb/s. If all RTCP packets are 1 kbit long, packets should be sent at a total rate of one per second. Under steady state conditions, if there are 100 group members, each member will send a packet once every 100 seconds, and everything works. However, if 100 group members all join the session at about the same time, each thinks they are initially the only group member. Each therefore sends packets at a rate of 1 per second, yielding an aggregate rate of 100 packets per second, or 100 kb/s, to the group.

Congestion can also occur when a large number of users all leave a group at nearly the same time. This is possible for the same reasons a step-join can occur. The RTP specification includes a BYE packet, sent when a member leaves a group. There are no controls or rules on sending these packets. So, should 500 participants leave the group at once, 500 BYE packets will be sent at once.

3.3.2 State Storage

In order to estimate the group size, hosts must listen to the multicast group and count the number of distinct end systems which send an RTCP packet. To make sure an end system is counted only once, its unique identifier (SSRC) must be stored. Clearly, this does not scale well to extremely large groups, which would require megabytes of memory just to track users. Alternate solutions must be found, particularly for set top boxes, where memory is limited.

3.3.3 Delay

As the group sizes grow, the time between RTCP reports from any one particular user becomes very large (in the example above, if there were 3000 group members, each would get to send an RTCP packet about once an hour). This interval may easily exceed the duration of group membership. This means that timely reporting of QoS problems from a specific user will not occur, and the value of the actual reports is reduced.

3.4 Requirements of a Solution for IP telephony

Clearly, a range of solutions are possible for the scaling problems of RTP (our classification in the next section helps to outline them). Such solutions vary from turning off RTCP completely, to sending it to a specific feedback point, to adding aggregation nodes that collect, summarize, and distribute the data. In order to choose the most effective solution, it is critical to first outline the requirements of the solution.

Our requirements are based on our broad definition of IP telephony in the introduction. In this definition, IP telephony covers all sorts of multimedia, real time communications between any number of participants, from two to two thousand. From this, several requirements emerge:

- Since IP telephony endpoints can range from dumb, Plain Old Telephone System (POTS) phone-like-appliances to special purpose computers, the solution for feedback must be simple to implement.

- Since the number of participants in an IP telephony conference can vary tremendously, the solution must be appropriate and reasonable over a wide range of sizes. It is particularly advantageous for the solution to work extremely well for small groups, work very well for medium sizes, and gradually and smoothly become less effective, but still be useful, for very large groups.

- Since conference sizes are dynamic, the solution cannot be based on a priori knowledge of group sizes.
- Ideally, the solution should be backwards compatible with the existing RTP algorithms.
- Since conferences can be between members that may be geographically close or sparsely distributed, the solution must work over a wide area network, and make no assumptions about the distribution of the session participants throughout the network.
- The mechanism must be *safe*. This means that it does not introduce significant new denial of service attacks, beyond those possible in general for any multicast application.

3.5 Taxonomizing the Solution Space

We begin by attempting to taxonomize the solution space for generating feedback for real time QoS. Our taxonomy helps us to quickly identify possible solutions, and then evaluate them based on our above requirements.

The mechanisms for feedback can be classified based on a tuple, with each entry in the tuple representing a different aspect of the feedback mechanism. Each component of the tuple represents an axis in the space of multicast feedback protocols. Each subsection below discusses one of the axes. Broadly speaking, our axes represent answers to the questions *who, what, where, when* and *how*.

3.5.1 Feedback Destination: Where

Where does the feedback eventually go? At one extreme, the entire set of participants in the group can receive the feedback. This is the current operating point for RTCP. At the other extreme, a single user (whether they are a member of the group or not) can receive the feedback. In the middle are cases where only partial subsets of users receive the feedback. Selection of the subset of recipients of feedback can be based on any number of criteria. For IP telephony, some sensible choices are (1) only those users who send media into the session, (2) feedback from a participant about reception from A is only sent to A (3) only those users connected via high speed access links, (4) a network operator.

So, in summary, the feedback destination can generally be one of one, subset, or all.

3.5.2 Feedback Mechanism: How

The feedback mechanism is closely related to, but not the same as, feedback destination. It refers to the “how” of the feedback distribution.

The feedback mechanism can be multicast, to the same group as the data was sent to. This is useful when the feedback destination is the entire group, and is how RTCP works now. Alternatively, the feedback mechanism can be multicast, but to a different group (perhaps one that is administratively scoped), or to the same group but with a different TTL. This would allow the feedback to target a subset of the receiver population or a set of network managers. Use of such a mechanism usually requires a mapping from the feedback to the specific group or scope. Such mappings can be dynamic (based on group size, for example), or static.

In other cases, the feedback can be sent to a unicast address. This address may be the address of the sender, or perhaps of some separate collection station (providing feedback to a specific user). When the address is a separate station, some means is needed to obtain this address. This can be done by placing the address in the RTP or RTCP packets. Or, the address can be determined through some separate state distribution protocol, such as SAP. For example, in the reliable multicast space, LGMP [74] distributes the identity and status information of active collection stations (Group Controllers) through a separate protocol. Receivers use this information for localized self-evaluation and for selection of an appropriate collection station.

Unicast can also be used as a means for feedback to all group members or a subset therein. A separate station (or set of stations), can act as a collection point. Receivers unicast the feedback to the station, and the station unicasts it back to the specified set of receivers. This, of course, requires the station to have knowledge of group participants.

Hybrid schemes exist as well. In particular, unicast can be used to send the feedback to a collection station, which then distributes it to the receivers using multicast.

In summary, the set of possible feedback mechanisms includes multicast (same group), multicast (different group), unicast, or hybrid.

3.5.3 Feedback Source: Who

Who sends feedback? There is a wide range of possibilities here. All receivers may be required to send feedback. This is how the current RTCP implementation works. In addition, when intermediate servers provide aggregation of information, all receivers may be required to provide their feedback to an aggregation station.

At the other extreme is feedback from a single member. This type of feedback is often used in conjunction with redundant content (see Section 3.5.4).

In between are cases where a subset provides feedback. This subset can be determined in many ways:

statistical sampling: In this case, each receiver randomly decides to send feedback. The probability that they send feedback can be set by an external entity [71], or be computed in a distributed fashion based on some other parameters. Statistical sampling is useful in large groups where feedback from all participants is not needed, or where the bandwidth for feedback from all participants is too large. Note that very large feedback intervals, such that a participant may never get around to sending feedback, effectively results in statistical sampling.

parameterized: In this case, the receivers send feedback if they meet some kind of criteria. For example, all receivers with loss rates exceeding 5% might send feedback, while all others do not.

explicit selection: In this case, some external third party may explicitly select the group of receivers who should send feedback.

3.5.4 Feedback Content: What

What information does the feedback contain? We refer not to the actual parameters, but to the high-level semantics it conveys.

The feedback content can be one of the following:

individualized: In this case, the content of the feedback is the reception quality, loss reports, or other feedback as reported by individual stations. This is the case in RTCP right now.

redundant: In this case, the content of the feedback is the reception quality, loss reports, or other feedback as reported by individual stations. However, the nature of the feedback is that it is identical across those receivers providing the feedback. As a result, it may only be necessary for a single receiver to actually send the feedback. An example of this case are reliable multicast protocols where the feedback is the NAK for a specific packet. Each participant that has not received the packet sends an identical NAK. The information is redundant, since only one of these is needed if the retransmission is multicast. This case also works for feedback in RTP; those receivers with similar QoS performance need not all send reports.

aggregated: The feedback delivered to the receivers is aggregate information. It represents a summary of the information which may have been provided by the data receivers. Use of aggregated feedback usually implies a unicast or hybrid delivery mechanism, with intermediate aggregating nodes. Placement, configuration, and communications between these nodes can be complex. As an alternate, group participants can act as aggregators. However, this still requires election procedures to choose the aggregators. Furthermore, the feedback from a set to the aggregator must be limited, and therefore cannot use the same multicast group and scope as the original request.

3.5.5 Congestion Control: When

Congestion control is an integral component of multicast feedback. Since the number of receivers reporting feedback can be large, the potential for congestion in the network and at specific hosts is large. In this section, we consider some of the congestion control mechanisms which have been applied to help alleviate this problem.

One approach is to use periodic feedback. Periodic feedback is useful when the information it contains is timely and needed continuously, and must be transmitted often. For scalability purposes, when the feedback is delivered to the entire receiver population, the interval is made to scale with the number of participants sending feedback. This requires each participant to hear the feedback, in order to obtain a group estimate. Or, a single entity can maintain group size and distribute it to the other participants. This is the mechanism used in RTP. When periodic feedback is sent to specific collection stations or a subset of the population, scaling of the period may not be needed.

In other cases, the feedback is in response to a particular event, which we call a poll. A poll might be an explicit request for feedback. In these cases, periodic feedback is not appropriate. To provide some amount of congestion control, receivers who wish to send feedback do not do so immediately after the generating event. Rather, they wait some amount of time randomly selected, and then send feedback. When a participant sees a feedback from another participant, it cancels transmission of its own packet.

Another mechanism for congestion control, similar to the periodic approach, is to use a token passed among the receivers. Only receivers with the token can send feedback. The mechanism requires the establishment of a virtual ring around the receivers, and works well for small to medium sized groups [75].

3.6 Solution Space

With our taxonomy in hand, we are now in a position to examine the potential solutions to the feedback problem. We begin by exploring solutions that have been proposed in the literature to date, and classifying where they fit in the taxonomy. From this point, we can examine other points in the taxonomy to determine what other solutions might be appropriate.

3.6.1 Existing Solutions

A number of solutions for scaling feedback have been proposed, both in general and in the context of RTP. These solutions include polling and summarization as general solutions, and for RTCP specifically, they include adding summarizers to aggregate reports from regional or administrative areas, turning off RTCP reporting entirely, TTL scoping, using separate multicast groups for RTCP reports, and using the BYE message for QoS reporting, among others [76].

The subsections which follow discuss each of these alternatives in turn.

3.6.1.1 Summarizers

The idea of summarizers was proposed by Aboba [76]. Research by El-Marakby and Hutchison later fleshed out the details [77]. The basic idea is to develop a tree-based hierarchy of report summarizers. At the lowest level of the tree, session participants send receiver reports. A node, acting as an aggregator, collects these reports, and summarizes them. These summaries are then passed up towards the next highest node in the tree, until finally they reach the sender or some other appropriate feedback point. The summarization is most useful when the nodes at one level of a sub-tree see similar network performance. El-Marakby uses network hop counts to a summarizer, measured through TTLs, to group hosts together in the tree. She also proposes a dynamic scheme for building the tree.

The summarization approach introduces many issues that need to be resolved. First off, it is still necessary to rate control the feedback. How is this done? At each level of the tree, the rate of feedback to its parent can be constant, can scale inversely to the number of children of the same parent, or can scale with the number of end nodes in the sub-tree rooted in the summarizer. In addition, these summarizers must be elected, and re-elected in the case of changes in session dynamics.

The summarizer approach appears to be quite attractive; the hierarchical nature of the system allows for scalability, less information would need to be maintained at any given node, and convergence times would be reduced [77]. However, the scheme has a number of drawbacks for IP telephony:

1. It requires centralized processing in the summarizer. Therefore, protocols for discovering and electing such summarizers will be required, adding additional complexity. Administrative issues are introduced. Who runs these summarizers? Is there a fee? Furthermore, new summary messages must be defined, and new behaviors for the summarizers standardized.

2. It may have no impact when group members are sparsely distributed, since the tree that gets built can potentially be very deep but not very wide.

3. It also has no impact when group members are very densely distributed within a small network, since the tree will end up with one node at the root, and all others as its children. This means there will need to be significant rate controls to control feedback from the children to the single summarizer.

4. Convergence times are only reduced if the packet sizes for the summarizers grow smaller than the group sizes. However, if we still desire SDES information, summarization cannot occur, and we will still have to wait a long time before learning the names of all multicast group members.

5. It is not backwards compatible with existing RTP deployments.

6. It is overkill for small groups, which is a common case for IP telephony.

7. Its operation is unclear in the case of feedback for multiple senders.

Due to its complexity, and the fact that other congestion control mechanisms may still be needed, we believe that summarizers are not a ubiquitously useful solution for IP telephony. In applications such as broadcast-only media, they are more appropriate.

Table 3.1 indicates how the summarizer approach fits into our taxonomy.

3.6.1.2 Polling

A second approach, applied in the context of video session group size estimation [71] is to use a form of polling. In this mechanism, a single polling station sends a poll request to the set of receivers in the session. The poll contains information that allows the poller to control the

Axis	Value
Where	A single feedback collection point
How	Hybrid unicast and multicast
Who	All
What	Aggregated information
When	Not addressed by El-Marakby [77]; there are many possibilities

Table 3.1: Operating Point of Summarizers in the Feedback Taxonomy

Axis	Value
Where	A single feedback collection point
How	Multicast
Who	Sampled subset
What	Individual information
When	Based on poll interval of poller

Table 3.2: Operating Point of Polling in the Feedback Taxonomy

fraction of receivers that respond. The responses can contain feedback about reception quality. By adjusting the parameters in the poll, the amount of bandwidth used can be controlled.

This general feedback mechanism is not appropriate for general purpose IP telephony. In conferences with many senders, each of which needs feedback, the result are many independent polls, which is wasteful. Furthermore, for large conferences, it introduces denial of service attacks. Pollers can set the poll parameters inappropriately and cause network congestion.

Table 3.2 specifies where polling fits in our taxonomy.

3.6.1.3 Separate Multicast Groups

In the specific case of RTCP, an attractive alternative is to use a separate multicast group for receiver reports, to which only senders subscribe. This means that receivers will not see each others receiver reports, and the amount of network traffic is reduced. Since receivers no longer hear from each other, they will have to receive group size estimates from the senders (who presumably will join the receiver report group). This will introduce additional propagation, queuing and processing delays between the transmission of a receiver report and its increasing the group

Axis	Value
Where	A single feedback collection point (the sender)
How	Multicast
Who	All
What	Individual information
When	Based on period sent out by sender

Table 3.3: Operating Point of Separate Multicast Groups in the Feedback Taxonomy

size as perceived by the receivers. Analysis in future sections will demonstrate that congestion is strongly dependent on network delays. Thus, this approach worsens congestion for the senders (of course, the receivers will be spared).

Additional protocol structures will need to be introduced to allow the senders to inform the receivers of the group size estimates. This means the approach is not backwards compatible. It also introduces potential denial of service attacks, similar to the ones introduced by the polling approach above. A malicious user can join the receiver report multicast group and send out false, low estimates of group sizes, causing congestion for itself, but for any other receiver report group members as well. This means that authentication mechanisms will need to be added.

Table 3.3 specifies the operating point of separate multicast groups in our taxonomy.

3.6.1.4 Event-Based Reporting

This approach is a variant on the current RTCP algorithm. Feedback is sent to the group only when some specific event occurs, rather than periodically. These events can include (1) QoS degradation beyond some threshold, or (2) departure or joining of a group by a participant. In theory, this would allow feedback to occur only when needed, rather than periodically. This can help alleviate the congestion problem. In practice, the approach has problems:

1. If joins or leaves are deemed as events, then we still have the congestion problem with step joins. Since RTCP is also used for conveying identification information (name, email, phone number), it is desirable to send data on join events, at least.

2. Even if receiver reports are only sent in the event of a QoS problem, congestion can still

Axis	Value
Where	All
How	Multicast
Who	All
What	Individual information
When	When a specific event occurs

Table 3.4: Operating Point of Event Based Reporting in the Feedback Taxonomy

occur. If the congestion is being caused at a link near the root of the multicast tree, nearly all receivers will suffer loss. As a result, all users will send QoS reports back to the source. This, of course, only exacerbates whatever problem may have been causing the QoS degradation in the first place.

Table 3.4 conveys the operating point for event-based reporting in our taxonomy.

3.6.2 Additional Approaches

Based on our taxonomy, we can quickly consider additional ranges of solutions.

Choosing alternate choices for the “who” axis can introduce mechanisms that reduce the congestion problem by reducing the number of participants that send feedback. The polling and event-based reporting approaches take this approach. The general difficulty with this direction is that for small groups, it is extremely desirable to have feedback from all participants. This is because RTCP contains important participant identification information. This would seem to argue for a solution where every member sends when the group membership is small, but a select subset send as the group size increases. This solution works well when there is a centralized agent coordinating the process. However, in a distributed system, it is not possible. In the distributed system, each participant maintains its own group size estimate. The estimate is obtained by observing packets from other users. If only a subset of users send packets when the group size goes above some threshold, the group size estimates computed by each user will decrease (new users will also perceive the wrong group size), and each user once more begins to send feedback. The result is an oscillatory behavior that is undesirable. Since we believe it is also desirable to maintain the distributed nature of RTCP, effectively adjusting the “who” component is problematic.

Alternate choices for the “where” axis can also reduce bandwidth usage. However, they cannot avoid fundamental congestion problems. If all participants send feedback at the same time, but it’s to a single location, congestion will still occur. Adjusting the “where” just means that the congestion is in one place rather than many.

Changing the values on the “how” axes can’t address fundamental congestion problems. These depend on the operating points of the “who”, “when”, and “where” axes. Using differing “how” values can exacerbate the problem; i.e., using multi-unicast when multicast is more efficient.

The axes that remain are “what” and “when”. By adjusting the “what” towards summary information rather than individual information, congestion and latency issues can be alleviated. However, as we have discussed above, these solutions require significant additional infrastructure that will often be unjustified.

Our approach, as a result, is to consider solutions along the “when” axis. These, fundamentally, reduce congestion by sending data less frequently. This improvement comes at the cost of potentially increased latency. However, it is our opinion that reducing congestion, and maintaining the simple, distributed nature of the feedback, is more important than reducing feedback latency.

3.7 Reconsideration Algorithm

Our approach for solving the congestion problem is to work within the existing RTP framework. We do not define any new functional elements or messaging, nor do we change the way in which the basic RTCP mechanism works. Rather, we adjust the algorithm used to transmit RTCP packets, in order to reduce congestion. This approach has the advantage of being maximally compatible with existing implementations. As with the original RTCP algorithm, it is fully distributed. The feedback works very well for small groups, and with our algorithm, avoids the congestion problem with large groups.

Our algorithm is called *reconsideration* [78]. The effect of the algorithm is to reduce the initial flood of packets which occur when a number of users simultaneously join the group. It therefore addresses the congestion problem, but not the state storage or delay problems. This algorithm operates in two modes, conditional and unconditional. We first discuss conditional reconsideration.

At time t_n, as defined above in Equation 3.2, instead of sending the packet, the user checks if the group size estimate $L(t)$ has changed since t_{n-1}. If it has, the user *reconsiders*. This means that the user recomputes the RTCP interval (including the randomization factor) based on the current state (call this new interval $T' = R(\alpha)\max(T_{\min}, CL(t_n)),$), and adds it to t_{n-1}. If the result is a time before the current time t_n, the packet is sent, else it is rescheduled for $t_{n-1} + T'$. In other words, the state at time t_n gives a user potentially new information about the group size, compared to the state at time t_{n-1}. Therefore, it redoes the interval computation that was done previously at time t_{n-1}, but using the new state. If the resulting interval would have caused the packet to be scheduled before the current time, it knows that its interval estimate was not too low. If, however, the recomputation pushes the timer off into the future, it knows that its initial timer estimate was computed incorrectly, and it delays transmission based on its new timer. A pseudo-code specification of the algorithm is given in Figure 3.3.

Intuitively, this mechanism should help alleviate congestion by restricting the transmission of packets during the convergence periods, where the perceived group sizes $L(t)$ are rapidly increasing.

In unconditional reconsideration, the user reconsiders independently of whether the number of perceived users has changed since the last report time. A pseudo-code description of the algorithm is given in Figure 3.4. The RTCP interval is always recomputed, added to the last transmission time t_{n-1}, and the packet is only sent if the resulting time is before the current time. Clearly, when the group sizes are increasing, this algorithm behaves identically to conditional reconsideration. However, its behavior differs in two respects. First, consider the case where group size estimates have converged, and are no longer changing. In conditional reconsideration, no timer recomputation is done. But for unconditional, it is still redone. Since group sizes have not changed, the deterministic part of the interval remains the same. However, the random factor is redrawn each time. This means that packets will be transmitted when the recomputed random

```
new_interval := C * current_group_size_estimate;
new_interval := max(new_interval, $T_{min}$);
new_interval := new_interval * random_factor;

if ((last_transmission + new_interval < current_time) $\|$
  (current_group_size_estimate $\leq$ previous_group_size_estimate)) {
    send_packet();
    new_interval := C * current_group_size_estimate;
    new_interval := max(new_interval, $T_{min}$);
    new_interval := new_interval * random_factor;
    schedule_timer(current_time + new_interval);
    last_transmission := current_time;
    previous_group_size_estimate := current_group_size_estimate;
}
else {
    schedule_timer(last_transmission + new_interval);
    previous_group_size_estimate := current_group_size_estimate;
}
```

Figure 3.3: Conditional Reconsideration

factor is smaller than the previous factor, and packets will be delayed when the recomputed random factor is greater than the previous one. Note that since the random factor is of finite extent (between $1/2$ and $3/2$), packets are guaranteed to eventually be sent. However, the result is an average increase in the interval between RTCP packets.

The behavior of unconditional reconsideration differs during the initial transient as well. Consider N users who simultaneously join the group at time 0. They all schedule their first RTCP packets to be sent between $t = 1.25$ and $t = 3.75$. The users whose packets were scheduled earliest (at a time a little bit after $t = 1.25$) will not reconsider with conditional reconsideration, and will always send their packets. This is because no one else has sent any packets yet, and thus they have not perceived the group size to have changed. In fact, because of network delays, many users may send packets without reconsidering. Once the first transmitted packet has reached the end systems, conditional reconsideration “kicks in”, since users will perceive a change in group size only then. With unconditional reconsideration, those first few users do not wait for

```
new_interval = C * current_group_size_estimate;
new_interval = max(new_interval, $T_{min}$);
new_interval = new_interval * random_factor;

if (last_transmission + new_interval $<$ current_time) {
    send_packet;
    new_interval = C * current_group_size_estimate;
    new_interval = max(new_interval, $T_{min}$);
    new_interval = new_interval * random_factor;
    schedule_timer(current_time + new_interval);
    last_transmission = current_time;
}
else {
    schedule_timer(last_transmission + new_interval);
}
```

Figure 3.4: Unconditional Reconsideration

the first packet to arrive before using the reconsideration algorithm. They will all recompute the timer. Obviously, the group size estimate hasn't changed, but the random variable will be redrawn. For the first few users, the random factor was initially extremely small (that's why they are the first few users to send). In all likelihood, when the factor is redrawn, it will be larger than the initial factor, and thus the resulting interval will be larger. This will delay transmission of RTCP packets for those users. As time goes on, it becomes less likely than the new random factor will be greater than the initial one. However, by then, any RTCP packets which may have been sent will begin to arrive, increasing the group size estimates for each user. In this fashion, unconditional reconsideration alleviates the initial spike of packets which are present in conditional reconsideration. These arguments are all quantified in later sections.

It is worth noting that for both conditional and unconditional reconsideration, the algorithms recompute the interval yet another time once a packet is sent, in order to schedule the next packet. The reason for this is a subtle effect on the random factor. In both algorithms, the packets are sent conditionally based on the interval being below some threshold. If the packets are sent, it implies a conditioning on the interval and on the random factor. As a result, the value of the first random factor within the code block which sends the packets is, on average, smaller than its unconditioned value. In order to maintain the statistical properties of the random value, we recompute it again.

Both modes of the algorithm are advantageous in that they do not require any modifications to the current RTCP protocol structure. In fact, they operate properly even when only a subset of the multicast group utilizes them. As more and more members of a group use the algorithm, the amount of congestion is lessened in proportion. This leaves open a smooth migration path which is absent for most of the other proposed solutions.

3.7.1 Ideal Behavior

Before considering a performance analysis of our algorithms, we need to define the “ideal” behavior with which to measure performance. The flood of packets caused by the current RTCP algorithm with a step join has both good and bad consequences. Clearly, the congestion which results is not desirable. However, the flood allows the end systems to very rapidly learn about the group sizes and group membership, which is desirable. There is a fundamental tradeoff between the convergence time (i.e., the time until the observed group size $L(t)$ equals the actual group size) and the bandwidth used to achieve convergence.

Our approach is to define the ideal behavior as the one where feedback into the group never exceeds its specified threshold (5% for RTCP). This implies that convergence times will grow as the group sizes grow. However, it is the most social solution, in the sense that it will never congest the network, no matter how large the group sizes become. If we define the ideal behavior as convergence within any amount of time that grows less than linearly with the group size, the result is a protocol that does not scale and can eventually result in congestion.

We also consider congestion avoidance to be more important because we expect many users to be connected via low-speed dialup lines. In that case, bandwidth is at a premium, and it is in the self-interest of users to make the best use of it. Most users probably consider RTCP feedback much less important than the video or audio data itself, and therefore it is important to keep the feedback below the required 5%.

We add another dimension to the ideal behavior by defining a quantity called the *effi-*

ciency, $e(t)$. This quantity represents, for a step join, the size of the learning curve $L(t)$ divided by the total number of packets injected into the multicast group. It captures both loss (which reduces efficiency), and the capability of the algorithms we propose to make the best use of the bandwidth available to them by first transmitting packets from users who have not yet been heard from. The efficiency only makes sense for the convergence period; once converged, the learning curve will stop at the group size. From then on, packets will continue to be sent, so the efficiency as defined would decrease.

With this definition, we now state the desired ideal behavior:

1. The learning curve $L(t)$ grows linearly at a rate of $1/C$ users per second, until it reaches the group size N, at which point it becomes flat, and remains at N.

2. The efficiency $e(t)$ is equal to one during the convergence period.

3. The bandwidth used by all feedback is always equal to $1/C$ packets per second during the convergence period.

The third item is a mathematical consequence of the first two, but we state it for emphasis.

Clearly, as group sizes grow, the longer convergence times and larger RTCP intervals may make the feedback less useful for some applications. In cases where the group sizes are known to be large a priori, the polling solutions described below may be more desirable.

3.7.2 Simulations

We ran a number of simulations to examine the performance of the reconsideration algorithms.

The model implicit in all of our simulations and analysis is depicted in Figure 3.5. All users are identical in their view of the network, and so we focus on the state evolution as seen by any particular user. Each user is connected to the network via an access link of 28.8 kb/s downstream (i.e., from the network to the user). We assume upstream links are infinitely fast. In the downstream direction, congestion occurs because the RTCP reports from all of the other users are being received. In the upstream direction, each user sends only his own report; thus congestion never occurs in that direction. Therefore, since the upstream link is not critical in

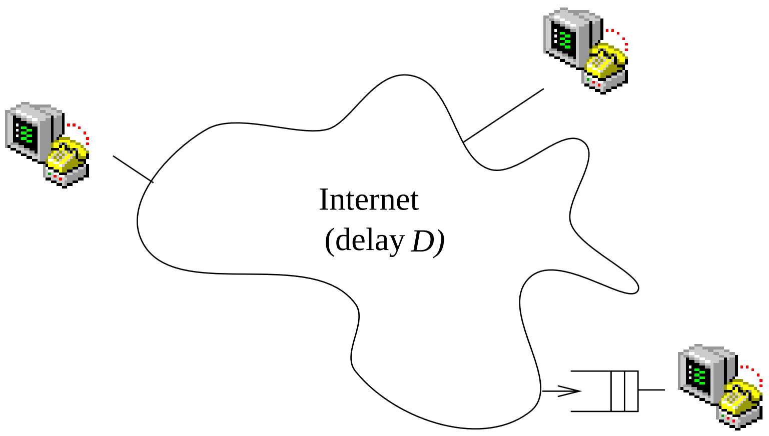


Figure 3.5: Network Model

the system behavior, we assume it is infinitely fast in order to simplify analysis. It should be noted that the asymmetry in access technologies such as ADSL are therefore beneficial for this particular problem; they allow more downstream bandwidth to relieve the congestion.

Multicast join latencies are ignored; this is reasonable in protocols such as DVMRP [79] since initial packets are flooded. Even in protocols like PIM [80], where explicit joins are required, the model still holds. This is because the multicast join will occur the instant the user requests to the join the group. However, the RTCP packets will not be sent for at least 1.25 s after such a join. This should be ample time to establish the multicast distribution tree, therefore providing full connectivity by the time the user sends a packet.

We assume that the network introduces a delay of D seconds, where D is uniformly distributed between 0 and 600 ms, with an average of 300 ms. We use the uniform distribution because it simplifies our analysis, and because the results from Chapter 2 showed that RTT distributions were approximately uniform over the middle parts of the delay range. The average value represents the upper ranges of the delays we observed in our traces. However, the network delays have a strong impact on performance of the algorithm, so we felt it important to be conservative.

Each user has buffering of 100 kB on the downstream access link. We assume all RTCP packets are 128 bytes in size, and that the session bandwidth is 28.8 kb/s.

Figure 3.6 and Figure 3.7 depict state evolution for a single user when 10,000 users simultaneously join a multicast group at $t = 0$. The figures depict the system with no reconsideration

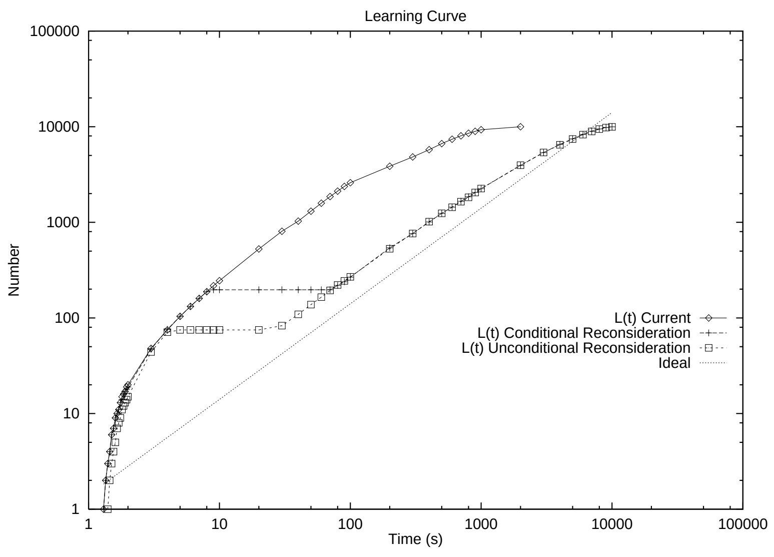


Figure 3.6: Learning curve, step join with N=10,000

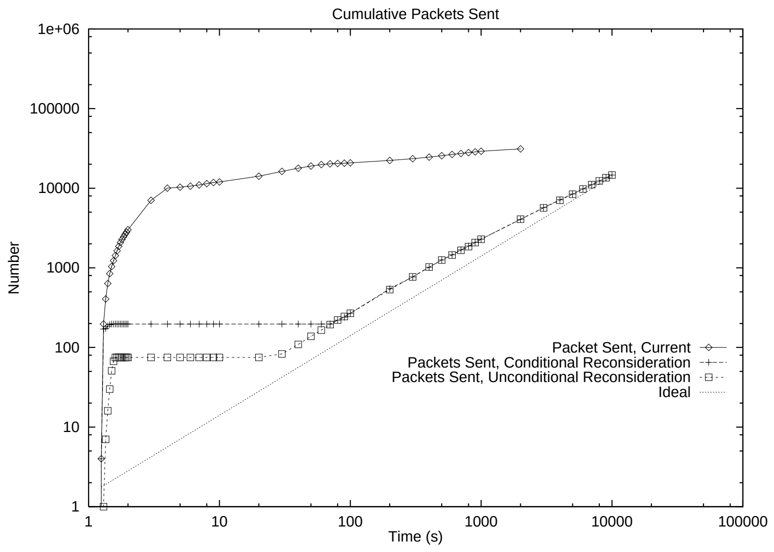


Figure 3.7: Total packets sent, step join with N=10,000

(the current specification), conditional reconsideration, unconditional reconsideration, and the ideal behavior. The graphs are plotted on a log-log scale to emphasize the beginning and complete evolution of the system. Figure 3.6 depicts the learning curve, and Figure 3.7 shows the total number of packets sent to the multicast group at time t. Note that this quantity is the integral of $r(t)$, given that $r(t)$ is the total packet transmission rate into the multicast group. Note the burst of packets sent in the beginning by the current algorithm. Exactly 10,000 packets are sent out in a 2.5 s interval. This is almost 3000 *times* the desired RTCP packet rate. However, this burst is reduced *substantially* by the reconsideration mechanisms. Conditional reconsideration causes only 197 packets to be sent over a 210 ms interval, and unconditional reconsideration causes merely 75 packets to be sent over a 327 ms interval. We also observed similar improvements in the relative number of packets transmitted, over a variety of different link speeds, delays, group sizes, and delay distributions.

We noted that the startup burst with reconsideration was particularly disturbing when network delays were deterministic instead of uniformly distributed. This is demonstrated in Figure 3.8, which looks at the cumulative number of packets sent when $10,000$ users simultaneously join at $t = 0$. In all cases, the mean network delay was 300 ms, but the distribution varies. Exponentially distributed delays exhibited nearly identical performance to a uniform distribution. Later sections will demonstrate that the spike is dependent on the amount of time until the first packet arrives. As the number of users in the step join becomes large, the number of users who send their packets within the first ϵ seconds after $t = 1.25$ grows large for any ϵ. Consider an ϵ much smaller than typical network delays, say 10 μs. As far as computing arrival times at end stations, these packets can be treated as though they were all sent at the same time. The amount of time until the first of these packets arrives at any end system is thus the *minimum* network delay experienced by all of those packets. If the network delays are exponential, the expected minimum of N exponential random variables goes to zero as N grows. The same is also true for a uniform random variable. For a deterministic variable, this is not the case; the minimum is always the same. Therefore, the performance is worse for network delays which are fixed.

We have also observed that the reconsideration mechanisms cause a complete pause in packet transmissions after the initial spike. This pause (which we call the “plateau effect”) lasts for a time proportional to the number of packets in the spike. This has both positive and negative implications. On the plus side, it gives network buffers time to clear. However, it also causes the send rate to deviate from our desired fixed $1/C$ packets per second. The phenomenon occurs because the spike of packets in the beginning causes the system to reconsider, and not send, all packets after the spike. A more detailed explanation of the phenomenon is given in Section

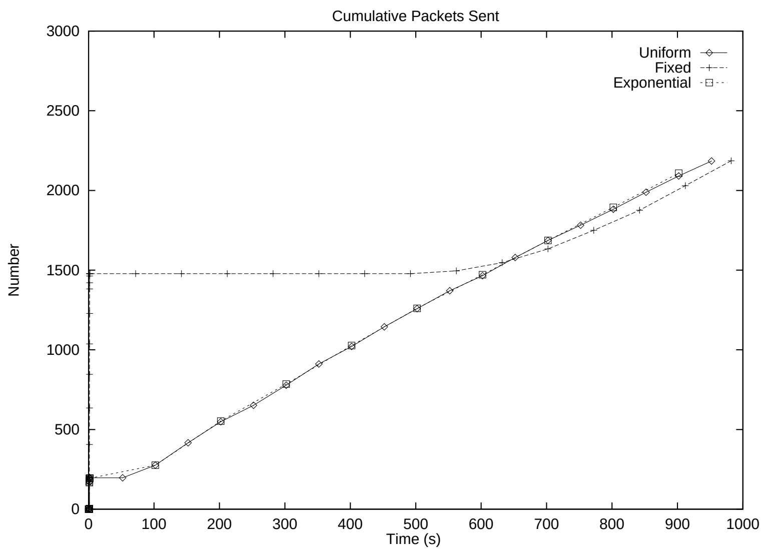


Figure 3.8: Effect of delay distribution on transient for conditional reconsideration

3.7.3.2. However, after the spike and plateau, the packet rate behaves fairly well, sending packets at a nearly constant rate.

We also ran simulations to observe performance in linear joins, where users join the group gradually at a rate of γ users per second. The results are shown in Figure 3.9 and Figure 3.10. Both plots depict the cumulative number of packets sent by all users. The simulation parameters are identical to the above cases, except network delays are deterministic at 300 ms. The first plot depicts conditional reconsideration, and the second, unconditional. In all cases, 2500 users join the system, but the rate that they do so is varied. Both plots depict the step join, and joins at a rate of 5000, 2500, and 500 users per second. The plots indicate that linear joins quickly eliminate the initial transient of packets and the plateau period, with the reduction being better for unconditional reconsideration.

Section 3.7.3.3 analyzes how the behavior of reconsideration changes under linear joins. Our analysis has shown that as soon as the join rate γ drops below the group size divided by the minimum RTCP interval, the initial bursts in the reconsideration algorithms begin to disappear, whereas they remain for the current specification. All other aspects of the system performance (including long term growth of $L(t)$) are identical to the step-join case.

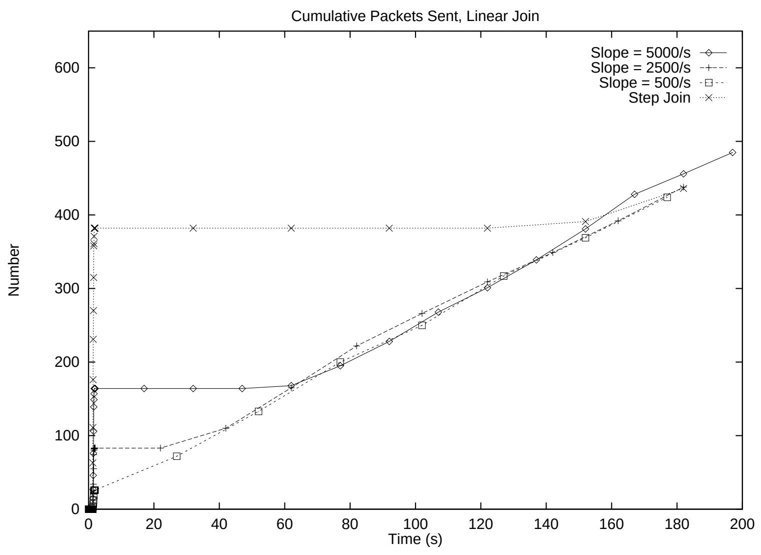


Figure 3.9: Linear joins: conditional reconsideration

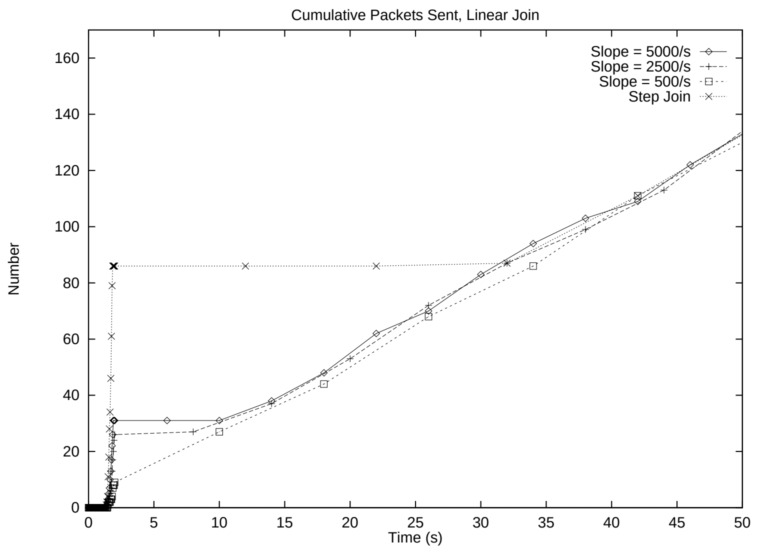


Figure 3.10: Linear joins: unconditional reconsideration

3.7.3 Analysis

In this section, we present a mathematical analysis of the reconsideration mechanism. We first consider the case where there are no network delays. This results in a differential equation which describes the learning curve. The analysis also applies to networks with delay, but only models the post-transient behavior of the system. However, this is sufficient to compute the post-transient packet rate and system convergence times. We then extend this analysis to the case of network delays, and derive expressions which describe the transient spikes and plateaus in the learning curve. We also analytically demonstrate the reasons for improved performance from unconditional reconsideration, which only exist in the presence of network delays. We also consider linear joins, deriving bounds on their approximations as step joins.

3.7.3.1 No Delay

The system we consider here is one where all of the users join the network at the same time, $t = 0$. It is assumed that the network introduces neither delay nor loss, and that access links have infinite bandwidth. The result is that when a user sends an RTCP packet, it is received by all of the users simultaneously at the time it was transmitted.

We also assume that all users are receivers. In the model considered here, all users will have exactly the same state (in terms of $L(t)$) at all times. Thus, we trace state evolution as seen by a particular user. The user estimate has converged when $L(t) = N$, the number of users actually in the group.

Whenever a packet is reconsidered, it is either sent, or it is not, depending on whether the newly computed send time is before or after the current time. We can therefore view the reconsideration mechanism as causing any packet to be sent with some probability P_{send}. In the most general case, P_{send} is a function of the current time t, the time of the last RTCP report, and the number of users observed at t, $L(t)$. Once sent, these packets are received instantly by all other participants. If a packet is the first RTCP packet sent by some user X (which we call an initial packet), the result is an increase in $L(t)$ by 1 for all receivers. Consider the evolution of $L(t)$ for any particular user U. For any subsequent packet received by U sent by X, $L(t)$ at U is unaffected. Therefore, we need only consider the value of P_{send} for initial packets. For these packets, the last report time is always $t = 0$, so that the send probability is a function of t and $L(t)$ only.

If we consider some small interval of time, the change in $L(t)$ is equal to the number of initial packets scheduled to be sent during this interval across all N members, times the probability of sending a packet in that interval. Based on this, we can immediately write the differential equation:

$$\frac{dL}{dt} = d(t)P_{send}(t, L(t)), \tag{3.3}$$

where $d(t)$ is the rate of packets scheduled for transmission during some time interval. What remains is the evaluation of the scheduled rate $d(t)$ and send probability $P_{send}(t, L(t))$. We first consider the send probability.

3.7.3.1.1 Computing the Send Probability Consider an initial packet scheduled to be transmitted by a user at time t. We assume a "fluid" model of packet transmissions, so that $L(t)$ is continuous. In this case, $L(t)$ increases over any time interval, causing all packets to be reconsidered[1]. At time t, the user perceives $L(t)$ other users in the system. It thus calculates a new packet interval, which is equal to:

$$T = R(\alpha) \max(T_{\min}, CL(t)) \tag{3.4}$$

Since $CL(t)$ is larger than $T_{\min}$ most of the time, we ignore the $\max$ operator:

$$T = R(\alpha)CL(t) \tag{3.5}$$

Keeping in mind that the previous report time is always $t = 0$, the result is that the new transmission time is T. If this time is less than the current time t, the packet is sent, otherwise, it is not. This implies that P_{send} equals the probability that T is less than t. Since T is uniformly distributed between $(1 - \alpha)CL(t)$ and $(1 + \alpha)CL(t)$, we can immediately write the probability of sending a packet:

$$P_{\text{send}} = \tfrac{t-(1-\alpha)CL(t)}{2\alpha CL(t)}, \quad \text{for} \qquad (1-\alpha)CL(t) < t < (1+\alpha)CL(t) \tag{3.6}$$

The numerator represents the range of times in the interval widow which fall below the current time t, while the denominator represents the total range over which the times for the interval are selected. This is illustrated in Figure 3.11. Note that this probability only makes sense when t is between $(1 - \alpha)$ and $(1 + \alpha)$ of $CL(t)$. When t is to the left of this "reconsideration window", the probability is zero, and when t is to the right of the window, it is one.

[1]It is for this reason that we make no distinction between conditional and unconditional reconsideration here

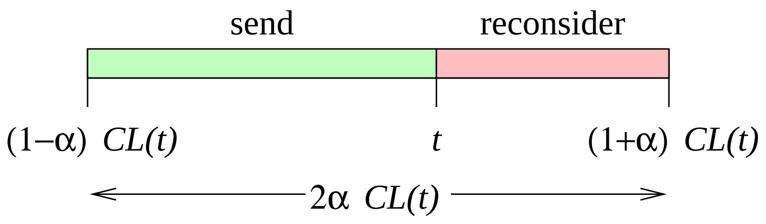


Figure 3.11: Computing P_{send} with reconsideration

An important implication of this equation is that the send probability is zero when $t < (1-\alpha)CL(t)$. This places an upper bound on the learning curve; if the learning curve should reach this bound, no initial packets would be sent, and the curve would remain flat until it fell back below this upper bound. We therefore define the *maximum learning curve* $L_{\max}(t)$ to be:

$$L_{\max}(t) = \frac{1}{(1-\alpha)C}t \tag{3.7}$$

The actual learning curve $L(t)$ is always below $L_{\max}(t)$.

3.7.3.1.2 Computing the Scheduled Rate The next step is to compute the scheduled rate $d(t)$, which is significantly harder. Ideally, the scheduled rate at any point in time is the number of users in the system, N, divided by the average RTCP interval size perceived by those users at time t, namely $CL(t)$. However, there is a delay between the time t when the users observe there to be $L(t)$ other users, and when packets are to be sent which were scheduled at t. On average, this delay is $CL(t)$. In other words, at time t, each user perceives $L(t)$ other users. When a packet is sent at time t, the next RTCP packet is scheduled for time $t + CL(t)$ on average. At that time, the scheduled rate of packets will be $\frac{N}{CL(t)}$, and of initial packets, $\frac{N-L(t+CL(t))}{CL(t)}$. We thus have:

$$d(t+CL(t)) = \frac{N - L(t+CL(t))}{CL(t)} \tag{3.8}$$

Then, we define t' as the time for which $t = t' + CL(t')$, so that:

$$d(t) = \frac{N - L(t)}{CL(t')} \tag{3.9}$$

Note that it is always the case that $t' < t$ as long as $L(t) > 0$. Without further assumptions on $L(t)$, no further simplification is possible.

The curves of Figure 3.6 show that when reconsideration is used, $L(t)$ exhibits linear behavior between roughly $t = 100$ and $t = 9000$ (thus ignoring the transient behavior in the beginning few seconds). We therefore attempt to determine the slope a of this line based on the differential equation given in Equation 3.3. Substituting $L(t) = at$ into (3.3), along with our expressions for P_{send} and $d(t)$:

$$a = \frac{N - L(t)}{CL(t')} \frac{1 - (1 - \alpha)Ca}{2\alpha Ca} \tag{3.10}$$

For small t, $L(t) < N$, so we can ignore the L in the first term's numerator. Thus:

$$\frac{2\alpha C^2 L(t')}{N} a^2 + a(1 - \alpha)C - 1 = 0 \tag{3.11}$$

Note that since $t' < t$, and $L(t)$ is non-decreasing, $L(t') \leq L(t)$. Thus, for large N and small t, $L(t') \leq L(t) \ll N$, we can neglect the a^2 term, and obtain the desired result:

$$a = \frac{1}{(1 - \alpha)C} \tag{3.12}$$

Not coincidentally, this is also the slope of the maximum learning curve. The equation indicates, therefore, that $L(t)$ grows at its maximum rate until the approximation is no longer valid, at which point its growth tapers off.

The point at which the approximation is no longer valid is also readily computed. It occurs when the first term in the polynomial is no longer small. This happens when the first term is on the order of one. Recall the precise formulation of the polynomial is:

$$\frac{2\alpha C^2 a^2 L(t')}{N - L(t)} + (1 - \alpha)Ca - 1 = 0 \tag{3.13}$$

Using (3.12) as a solution for a, we set the first term in the polynomial to 1:

$$\frac{2\alpha}{(1 - \alpha)^2} \frac{L(t')}{N - L(t)} = 1 \tag{3.14}$$

Unfortunately, the recursive definition of t' will not allow us to proceed further. As an approximation, we assume that $L(t) = L(t')$. This assumption is reasonable if $C \ll 1$. If this condition is not true, it will yield an underestimate of when the linear assumption is no longer valid. With this assumption, the linear approximation of $L(t)$ is valid until:

$$L(t) = N\left(\frac{(1-\alpha)^2}{1+\alpha^2}\right) \tag{3.15}$$

For α equal to $1/2$, the implication is that $L(t)$ will remain linear roughly until it reaches 20% of its maximum value, and at that point it will be significantly non-linear. The period of linearity is an appreciable fraction of time until convergence.

With a linear approximation to $L(t)$, we can now approximate the scheduled rate $d(t)$. We again start with the initial definition:

$$d(t) = \frac{N - L(t)}{CL(t')} \tag{3.16}$$

With:

$$t = t' + CL(t') \tag{3.17}$$

If we assume that $L(t) = t/(1-\alpha)C$, and plug this back in, we obtain:

$$t = t' + \frac{t'}{1-\alpha} \tag{3.18}$$

$$t = t'\left(1 + \frac{1}{1-\alpha}\right) \tag{3.19}$$

$$t = t'\left(\frac{2-\alpha}{1-\alpha}\right) \tag{3.20}$$

Substituting this into Equation 3.16, we obtain:

$$d(t) = \frac{N - L(t)}{CL(\frac{1-\alpha}{2-\alpha}t)} \tag{3.21}$$

Finally, we assume that $L(t)$ is linear, and therefore:

$$d(t) = \frac{N - L(t)}{C\frac{1-\alpha}{2-\alpha}L(t)} \tag{3.22}$$

This equation is *very* approximate, for a number of reasons:

1. It has been obtained via linear approximations, but we are attempting to use it to model non-linear behavior.

2. Fundamental to its definition is the concept that the delay between now, and the time when the perceived state causes the density to be $N - L(t)/CL(t)$ is $CL(t)$. This is true if the packet is sent at time t; if it is reconsidered, the delay may be less than this.

3. It is also assumed that the mix of initial and non-initial packets in the scheduled rate at any point in time is uniform. This is likely not true; we expect initial packets to be scheduled earlier than non-initial.

Despite these inaccuracies, we have observed that it gives good results in the final differential equation, and appears to capture the density reasonably well. This can be seen by the results in Figure 3.12. The figure depicts the experimentally observed integral of the scheduled rate when $10,000$ users join the system at $t = 0$[2]. Network delays are zero, and the links are infinitely fast. The figure also depicts the integral of the scheduled rate as computed by analysis. To do this, we use the numerical solution for $L(t)$ as obtained below from the ODE, and plug it into the formula for the scheduled rate. Of course, this is somewhat of a circular approach; but it suffices to show that we have made a reasonable assumption. From the figure, it is clear that the shape of the analytical solution is roughly correct, but the derivative of the analytical curve is generally less than that of the experimental. This implies that our approximation will generally underestimate the scheduled rate.

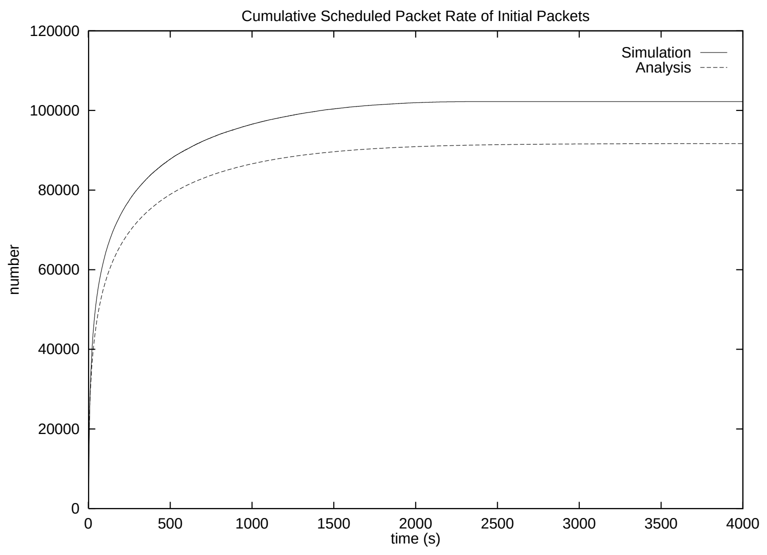


Figure 3.12: Experimental vs. Analytical Scheduled Rate Integral

[2]We use the integral since measuring instantaneous quantities requires significant averaging, which would destroy the shape of the curve we are looking for.

3.7.3.1.3 Obtaining the ODE Now, with the density and send probabilities computed, we can write the final differential equation, which is:

$$\frac{dL}{dt} = \frac{N - L(t)}{C\frac{1-\alpha}{2-\alpha}L(t)} \frac{t - (1-\alpha)CL(t)}{2\alpha CL(t)} \tag{3.23}$$

This ODE allows us to compute a numerical solution, which can be compared against the simulations. Figure 3.13 shows the learning curve, with 10,000 users joining at $t = 0$, for both analysis and simulation. In the simulation, however, we take into account non-zero delays and finite link speeds; network delays are a deterministic 300 ms, and link speeds are 28.8 kb/s. Note that despite this change in assumptions, the analytical expression *still* comes extremely close to the simulations for a large portion of the convergence period. In particular, it is very close during the period of linearity of $L(t)$ and less accurate afterwards. In addition, the differential equation does not capture the behavior of $L(t)$ for $0 \leq t \leq 20$, where the simulated curve exhibits the spike and plateau (this is difficult to see in Figure 3.13 because of the x axis scale).

We believe that network delays only impact the behavior of $L(t)$ when they are on the order of $CL(t)$. This is somewhat intuitive; the timescale of transmission events for any particular user is $CL(t)$. If network delays are much smaller than this, they are almost instantaneous as far as sending packets goes, and therefore do not affect the system behavior. It is for this reason that network delays only impact the learning curve during the first minute or so.

3.7.3.1.4 Computing the Level of Congestion With an understanding of the behavior of $L(t)$, we are now in a position to discuss the real quantity of interest: the aggregate bit rate generated by these sources as they move towards convergence (recall that convergence is the time when $L(t)$ equals the actual group size N). We call this quantity $r(t)$. Since the integral of this quantity is the total number of packets sent, we have, as an immediate consequence:

$$r(t) \geq \frac{d}{dt}L(t) \tag{3.24}$$

This is because $L(t)$ always increases by one due to the arrival of an initial packet. Since not all packets are initial, the total number of packets sent is always greater than or equal to the learning curve.

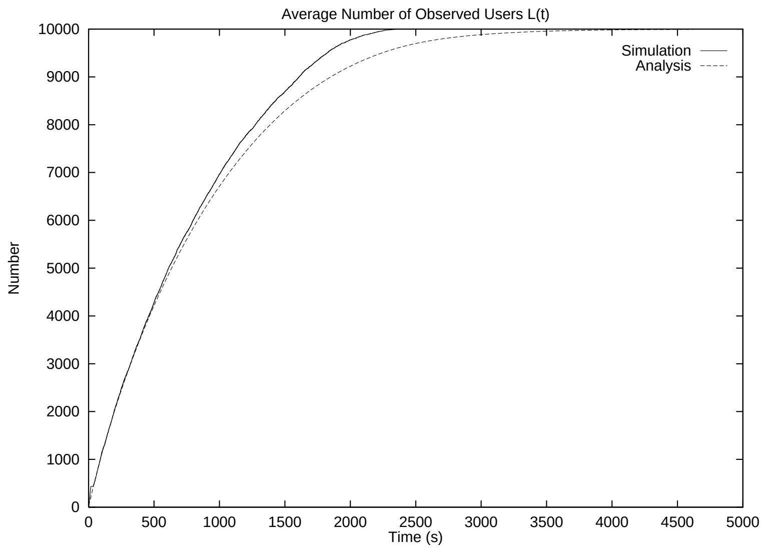


Figure 3.13: Experimental vs. analytical learning curve

Experimentally, we have observed that $r(t)$ is actually *equal* to the derivative of $L(t)$ for a large fraction of the time until convergence. This can be seen in Figure 3.14. That figure depicts the learning curve and the integral of the packet rate. The scenario is a step join of $10,000$ users at $t = 0$, uniform network delays with a mean delay of 300 ms, and 28.8 kb/s access links. Note how the two are exactly the same between $t = 0$ and $t = 4000$, which nearly 40% of the time between the beginning of the run and convergence.

The reason for this lies with the reconsideration mechanism's "favoring" of packets from users who have not yet sent a packet (initial packets). Consider two packets, both scheduled to be sent at some time t. One is an initial packet from user A, and the other is from user B, who has sent a packet previously. For user A, since no packet has been sent, the last report time is at $t = 0$. For user B, the last report time is at some time t^*, not equal to zero. In the latter case, the bottom edge of the reconsideration window for user B is at $t^* + C(1 - \alpha)L(t)$. Thus, the probability of user B sending a non-initial packet at time t is:

$$P_{\text{sendold}} = \frac{t - t^* - C(1-\alpha)L(t)}{2\alpha C L(t)} \tag{3.25}$$

This quantity is *always* less than the send probability for an initial packet from user A as given in Equation 3.6. In fact, for small t, $L(t)$ is equal to $t/C(1 - \alpha)$. Substituting this into Equation 3.25, we get that the numerator of the fraction is negative, so the send probability

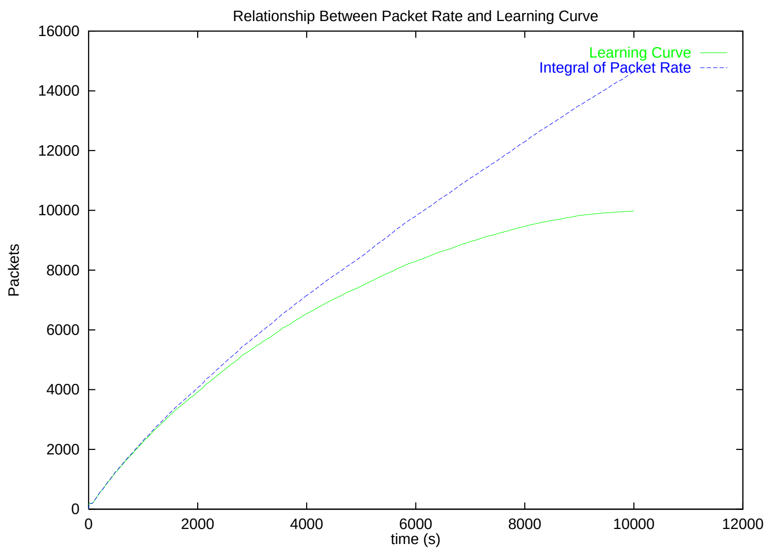


Figure 3.14: $L(t)$ vs. $\int r(t)$

is exactly zero.[3]. Therefore, $r(t)$ is exactly equal to the derivative of $L(t)$ while $L(t)$ is linear. We expect it to continue to track the derivative closely even as $L(t)$ tapers off. The result is that reconsideration demonstrates large efficiencies, a very nice side effect.

Once $L(t)$ has converged to N, packets are sent at a rate of $1/C$ with conditional reconsideration. With unconditional reconsideration, this rate is somewhat less; later sections explore this issue further. Therefore, $r(t)$ exhibits a dual-constant behavior; it starts at $1/(1-\alpha)C$, stays there for some time, then reduces to $1/C$, where it remains from then on.

3.7.3.1.5 Reconsideration as a Control Mechanism We can gain further insight into the behavior of the reconsideration algorithm by viewing it as a control mechanism. To see this, we can compute the send probability when $L(t)$ exceeds the ideal t/C. Looking at the send probability in Equation 3.6, as $L(t)$ increases, the numerator decreases, and the denominator increases, so the quotient decreases. Plugging $L(t) > t/C$ into Equation 3.6:

$$P_{send} \leq \frac{t-(1-\alpha)t}{2\alpha t} \tag{3.26}$$

$$P_{send} \leq \frac{\alpha t}{2\alpha t} \tag{3.27}$$

[3]Note that plugging in $L(t) = t/C(1-\alpha)$ into Equation 3.6 yields a numerator of zero, and thus a probability of zero also. However, the send probability is zero only in the limit for $N = \infty$; it is slightly positive for all real cases. This is in contrast to the send probability for non-initial packets, which is exactly zero for finite N.

$$P_{send} \leq \frac{1}{2} \tag{3.28}$$

Conversely, if $L(t)$ is below t/C, the send probability is greater than $1/2$. Therefore, the send mechanism is acting as a controller, attempting to keep the learning curve as close as possible to the ideal by adjusting the send probability. Furthermore, this behavior depends only on the formulation of P_{send}, which is *exact* and not dependent on network delays. This implies that this control behavior carries into real systems as well.

3.7.3.1.6 Computing the Convergence Time The final step is to approximate the convergence time. Unfortunately, the precise time depends on the non-linear regime of $L(t)$, which we cannot capture adequately. However, we can bound the convergence time by assuming linear behavior until $L(t)$ equals N. Since the actual $L(t)$ is less than this curve, the convergence time T_c is easily bounded on the left by

$$T_c \geq NC(1-\alpha). \tag{3.29}$$

This bound grows linearly with the group size, as expected.

It is possible to compute an upper bound as well. Consider the last initial packet to be sent. Before it is sent, $L(t) = N - 1$. As long as the send probability is less than one, it is possible that this last initial packet will not be sent. But, according to (3.6), the send probability is one when $t > (1+\alpha)CL(t)$. This means that the last initial packet must be sent as soon as $t = (1+\alpha)C(N-1)$. This gives us an upper bound of

$$T_c \leq NC(1+\alpha). \tag{3.30}$$

3.7.3.2 Modeling Delay and Loss

Of course, in a real network, delays are an important concern; in fact, they are primarily responsible for the transient congestion conditions seen in Figure 3.7 over the first few seconds.

In this section, we consider the reconsideration algorithm in the presence of network delay and link bottlenecks. It is readily demonstrated that delay and link congestion have an impact on the startup behavior of the learning curve. The result is a bouncing effect of rapid sending followed by complete quiet. However, these phenomenon last for only a small fraction of the duration of the convergence period. We quantify these anomalies, and back up the analytical results with simulation. We also demonstrate the superiority of unconditional reconsideration in the presence of network delays.

The transient condition present in Figure 3.7 appears to consist of two parts. In the first, a large number of packets are initially transmitted. After this burst, there is a quiet period, the “plateau effect”, where no packets are sent at all. After this effect subsides, packet transmission continues at some nominally linear rate, as predicted by the equations in the previous section.

This transient phase is easily explained. At $t = 0$, all N users join the system. They schedule their packets to be sent between $(1-\alpha)T_{min}$ and $(1+\alpha)T_{min}$. At time $(1-\alpha)T_{min}$, packets begin to be sent. Assume that the network introduces a delay of D seconds. This means that no packets will arrive at any end system until time $(1-\alpha)T_{min} + D$. During these D seconds, many packets will be sent by end-systems, causing the initial spike of packets. After D seconds, this burst of packets will arrive. This causes a sharp increase in the perceived group size $L(t)$. This, in turn, increases the packet transmission interval, and moves the left hand side of the reconsideration window well beyond the current time. The result is a complete halt in transmissions until real time catches up with the left hand side of the reconsideration window.

This qualitative description of the system is easily quantified. For a large enough N, the flood of packets starting at time $(1-\alpha)T_{min}$ will saturate the access links D seconds later, independent of whether conditional or unconditional reconsideration is used. While the links remain saturated, packets arrive at a continuous rate at the link speed, which we denote as m packets per second. We can therefore express the time of the n^{th} packet arrival by

$$t_n = (1-\alpha)T_{min} + D + \frac{n}{m}. \tag{3.31}$$

The arrival of the n^{th} packet causes $L(t)$ to increment by one, so that if n packets have arrived by t_n, $L(t_n) = n$. We can therefore substitute n in the above equation with $L(t)$:

$$L(t) = m(t - (1-\alpha)T_{min} - D) \tag{3.32}$$

The result is a rapid linear increase in $L(t)$, well beyond its maximum as given in Equation 3.7. When the learning curve exceeds this maximum, all sending will stop. Call this stopping time

t_{stop}. It is the time at which the left hand side of the reconsideration equals the current time:

$$(1-\alpha)CL(t_{stop}) = t_{stop} \tag{3.33}$$

$$t_{stop} = (1-\alpha)T_{min} + D + \frac{(1-\alpha)T_{min} + D}{(1-\alpha)Cm - 1} \tag{3.34}$$

We can then substitute t_{stop} into Equation 3.32 and solve for the number of packets which have arrived up to t_{stop}, $n_{stop} = L(t_{stop})$:

$$n_{stop} = \frac{(1-\alpha)T_{min} + D}{(1-\alpha)C - 1/m} \tag{3.35}$$

The next step is to determine the number of packets sent by all participants by time t_{stop}. This quantity depends on whether the reconsideration mechanism is conditional or unconditional. We first look at conditional.

3.7.3.2.1 Number of Packets Sent for Conditional Reconsideration

The number of packets sent by time t_{stop} for conditional reconsideration, n_{sc}, consists of two terms n_{sca} and n_{scb}. Before the arrival of the first packet (at time $(1-\alpha)T_{min} + D + 1/m$), all packets scheduled to be sent are actually sent, since no users have observed a change in the group size estimate $L(t)$ (which would activate the reconsideration mechanism). The number of packets sent is then the density of packets scheduled to be sent (which is $N/2\alpha T_{min}$) times the amount of time until the first packet arrives. We call this quantity n_{sca}, and it is:

$$n_{sca} = \frac{N}{2\alpha T_{min}}\left(D + \frac{1}{m}\right) \tag{3.36}$$

Once the first packet arrives, reconsideration kicks in, and not all packets will be sent. Each will be sent with some probability, P. Unfortunately, this is not the same probability P_{send} as defined in Equation 3.6. That equation ignored the max operator, assuming $L(t)$ was large most of the time. This is not true in the very beginning, where it takes a few packets to increase $CL(t)$ beyond T_{min}. We assume that once enough packets have arrived to do this, the result will be to move the left hand side of the reconsideration window ahead of the current time (this is true when $D < C$). In other words, we assume the left hand side of the reconsideration window is always at $(1-\alpha)CT_{min}$ until t_{stop}.

With this in mind, the send probability between the arrival of the first packet and t_{stop} is given by:

$$P_{send} = \frac{t - (1-\alpha)T_{min}}{2\alpha T_{min}} \tag{3.37}$$

The number of packets sent is given by the integral of the scheduled packet rate times the send probability from time $(1-\alpha)T_{min} + D + 1/m$ to the stopping time:

$$n_{scb} = \int_{(1-\alpha)T_{min}+D+1/m}^{t_{stop}} d(t) P_{send} dt \tag{3.38}$$

Since the scheduled rate $d(t)$ is $N/2\alpha T_{min}$ between $(1-\alpha)T_{min} + D + 1/m$ and t_{stop}, the number of packets sent is obtained by:

$$n_{scb} = \int_{(1-\alpha)T_{min}+D+1/m}^{t_{stop}} \frac{N}{2\alpha T_{min}} \frac{t - (1-\alpha)T_{min}}{2\alpha T_{min}} dt \tag{3.39}$$

This integral results in a growth in the number of sent packets as t^2 until complete cutoff at t_{stop}. The solution to the integral is:

$$n_{scb} = \frac{N}{8\alpha^2 T_{min}^2} \left(\left(\frac{(1-\alpha)T_{min} + D}{(1-\alpha)Cm - 1} + D \right)^2 - \left(D + \frac{1}{m} \right)^2 \right) \tag{3.40}$$

And the total number of packets sent, using conditional reconsideration, during this transient spike is:

$$n_{sc} = n_{sca} + n_{scb} \tag{3.41}$$

These analytical results are compared with simulation in Figure 3.15. The figure displays the cumulative number of packets sent for a step join. For the simulation, 100,000 users join the system at $t = 0$. Network delays are deterministic and equal to 300 ms, and access link speeds are 28.8 kb/s. The plot shows only the initial transient. The linear and then t^2 behavior is clear from the simulation. The horizontal line labeled nsc1 represents the quantity n_{sca} as predicted by our analysis, and the line labeled nsc represents the quantity n_{sc} as predicted by our analysis. Note that our analysis predicts these quantities quite well. The analysis also predicts that sending will stop at $t_{stop} = 1.72s$, which agrees with the simulation. Also note that the number of packets sent is dominated by the n_{sca} term.

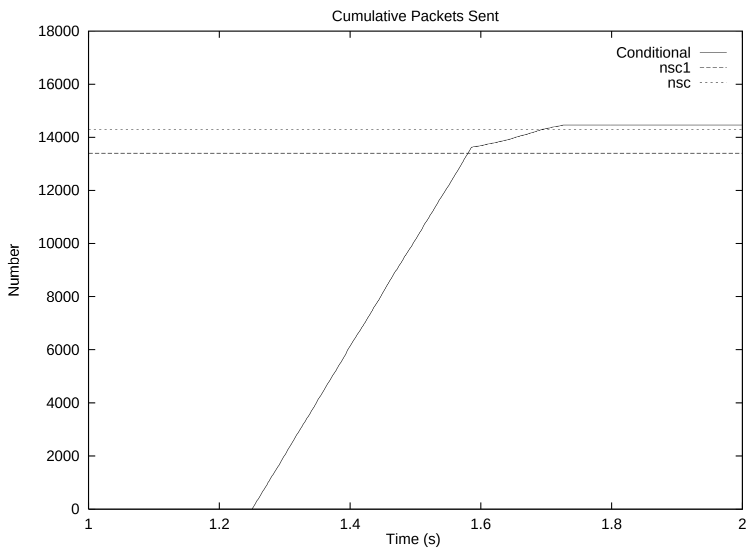


Figure 3.15: Transient with Conditional Reconsideration

3.7.3.2.2 Number of Packets Sent for Unconditional Reconsideration For unconditional reconsideration, the number of packets sent during the transient (n_{su}) is different. In the conditional case, the total consisted of two parts; one before the arrival of the first packet (as the reconsideration mechanism had not "kicked in" yet), and one after. In the case of unconditional, we do not need to wait for the arrival of a packet for the mechanism to activate. Therefore, the number of packets sent is given by an equation similar to that for n_{scb} above. It is the integral of the scheduled rate, times the send probability. In this case, the integral is between $(1-\alpha)CT_{min}$ and t_{stop}, instead of just between the arrival of the first packet and t_{stop}. The number of packets sent for unconditional is therefore:

$$n_{su} = \int_{(1-\alpha)T_{min}}^{t_{stop}} \frac{N}{2\alpha T_{min}} \frac{t-(1-\alpha)T_{min}}{2\alpha T_{min}} dt \tag{3.42}$$

Solving, we obtain:

$$n_{su} = \frac{N}{8\alpha^2 T_{min}^2} \left(\frac{(1-\alpha)T_{min}+D}{(1-\alpha)Cm-1} + D \right)^2 \tag{3.43}$$

This quantity is small compared to n_{sca} for conditional reconsideration, thus the improved performance. These results are compared with simulation in Figure 3.16. The simulation model is identical to that in Figure 3.15, except unconditional reconsideration is used. As the plot indicates, only the t^2 behavior is present here. The horizontal line labeled nsu indicates the value of n_{su} as predicted by our analysis. The total number of packets sent during the transient is much reduced compared to conditional reconsideration (almost by a factor of seven), and reasonably well predicted by our analysis.

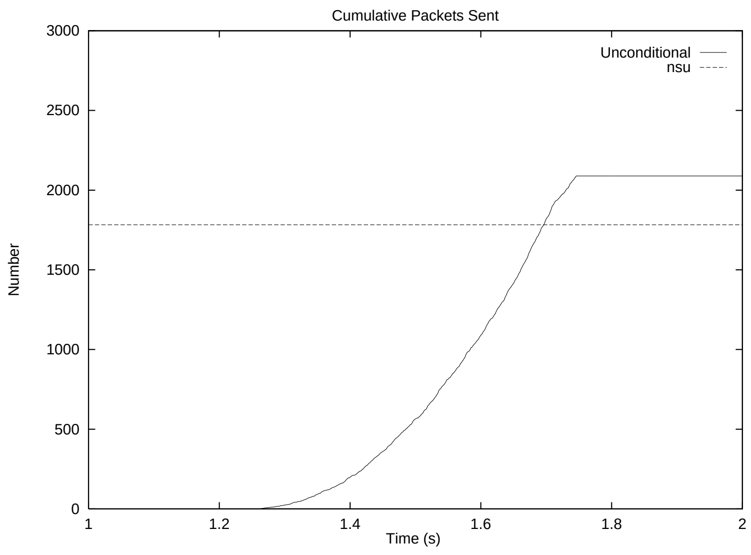


Figure 3.16: Transient with Unconditional Reconsideration

3.7.3.2.3 Duration of Plateau Period The next step is to determine the duration of the plateau period. Packet sending will start again when the current time catches up with the left hand side of the reconsideration window, which will have quickly advanced to $(1-\alpha)Cn_{sc}$ for conditional reconsideration, and $(1-\alpha)Cn_{su}$ for unconditional reconsideration. We therefore define t_{start} as the time at which sending of packets begins again. We have, for conditional reconsideration,

$$t_{startc} = (1-\alpha)Cn_{sc}, \tag{3.44}$$

and for unconditional reconsideration,

$$t_{startu} = (1-\alpha)Cn_{su}. \tag{3.45}$$

For conditional reconsideration, if we assume $n_{sc} \approx n_{sca}$, we obtain

$$t_{startc} = \frac{C(1-\alpha)N}{2\alpha T_{min}} \left(D + \frac{1}{m} \right), \tag{3.46}$$

Group Size N	Conditional		Unconditional	
	n_{sent}	T_{plat}	n_{sent}	T_{plat}
1000	143	49 s	18	5 s
10000	1430	506 s	178	61 s
100000	14305	5083 s	1784	632 s

Table 3.5: Transient Behavior for Various Group Sizes

and for unconditional reconsideration,

$$n_{su} = \frac{(1-\alpha)CN}{8\alpha^2 T_{min}^2}\left(\frac{(1-\alpha)T_{min}+D}{(1-\alpha)Cm-1}+D\right)^2 . \tag{3.47}$$

The duration of the plateau period itself is given by

$$T_{plat} = t_{start} - t_{stop}. \tag{3.48}$$

The following table lists the values of the parameters derived above for various group sizes. In all cases, $\alpha = 1/2$, T_{min} is 2.5 s, C is 0.711 s, and D is 300 ms. The unconditional mechanism provides clear gains in terms of reducing the number of packets sent during the transient, and the duration of the plateau effect.

3.7.3.3 Linear Joins

The model for the step-join was motivated by applications where the user population joins a multicast group in response to some synchronous event, such as a television program or a session announcement. However, due to a variety of real world factors, users will not really join the group at exactly the same time. Instead, their joins are likely to be scattered, perhaps uniformly distributed over some short interval of time. The result would be that the group membership would grow linearly in time until reaching its peak, as opposed to occurring in a single step.

Because of this, it is critical to determine the impact of linear joins on our assumptions and on the performance of the reconsideration algorithms. In this section, we consider the modeling of the system where users join the group in a linear rate. Our goal will not be an exact solution of any equations of constants, but rather identification of transition points where linear behavior appears as a step.

What differentiates the linear join from the step join is the initial shape of the scheduled rate $d(t)$. In the case of the step join, the scheduled rate of packets is flat between $(1-\alpha)T_{min}$ and $(1+\alpha)T_{min}$. During this time, it is equal to $N/2\alpha T_{min}$. In the case of linear joins, however, we would not expect the scheduled rate to be flat. Instead, we expect it to increase until it reaches a peak, and then to decrease from there.

Consider the computation of the scheduled rate at some time t. Since we are only interested in the shape of the scheduled rate for small t, we assume that $d(t)$ is only governed by initial packets, and not by packets that have been reconsidered or sent previously. The equations presented above dealt with the shape of the scheduled rate during this period of time.

The scheduled rate at t is a function of users who join the system between times $t-(1+\alpha)T_{min}$ and $t-(1-\alpha)T_{min}$ only. For a user who joins the system during this interval, the probability that they actually schedule their packet to be sent in an infinitesimal window around t is $dt/2\alpha T_{min}$. Therfore, if we define dJ/dt to be the increase in the number of users at time t (so that $J(t)$ is the total number of users who have joined up until time t), we can obtain the scheduled rate directly by integration:

$$d(t) = \int_{t-(1+\alpha)T_{min}}^{t-(1-\alpha)T_{min}} \frac{d\tau}{2\alpha T_{min}} \frac{dJ}{d\tau} \tag{3.49}$$

$$d(t) = \frac{J(t-(1-\alpha)T_{min}) - J(t-(1+\alpha)T_{min})}{2\alpha T_{min}} \tag{3.50}$$

When $J(t) = Nu(t)$, i.e., a step join of N users at $t=0$, the above equation correctly defines $d(t)$:

$$d(t) = \begin{cases} 0, & t < (1-\alpha)T_{min} \\ \frac{N}{2\alpha T_{min}}, & (1-\alpha)T_{min} \le t \le (1+\alpha)T_{min} \end{cases} \tag{3.51}$$

The behavior of $d(t)$ after $t = (1+\alpha)T_{min}$ depends on the reconsideration algorithm, as discussed in previous sections.

For a linear join, we have the following definition for $J(t)$:

$$J(t) = \begin{cases} \gamma t, & t < \frac{N}{\gamma} \\ N, & t \ge \frac{N}{\gamma} \end{cases} \tag{3.52}$$

We can substitute this into Equation 3.49, and then obtain the solution for $d(t)$. The solution differs depending on the relationship between N/γ and $2\alpha T_{min}$. When $N/\gamma < 2\alpha T_{min}$, we obtain:

$$d(t) = \begin{cases} 0, & 0 \leq t < (1-\alpha)T_{min} \\ \frac{\gamma(t-(1-\alpha)T_{min})}{2\alpha T_{min}}, & (1-\alpha)T_{min} \leq t < (1-\alpha)T_{min} + \frac{N}{\gamma} \\ \frac{N}{2\alpha T_{min}}, & (1-\alpha)T_{min} + \frac{N}{\gamma} \leq t \leq (1+\alpha)T_{min} \\ \frac{N-\gamma(t-(1+\alpha)T_{min})}{2\alpha T_{min}}, & (1+\alpha)T_{min} \leq t < (1+\alpha)T_{min} + \frac{N}{\gamma} \end{cases} \tag{3.53}$$

This somewhat complicated equation actually describes a trapezoid. The height of the trapezoid is $N/2\alpha T_{min}$, the top has a width of $2\alpha T_{min} - N/\gamma$, and the bottom has a width of $2\alpha T_{min} + N/\gamma$. This is somewhat intuitive. In the case of the step join, the scheduled rate was a rectangle, with the same height as this trapezoid. The affect of the linear joins is to cause the rectangle to bend in on itself, forming a trapezoid. We note that in the case above, the height still remained fixed at $N/2\alpha T_{min}$.

When N/γ is greater than $2\alpha T_{min}$, we obtain the following solution for $d(t)$:

$$d(t) = \begin{cases} 0, & 0 \leq t < (1-\alpha)T_{min} \\ \frac{\gamma(t-(1-\alpha)T_{min})}{2\alpha T_{min}}, & (1-\alpha)T_{min} \leq t < (1+\alpha)T_{min} \\ \gamma, & (1+\alpha)T_{min} \leq t < (1-\alpha)T_{min} + \frac{N}{\gamma} \\ \gamma - \frac{\gamma(t-((1-\alpha)T_{min}+\frac{N}{\gamma}))}{2\alpha T_{min}}, & (1-\alpha)T_{min} + \frac{N}{\gamma} \leq t < (1+\alpha)T_{min} + \frac{N}{\gamma} \end{cases} \tag{3.54}$$

This even more complex looking equation also describes a trapezoid. Here, however, the height of the trapezoid is equal to γ, and the slopes of the sides are equal to $\gamma/2\alpha T_{min}$. As the slope of the linear joins decreases, the trapezoid flattens out and grows wider. This makes sense; the lower the slope of the linear join, the less of a "step" the join becomes, and the more spread out the initial packets from these users become.

With these two equations, we can now look at the size of the initial spike of packets. We first consider conditional reconsideration. In the case of the step join, the dominant cause of the initial flood was the n_{sca} term. This term essentially represented the product of the scheduled packet rate times the amount of time until the reconsideration mechanism kicks in (roughly $D + 1/m$). The more general formulation of this quantity is:

$$n_{sca} = \int_{(1-\alpha)T_{min}}^{(1-\alpha)T_{min}+D+\frac{1}{m}} d(t)dt \tag{3.55}$$

We can therefore plug either of the two equations for $d(t)$ (Equation 3.54 or Equation 3.53) into this integral and obtain the number of packets sent. For the case of rapid linear joins where $N/\gamma \ll 2\alpha T_{min}$, the first definition of $d(t)$ applies. Furthermore, if $N/\gamma \ll D$, the integral above will mostly be over the flat region where $d(t) = N/2\alpha T_{min}$, and the result will be nearly the same as for a step join. This allows us to define an important breakpoint; linear joins appear as step joins as long as $\gamma \gg N/D$. Below this, they begin to cause less transient spikes.

As the slope of the joins decrease, we reach the point where $\gamma < N/2\alpha T_{min}$ and thus the second definition of $d(t)$ applies. Consider the case where $D + 1/m < 2\alpha T_{min}$ (since T_{min} is on the order of seconds, this is a reasonable assumption for even the slowest of access links). The integral of Equation 3.55 is then over only the linear region of $d(t)$ which makes up the left side of the trapezoid. With a change of variables, the integral becomes:

$$n_{sca} = \int_0^{D+\frac{1}{m}} \frac{\gamma t}{2\alpha T_{min}} dt \tag{3.56}$$

$$n_{sca} = \frac{\gamma}{4\alpha T_{min}} \left(D + \frac{1}{m}\right)^2 \tag{3.57}$$

This is a substantial reduction in the number of packets sent. Furthermore, the size of the plateau is linearly related to the number of packets sent before t_{stop}, as given by Equations 3.44 and 3.48. If we conservatively approximate t_{stop} as 0, we can get an upper bound on the duration of the plateau for linear joins:

$$T_{plat} = \frac{(1-\alpha)C\gamma}{4\alpha T_{min}} \left(D + \frac{1}{m}\right)^2 \tag{3.58}$$

Note that this quantity is always less than the case of a step join. Thats because in this case, $\gamma < N/2\alpha T_{min}$ and $D + 1/m < 2\alpha T_{min}$.

In the case of unconditional reconsideration, the number of packets sent is similar to that for conditional. The difference is that we must incorporate the send probability into the integral for the number of transmitted packets. The resulting integral is given by:

$$n_{su} = \int_{(1-\alpha)T_{min}}^{t_{stop}} d(t) \frac{t - (1-\alpha)T_{min}}{2\alpha T_{min}} dt \tag{3.59}$$

The plateau is proportional to n_{su}, as in the conditional case. We do not compute an explicit solution, but note that the change of $d(t)$ from flat to linear will cause n_{su} to be proportional to the cube of the network delays. This too is also a reduction from the value in the case of the step join.

The conclusion, therefore, is that linear joins reduce substantially the initial burst of packets sent when reconsideration is used, when the slope of the linear join is less than $N/2\alpha T_{min}$.

This result is demonstrated in Figure 3.9. The plots depict the cumulative number of packets sent by all users. In all four cases, a total of 2500 users join the system. The network delays are a fixed 300 ms, access link speeds are 28.8 kb/s, network access buffers are 100 kB, and conditional reconsideration is used. Three of the plots depict linear joins of varying slope, and the fourth is a step join, In the case of the 5000 users/s linear join, the above equation predicts that n_{sc} is around 112 packets, it is roughly 140 in the simulation. For a slope of 2500/s, theory predicts 56, and the simulation shows around 75. For a slope of 500/s, the theory predicts 12 packets, and the simulation indicates around 25. In terms of the duration of the plateau's: in the case of 5000 users/s linear join, the equation predicts 40 s, and the simulations show 58 s. For a slope of 2500/s, the theory predicts 20 s and the simulations show 28 s. For a slope of 500/s, the theory predicts 4 s and the simulations show 8 s. It is interesting to note that for slopes even as large as 500 users per second, the initial transients have almost completely disappeared.

The same simulation was run using unconditional reconsideration. The results are plotted in Figure 3.10. In this case, a join rate of 5000 users per second results in a spike of 30 packets and a plateau of around 4 s, almost negligible. The size of the spike and the plateau are reduced even further as the join rate falls below 5000 users per second.

The conclusion is that linear joins significantly reduce the initial transient spike and plateau period. This is especially true for unconditional reconsideration, which exhibits a spike and plateau for only the most extremely sharp linear join rates.

3.7.3.4 Steady State Behavior

It is important to consider the behavior of the reconsideration algorithms when the size of the group has reached a steady state. The ideal behavior is for the total send rate of the group to be $1/C$ RTCP packets per second, equally divided among all users.

There are two situations which can be reasonably deemed as steady state. The first of these is a group size which remains exactly fixed. However, in real systems, users come and go, so a second definition of steady state is a group whose membership oscillates slightly about some large value.

In the first scenario, conditional reconsideration will behave exactly as the system without reconsideration. Since the group size has not increased since the last packet transmission, the reconsideration mechanism will not be activated at all. However, with unconditional reconsideration, transmissions are always reconsidered, even when steady state has been achieved. The result of this will be an increase in the average interval for each group member, and thus a reduction in the overall send rate.

Figure 3.17 depicts the steady state packet rate when $10,000$ users join the system at time 0. The graph depicts the average packet rate (averaged over 100 seconds), from $t = 5000$ to $t = 100,000$. Lines are given for unconditional reconsideration, conditional reconsideration, no reconsideration, and the ideal packet send rate. Note that both conditional reconsideration and no reconsideration operate perfectly, sending packets at 1.4 packets per second (5% of 28.8 kb/s divided by 1 kB is 1.4 packets/s). On the other hand, unconditional reconsideration converges to a lower send rate, around 1.149 packets per second.

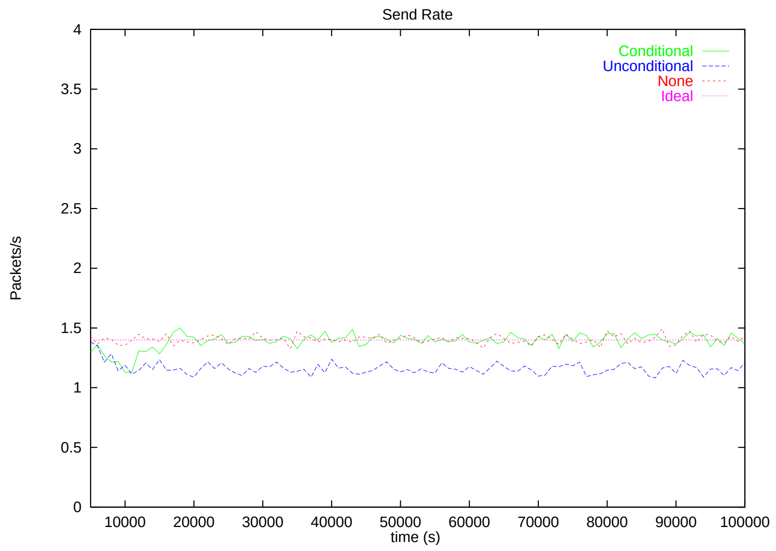


Figure 3.17: Steady State RTCP Packet Rate

Normally, the RTCP packets are sent with an interval which is between $1/2$ and $3/2$ of the deterministic interval. With unconditional reconsideration, an initial interval between $1/2$ and $3/2$ of the deterministic interval is chosen randomly. Then, when it is time to send the packet, a second interval, between $1/2$ and $3/2$ of the same deterministic interval, is chosen. If the second is greater than the first, a third is chosen, and so on, until an interval is chosen which is less than the current. When this happens, the packet is sent.

We can represent this process mathematically as follows. Let x_i be a random variable uniformly distributed between 0 and 1. First, x_0 is chosen, and then x_1. If $x_1 < x_0$, we assign y to x_0, and the process terminates. Otherwise, we draw x_2, and if $x_2 < x_1$, assign $y = x_1$, and the process terminates, etc. The problem is to determine the distribution of y.

The approach we will use is due in large part to Daniel Rubenstein, and proceeds as follows. We first examine the problem where the x_i are discrete, taking on values between 1 and M. We will label discrete random variables with a hat. We then take various limits to relate the distribution of $\hat{y}$ to y. The probability that $\hat{y}$ equals some value, say n, is the probability that we chose i random variables before n which were in order, then we chose n, and then we chose a number less than or equal to n, summed over all possible i. This can be expressed as:

$$p(\hat{y} = n) = \sum_{i=0}^{n-1} \frac{\begin{pmatrix} n-1 \\ i \end{pmatrix}}{M^i} \frac{1}{M} \frac{n}{M} \tag{3.60}$$

The combinatorial represents the number of different ways i random variables less than n can be chosen such that they are ordered. The second term, $1/M$, then represents the probability of choosing $\hat{x} = n$ next, and then the final term represents the probability of choosing $\hat{x}$ less than or equal to n.

We can collapse this equation significantly by noting that the combinatoric and the $1/M^i$ term can be seen as the binomial expansion of a polynomial:

$$p(\hat{y} = n) = \frac{n}{M^2} \left(1 + \frac{1}{M}\right)^{n-1} \tag{3.61}$$

Next, we convert this to a cumulative distribution:

$$p(\hat{y} \leq y_0) = \sum_{n=0}^{y_0} \frac{n}{M^2} \left(1 + \frac{1}{M}\right)^{n-1} \tag{3.62}$$

We define R as $1 + 1/M$, so that:

$$p(\hat{y} \leq y_0) = \frac{1}{M^2} \sum_{n=0}^{y_0} nR^{n-1} \tag{3.63}$$

We can then compute this sum by expressing as the sum of the derivative of a more manageable function:

$$p(\hat{y} \leq y_0) = \frac{1}{M^2} \frac{d}{dR} \sum_{n=0}^{y_0} R^n \tag{3.64}$$

Computing the sum:

$$p(\hat{y} \leq y_0) = \frac{1}{M^2} \frac{d}{dR} \frac{1 - R^{y_0+1}}{1 - R} \tag{3.65}$$

And then taking the derivative:

$$p(\hat{y} \leq y_0) = \frac{1}{M^2} \frac{1}{(1-R)^2} \left[1 - y_0 R^{y_0} - R^{y_0} + y_0 R^{y_0+1}\right] \tag{3.66}$$

Substituting back in $R = 1 + 1/M$:

$$p(\hat{y} \leq y_0) = 1 + y_0 \left(1 + \frac{1}{M}\right)^{y_0} \left[\frac{1}{M}\right] - \left(1 + \frac{1}{M}\right)^{y_0} \tag{3.67}$$

This $\hat{y}$ takes on discrete values between 1 and M. We would like to change to a distribution which takes on values between 0 and 1, say $\hat{z}$:

$$p(\hat{z} \leq z_0) = p(\hat{y} \leq M z_0) \tag{3.68}$$

Substituting in:

$$p(\hat{z} \leq z_0) = 1 + z_0 \left(1 + \frac{1}{M}\right)^{z_0 M} - \left(1 + \frac{1}{M}\right)^{z_0 M} \tag{3.69}$$

Now, we take the limit as M goes to infinity, and equate this to the distribution of the actual y we are looking for:

$$\lim_{M \to \infty} p(\hat{z} \leq z_0) = 1 + z_0 e^{z_0} - e^{z_0} = p(y \leq z_0) \tag{3.70}$$

We can finally take the derivative and get the pdf for y:

$$p(y = y_0) = y_0 e^{y_0} dy_0 \tag{3.71}$$

The mean of this quantity is $e - 2$.

The implication for unconditional reconsideration is that the interval is not uniform between $1/2$ and $3/2$ of the deterministic interval, but instead distributed as $(y - 1/2)e^{(y-1/2)}$

between $1/2$ and $3/2$. Instead of the mean being equal to the deterministic interval, the mean is now equal to 1.218 times the deterministic interval. This causes a reduction in the total packet rate by $1 - \frac{1}{e-3/2}$, or 18%.

When we examine behavior in the second form of steady state (that is, small oscillations in the user population about the group size), the results are slightly different, and are depicted in Figure 3.18.

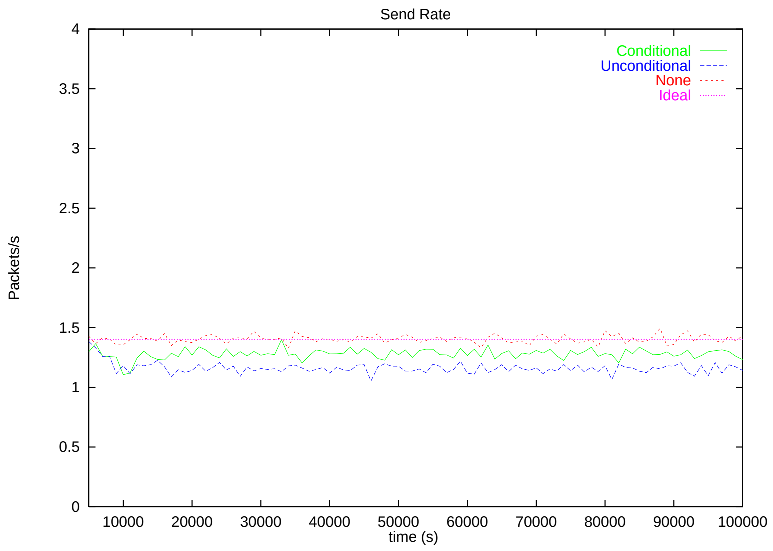


Figure 3.18: Oscillating Steady State RTCP Packet Rate

As expected, the behavior for the no-reconsideration case is unchanged, and ideal. Furthermore, the unconditional case is unaffected. This is because the slight oscillations cause only a few more or less packets to be sent, resulting in no effect on the overall packet rate. For conditional reconsideration, the “churning” in user population causes reconsideration to occur for around $1/2$ of the packets scheduled. The result is that the packet rate is halfway between unconditional and no reconsideration.

3.7.3.5 Fairness

Another important issue to look at is the impact of the reconsideration algorithms on fairness. We define fairness as the ability of the algorithm to distribute the RTCP packet rate equally among all group members. Quantitatively, we can measure fairness by computing the coefficient of variation in Z_t, where $Z(t)$ is defined as the total number of packets sent from any user up until time t, divided by t.

The original RTCP algorithm is extremely fair, and we do not anticipate deviation from this as a result of either conditional or unconditional reconsideration. This intuition was demonstrated with simulations, the results of which are presented in Figure 3.19. The figure presents the time evolution of the coefficient of variation of $Z(t)$ for three cases: no reconsideration, conditional reconsideration, and unconditional reconsideration. The system which was simulated consists of 1000 users simultaneously joining at time 0. Access links are 28.8 kb/s, network delays are uniform between 0 and 600 ms, access buffers are 100 kB, and occasional users join and leave the system, causing slight oscillations in group size, after time 1000 s. As the graphs indicate, the fairness is excellent in all three cases, with unconditional reconsideration actually exhibiting better fairness than the other algorithms.

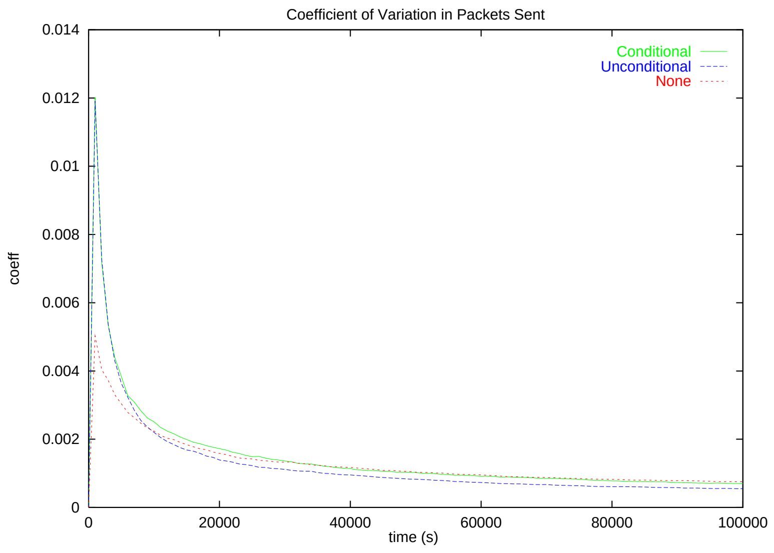


Figure 3.19: Coefficient of Variation of Packets Transmitted

3.7.3.6 Single User Joins Late

We present in this section an analysis of a different scenario using the mathematical tools developed in the past section. Consider a large group of N users, all of which have already converged to $L(t) = N$. At some point in time, a new member joins the group. We are interested in determining how long this member takes to converge, and how many packets he will emit until convergence.

Again, we ignore network delays and losses. The first step is to compute the learning curve for the new member. Since everyone else has already converged, the aggregate send rate of feedback is $1/C$. Each packet will be initial as far as the new member is concerned, so his learning curve increases as $1/C$ as well. We therefore have

$$L(t) = t/C. \tag{3.72}$$

The next step is to compute the times of packet emissions. The quantity depends on whether reconsideration is being used. We consider the current algorithm, with no reconsideration, as it is analytically tractable.

If t_n was the time of the last packet emission, the user will schedule the next one based on the learning curve at time t_n, so that:

$$t_{n+1} = t_n + CL(t_n) \tag{3.73}$$

Plugging in the value of $L(t)$ from (3.72):

$$t_{n+1} = 2t_n \tag{3.74}$$

This equation is easily solved recursively:

$$t_n = t_0 2^n \tag{3.75}$$

The number of packets sent can be solved by setting t_n equal to the convergence time, which is CN, and solving for n:

$$n = t_0 \log_2 CN \tag{3.76}$$

On average, the time of first emission is T_{min}, which is 2.5 s. Thus:

$$n = T_{min} \log_2 CN \tag{3.77}$$

When reconsideration is turned on, the convergence time is unaffected, but the number of packets emitted by the user during the convergence period is reduced. To see this, we compute P_{send} from Equation 3.6, using $L(t) = t/C$. The result is that:

$$P_{send} = \frac{1}{2} \tag{3.78}$$

Unfortunately, an analytical solution to the number of packets sent is intractable. However, the presence of the $P_{send} = 1/2$ in the system guarantees that fewer packets are sent than without reconsideration.

3.8 BYE Reconsideration Algorithm

The reconsideration problem of the previous section deals very well with the congestion problem that is caused by many users simultaneously joining a group. However, a similar problem happens when many users simultaneously leave a group. Since an RTP client sends an RTCP BYE packet when it leaves the group, this causes a flood of BYE packets, which congests the network. We refer to this as the *BYE flood problem*. Users can be expected to leave a group simultaneously for much the same reasons they might join simultaneously, namely an automatic leave as a result of the end of the session (as indicated in the Session Description Protocol (SDP) [81] description of the session), or because the “show” is over, and the users manually exit their applications.

A number of aspects of the BYE flood problem make it different than the simultaneous join problem. These must be taken into consideration when designing an algorithm to reduce the flood. We therefore state the goals of the BYE flood prevention algorithm as follows:

- Users often terminate their applications just after leaving the session. The algorithm must be aware of this possibility, and define the appropriate behavior if an application decides to terminate.
- The algorithm should behave gracefully; when very few users are leaving the group simultaneously, users should generally be allowed to send their BYE packets right away. It is only in the presence of a large number of BYE packets that the algorithm should “kick in”, and force users to hold back on sending their BYE packets.
- The algorithm should be simple, requiring minimal computation and storage.

We propose an algorithm called *BYE reconsideration* to accomplish these goals. The algorithm operates much like standard reconsideration (which we also refer to as forward reconsideration). However, instead of counting other users, and using the resulting count as a multiplier for the packet transmission interval, the client counts BYE packets, and uses the number of BYE packets received thus far as the multiplier for the interval. The operation of the algorithm is as follows.

At some time t_l, the user decides to leave the session. The application first checks to see if it has ever sent an RTCP packet. If it has not, the application must not send a BYE packet. Instead, it should leave the session silently. Without having sent an RTCP packet, the BYE packet provides no useful information. Next, the application checks to see if the group size is less than some threshold, B_t (a value of 50 seems reasonable). If it is, the application may send a BYE packet immediately, and then leave the session. For small groups, BYE packets are of significant value (for loose session management), and sending them immediately is quite important. Since the group contains only a small number of participants, the flood of packets is limited.

It is possible that a user has a low group size estimate if they leave a session quickly after joining it. If this session is large, and there are many users coming and going fairly quickly (typical of a channel surfer), it might appear that this can cause a steady flow of BYE packets. However, if these clients implement the forward reconsideration algorithms, they generally will have never sent an RTCP packet. This is because the new users' RTCP packet transmission will be constantly reconsidered, as the new user will be receiving RTCP packets at a steady rate from other users already in the group. This constant reception of packets will cause the new user to see a steady growth in the group size, causing its own RTCP packet transmission to be pushed into the future. Since a user who never sends an RTCP packet cannot send a BYE packet, this will generally cause these "channel surfers" to neither send RTCP SDES or RR information, nor a BYE packet.

If the user has sent an RTCP packet previously, and the group size exceeds B_t, the application computes a time interval T as:

$$T = R(1/2)\max(T_{min}, n_l * C) \tag{3.79}$$

Where T_{min} is 2.5 s, $C = p/(bw * .05)$, $R(1/2)$ is a random variable uniformly distributed between $1/2$ and $3/2$, p is the average size of all BYE packets received thus far, and bw is the session bandwidth. n_l is the number of users that have sent a BYE packet since the user decided to leave the session. It is initialized to 1. The average packet size is computed using the same exponential weighted average filter used to compute the average RTCP packet size in the RTP specification [2]. The value is updated through the filter every time a BYE packet is received or transmitted (not when it is reconsidered).

The user then schedules the BYE packet to be sent at time $t_l + T$. Between t_l and this time, the user increments n_l for each BYE packet that is received. In this fashion, n_l counts the number of BYE's from other users since deciding to leave the session.

When time t_l arrives, the user recomputes T according to the previous equation (including redrawing the random factor). If $t_l + T$ is less than the current time, the BYE packet may be sent. If $t_l + T$ is more than the current time, the BYE packet transmission is rescheduled for time $t_l + T$. At that time, the computation and comparison are repeated. All along, n_l is incremented for each BYE packet received.

A BYE packet which is from an SSRC which already sent a BYE (a duplicate) is ignored. Furthermore, the application should not increment n_l if it receives a BYE from a user which has never sent an RTCP packet. Under normal situations, an application should never send a duplicate BYE packet, or send a BYE if an RTCP packet was never sent. However, a malicious user may send many BYE packets. If this check were not made, these BYE's would cause the variable n_l to increase, and effectively prevent any other user from sending a BYE.

The effect of this algorithm is to restrict the BYE packet transmission rate to at most an additional 10% of the session bandwidth (assuming a very large simultaneous leave). At the same time, if only a few users are leaving the group (even for a large group), they will get to send their BYE packets in a timely fashion. This meets the design objectives described in the beginning of the section.

We ran numerous simulations to verify the performance of the algorithm. Even with as many as 10,000 users simultaneously leaving the session, the BYE reconsideration algorithm maintained the BYE transmission rate at 10% of the session bandwidth. This is demonstrated in Figure 3.20, which depicts the cumulative number of RTCP packets (BYE and others) sent to the multicast group over time. At time 10,000, almost all of the users leave the group. The top line depicts the performance without BYE reconsideration, where some 10,000 BYE packets are sent all at once. The lower curve shows performance for BYE reconsideration. Note how there is only a small increase in packet transmission rates.

In fact, the increase in packet transmission rates are readily quantified. Since BYE reconsideration basically uses forward reconsideration, but with BYE packets, the upper bound on the learning curve for forward reconsideration is also an upper bound on the BYE packet rate for BYE reconsideration. Thus, the BYE rate is limited to $\frac{1}{(1-\alpha)C}$. For the normal case of $\alpha = .5$, BYE packet rates are limited to $2/C$, twice the desired rate.

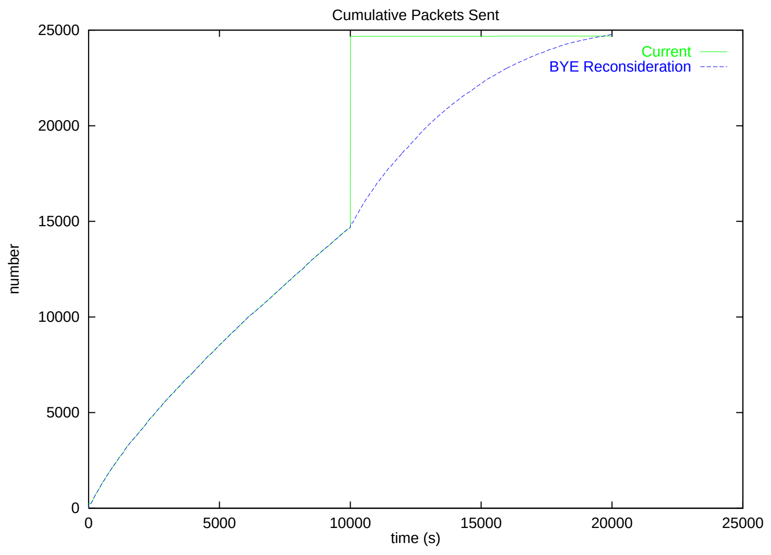


Figure 3.20: BYE Reconsideration Performance

3.9 Reverse Reconsideration

We have observed a secondary effect when many users simultaneously leave a group. There are many applications where not all of the users will leave, and some will stick around. An example is a distance learning application. There are perhaps several hundred students in the class. When the class ends, most of the students leave at about the same time. However, some stay behind to talk with the professor.

We have observed that rapid leaves can cause the remaining users to time each other out. It can take a significant amount of time for the users to return. This implies that each user will not see the other users, which is undesirable for the post-class discussion scenario just mentioned.

3.9.1 Quantifying the Problem

The difficulty is related to the way timeouts are handled. In the current RTP specification [2], a user is timed out if they have not sent an RTP or RTCP packet within the last five RTCP intervals. In dynamic groups, the interval itself is dynamic. As many users leave a group, their BYE packets cause the group size estimate to rapidly decrease. This, in turn, decreases the timeout interval. If the number of users who leave the group is sufficiently large, the timeout interval may decrease so much that the remaining users will time out. We call this phenomena *premature timeouts*.

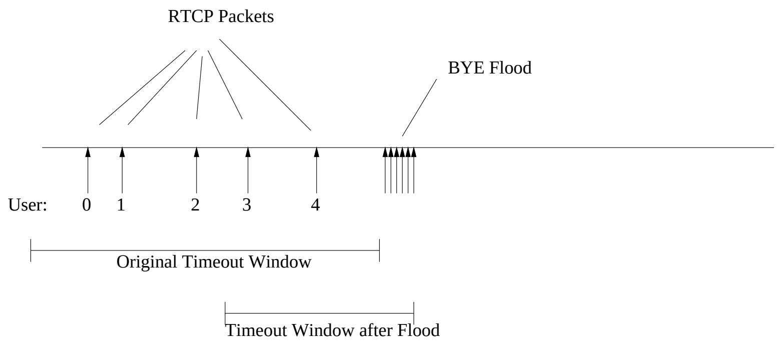


Figure 3.21: Premature Timeout Problem

The problem is depicted graphically in Figure 3.21. There are six users in the system. The chart depicts RTCP packet arrivals as seen by user 6. Before the BYE flood, the timeout window for user 6 contained RTCP packet receptions from every other user. However, after the flood, the window of user 6 has shrunk so that users 0, 1 and 2 have their last packets outside of the window. This will cause them to be timed out.

For example, consider a group of 505 users. If the total RTCP interval is to be limited to 1 packet per second, each user sends RTCP packets once every 505 seconds (on average). Assume user 1 last sent an RTCP packet at time 0. User 1 schedules the next RTCP packet for time 505. At time 490, 500 of the 505 members (not including user 1) leave the group, and send BYE packets (assume for the moment that there is no BYE flood prevention algorithm). Shortly thereafter (say time 500) the BYE packets have been received, and the remaining 5 users perceive the group size to be 5. Based on this, the timeout interval is 25 seconds. Any user who has not sent a packet since time 475 will therefore be timed out. User 1 last sent a packet at time 0, so user 1 is timed out. In fact, odds are good that most of the remaining 5 users sent their last packet before time 475. Thus, every user will time out all of the other users. Furthermore, it may take a long time for those users to come back. Consider user 2, who did not leave the group, and who was unfortunate enough to have last sent an RTCP packet at time 450. They then scheduled their next RTCP packet for time 955 (since there were still 505 users at the time). After the exodus at time 500, user 2 will remain, but will not send the next RTCP packet until time 955 (unless the user sends data, in which case they will be known via the RTP packet).

The first question to ask is whether BYE flood prevention helps alleviate this problem. Since the algorithm is designed to reduce the flood of BYE packets, the group size cannot drop so rapidly. This does help, of course, but not completely. With network delays, there still can be spikes in BYE packets. It does not require many BYE packets for this phenomenon to surface; it only requires that the ratio of users left after the leave to the number before the leave be less than around $1/5$. This can occur in both small and large groups alike.

Furthermore, we performed a simple analysis to help quantify the scope of the effect. Our analysis, like those above, is only approximate. Its main function is to help assess at what slopes of change in the group membership the phenomena becomes noticeable.

We start by assuming that the group membership begins decreasing at time $t = 0$ at a rate of r users per second. The initial group size is N_s, and the final group size is N_f, which is some fraction f of the initial group size, so that $N_f = fN_s$. Based on this, the learning curve can be expressed as:

$$L(t) = \begin{cases} N_s & t < 0 \\ N_s - rt & 0 < t < \frac{N_s(1-f)}{r} \\ fN_s & t > \frac{N_s(1-f)}{r} \end{cases} \tag{3.80}$$

At any point in time, the timeout window stretches from $t - MCL(t)$ to t. During this interval, each participant must have sent at least one packet, else they are timed out. On average, each user should have sent M packets during this window. In normal RTP operation, M is 5.

We consider two cases: long, gradual declines in group membership, and more rapid declines. Our definition of long, gradual declines are those where the timeout window is much shorter in duration than the amount of time during which the decline in group membership is linear. Short declines is where the timeout window is much larger than the time during which the decline in group membership is linear.

3.9.1.1 Long Declines

Mathematically, we can express the long decline case as:

$$t - MCL(t) > CN_s \tag{3.81}$$

For most of the duration of the decline. This condition guarantees that there is a period of time for which the packet scheduling is only based on the group membership during the linear decline. This means the equality must hold at least for $t = \frac{N_s(1-f)}{r}$. Using the expression for $L(t)$ from 3.80 in Equation 3.81, and evaluating the result at $t = \frac{N_s(1-f)}{r}$, we obtain:

$$rC < \frac{1-f}{1+fM} \tag{3.82}$$

To determine the impact of this decline on timeouts, we need to compute an estimate of the number of packets sent from each user during this window. Ideally, this quantity is M, but if it is substantially less, premature timeouts may occur.

To compute the number of packets sent by each user, we use a fluid model of packet transmissions. The packet rate from any user is ideally $1/CL(t)$ on average. However, we have observed that there is a delay between when group sizes change, and when this is reflected in the rate of packet transmissions. This delay is based on the interval size. So, the scheduled rate of packets from a particular user at time t can be given by:

$$d'(t + CL(t)) = \frac{1}{CL(t)} \tag{3.83}$$

This equation is similar to Equation 3.8. However, here we consider the scheduled rate from a single user (and thus the absence of the N from Equation 3.8), and we also consider the packet rate, not just the initial packet rate, as is done in Equation 3.8.

Combining with the expression for $L(t)$ from Equation 3.80, we can obtain an equation for the scheduled rate from a particular user:

$$d'(t) = \frac{1 - rC}{CN_s - rCt} \tag{3.84}$$

The total number of packets sent by the user during the window is the integral of the scheduled rate over the duration of the window, times the send probability. Since group sizes are declining, packets won't be reconsidered in conditional reconsideration, but they may be for unconditional. However, we make the simplifying assumption that packets are not reconsidered, so that the send probability is 1. In this case, the expression for the number of packets sent during the timeout window is:

$$N_{sent} = \int_{\frac{N_s(1-f)}{r} - MCN_f}^{\frac{N_s(1-f)}{r}} \frac{1 - rC}{CN_s - rCt} \tag{3.85}$$

This integral is readily evaluated to:

$$N_{sent} = \left(\frac{1}{rC} - 1\right) ln(1 + rCM) \tag{3.86}$$

Note that this quantity is positive only if rC is less than one. However, the conditions for long declines described in Equation 3.82 guarantee that this is the case.

We observe that in the limit as r goes to zero, N_{sent} approaches M, as it should. This means that when the group sizes are not declining, M packets are sent during the timeout window. It is also clear that the number of packets sent begins to decline below M as rC becomes significant. Unfortunately, our formulation is no longer valid once rC approaches 1. However, the formulation does indicate that there is a significant reduction in the number of packets sent in the timeout window even for gradual rates of decline. For example, with $rC = .5$, the number of packets sent during the window is 1.25 (assuming $M = 5$).

3.9.1.2 Rapid Declines

We also consider the number of packets sent during the timeout window when the rate of decrease in group membership is rapid. Here, we define rapid as the case where the amount of time it takes for the group membership to decline to N_f is less than the RTCP interval before the decline began. Mathematically, this implies:

$$\frac{Ns(1-f)}{r} < CN_s \tag{3.87}$$

Some simple algebra indicates that this occurs when $rC > 1 - f$. It can be readily demonstrated that in this case, the number of packets sent during the timeout window at $t =$

$\frac{N_s(1-f)}{r}$ is identical to the case of an instant departure. This is due to the inherent “delay” in the density function, where changes in group membership don’t get reflected in the transmission rate until some time later.

In the case of an instant departure, the group membership instantly drops to $N_s f$ from N_s at $t = 0$. Before this drop, and for a substantial time later, the rate of packets from any specific user is $1/CN_s$. After the drop, the timeout window is fN_sMC. The number of packets sent is readily computed as the product of the packet rate before the drop times the size of the window. The result is:

$$N_{sent} = fM \tag{3.88}$$

Consider once more the example where $rC = 0.5$. In the case of a long decline, we found that the number of packets sent was 1.25. The constraint of Equation 3.82 also implies an upper bound on f for long declines. In particular:

$$f < \frac{1 - rC}{1 + rCM} \tag{3.89}$$

With $rC = 0.5$, f must be less than 0.142. This implies that the number of packets sent in the case of a rapid decline with the same value of f is less than $fM = 0.714$ with $M = 5$. This result is interesting; the implication is that the number of packets sent rapidly decreases from its maximum of $M = 5$ to 1.25 as rC goes from 0 to 0.5. As rC continues to increase from 0.5 to infinity, the number of packets sent drops only slightly more, from 1.25 to 0.714. This means that the premature timeout phenomena quickly becomes important even for relatively slow declines.

3.9.2 Reverse Reconsideration Algorithm

One of the major factors contributing to the premature timeout effect is the delay between when the group size decreases, and when users begin to send packets using the resulting smaller interval. In the example of Section 3.9.1 user 2 sent an RTCP packet at time 450, and scheduled the next for time 955. After the exodus at time 490, user 2 knows that the group size has dropped, but does nothing. The user instead waits until time 955, sends the packet, and then computes the next send time. Since the group size is now 5, user 2 will schedule the next packet 5 seconds later, on average. There is thus a 500 second “delay” between the exodus and when user 2 gets around to scheduling a packet using the new, smaller interval.

To resolve this problem, we propose an algorithm called *reverse reconsideration*. The idea is simple. If the group membership decreases, each user reschedules their next packet immediately. The packet is rescheduled for a time earlier than it was scheduled previously. The amount earlier is made to depend on how much the group size has decreased.

More specifically, assume that the last time a user sent an RTCP report is t_p. The next report is scheduled for time t_n, and t_c is the current time. Before the arrival of a BYE packet at the current time t_c, there were n_p users. There are now n_c users. Before the BYE, RTCP packet transmissions should be uniformly scheduled over time. That means that there should have been n_c packet transmissions scheduled between t_c and $t_c + Cn_c$. Now, however, the group size has decreased to n_p. This should cause there to be n_p packet transmissions scheduled between t_c and Cn_p. To accomplish this, every user should compress the interval between the current time and their next packet transmission by n_c/n_p. This implies that the new next packet transmission time should be rescheduled for time t'_n:

$$t'_n = t_c + \frac{n_c}{n_p}(t_n - t_c) \tag{3.90}$$

This new value for t_n has two key properties:

1. The new time is always earlier than the previous time.

2. The new time can never be before the current time.

The second property is key; it guarantees that there will not be a spike of packets transmitted due to a sharp decrease in group size.

On the surface, it would seem that an alternate algorithm might be a direct application of regular (i.e., forward) reconsideration. Such an implementation might work as follows. At t_c, when the BYE arrives, the user recomputes the transmission interval T, based on the new group size n_c. This interval is then added to the previous packet transmission time t_p. If the result is a time before the current time, the packet is sent, else it is rescheduled for $t_p + T$. This algorithm does not work. Even a moderate decrease in the group size would cause many users to send their RTCP packets immediately, causing an additional spike. This is because this version of the algorithm does not maintain property 2; the new transmission time can be before the current time.

There is one additional aspect to the reverse reconsideration algorithm that must be considered, which is how it interacts with forward reconsideration. Consider the following example. There are 100 users in a group. The constant C is equal to 1 packet per second. At time 0, user A sends an RTCP packet, and schedules the next for time 100. At time 50, 50 users leave the group. User A executes the reverse reconsideration algorithm, and reschedules their packet for time 50 + (50/100)(100 - 50) = 75. At time 60, one more user joins the group. At time 75, user A executes the forward reconsideration algorithm (we assume conditional reconsideration). Since the group size has increased (51 vs. 50), user A recomputes the interval, which is now 51 seconds on average. This is then added to the previous transmission time, which is time 0. The result is $t = 51$, significantly earlier than the current time $t = 75$. This will cause the user (and in fact, all other users) to send their packets immediately, even if the group size further increases. The problem is that while we have adjusted the next packet transmission time, t_n, with reverse reconsideration, we have not adjusted the value of the previous packet transmission time, t_p. This quantity is used for forward reconsideration, and must be adjusted as well in order to maintain consistency.

The adjustment algorithm is simple. The value for t_p is updated when t_n is updated by reverse reconsideration. Its value is adjusted to t_p':

$$t_p' = t_c - \frac{n_c}{n_p}(t_c - t_p) \tag{3.91}$$

The nature of this adjustment is the same as for t_n. In the previous example, it would have caused t_p to be adjusted from 0 to (50 - (50/100)(50 - 0)) = 25. At time 75, when the user is performing the forward reconsideration algorithm, the interval $T = 51$ is added to $tp = 25$, yielding $t = 76$, slightly ahead of the current time $t = 75$, as expected (since the group has only increased by one member).

The complete algorithm is described in Figure 3.22. The algorithm is executed every time the group size changes.

```
$t_c$ := current_time
$n_c$ := current_group_size
if($n_c < n_p$) {
$t_n := t_c + \frac{n_c}{n_p}(t_n - t_c)$
$t_p := t_c - \frac{n_c}{n_p}(t_c - t_p)$
$n_p := n_c$
}
```

Figure 3.22: Reverse Reconsideration Algorithm

3.9.3 Performance

What is the improvement due to reverse reconsideration? It can be shown that the number of packets sent by a single user during the timeout window has an achievable lower bound of:

$$N_p = (1/rC) * \ln(1 + rCM) \tag{3.92}$$

This time, independent of the relative value of rC. We omit the proof for brevity; it closely follows the procedure used for the single user joins late computation of Section 3.7.3.6. Again, this holds only when the actual number of users who leave is a significant fraction of the current group size ($N_f/N_s < 1/(1+MCr)$). Assuming that BYE reconsideration is being used, Section 3.8 demonstrated that the BYE rate is limited to around $r = 2/C$ when network delays are small. Substituting this into Equation 3.92, we obtain:

$$N_p = 1/2 \ln 11 = 1.19 \tag{3.93}$$

This means that a user will send 1.19 packets on average, during the timeout window. Note that this only holds when $N_f/N_s < 1/(1+MCr)$. With $r = 2/C$ and $M = 5$, this implies $f < 1/11$. Without reverse reconsideration, N_p is equal to fM. For a fair comparison, considering the case with $f = 1/11$, $N_p = 5/11$. Therefore, reverse reconsideration affords a factor of three improvement in performance. Now, no users will timeout under normal circumstances (on average). However, a single packet loss may cause a user to timeout prematurely.

Our simulations confirm the performance gains from reverse reconsideration. Figure 3.23 depicts the learning curve $L(t)$ as seen by a single user present for the entire session. At time

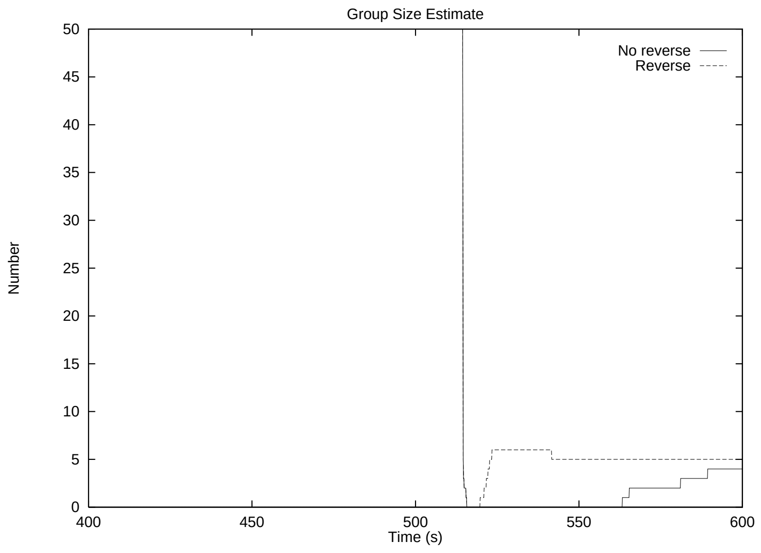


Figure 3.23: Group Size Estimate with Reverse Reconsideration

0, 105 users join the system, and all but five leave at time 500. Two curves are depicted, one for reverse reconsideration, and one without. The curve without reconsideration shows how the group size estimate drops to zero (the user does not include themselves in this estimate), and does not return to five for almost two minutes. However, with reverse reconsideration, the group size estimate returns to the correct value within a few seconds.

3.10 Group Sampling

As we have seen, the original RTCP transmission algorithms and our improvements with reconsideration both require a participant to maintain a group size estimate $L(t)$ of the number of participants in the group. In order to do this, a participant must maintain a table of SSRC values for other participants in the session. When an RTCP packet arrives, if the SSRC is in the table already, nothing is done. Otherwise, the SSRC is added to the table, and the group size estimate $L(t)$ is incremented by one. If a BYE packet arrives with an SSRC currently in the table, the estimate is decremented by one.

For large multicast sessions, such as an MBone broadcast or IP-based TV distribution, group sizes can be extremely large, on the order of hundreds of thousands to millions of participants. In these environments, RTCP may not always be used, and thus the group membership table isn't needed. However, it is highly desirable for RTP to scale well for groups with one member to groups with one million members, without human intervention to "turn off" RTCP when it's no longer appropriate. This means that the same tools and systems can be used for both small conferences and TV broadcasts in a smooth, scalable fashion.

Storage of an SSRC table with one million members, for example, requires at least four megabytes. As a result, embedded devices with small memory footprints may have difficulty under these conditions. To solve this problem, SSRC sampling has been proposed. SSRC sampling uses statistical sampling to obtain a stochastic estimate of the group membership. In this section, we discuss SSRC sampling, and in particular, focus on the issues in using it for highly dynamic groups.

3.10.1 Basic Operation

The basic idea behind SSRC sampling is simple. Each participant maintains a key K of 32 bits, and a mask M of 32 bits. Assume that m of the bits in the mask are 1, and the remainder are zero. When an RTCP packet arrives with some SSRC S, rather than placing it in the table, it is first sampled. The sampling is performed by ANDing the key and the mask, and also ANDing the SSRC and the mask. The resulting values are compared. If equal, the SSRC is stored in the table. If not equal, the SSRC is rejected, and the packet is treated as if it were never received.

The key can be anything, but is usually chosen to be the SSRC of the user who is performing the sampling.

This sampling process can be described mathematically as:

$$D = (K \wedge M == S \wedge M) \tag{3.94}$$

Where the $\wedge$ operator denotes AND and the $==$ operator denotes a test for equality. D represents the sampling decision.

According to the RTP specification, the SSRC's used by session participants are chosen randomly. If the distribution is also uniform, it is easy to see that the above filtering will cause 1 out of 2^m SSRC's to be placed in the table, where m is the number of bits in the mask, M, which are one. Thus, the sampling probability p is 2^{-m}.

Then, to obtain an actual group size estimate, L, the number of entries in the table N is multiplied by 2^m:

$$L = N2^m \tag{3.95}$$

Care must be taken when choosing which bits to set to 1 in the mask. Although the RTP specification mandates randomly chosen SSRC, there are many known implementations which do not conform to this. In particular, the ITU H.323 [82] series of recommendations allows the central control element, the gatekeeper, to assign the least significant 8 bits of the SSRC, while the most significant are randomly chosen by RTP participants.

The safest way to handle this problem is to first hash the SSRC using a cryptographically secure hash, such as MD5 [83], and then choose 32 of the bits in the result as the SSRC used in the above computation. This provides much better randomness, and doesn't require detailed knowledge about how various implementations actually set the SSRC.

3.10.1.1 Performance

The estimate is more accurate as the value of m decreases, less accurate as it increases. This can be demonstrated analytically. If the actual group size is G, the ratio of the standard deviation to mean of the estimate L (coefficient of variation) is:

$$\sqrt{\frac{2^m - 1}{G}} \tag{3.96}$$

This equation can be used as a guide for selecting the thresholds for when to change the sampling factor, as discussed below. For example, if the target is a 1% standard deviation to mean, the sampling probability $p = 2^{-m}$ should be no smaller than $.5$ when there are ten thousand group members. More generally, to achieve a desired standard deviation to mean ratio of T, the sampling probability should be no less than:

$$p > \frac{1}{1 + GT^2} \tag{3.97}$$

3.10.2 Increasing the Sampling Probability

The above simple sampling procedure would work fine if the group size was static. However, it is not. A participant joining an RTP session will initially see just one participant (themselves). As packets are received, the group size as seen by that participant will increase. To handle this, the sampling probability must be made dynamic, and will need to increase and decrease as the group size varies.

The procedure for increasing the sampling probability is easy. A participant starts with a mask with $m = 0$. Under these conditions, every received SSRC will be stored in the table, so there is effectively no sampling. At some point, the value of m is increased. This implies that approximately half of the SSRC already in the table will no longer match the key under the masking operation. In order to maintain a correct estimate, these SSRC must be discarded from the table. New SSRC are only added if they match the key under the new mask.

The decision about when to increase the number of bits in the mask is also simple. Let's say an RTP participant has a memory with enough capacity to store C entries in the table. The best estimate of the group is obtained by the largest sampling probability. This also means that the best estimate is obtained the fuller the table is. A reasonable approach is therefore to increase the number of bits in the mask just as the table fills to C. This will generally cause its contents to be reduced by half. Once the table fills again, the number of bits in the mask is further increased.

3.10.3 Reducing the Sampling Probability

If the group size begins to decrease, it may be necessary to reduce the number of bits in the mask. Not doing so will result in extremely poor estimates of the group size. Unfortunately, reducing the number of bits in the mask is more difficult than increasing them.

When the number of bits in the mask increases, the user compensates by removing those SSRC which no longer match. When the number of bits decreases, the user should theoretically add back those users whose SSRC now match. However, these SSRC are not known, since the whole point of sampling was to not have to remember them. Therefore, if the number of bits in the mask is just reduced without any changes in the membership table, the group estimate will instantly drop by exactly half.

To compensate for this, some kind of algorithm is needed. Two approaches are analyzed here: a corrective-factor solution, and a binning solution. Our results show that the binning solution generally outperforms the corrective factors solution.

3.10.3.1 Corrective Factors

The idea with the corrective factors is to take one of two approaches. In the first, a corrective factor is added to the group size estimate, and in the second, the group size estimate is multiplied by a corrective factor. In both cases, the purpose is to compensate for the change in sample mask. The corrective factors should decay as the “fudged” members are eventually learned about and actually placed in the membership list.

The additive factor starts at the difference between the group size estimate before and after the number of bits in the mask is reduced, and decays to 0 (this is not always half the group size estimate, as the corrective factors can be compounded, see below). The multiplicative corrective factor starts at 2, and gradually decays to one. Both factors decay over a time of $CL(t_s^-)$, where C is the average RTCP packet size divided by the RTCP bandwidth for receivers, and $L(t_s^-)$ is the group size estimate just before the change in the number of bits in the mask at time t_s. The reason for this constant is as follows. In the case where the actual group membership has not changed, those members which were forgotten will still be sending RTCP packets. The amount of time it will take to hear an RTCP packet from each of them is the average RTCP interval, which is $CL(t_s^-)$. Therefore, by $CL(t_s^-)$ seconds after the change in the mask, those users who were fudged by the corrective factor should have sent a packet and thus appear in the table. We chose to decay both functions linearly. This is because the rate of arrival of RTCP packets is constant, so that the group membership count should increase linearly as these users are learned.

What happens if the number of bits in the mask is reduced once again before the previous corrective factor has expired? In that case, we *compound* the factors by using yet another one. Let $f_i()$ represent the i^{th} additive correction function, and $g_i()$ the i^{th} multiplicative correction function. If t_s is the time when the number of bits in the mask is reduced, we can describe the additive correction factor as:

$$f_i(t) = \begin{cases} 0 & t < t_s \\ (L(t_s^-) - L(t_s^+))\,\frac{t_s + CL(t_s^-) - t}{CL(t_s^-)} & t_s < t < t_s + CL(t_s^-) \\ 0 & t > t_s + CL(t_s^-) \end{cases} \tag{3.98}$$

and the multiplicative factor as:

$$g_i(t) = \begin{cases} 1 & t < t_s \\ \frac{t_s + 2CL(t_s^-) - t}{CL(t_s^-)} & t_s < t < t_s + CL(t_s^-) \\ 1 & t > t_s + CL(t_s^-) \end{cases} \tag{3.99}$$

Note that in these equations, $L(t)$ denotes the group size estimate obtained including the corrective factors except for the new factor. If this were not the case, the above definitions would be recursive. t_s^- is the time right before the reduction in the number of bits, and t_s^+ the time after. As a result, $L(t_s^-)$ represents the group size estimate before the reduction, and $L(t_s^+)$ the estimate right after, but not including the new factor.

Finally, the actual group size estimate $L(t)$ is given by:

$$L(t) = N2^m + \sum f_i(t) \tag{3.100}$$

for the additive factor, and:

$$L(t) = N2^m \prod g_i(t) \tag{3.101}$$

for the multiplicative factor.

As an example, consider computation of the additive factor. The group size is 1000, C is 1 second, and m is two. With a mask of this size, a participant will, on average, observe 250 ($N = 250$) users. At $t = 0$, the user decides to reduce the number of bits in the mask to 1. As a result, $L(0^-)$ is 1000, and $L(0^+)$ is 500. The additive factor therefore starts at 500, and decays to zero at time $t_s + CL(t_s^-) = 1000$. At time 500, lets assume N has increased to 375 (this will, on average, be the case if the actual group size has not changed). At time 500, the additive factor is 250. This is added to $N2^m$, which is 750, resulting in a group size estimate of 1000. Now, the user decides to reduce the number of bits in the mask again, so that m=0. Another additive factor is computed. This factor starts at $L(t_s^-)$ (which is 1000), minus $L(t_s^+)$. $L(t_s^+)$ is computed without the new factor; it is the first additive factor at this time (250) plus 2^n (1) times N (375). This is 625. As a result, the new additive factor starts at 1000-625 (375), and decays to 0 in 1000 seconds.

3.10.3.2 Binning Algorithm

In order to more correctly estimate the group size even when it was rapidly decreasing, a binning algorithm can be used. The algorithm works as follows. There are 32 bins, same as the number of bits in the sample mask. When an RTCP packet from a new user arrives whose SSRC matches the key under the masking operation, it is placed in the m^{th} bin (where m is the number of ones in the mask), otherwise it is discarded.

When the number of bits in the mask is to be increased, those members in the bin who still match after the new mask are moved into the next higher bin. Those who don't match are discarded. When the number of bits in the mask is to be decreased, nothing is done. Users in the various bins stay where they are. However, when an RTCP packet for a user shows up, and the user is in a bin with a higher value than the current number of bits in the mask, it is moved into the bin corresponding to the current number of bits in the mask. Finally, the group size estimate $L(t)$ is obtained by

$$L(t) = \sum_{i=0}^{i=31} B(i)2^i, \tag{3.102}$$

where $B(i)$ are the number of users in the ith bin.

The algorithm works by basically keeping the old estimate when the number of bits in the mask drops. As users arrive, they are gradually moved into the lower bin, reducing the amount that the higher bin contributes to the total estimate. However, the old estimate is still updated in the sense that users which timeout are removed from the higher bin, and users who send BYE packets are also removed from the higher bin. This allows the older estimate to still adapt, while gradually phasing it out. It is this adaptation which makes it perform much better than the corrective algorithms. The algorithm is also extremely simple.

3.10.3.3 Comparison

We ran simulations to compare the performance of these algorithms. In the simulation, 10,001 users join a group at t=0. At t=10,000, 5000 of them leave. At t=20,000, another 5000 leave. All implement an SSRC sampling algorithm, unconditional forward and BYE reconsideration. When sampling is used, a memory size of 1000 SSRC was assumed.

Figure 3.24 shows the group size estimate as seen by the single user present throughout the entire session. The performance without sampling, and with sampling with the additive, corrective, and bin-based correction are depicted. The horizontal axis represents time in seconds, and the vertical axis represents the group size estimate $L(t)$.

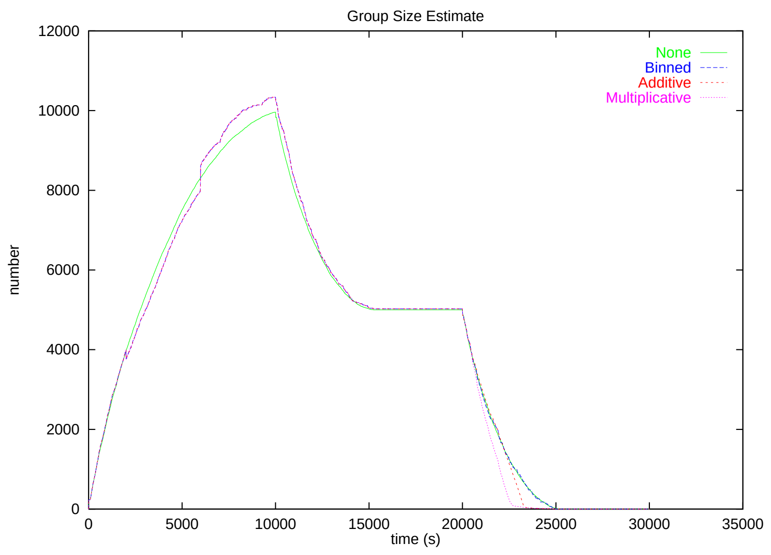


Figure 3.24: Comparison of SSRC Sampling Algorithms

From the graph, it can be seen that all three mechanisms do a fairly good job as estimating the group size throughout most of the session. However, during the period of rapid group size reduction (as a result of 10,000 users eventually leaving), the estimates from the corrective factors algorithms underestimate the group size. This is shown in more detail in Table 3.6, which tabulates the group size estimates from time 20,000 to time 25,000.

As the table shows, the binning algorithm comes closest to the exact (unsampled) group size estimate, particularly as the estimate drops below 1000. From the table, it can also be seen that the multiplicative correction factor consistently performs worse than the additive factor.

Time	No Sampling	Binned	Additive	Multiplicative
20000	5001	5024	5024	5024
20250	4379	4352	4352	4352
20500	3881	3888	3900	3853
20750	3420	3456	3508	3272
21000	3018	2992	3100	2701
21250	2677	2592	2724	2225
21500	2322	2272	2389	1783
21750	2034	2096	2125	1414
22000	1756	1760	1795	1007
22250	1476	1472	1459	582
22500	1243	1232	1135	230
22750	1047	1040	807	80
23000	856	864	468	59
23250	683	704	106	44
23500	535	512	32	32
23750	401	369	24	24
24000	290	257	17	17
24250	198	177	13	13
24500	119	129	11	11
24750	59	65	8	8
25000	18	1	2	2

Table 3.6: Group Size Estimate with Sampling Algorithms

3.10.4 Sender Sampling

Care must be taken in handling senders when using SSRC sampling. Since the number of senders is generally small, and they contribute significantly to the computation of the RTCP interval, sampling should not be applied to them. However, they must be kept in a separate table, and not be "counted" as part of the general group membership. If they are counted as part of the general group membership, and are not sampled, the group size estimate will be inflated to overemphasize the senders.

This is easily demonstrated analytically. Let N_s be the number of senders, and N_r be the number of receivers. The membership table will contain all N_s senders and $\frac{1}{2}^m$ of the receivers. The total group size estimate in the current draft is obtained by 2^m times the number of entries in the table. Therefore, the group size estimate becomes:

$$L(t) = N_r + (2^m)N_s \tag{3.103}$$

which exponentially weights the senders.

This is easily compensated for in the binning algorithm. A sender is always placed in the 0^th bin. When a sender becomes a receiver, it is moved into the bin corresponding to the current value of m, if its SSRC matches the key under the masked comparison operation.

3.11 Conclusions

In this chapter, we considered the problem of feedback for multicast multimedia sessions. We demonstrated the problems of congestion, latency, and state storage for the feedback mechanism currently defined in RTP. After an examination of other proposed solutions, we propose a set of algorithms broadly deemed *reconsideration*, which work well in adapting feedback in dynamic groups to control congestion. Through simulation and analysis, we demonstrated that reconsideration significantly reduces the congestion problem in RTP feedback. We also demonstrate how it remains backwards compatible with existing RTP mechanisms, in addition to meeting all the other requirements outlined for IP telephony.

We have also observed that our work has broader applicability than just feedback control for RTP. The reconsideration algorithms, at their core, provide a means for fair bandwidth sharing among a dynamic set of participants on a shared medium. As such, they can be considered a completely distributed form of fair queuing [84]. In fact, by adding simple weights to C which vary from participant to participant, our algorithms can implement a distributed form of weighted fair queuing. Such a distributed algorithm would has useful applications for bandwidth sharing on ethernets, distributed video rate control, and distributed gaming, to name a few. Our analysis is also applicable to these applications, as it effectively shows how close we come to true WFQ as a function of the network latency.

This chapter has also shown how statistical sampling can vastly reduce memory requirements for feedback, with almost no penalty in performance. Like reconsideration, this work has applicability to any distributed sampling application.

Forward reconsideration, BYE reconsideration, and reverse reconsideration have all been accepted by the IETF as improvements upon RTP. They have been folded into the specification [85] and will be part of the draft standard version of RTP. The SSRC sampling algorithms have also been accepted by IETF and have been published as an experimental RFC, RFC 2762 [86].

Chapter 4

Signaling Protocols

4.1 Introduction

So far, we have considered the problem of providing scalable Internet telephony from the perspective of the transport and transport control layers. As we have discussed in the introduction, Internet telephony is fundamentally different from traditional circuit switched telephony in that there is a complete separation of media transport from call control, signaling, and services, except at the end systems.

Central to call control, features, services and applications is the *signaling protocol*. Signaling protocols are responsible for the establishment and maintenance of calls, and the invocation and delivery of services. In the PSTN, protocols such as the ISDN User Part (ISUP) [87] have played this role, as part of the SS7 stack. We believe that protocols for signaling within the Internet can, and should, be significantly different in structure from those used in the PSTN. This difference is mandated by two architectural differences between Internet telephony and circuit switched telephony. First is the distributed nature of Internet telephony, and the second is the ability of Internet telephony to deliver *combined services* (also referred to as hybrid services by Gbaguidi [88]) that make use of web, email, instant messaging, and other Internet applications as components of voice and video communications services.

In this chapter, we investigate the problem of designing a signaling protocol for Internet telephony. We begin by discussing the requirements for such a protocol. This is followed by a review of existing work in this area, with an emphasis on if, and how, existing techniques meet the requirements we have outlined. This is followed by a presentation of a new protocol that we have co-developed, called the Session Initiation Protocol (SIP), which is currently standardized by the IETF in RFC 2543 [89]. Finally, we discuss an implementation of a SIP server that was built to enable these services, called *gosSip*, and then we conclude and discuss future work.

4.2 Requirements for a Signaling Protocol

Some of the required functionality for an Internet telephony signaling protocol has been outlined by Rosenberg and Schulzrinne [90]:

Name translation and user location: When one user wishes to communicate with another, they start with a location independent identifier for the other user. Examples of such identifiers are phone numbers and email addresses. It is the responsibility of the network to translate this identifier to an address which represents the network location(s) where the user is resident. We pluralize locations since, especially in the Internet, a user may be connected to the network through numerous devices, such as a desktop PC, a laptop, a PDA, and an IP telephone. Each of these devices differs in its capabilities and appropriateness for communications of different sorts. The signaling protocol needs to convey sufficient information to allow a user to be contacted, in a timely fashion, at the appropriate point(s) of contact.

Call state modification: The signaling protocol needs to allow all parties in a communications session to maintain a state machine reflecting call progress. This includes states like "ringing", "active" and "terminated". To support this, the protocol needs messages that can convey the events that causes transitions between these states in a reliable fashion, guaranteeing convergence of the states amongst all parties.

Media information exchange: The end systems need to agree on what media types to exchange (audio, video, text), which codecs to use for each media stream, and the parameters for those encodings. Since media is sent end-to-end, the end systems also need to exchange the IP addresses and ports for those media streams.

Media changes: It must be possible to adjust the composition of media sessions during the course of a call, either because the participants require additional or reduced functionality or because of constraints imposed or removed by the addition or removal of call participants.

User feature invocation: The signaling protocol must provide mechanisms for users to request the system to deliver call features, such as hold and call forward.

Network feature invocation: Many features and call services are not explicitly invoked by users, but are invoked by the service provider during processing of the call. As an example, call screening features can be provided by network servers. They are invoked by the provider during call processing, resulting in either rejection of the call, or completion of the call. The signaling protocol must provide sufficient capabilities for network-invoked features and applications.

Many other functions are needed for a complete communications management system. Examples include distributed queue management for floor control (needed only for multiparty conferences) and distributed counting for voting. We consider these outside the scope of signaling function because they are session specific. In our model, the signaling protocols are independent of the type of sessions they establish.

Perhaps the most critical component of a signaling protocol is its ability to deliver services. Services can range from the simple and mundane (call hold, call waiting, caller ID, call transfer) to extremely complex and valuable (unified messaging, interactive voice response, voice web browsing). We believe that one of the primary benefits of Internet telephony is its ability to deliver a wide range of new services. This ability stems from the existence of other applications on the Internet that can be used in conjunction with voice, video, and other communications mechanisms. These applications include web, email, instant messaging, and *presence*. Presence [91] is defined as the ability to *subscribe* to the communications state of another user, and then to receive *notifications* as that state changes. Presence systems have been in existence for quite some time on the Internet; Zephyr, for example [92], has been around since the late 1980s. At the time of writing, the America Online (AOL) presence service had close to 100 million users. Presence has traditionally been associated with instant messaging services, but its potential application is far broader than that.

We refer to services that combine web, email, instant messaging, presence, and other IP applications with communications as *combined services*. Chapter 6 gives numerous examples of such services. A simple one is Call Forward No Answer to Web (CFNA-W). In the PSTN, Call Forward No Answer (CFNA) is a widely used and simple service. Users of this service call a specific number, and if no one answers, the call is forwarded to some other number. In CFNA-W, users call a specific user, and if there is no answer, the user is returned a web page. This web page may, for example, provide information on how to send the callee email, or a URL to click on for sending voicemail.

Combined services are critical for Internet telephony, since they offer a source of revenue for Internet telephony service providers which is not possible for traditional circuit switched telephony providers. For consumers, Internet telephony started out primarily as a tool for cheap long distance and international calls. Service providers used it for arbitrage of local access charges. However, cost differentials often disappear over time as industries evolve, and so Internet telephony must have a value proposition beyond “cheap”.

Providing combined services requires a signaling protocol with the primitives needed to deliver them. It also requires a service architecture that can take advantage of the capabilities of the protocol to actually provide the services. In this chapter, we consider the primitives that need to be built into the signaling protocols. In Chapter 6, we consider how to build a service architecture around these primitives.

4.3 Existing Signaling Protocols

In this section, we consider the existing body of work on Internet telephony signaling protocols. Not surprisingly, much of this work has been done by standards bodies. We focus here on the Bearer Independent Call Control (BICC) protocol [93], developed by the ITU and the H.323 suite of protocols [94], also developed within the ITU.

4.3.1 BICC

The most widely deployed signaling protocol to date is the ISDN User Part (ISUP) [87], which is the network to network protocol used between telephone switches as part of the digital Signaling System 7 (SS7) network. ISUP provides many of the functions described in Section 4.2, which are independent of the actual bearer circuit used to carry the call. A few components of ISUP, however, are specific to creation of bearer circuits, and thus ISUP is not appropriate in its native format for Internet telephony signaling. However, this is being rectified by recent activity within ITU called Bearer Independent Call Control (BICC) [93], which is aimed at extracting the call control components from ISUP, and allowing other bearers, such as ATM and RTP/UDP/IP, to be used instead of circuits.

Clearly, BICC can provide support for name translation and user location, call state modification, user feature invocation, and network feature invocation, purely as a consequence of being based on ISUP. Its ability to support media changes and media capability negotiation are dependent on what support it has for establishment of RTP bearers. At the time of writing of this dissertation, that work was in its early stages.

BICCs strength is ease of interoperability with the PSTN, and the ability to transparently deliver existing circuit switched features and services to IP endpoints.

BICCs primary drawback is that it it cannot deliver any services beyond existing circuit switched features and applications. This includes combined services. As we discuss in Section 4.4.8, supporting combined services requires support for MIME object transport and URL addressing, amongst other capabilities. These are absent in BICC.

4.3.2 H.323

The ITU has also developed the H.323 series of recommendations [94]. H.323 was originally specified as a signaling protocol for packet communications on LANs with no support for QoS. However, version 2 expanded its scope to cover the Internet specifically, and it gained widespread support as the industry standard for Internet telephony signaling. Version 4 of H.323 [95] was approved at the end of 2000.

H.323 is actually an umbrella specification, covering a suite of recommendations that define a complete system for packet-based multimedia communications. These recommendations include:

H.225.0: H.225.0 [96] covers basic call establishment and termination, registration, admission, and status (RAS), and media transport (which uses RTP).

H.245: H.245 [97] covers session control, including capabilities negotiation, logical channel signaling, conference control and floor control.

H.235: H.235 covers security for H.323 systems [98].

H.246: H.246 [99] covers interoperability with other multimedia systems, include H.324 (PSTN multimedia) and H.320 (ISDN multimedia).

H.450: H.450 is a series of recommendations itself, each of which covers a particular supplementary service. Examples include call transfer, call diversion, call hold, and call park. H.450.1 [100] outlines the framework for the series.

The central element in the H.323 network is the gatekeeper. It is responsible for call routing, call management, endpoint management, and overall resource management for the collection of terminals and gateways under its control (the collection of which is referred to as a *zone*).

A complete treatment of H.323 is well beyond the scope of this dissertation, as the sum total lengths of the specifications themselves are well into the thousands of pages. Toga and Ott provide a good tutorial [101], as does Toga and ElGebaly [102].

The strengths and weaknesses of H.323 (especially compared to SIP) have been documented by Rosenberg and Schulzrinne [103], Pagurek and White [104], Agboh [105] and Sisalem [106]. H.323's strengths are its support for video conferencing systems. Its capability negotiation features are well beyond those provided by SIP and SDP, and these are important for video codecs. H.323 provides some specific features for lip synchronization and playout buffer management that are absent in SIP. Levin documents several of the missing capabilities for video in SIP, which are present in H.323 [107]. It is worthwhile to note that most are issues with SDP or RTP, and not with SIP itself.

H.323's drawbacks are its substantial (and continually growing) complexity, and the wealth of different ways to accomplish the same function, both of which have led to interoperability problems. Its central network element, the gatekeeper, needs to maintain call state in order to participate in call signaling, which hampers scalability and fault tolerance. It lacks advanced user location functions, such as forking, which allow a call to ring numerous phones simultaneously. Its model for QoS is that the gatekeeper manages the capacity of the network. Primitives are provided for admission and bandwidth control. These are not useful outside of LAN-based IP telephony, since the gatekeeper cannot determine capacity for media calls in a general purpose IP network. H.323's extensibility relies on protocol versioning and vendor specific attributes scattered throughout the protocol, which is very limited.

Most importantly, H.323 has difficulty delivering new services. The standardized vehicle for services is the H.450 series of recommendations, which provide only the most basic features. A specification is required for every feature. This limits the speed at which features can be added, and limits vendor creativity. Mechanisms for third party call control are absent, and these are critical for services, as we shall see in Chapter 4. H.323 is also unable to delivery many combined services. As with BICC, it lacks support for MIME transport and ubiquitous URL addressing[1].

4.4 SIP Overview

To resolve the limitations of existing signaling protocols, we co-developed, along with colleagues from Columbia University, Aciri and Caltech, the Session Initiation Protocol (SIP), which is currently an IETF proposed standard, RFC 2543 [89]. Numerous papers describe the functions and capabilities of SIP [108, 109, 90, 110, 111], including a book [112].

We provide here only a brief review of SIP, focusing on the capabilities that resolve the issues we have raised above.

SIP is used to establish, change, and tear down calls between one or more endpoints in an IP-based network. It is based heavily on the Simple Mail Transfer Protocol (SMTP) [113, 114],

[1]H.323 alias addresses can include URLs. However, there is no documentation on the usage of URLs in these fields. Furthermore, alias addresses are not used throughout the specifications. Call transfer, for example, is not based on alias addresses, and cannot use a URL

the basis for email, and the HyperText Transfer Protocol (HTTP) [115], the basis of the web. SIP was also designed to leverage off the work on distributed conferencing services [116, 117, 118].

Like HTTP, SIP is a textual client-server protocol, with requests issued by the client and responses returned by the server. SIP actually reuses much of the syntax and semantics of HTTP, including its message structure, response code hierarchy, authentication framework, and client-server operation. SIP is used for IP telephony functions by mapping each function to one or more *transactions*. A SIP transaction consists of a single request issued by a client, and one or more responses returned by one or more servers. SIP transactions are *idempotent*, just like HTTP transactions. Also like HTTP, each SIP request is an attempt to invoke some *method* on the server. RFC 2543 [89] defines six SIP methods. The most important of these is the INVITE method, which is used to initiate a session between the client and the server.

Unlike HTTP and SMTP, SIP can run on top of either TCP or UDP. In the case of UDP, the protocol provides its own mechanisms for reliability. The use of UDP also means that the SIP messages can actually be multicast. Multicast allows for, among other features, group invitations and basic Automatic Call Distribution (ACD) functions that don't require a distribution server. The use of UDP also provides for fast operation (avoiding the TCP SYN handshake) and better scalability (no need to maintain TCP connection state in the kernel). When used with TCP, SIP allows many requests and responses to be sent over the same TCP connection, as in HTTP 1.1.

4.4.1 Protocol Components

There are two components in a SIP system, user agents and network servers. A user agent is an end system which acts on behalf of some person or automata that wishes to participate in calls. A user agent contains, in general, both a protocol client (called a User Agent Client, or UAC) and a protocol server (called a User Agent Server, or UAS). The UAC is used to initiate a call, and the UAS is used to receive a call. The presence of both in a user agent allows for peer-to-peer operation using a client-server protocol.

In addition to user agents, SIP provides for network servers. These servers are of three different types, namely *proxy*, *redirect* and *registrar*. A SIP proxy acts in much the same way as an HTTP proxy or an SMTP Message Transfer Agent (MTA). It receives a request, makes a determination about the next server to send it to, and forwards the request, possibly after modifying some of the header fields. A SIP proxy has no way of knowing whether the next server to receive the request is another proxy server, redirect server, or user agent server. As such, SIP requests can traverse many servers on their way from UAC to UAS. Responses to a request always travel along the same set of servers the request followed, but in reverse order.

A redirect server receives requests, but instead of forwarding them to the next hop server, it tells the client to contact the next hop server directly. It does this by responding to the request itself using a *redirect response*, containing the address(es) of the next hop server(s). This is analogous to iterative searches in the Domain Name System (DNS) [119, 120], just as a proxying is analogous to recursive searches.

A registrar receives SIP REGISTER requests. These requests are used to establish location state that is accessible by proxy and redirect servers for the purpose of request routing. A UAC will periodically send REGISTER messages, binding the logical name of the entity represented by the UA (e.g., jdrosen@columbia.edu) to the host where the UA is present (e.g., jdrosen@1.2.3.4).

It is important to note that the roles of user agent, proxy, redirect server and registrar are *logical* ones. That is, a physical box may take on different roles for different calls, depending on the policy and service requirements.

4.4.2 SIP Network Servers

The main function of a SIP network server is to provide for name resolution and user location, or more generally, call and request routing. When a user wishes to place a call, it must send an INVITE request to the UAS of that user. However, the caller in general will not know the IP address or hostname of the UAS for the given user; it will have only some name which represents the caller (usually an email address, but it can be a telephone number or some other local identifier). Using this name, the UAC can determine a network server which may be able to resolve the name to an IP address. This network server may, in turn, proxy or redirect the call to additional servers, eventually arriving at one which definitively knows the IP address the user is to be contacted at. The process of determining the next-hop server is known as *next-hop routing*. SIP provides facilities for loop detection and prevention.

A SIP network server can use any means at its disposal to determine the next-hop server. These include DNS, accessing databases, executing programs, and prompting users. The final UAS contacted by the caller is determined by the composition of the decisions made at all the servers from callee to caller. It is this fact which makes SIP servers the basis for powerful mobility and forwarding services.

A SIP network server may determine, as a result of its next-hop routing decision, that there are actually multiple next-hop servers which may be able to eventually contact the user. SIP allows for a proxy server to *fork* an incoming request, sending it to multiple next-hop servers. Each of these *branches* may generate responses; SIP provides rules for merging and passing back the best response(s) upstream, towards the UAC. Two different types of forking are specified, namely parallel and sequential. In parallel forking, the proxy sends a request to multiple next hop servers at the same time. This allows for multiple phones to ring at once, for the same user. In sequential forking, the first address is contacted, and if it does not generate a successful response, the next address is tried.

Each SIP transaction can take a different path through servers in the network. In a typical call, the first request is an INVITE, which may traverse many network servers on its way to the callee. The response to the INVITE contains an address that the UAC can use to communicate directly with the UAS. The implication of this is that SIP network servers do not need to maintain call state. Once a transaction is complete, a SIP server has no recollection of the caller or callee. This results in good scalability and reliability. A SIP server can crash and recover (or a backup swapped in), and none of the calls initiated through it are affected. This also means that the duration of and amount of state maintained at a server is small in comparison to the PSTN, where a switch must maintain call state for the entire call duration.

Interestingly, it is not even required for a SIP network server to be stateful during a transaction. A proxy or redirect server can be completely stateless. This means it receives a request, and either generates a response, or proxies the request, and then forgets everything. Proxied requests contain all the state required for correct protocol operation embedded within the message. This is nicely aligned with the Internet datagram architecture where packets contain sufficient information to be individually routed. Furthermore, a stateful proxy can decide to become stateless at any time during the transaction, and the system still operates correctly. The decision to be stateless or stateful can be made on a call-by-call basis as desired by the administrator. This allows for large, central SIP servers to be stateless, but for smaller, localized servers to be stateful.

Figure 4.1 depicts a typical SIP deployment of network servers, and the message flow between them. Here, there are three domains (A, B, and C), each of which has a single SIP server acting as an access point into and out of their networks. A user agent, Joe, in domain A makes a call to another user, Bob. The call invitation is forwarded to domain A's access server (1), which attempts to find the callee in domain B and C by forking the request. The request arrives at domain C (3), but the user is unknown at this location, so an error response is returned (4). The request at domain B (2), however, is forwarded to a local server internal to domain B (5), where it finally reaches the UAS (6). The response is then forwarded back along the same path to the caller (7,8,9,10).

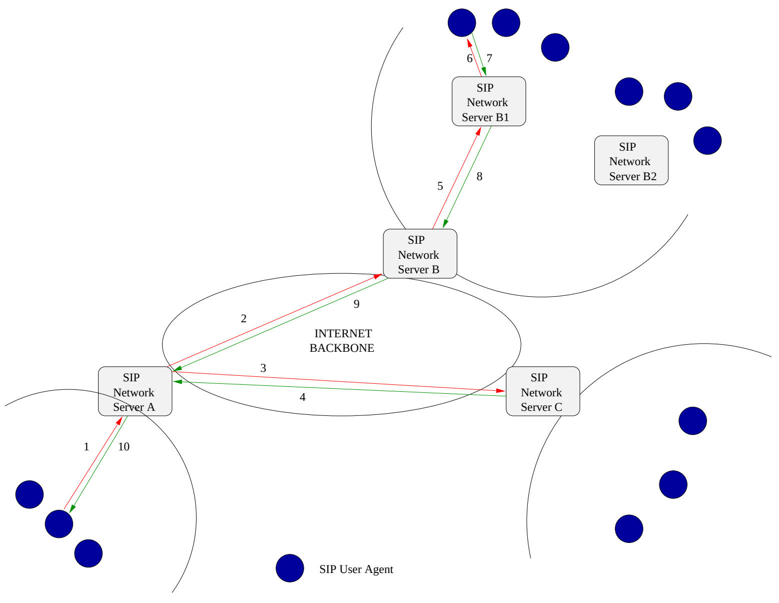


Figure 4.1: Typical SIP deployment

Recent work on SIP has described a network component referred to as a *back-to-back user agent*, or B2BUA [121]. This element is a completely stateful device which acts as a UAS for calls it receives, and then re-initiates them to the callee, acting as a UAC. A B2BUA enjoys none of the scale and fault tolerance benefits that a proxy does, but is capable of owning complete control over the call, much like a PSTN switch (even though the media does not flow through it).

4.4.3 SIP Messages

SIP messages are either requests or responses. Messages are sent as text. Requests start with a *request line*, and responses start with a *status line*. Both message types are followed by a series of headers, each of which is a header name, followed by a colon, followed by a header value. Headers are terminated by a carriage-return and line-feed. After the headers, the messages contain an optional body, separated from the headers by a blank line. Headers are used to convey information needed by SIP entities for processing of the request or response. The bodies are opaque to SIP, and can contain anything. Frequently, they contain Session Description Protocol (SDP) [81] messages, as we discuss below. However, they can contain any type of content.

```
INVITE sip:ann@lucent.com SIP/2.0
Via: SIP/2.0/UDP 131.215.131.13;maddr=239.112.3.4;ttl=16
Via: SIP/2.0/TCP 10.0.1.1;received=128.13.44.52
From: John Smith <sip:jsmith@lucent.com>
To: Arun Netravali <sip:ann@lucent.com>
Subject: Raise
Call-ID: 132059753@mypc.domain.lucent.com
Content-Type: application/sdp
CSeq: 4711 INVITE
Content-Length: 187

v=0
o=user1 51633745 1348648134 IN IP4 128.3.4.5
s=Interactive Conference
c=IN IP4 224.2.4.4/127
t=0 0
m=audio 3456 RTP/AVP 0 22
a=rtpmap:22 application/g723.1
```

Figure 4.2: Typical SIP INVITE message

A typical SIP INVITE request message is shown in Figure 4.2. The request line in a SIP request message contains the method, the Request-URI, and the protocol name and version (which is always SIP/2.0). The Request-URI plays a central role in SIP. It identifies the target of the request at the next-hop server. The request method specifies the processing desired by the client. The SIP specification [89] defines several methods. These are INVITE, BYE, OPTIONS, ACK, REGISTER and CANCEL. INVITE is used to invite a user to a session, or to modify the media components of an existing session. BYE is used to “hang up”, or terminate a session. OPTIONS is used to solicit information about the capabilities of the callee, but does not set up a call. ACK is used for reliable message exchanges. REGISTER creates address bindings, and is discussed in Section 4.4.6. CANCEL is used to terminate a pending request (almost always an INVITE), but does not undo a completed request.

Responses are formatted almost identically to requests. They differ in the first line. This line, called the *status line*, conveys a *response code* and a *reason phrase*. The response code is a numerical code that indicates the results of the request processing. Responses codes are integers from 100 to 699. Only the hundreds digit is significant as far as mandatory processing rules are concerned (this follows HTTP 1.1 operation [115]). The reason phrase is a textual string, fit for display to a human user, that describes the results of the request processing. Response codes from 100 to 199 are known as *provisional responses*. These responses contain optional progress information. They are delivered unreliably from the UAS to the UAC (although we have defined a SIP extension which allows them to be delivered reliably [122]). An example provisional response code is 183, which usually carries a reason phrase of “Call Progressing”. Another example is 180, which usually carries a reason phrase of “Ringing”. A UAS can send as many provisional responses as it likes. However, it can send only a single *final response*. Final responses, which have response codes from 200 to 699, complete the transaction. Codes 200-299 indicate success (i.e., the call was accepted, in the case of INVITE). Codes 300 to 399 indicate redirection. Codes 400-499 indicate client error (such as a malformed request). Codes 500-599 indicate server failure (such as overload). Finally codes 600-699 indicate global failure.

As SIP is a textual protocol, generation and parsing of its messages is easily done with text processing languages such as perl. Furthermore, its compliance to RFC822 [123] formatting rules means existing HTTP or SMTP parsers can be used directly for lexigraphic analysis of messages. It also simplifies debugging, development, and extensions.

4.4.4 Addressing and Naming

To be invited and identified, the callee has to be named. Since it is the most common form of user addressing in the Internet, SIP defines its addresses as URLs [124], similar in structure to the mailto URL [125]. These are generally of the form “sip:user@address”, where the address can either be a valid DNS name, or an IP address. When identifying a user at a specific host, the address will typically be a hostname or IP address. When used as a location independent identifier, the address is usually a domain name. The user portion of the URL identifies the user, process, system, or service at the specific address. The user portion can be a telephone number (useful when the address is that of a gateway to the PSTN), or any character string.

The use of email-style addresses as SIP addresses enables a scalable means by which a UAC can deliver a request to a SIP server which likely knows how to forward the request to the final callee - the Domain Name System (DNS). By using a series of DNS lookups, searching for service (SRV), canonical name (CNAME), and address (A) records, the caller can determine the address of a server which has naming authority for all users within the domain [126].

4.4.5 Initiating, Modifying, and Terminating Calls

To initiate a call, the UAC formulates an INVITE request. The From header is populated with the SIP URL identifying the caller, and the To header is populated with the SIP URL identifying the callee. The Contact header is populated with the address to be used for SIP message exchanges for the rest of the call (similar to the Reply-To header in email). The Request-URI is also populated with the SIP URL identifying the callee. The body of the request might contain SDP (see Section 4.4.7). The request is then sent to the host identified by the address in the SIP URL in the Request-URI. This host is likely a proxy server. The proxy server will make a routing decision, and determine the next-hop server to forward the request to. It then rewrites the Request-URI to contain the URL identifying the resource to contact at the next-hop server. As the request traverses through the proxy network, the Request-URI is rewritten by each proxy. As a result, it is the Request-URI which is used as the input to the routing and policy decision process at each element, including proxies and the UAS.

Once the INVITE arrives at the UAS, the end user or service is alerted about the incoming call. If the call is accepted, a 200 “OK” response is generated. This response contains copies of the To and From headers from the request, amongst others. The UAS also adds the Contact header, which indicates the address to use for message exchanges for the remainder of the call. If the call is not accepted, a 400-class response (meaning any response from 400-499) is returned. Typically, the 486 “Busy Here” status code is returned to indicate that the callee is busy. A 200 OK response will almost always contain SDP. Finally, the caller generates an ACK request to acknowledge receipt of the response. The ACK may also contain SDP.

To deliver these messages reliably, the UAC retransmits the INVITE periodically (with an exponentially increasing retransmission interval), until receipt of a provisional response. At that point, retransmissions cease. When the final response is sent by the UAS, it retransmits the response periodically (also with an exponentially increasing retransmission interval), until receipt of the ACK.

At any time during the call, either the caller or callee can issue a re-INVITE. A re-INVITE is not a new SIP method. It is simply a regular INVITE that just happens to be sent in the middle of a call. It is used to update the media session associated with the call. New media sessions can be added, existing ones can be removed, and the IP address and ports for a particular media session can be changed.

Donovan has defined the INFO method [127] as a new SIP request message that can also be issued during a call. It is used to exchange application related information. The semantics depend on the headers any body in the request. For example, the Call-Info header, which can be present in any request or response message, informs the recipient of the message to render the content at the URL present in the header. If an INFO request contains this header, the recipient displays the content. The INFO method is also used to assist in interoperability with the telephone network [128].

To hang up the call, either caller or callee can issue a BYE request. The recipient responds with a 200 OK, and the call is terminated.

4.4.6 Registrations

The SIP REGISTER request message is used to convey location information to a SIP server, known as a registrar. Generally, the registrar is co-located with a proxy server. It acts as a repository of information on how to route calls. When a call arrives at a proxy, it can route the request based on static or programmed policies, or it can look up the Request-URI in the registrar's database, and determine the next-hop address(es) to use.

In the most general sense, the REGISTER request is a method for installing address mappings into the registrar. The address in the To field of the REGISTER request is mapped to the addresses in the Contact headers of the registrations. The Contact headers can also contain information to assist the proxy in selecting an address to use. Specifically, RFC 2543 defines the *q* parameter as a numerical value, from 0 to 1, that indicates a preference for a specific address. A *q* value of one indicates the highest preference. A proxy will generally try the highest preference address first, and if the user is not there, try the next lowest priority.

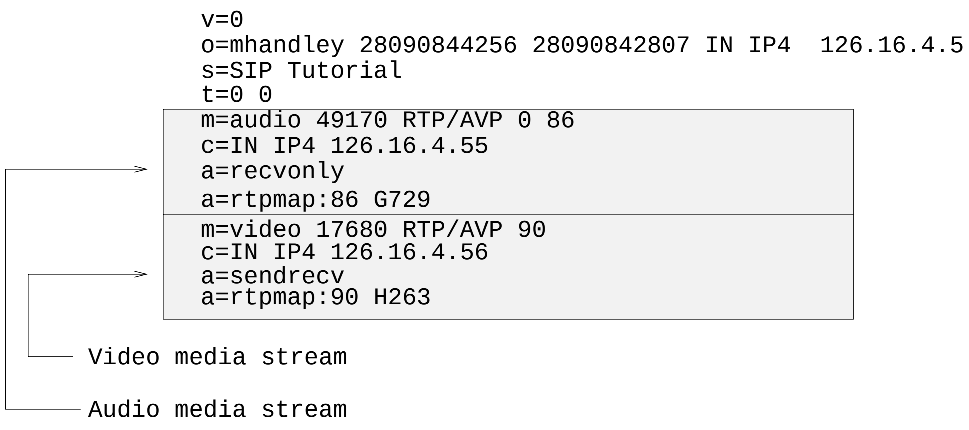
The address mappings installed by the REGISTER request are soft-state, and must be refreshed by the UA. As a result of this, the REGISTER message also acts as a form of connectivity status. If a user has an active registration, they are "online" and connected to the SIP system. If they do not have an active registration, they are not connected, and thus not available to receive calls.

Since REGISTER is a normal SIP request, it can contain a body, as can the response to registrations. The usage of bodies in REGISTER has been proposed as a means to upload call processing logic to proxy servers [129]. Furthermore, the body of a REGISTER response can contain configuration information useful to the user agent. These may include speed dial button configurations, additional addresses (allowing for PBX-like functionality, whereby the server chooses the addresses used by each client), or a call log. Using the multipart MIME formatting rules, and the capability negotiation features of HTTP, new types of information can be added as time passes.

4.4.7 Session Description Protocol Usage

The Session Description Protocol (SDP) [81] is not really a protocol at all. Rather, it is a syntax for a document that is used to describe multimedia sessions. It was initially applied to describe conferences on the MBone, by conveying it within Session Announcement Protocol (SAP) messages [68]. It later found application to unicast sessions within SIP.

A simple SDP message is shown in Figure 4.3. It is composed of a series of lines, each separated by a carriage-return line-feed. Each line is a single letter, representing a parameter,



```
v=0
o=mhandley 28090844256 28090842807 IN IP4  126.16.4.5
s=SIP Tutorial
t=0 0
m=audio 49170 RTP/AVP 0 86
c=IN IP4 126.16.4.55
a=recvonly
a=rtpmap:86 G729
m=video 17680 RTP/AVP 90
c=IN IP4 126.16.4.56
a=sendrecv
a=rtpmap:90 H263
```

Figure 4.3: SDP message example

followed by an equal sign, followed by the value of the parameter. The important sections of the SDP are in the shaded boxes. Each box begins with an "m line" which describes a media stream in the session. It indicates the type of stream, followed by the port number it is to be sent to, followed by a transport indicator (which is almost always RTP/AVP, indicating usage of RTP and its audio-visual profile [130]). After that is a listing of numbers, indicating the payload types supported for the stream.

The "c line" contains the connection address, and indicates the IP address where the media should be sent to. SIP allows this to be 0.0.0.0, indicating that media should not be sent at all. This is used to enable media-on-hold. An SDP with a c line indicating an address of 0.0.0.0 is referred to as "held SDP".

Streams can be send-only, receive-only, or bidirectional (sendrecv), as indicated by the "a line" which contains many different types of attributes. When used with SIP, a stream that is unidirectional will be send-only for one side, and receive-only for the other.

The rtpmap attribute is used to support *dynamic payload types*, where the mapping from an RTP payload type number to a specific codec is signaled for the call. For *static payload types*, the mappings are specified in RFC 1890 [130]. For example, payload type 0 is PCM μ-law.

SIP usage of SDP is based on a two-way offer-answer model. One side of the call "offers" an SDP for the session, and the other side "answers". Typically, the offer is carried in the INVITE, and the answer in the 200 OK. However, it is allowed for no SDP to be present in the INVITE, in which case the offer is made in the 200 OK, and the answer in the ACK.

The SDP for the answer has to follow a set of guidelines for its construction. For each m line in the offer, there must be a corresponding m line in the answer. If a media stream in the offer is to be rejected, the port number for the media stream in the answer is set to 0. The set of codecs listed for a stream in the answer has to overlap with the set of codecs offered in the offer. If a stream is offered as send-only, the answer must be receive-only, and vice-a-versa. The IP address and port for the stream in the offer indicate where the offering-party would like to receive media for the stream. The IP address and port for the stream in the answer indicate where the answering-party would like to receive media for the stream.

As the body is opaque to SIP, but described using MIME [131] syntax and semantics, SIP can use other media description formats besides SDP. Possibilities include H.245 capability descriptors, SMIL [132] presentation descriptions, or an XML [133] formatted description of a new video codec. As these and other descriptions [134] appear, they are as easily incorporated into SIP as a new image type is incorporated into HTTP.

4.4.8 SIP as a Tool for New Services

We have argued that the weakness of H.323 and BICC is their inability to readily provide new services. In particular, these protocols cannot provide services that combine voice and video with web, instant messaging, presence, and email. SIP, however, has several components that make it ideally suited to provide these kinds of services.

Broadly speaking, SIP supports combined services because of two features it provides. One is its support of MIME, and the other is its ubiquitous support for URLs and URIs as an addressing format. These are explained in detail below.

4.4.8.1 MIME

MIME stands for Multipurpose Internet Mail Extensions, and is specified in RFC 2045 [135]. MIME is now used in a number of other protocols, including HTTP. It has become the defacto standard for describing all sorts of content on the Internet. Nearly every audio format, every video type, and every image encoding is a registered MIME type. The MIME headers allow the type of the data, its encoding, and its size to be described. MIME headers also allow clients and servers to negotiate the types of content they understand. MIME also allows for multipart, which means email or a web response can contain multiple MIME objects. MIME also enables security; a protocol called S/MIME [136] allows an email message to contain a MIME part that contains a signature. At its core, MIME is important because of its sheer ubiquity; any type of data which can be reasonably carried over the web or in email is registered MIME type.

SIP uses MIME. SIP messages carry bodies, just like web and email do. In the simplest case of a basic call setup, the body contains information, encoded in the SDP, on the audio codecs, RTP ports and IP addresses used for the media. However, SIP messages can also contain other types of bodies; any registered MIME type is allowed. When used with SIP, these bodies can be used to convey a variety of important pieces of information:

Display data: The content contains information for display, to provide additional information related to features, services, or call signaling. The receiving SIP entity should optionally display it using the default viewer for that MIME type. As an example, a SIP INVITE message might contain a JPEG image for display, thus enabling a visually enhanced caller ID. A SIP INVITE message might also contain a vCard [137] to identify the caller.

Billing data: The information can be for billing or settlement purposes. It can contain tokens (such as an Open Settlement Protocol (OSP) token [138]) which authorize the user to make a call.

Signaling data: The information can convey additional signaling data. For example, when making a call from the PSTN to the PSTN with IP serving as the toll bypass, the SIP messages can contain ISUP messages. This allows for transparent operation of SS7 telephone network signaling, yet it also allows call completion to non-ISUP aware devices [128].

Code: The data in a SIP message can be code to execute in order to provide some enhanced service [139, 129].

URLs: The information can be a list of URLs that point to any of the above pieces of data.

Of course, SIP bodies might be used for additional purposes down the road. Using the MIME headers for negotiating supported types, backwards compatibility can be maintained even as new uses are discovered.

MIME is an important enabler of new services. Because many combined services involve web and email, enabling them, at the core, comes down to allowing SIP systems to add, process, and transmit MIME content.

As an example, the simple combined service we described in Section 4.2, CFNA-W, is enabled by having a SIP server return HTML content in a no-answer response.

4.4.8.2 URIs

A URI is a Uniform Resource Identifier [124]. Many users are familiar with the URL, which is a Uniform Resource Locator. A URI is a generalization of a URL. A URI identifies, in some way, a resource. A URL identifies a resource by its location, specifically, what host and path it exists on. Another way to identify a resource is by its name. For example, a book can be defined by its ISBN number. Uniform Resource Names, or URNs [140], are another type of URI which identify resources in this way.

URIs have become the standard Internet way of application-layer addressing. Web pages are identified by HTTP URLs. Email is sent by clicking on mailto URLs. As new applications are developed, they define URI types as well. As a result, URIs have become the universal way to access Internet applications.

Rather than defining some new type of address space, SIP uses URIs, and only URIs, for addressing. Despite the fact that SIP has its own URI, SIP messages can contain any other type of URI wherever a SIP URI can appear. It is this fact that enables many of the features discussed below.

4.5 Implementation

To verify the correctness of SIP, and validate it as a useful tool to enable new services, we implemented a SIP proxy and redirect server, named gosSIP. The server was written in ’C’ for Solaris, and contained roughly twenty five thousand lines of code. The server is a complete SIP proxy implementation, including support for UDP and TCP, record-routing[2], basic and digest authentication, proxy and redirect modes, recursion, loop detection, request merging, call cancellation, registrations, and parallel and sequential forking. To enable service creation, we implemented CPL [141] and SIP CGI [142] (to our knowledge, the first implementation of both of these). The server supports routing of requests that contain telephone numbers in the Request-URI, using a longest-prefix-match routing table. For example, a single table entry can direct the proxy to forward all calls for numbers beginning with +1212 (New York City) to `gateway.com`. The server supports a cache of call routing decisions, so that requests for a specific SIP URL that has appeared previously are resolved rapidly.

We wrote utilities for adding and configuring users to the system. We also defined an extensive configuration language, using XML, which allowed us to specify any one of over fifty configurable parameters. The server also supported flexible logging that allowed generation of arbitrary text logs, similar to the logging capabilities of the Apache [143] web server.

In the sections below, we provide an overview the architecture of this system and describe several services we implemented with it.

4.5.1 Events and Threading

The gosSIP server is an event-based multi-threaded software system. Asynchronous events are generated by the arrival of packets on the network, the firing of a timer, or the transition of certain state machines to the completed state. These events are placed into event queues, and processed by worker threads using the well-known worker thread pattern. The overall threading architecture is depicted in Figure 4.4. There are four types of threads:

Message thread: This thread listens on sockets for messages from the network. It is the only thread that ever listens for network packets, requests or responses, UDP or TCP. The message thread uses a single `select()` call to listen for packets. We felt that using a single select call, rather than using a number of threads each listening on a single socket, would be

[2]Record-routing is a SIP capability that allows a proxy to request to be included in subsequent signaling between participants

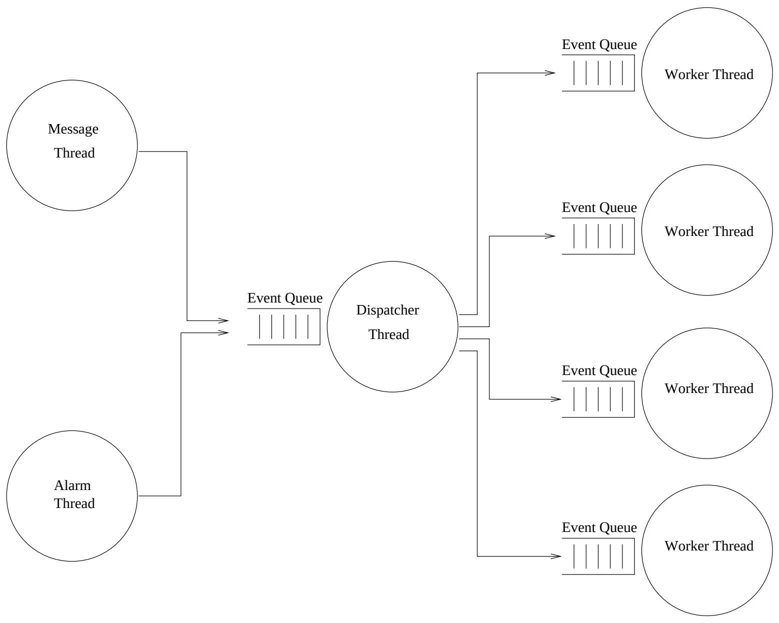


Figure 4.4: gosSIP threading architecture

more efficient. When a complete packet has arrived, it is parsed, and the resulting message encapsulated in an object and inserted into the event queue read by the dispatch thread.

Alarm thread: The alarm thread handles timers for all the other components in the system. Other threads can schedule timers to fire at a specific time, and pass an event to be placed onto the event queue once it fires. The alarm thread uses a heap data structure to order the timers. It sleeps until the next timer is to fire, and when it does, wakes up and places the scheduled event into the dispatchers event queue.

Dispatch thread: This thread reads from a queue of events, and dispatches them to the event queue that is processed by a specific worker thread. The dispatch thread may create a new worker thread if needed, or place the event into the queue of an existing worker thread. It can schedule work round-robin or randomly (but uniformly) distributed across worker threads.

Worker thread: This thread is responsible for the actual processing of events. It blocks until an event is inserted into its work queue. When that happens, it wakes up, and processes the event. Once its finished, it checks if there are more events in the queue. If there are, the next event is processed (and once that is completed, it checks for more events, and so on). If there are no events in the queue, the thread blocks, waiting for an event to be inserted into its work queue. The number of worker threads is configurable.

There are numerous events the system knows about:

Message event: a message has arrived from the network.

Client packet event: the client state machine reports a significant packet (i.e., not a retransmission).

Server packet event: the server state machine reports a significant packet (i.e., not a retransmission).

Client error event: the client state machine reports an error, typically a timeout or socket failure.

Server error event: the server state machine reports an error, typically a timeout or a socket failure.

Client completion event: the client state machine has completed.

Server completion event: the server state machine has completed.

Client timer event: a timer associated with the client machine has expired.

Server timer event: a timer associated with the server machine has expired.

Registration timer event: a registration timer has expired.

TCP close event: a peer has closed a TCP connection.

4.5.2 Processing Architecture

Worker threads handle events. The events above (with the exception of the registration timer) are all associated with a SIP transaction. A SIP transaction, as far as gosSIP is concerned, is the set of messages and timers initiated by a request (INVITE, OPTIONS, BYE, etc.), including CANCELs and ACKs and responses associated with that request. For example, a new transaction is initiated by an INVITE request. The responses received to that request, and any ACKs or CANCELs received for that request, are all part of the same transaction.

Handling of events for a transaction are handled by three cooperating sets of state machines, namely server machines, client machines, and the mediator machine. The purpose of these machines is as follows:

Server state machine: The server state machine receives requests from the network, and sends responses. It handles response retransmissions, TCP/UDP details, and automatic generation of responses for errored requests.

Client state machine: The client state machine sends requests on the network (more specifically, one request and any retransmissions, an ACK, or CANCEL for that one request) and receives responses to the request. It handles request retransmissions, TCP/UDP details.

Mediator state machine: The mediator state machine mediates between the client machines, server machines, and application. It coordinates these by passing information from one to the other after applying the appropriate behavior.

Each transaction has its own instance of these state machines. Transactions don't share state. The important implication of this is that gosSIP is not a call stateful server. In other words, if a call is setup with an INVITE request, and later terminated by a BYE request, gosSIP does not know about the call itself. As far as gosSIP is concerned, two independent transactions executed.

Each transaction contains a single mediator state machine. There is usually a single server machine, but there may be more than one. This happens when *request merging* occurs; that is, the same request is received from different upstream SIP servers. If the proxy forks, there are multiple client machines, one for each branch.

The interaction between these state machines are shown in Figure 4.5. In the figure, requests are shown in solid, responses as dotted, and application interactions as dashed lines. Requests from the network are received and processed by the server state machines. If the messages are significant events (i.e., not retransmissions), the server state machines create server packet events, which are sent to the dispatcher and later executed by the mediator machine. The server machine effectively hides retransmissions, timers, TCP/UDP and network level details from the

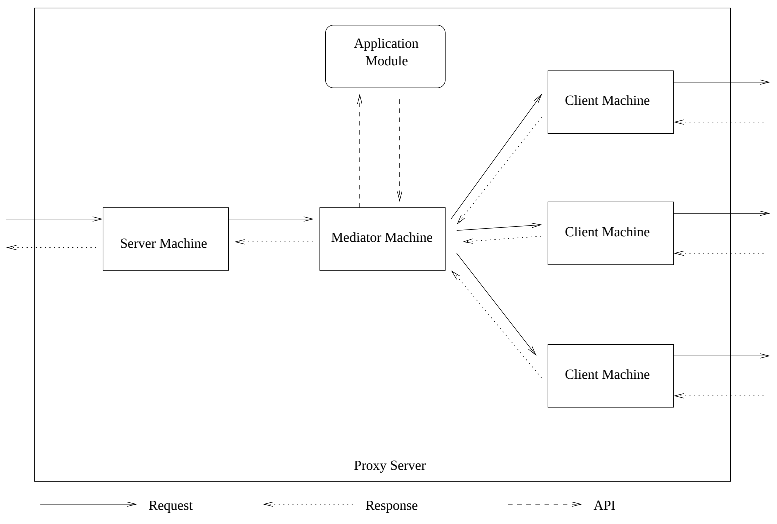


Figure 4.5: gosSIP state machine architecture

mediator. The mediator state machine is the central intelligence of the server. It uses a server API to communicate significant events to the application. The application can then instruct the mediator on how to proceed. The mediator can create client state machines, as needed. Each client machine handles forwarding of a request onto the network. So, if a server forks with two messages, two client state machines are created. As responses are received, they are handled by the appropriate client machine. If the response is significant (i.e., not a retransmission), it is forwarded to the mediator. The mediator uses the main server API to find out what to do next. If the response is to be forwarded downstream, the message is processed and then past to the server state machine, which sends it.

Communication between the state machines can occur in one of two ways. The first is by direct function call; one machine can call the update function of another machine. The other way is through event passing. An event can be generated, fed into the dispatcher, and then executed by a worker thread. As a general rule, communications from the client and server machines to the mediator are through event passing, and communications from the mediator to the client and server machines is by function call. This asymmetry is needed because the main processing routine in the mediator state machine is not re-entrant. If the client or server machines could directly call this processing routine, cases could arise where a re-entrant call is made. Specifically, if the mediator is processing an event, and calls the client machine processing routine, which tries to send a message, but fails immediately, so that the client state machine calls the mediator machine's processing routing to report the event, a re-entrant call to the mediator processing routine would occur.

The server API is used to communicate with the application module. The server API has numerous callbacks that inform the application of events; these events abstract much of the detail of SIP away, but still contain the full information on SIP messages as needed. The application can then exert control over the server by passing it instructions, or directives, on future transaction processing. The directives are service primitives, combined by the application for a complete service. They are described in Section 4.5.6.

Within gosSIP, one and only one application exists at a time. The initial application is always CPL+. CPL+ is an enhanced CPL, designed to support administrator-defined services. CPL was designed to be used for end-users to specify services [144]. In our application, the administrator uses it to specify the overall service delivered by the proxy. To support this application, we added additional switches and actions to CPL. Specifically, we have added switches based on the request method (CPL assumes INVITE, but CPL+ can handle any method), the action parameter from a registration, and the source of the request (either the network or an internally generated request). Additional lookups are defined for the telephone routing table and the cache. An authenticate action was added, allowing the administrator to specify whether the request needs to be authenticated for further processing. A register action was also added, which causes a REGISTER request message to be processed according to the SIP specification. Actions were also specified to allow the invocation of a CGI script or the CPL script uploaded by a user.

An example of a CPL+ script is shown in Figure 4.6. This script configures the server to accept registrations for the `mci.com` domain, and to reject all others. All incoming requests except REGISTER are processed by looking up the Request-URI in the registration database. If a match is found, a redirect response is issued, otherwise, a not-found response is issued.

4.5.3 Server State Machine

If the event being processed is a request from the network, it is handled by the server state machine. The function of the state machine is to receive requests from the network, handle retrans-

```
<call>
  <method-switch>
    <register>
      <string-switch field="to" subfield="domain" depth="2">
        <string is="mci.com">
          <register/>
        </string>
        <otherwise>
          <respond status="error"/>
        </otherwise>
      </string-switch>
    </register>
    <otherwise>
      <lookup source="registration">
        <success>
          <redirect/>
        </success>
        <otherwise>
          <respond status="notfound"/>
        </otherwise>
      </lookup>
    </otherwise>
  </method-switch>
</call>
```

Figure 4.6: Example CPL+ script

missions of requests, send and retransmit responses to the network, handle malformed requests from the network, and receive CANCEL and ACK messages for a request from the network. The server state machine worries about the details of retransmission timers, TCP vs. UDP rules, and responses to CANCELs. The server state machine hides these details from the mediator state machine, so that the mediator need not concern itself with transport details or timers. To accomplish this, the server machine informs the mediator machine of ”significant” events. Significant events are generally message arrivals that cause a state change in the server machine. Not all message arrivals cause state transitions in the server state machine.

The server state machine has seven states. These states mirror the states for server INVITE handling as specified in RFC 2543. However, we have integrated the non-INVITE state machinery as well, so that a single machine handles both cases, and also modified it to cover proxy handling as well. The states are:

SERVER_STATE_INITIAL: This is the state the server machine is in initially, before it has processed any messages or timers. Note that this state is transient. It is the initial state first entered when the machine is created. However, a machine is only created when a message arrives, so the machine will immediately transition to SERVER_STATE_CALL_PROCEEDING.

SERVER_STATE_CALL_PROCEEDING: In this state, the server has received a request for a new transaction, and generated a provisional response. No final response has been sent.

SERVER_STATE_FINAL_STATUS: The final response to the request has been sent. In SIP, INVITE requests are retransmitted periodically. So, this state handles these transmissions. In addition, for non-INVITE requests over UDP, the response is retransmitted when a request retransmission arrives. This state handles this case as well. The server exits this state if it receives an ACK (for INVITEs), or after a timeout (for non-INVITE).

SERVER_STATE_CONFIRMED: For an INVITE request, an ACK has been received. This state exists to handle any late messages for the transactions, such as ACKs or CANCELs that have been delayed. It is exited by means of a timer.

SERVER_STATE_TRANSPARENT: In this state, the server has sent a 200 response to an INVITE, said response having been received from a downstream server. The SIP spec specifies that reliability for 200-class responses to INVITEs is end-to-end, not hop-by-hop. For this reason, when the server state machine is in this state, it passes all ACKs it receives to the mediator (so they can be forwarded to a client machine and onto the network). Furthermore, any 200-class responses it receives from the mediator get sent to the network. In this regard, the ACKs and 200 OKs pass transparently through the server, and thus the name of this state.

SERVER_STATE_CANCELLING: The server state machine has received a CANCEL request for the transaction. It has sent a response to the CANCEL, and will retransmit the response to the CANCEL if a request retransmission is received. Retransmissions of the response to the original request no longer take place. Responses to the original request are ignored.

SERVER_STATE_TERM: The server state machine is finished. No further state changes can take place, and no further processing occurs. When the server machine first enters this state, it sends a server completion event to the mediator. The mediator uses this information to determine if it is safe to destroy the entire transaction and its associated mutexes.

The server state machine also makes use of some static variables to control behavior. These static variables include timeout values, the number of response retransmissions it will perform before giving up, and the IP address of the gosSIP server. The timers and retransmission counter takes on default values; these can be changed through simple API functions which simply set these variables. The address of the gosSIP server is used in the message validity checks performed by the server. Specifically, it is used for loop detection.

4.5.4 Client State Machine

When a response is received from the network, it is processed by the client state machine. The function of the machine is to send requests from the mediator onto the network, handle retransmissions of those requests, send CANCEL and ACK messages, and handle responses to the CANCEL and ACK messages. The client state machine worries about the details of retransmissions timers, transport protocols, and malformed responses. The client state machine hides these details from the mediator state machine. To accomplish this, the client machine informs the mediator machine of significant events. Significant events are generally response arrivals that cause a state change in the client state machine.

Each client state machine handles a single proxied request. If gosSIP forks a request to three downstream SIP servers, three independent client state machines are created. Each of these three state machines is independent. The state machines do no cooperate with each other. They report events to the mediator as if there were no other client machines. This allows the mediator to worry about forking issues, while the client machines can focus on getting the requests to the next server and receiving the responses.

The client state machine has eight states. The state machine represents the union of the INVITE and non-INVITE client machines specified in RFC2543, adapted for proxy behavior. The states are:

CLIENT_STATE_INITIAL: This is the state the client machine is in initially. It is in this state when the machine is first instantiated by the mediator. Like the initial server state, this state is transient. When a client machine is created, it is in this state. However, a client state machine is created only when there is a message to send, so it will transition to the CLIENT_STATE_CALLING state quickly.

CLIENT_STATE_CALLING: In this state, the client machine has sent a request on the network. It is retransmitting the request, but has not yet received any response.

CLIENT_STATE_CALL_PROCEEDING: In this state, the client machine has received a provisional response. If the request was non-INVITE, this causes the period of request transmissions to increase to 5 seconds. For INVITE requests, request retransmissions are stopped completely.

CLIENT_STATE_FINAL_STATUS: A final response to the request has been received. For INVITE requests, the client machine may be sending an ACK (it will not for 200-class responses).

CLIENT_STATE_CONFIRMED: In this state, the server machine is largely finished. It is just waiting around to handle any spurious response retransmissions (which it discards). After a specific amount of time, the client machine transitions to the CLIENT_STATE_TERM state.

CLIENT_STATE_TRANSPARENT: In this state, the client has received a 200-class response to an INVITE. This event is reported to the mediator. In addition, any subsequent 200-class responses (be they retransmissions or new ones) are reported to the mediator. Similarly, an ACK requests received by the mediator are forwarded to the next hop SIP server. This allows for ACKs and 200 OK's to be passed transparently through the client machine, and thus the name of the state. This is required for reliability of 200 OK's to INVITEs to be end-to-end.

CLIENT_STATE_CANCELLING: The client machine has sent a CANCEL message for the original request. It will no longer transmit the original request. Responses to the original request are ignored. The client machine awaits a response to the CANCEL.

CLIENT_STATE_TERM: The client machine is finished. No further state changes can take place, and no further processing occurs. When the client machine first enters this state, it sends a client completion event to the mediator. The mediator uses this information to determine if it is safe to destroy the entire transaction and its associated mutexes.

The server machine also makes use of some static variables to control behavior. These static variables include timeout values and the number of response retransmissions it will perform before giving up. The timeout value and retransmission counter take on default values; these can be changed through simple API functions.

4.5.5 Mediator State Machine

The mediator is the heart of the gosSIP server. It mediates between the client machines, server machines, and the application. The client and server state machines place significant events, as defined above, into the dispatcher's event queue. These get processed by worker threads, which invoke the main mediator event processing routine. The mediator informs the application of these events, and waits for instructions on how to process them. In its role as mediator, the mediator machine decides when and where to proxy requests to, and what to do with the responses that arrive. It chooses, among the responses received by the client state machines, the one(s) to forward upstream. It is for this reason that much of the mediator machine is concerned with forking proxy processing.

The interaction with the application is through the main server API. In gosSIP, CPL+ is the application that runs initially. The CPL+ application makes a determination about what to do, and then hands off control to primitive modules, such as proxy, redirect and respond functions. This is discussed in more detail below.

The mediator state machine has nine states. These states are extracted from the forking proxy behavior specified in the code in Section 12.4 of RFC 2543 [89]. These states are:

MEDIATOR_STATE_INITIAL: This is the state the mediator is in when first instantiated. This state is transient, but to a lesser degree than the client and server machines. When a request first arrives, and no context exists, one is created containing a server machine and mediator machine, both of which are in their initial states. The server machine immediately processes the request and then inserts an event to be processed by the mediator. This event will cause the mediator to transition out of the initial state. During the time between when this event is scheduled and executed, the mediator will remain in the initial state.

MEDIATOR_STATE_PENDING: The mediator has created at least one client state machine, and proxied at least one request. Some responses to these proxied requests may have been received, but none of them are 200 or 600-class responses. Furthermore, there is at least one proxied request for which no response has been received. No response has been sent yet.

MEDIATOR_STATE_GOT200: The mediator has gotten a 200-class response to a proxied request, and there are still proxied requests for which no response has been received. No response has been sent yet.

MEDIATOR_STATE_GOT600: The mediator has gotten a 600-class response to a proxied request, but no 200-class responses. There are still proxied requests for which no response has been received. No response has been sent yet.

MEDIATOR_STATE_FINISHED_200: The mediator has received responses for all the requests it proxied, and at least one of them was a 200-class response. No response has been sent yet.

MEDIATOR_STATE_FINISHED_600: The mediator has received responses for all the requests it proxied, and at least one of them was a 600-class response. None of them were a 200-class response. No response has been sent yet.

MEDIATOR_STATE_FINISHED: The mediator has received responses for all the requests it proxied. None of them were a 200 or 600-class response. No response has been sent yet.

MEDIATOR_STATE_TRANSPARENT: The mediator has received a 200-class response, and has sent a 200-class response.

MEDIATOR_STATE_TERM: The mediator has sent a non-200-class response.

The reason these various states exist is that one of the server API callback functions, `onReadyCallback()`, needs them. This callback is called whenever the server receives a response which leads it to believe that it is ready to send a final response. ”Ready” is defined in the SIP spec as having received:

- any 200-class response;
- a 600-class response, so long as no 200-class response has been received;
- the lowest-numbered response, so long as all responses have been received, none of which is a 200 or 600-class response.

The `onReadyCallback()` is also called whenever the response to be sent changes. Here is an example. The mediator causes three requests to be proxied, to A, B, and C, respectively. A 600-class response is received from A. This causes the mediator to transition to MEDIATOR_STATE_GOT600. The `onReadyCallback()` function is called, indicating the mediator believes it should send this response. However, the application does not tell it to do so. Then, B sends a 200-class response. This causes the mediator to transition to MEDIATOR_STATE_GOT200, and call the `onReadyCallback()` function once more, this time telling it that the 200-class response is the one it believes it should send. Again, the application says nothing. Finally, C also sends a response, which is a 600-class response. The mediator transitions to MEDIATOR_STATE_FINISHED_200. The `onReadyCallback()` function is not called, since the 600-class response is not ”better” than the 200-class response. However, the `onFinalCallback()` function is called (since a final response to a request has been received). Presumably, an application which does not send a response when `onReadyCallback()` is called will set the `onFinalCallback()` to ensure it is called again with a response from some other request.

Now, should the application tell the mediator to proxy the request to D, the mediator transitions back to MEDIATOR_STATE_GOT200, since there is now a proxied request for which no response is received.

Whenever almost any event occurs (server packet, client packet, server error, client error), the mediator notifies the application with a callback. The mediator performs any directives passed from the application in the return value of the callback. The mediator is also capable of a default behavior when the callbacks are not set, or when the application explicitly tells the mediator to perform the default behavior. The default behavior is hard coded into the state machine, and represents the recommended behavior from the SIP specification, as outlined above.

The mediator is also responsible for deciding when its time to destroy the transaction state. To do this, it keeps track of the server and client completion events it has received. Whenever another such event is received, it checks if completion events have been received from all client and server machines. If this is the case, and the mediator itself is done (MEDIATOR_STATE_TERM or MEDIATOR_STATE_TRANSPARENT), the mediator calls `DestroyContext()`, which sets the field marking the context for destruction. When the mutex around the transaction state is released, this flag is checked, and the state destroyed if no other threads are attempting to lock the mutex.

4.5.6 Server API

The server API allows applications to control the behavior of the proxy. The API is a *transactional* one. This means that its callbacks and controls are on a transactional basis, not on a call basis. The callbacks from the mediator to the application inform the application of all significant events in the lifecycle of the mediator machine. This includes final responses received to any proxied requests, reception of the best response (as discussed above), provisional responses, request cancellations, and acknowledgments.

The API also allows the application to control processing in the server through *directives*. Directives are structures that tell that proxy what to do. Directives can be passed to the mediator through direct function calls by the application. Alternatively, directives can be passed as return values from a callback. The directives supported by the server are:

PROXY: The proxy directive tells the server to proxy a request. It includes a *request fragment* that allows the application to modify the request in some way (by adding a removing a header, for example), the destination URI to send to, and a set of callbacks to be invoked with changes in the client state machine.

SEND: Send is like proxy, but does not use the client state machines. It is used to support stateless proxies.

RESPOND: Respond causes the proxy to send a response. The actual response is enclosed.

CANCEL: This tells the proxy to cancel all pending branches.

ACK: This tells the proxy to send the included ACK.

FORWARDRESPONSE: This tells the proxy to forward a previously received response upstream. The directive includes a reference to the response. If a reference of value zero is included, the server sends the best response received so far.

TERMINATE: This forcibly destroys all transaction state.

REGISTER: This tells the server to process the registration it just received. It only makes sense if the request that initiated the transaction was a REGISTER request. This directive allows the application to handle registrations, or the job can be delegated to the default processing in the server.

4.5.7 Memory Management

The SIP server memory manager is custom designed for the server. Rather than allocating memory directly from the kernel with malloc and calloc, the code uses memory manager versions of these functions, memmgr_malloc, memmgr_calloc, and so on.

The key idea of the memory manager is that most of the allocations performed in the server are for handling a transaction. Transactions have a finite lifetime, at most a few minutes.

After this time, the transaction is done, and none of the memory it used is needed. Furthermore, during the lifetime of a transaction, much of the memory allocated might be needed again before the transaction completes. This includes messages and packets, parsed structures, and so on. The nature of the memory usage means that we can allocate a large block of memory for each transaction. Allocations within a transaction use memory assigned to that transaction. Memory allocations for a transaction are not freed individually. Rather, the entire block of memory allocated for a transaction is freed when the transaction completes. This architecture provides many benefits:

Fast management. Since memory is never freed during the transaction, the memory manager does not need to worry about memory fragmentation. The manager keeps a pointer to the next free byte, and when an allocation is made, the pointer advances by that amount. This simple operation is fast.

Memory leak avoidance. Since memory for a transaction allocated from the memory manager is freed when the transaction terminates (when the whole block of memory is freed), the server code need not explicitly free any memory. This avoids memory leaks.

No copying. Since all memory allocated during a transaction is guaranteed to be valid for the lifetime of the transaction, pointers can be safely copied and de-referenced without copying the structures they point to.

Resource management. With a custom memory manager, it is easy to add resource management capabilities that limit the amount of usage allowed.

The only disadvantage of the manager is that it is wasteful. Memory which does not actually need to exist for the entire transaction will. This means that memory stays in use longer than needed. The result is an increase in the required memory on the server.

4.5.8 Services

When a request arrives at the server, processing begins by executing a CPL+ script specified at the command line of the server main. This script can go through a series of decisions and eventually pass off control of the server to a series of modules. Modules have been defined for basic proxy, redirect, and responding, in addition to CGI processing and phone routing.

The primary tool for creating services is the SIP CGI interface. The server will hand off request processing to a separate process spawned by the server. This process can generate responses or cause the request to be proxied, as specified in RFC 3050 [142]. We used the CGI interface to create several combined services.

One of the services we implemented was CFNA-W, which is described above in Section 4.2. We wrote a program in C which receives the initial request, and tells the server to proxy it. If the program receives a no answer response, it dynamically generates an HTML document, personalized with the identity of the caller, which asks the caller to leave a message by either clicking on one link to send email, or another like to go to their web page. The personalization is accomplished by extracting the name of the caller from the From field, and placing it into the HTML as a greeting. The HTML document is returned in a redirection response to the caller. The client software we used was capable of receiving HTML content in redirect responses, and passing it off to a web browser for display.

We also wrote services for find me, follow-me, call screening, and call redirect to email on busy. We found our architecture made development of these services easy. The integration of web and email, even though done simply, made the services far more attractive.

4.6 Conclusions and Future Work

We believe that SIP addresses the limitations in H.323 and BICC outlined above. Most important among them is the limited support for new services, particularly combined services. The usage of MIME and URIs are a key enabler of combined services, and these are capabilities lacking in both BICC and H.323. Our experience has also shown us that new services are facilitated by constructing signaling protocols with simple, well-defined, yet reusable operations. The simple offer/answer model in SDP, coupled with the flexibility to perform the exchange in either the INVITE/200 OK or 200 OK/ACK has enabled third party call control, although we had not considered third party call control when defining this capability in SIP.

Besides its advantages for constructing new services, SIP appears to be superior for building highly reliable and available Internet telephony networks. This benefit comes from the usage of stateless intermediate devices, along with the use of domain name lookups as a means of indirection. Together, these features allow for a proxy to fail, and for the downstream or upstream device to send the message to an alternate device. Since SIP can operate statelessly, the alternate can successfully process the message. Furthermore, the soft-state nature of re-INVITE requests (where the entire session state is included, rather than sending a delta) enables recovery of call and session state in user agents. More investigation is required to determine the scenarios when such recovery is actually feasible (other state in the device may make it impossible). Our soft-state approach and stateless proxy approach are not present in either H.323 or BICC.

Our implementation experience proved that SIP is generally correct (indeed, our experiences were used to help develop the specification in the first place). Of course, more extensive testing since the publication of the specification has yielded errors and inconsistencies that are in the process of being resolved. Our experience also demonstrated that the protocol was reasonably simple, at least in comparison to H.323. In all fairness to H.323, however, SIP is getting more complex as additional extensions are defined to fill in the gaps.

Much work remains in order to make SIP a complete protocol for Internet telephony signaling. We have proposed extensions or approaches to cover reliable delivery of provisional responses [122], integration of call signaling with network QoS [145], traversal of SIP through Network Address Translators (NAT) [146], construction of multiparty conferences [147], guidelines for defining extensions [148], and refresh of calls through a session timer [149]. Others have proposed extensions and techniques to cover a wide range of topics, many of which are needed for a complete solution. This work is all being carried out within the IETF.

Chapter 5

Gateway and Service Discovery

5.1 Introduction

The first two chapters of this dissertation have focused primarily on issues related to media transport; that is, how can voice and video be delivered from one host to another across the Internet with good quality. We assumed that the two hosts that are communicating have agreed to do so, and know each other's IP addresses, UDP ports, and available codecs. Exchanging this information between two hosts is the function of a signaling protocol, SIP, which we discuss in greater detail in Chapter 4. The discussion assumed that the Request-URI indicates a domain (which can be resolved through DNS if needed), so that the request can be forwarded to that domain. Only that domain can interpret the user portion of the Request-URI to further route the request. These assumptions, that the request can be forwarded to the domain of ownership, and only the domain of ownership can process the user portion, are not true when the Request-URI contains just a phone number (which can be represented with the tel URL [150]). This will be the case when a user on an IP device wishes to call a user connected to a terminal on the PSTN (known as PC-to-phone). It will also be the case when the long distance portion of a traditional phone call is to be carried over the Internet (known as phone-to-phone).

In the PC-to-phone scenario, the originating PC has only a telephone number, entered by a user. In the case of phone-to-phone via the Internet, the ingress gateway also has only a telephone number, which was the number dialed by the originating terminal on the telephone network[1]. In order to complete the call to the PSTN, the services of an Internet Telephony Gateway (ITG) are required. These gateways are capable of converting signaling and media session protocols between the Internet and the telephone network. Therefore, in order to complete the call, the telephone number must be converted into either an IP address of the gateway, the domain name of a gateway, or a SIP URL that routes to a SIP server capable of reaching a gateway. This problem, which we define as the *gateway discovery problem* is discussed in detail in this chapter.

The rest of this chapter is structured as follows. Section 5.2 defines the problem in more detail, outlines requirements for a solution, and then demonstrates how the problem is a subset of a more general problem of *wide area service discovery*, whose more general requirements are then outlined. Section 5.3 discusses related work in the general problem of service discovery. In Section 5.4, we review existing protocol solutions that have been used for service discovery, and show why they are inadequate. Then, in Section 5.5, we present our solution, called the *Wide Area Service Discovery Protocol*, or WASRV, which builds upon the Service Location Protocol [151, 152], and show how it meets the requirements outlined in Section 5.2.

5.2 Problem Definition

We first consider the specific problem at hand, and then view the problem from a more general perspective.

5.2.1 Gateways

Discovering services on the Internet is not a new problem. IP end-systems must be able to discover a DNS server, for example. This is typically done through either a static configuration, or via protocols like the Dynamic Host Configuration Protocol (DHCP) [153]. Unfortunately, ITG's provide a much different service than a DNS server. The nature of this service makes their discovery a much harder problem [154].

First and foremost, it is sufficient for an IP host to use the DNS server which is closest

[1]Actually, it may not be the actual number dialed, as a result of local number portability or 800 number translation services. However, that is not important for the purposes of this discussion.

to it, in terms of IP hops. The DNS server is usually administered by the IP transport provider, and clients use the server their administrator tells them (usually through DHCP) to use. There are no per-access charges associated with DNS lookups. Any cost for providing DNS service (like computer depreciation) is wrapped into monthly access charges, if any. There is generally no reason for a host to use a DNS server besides the one provided by its ISP.

This situation is almost the exact opposite for an ITG. Unlike DNS services, there is a cost associated with completing the call, since the ITG must dial the final endpoint on the PSTN. This will cause charges to be accrued by the ITG administrator, which must then be passed back to the user. These costs depend on the distance from the ITG to the final PSTN destination, the calling plan used by the ITG, the time of day, the volume of business, etc. To reduce these costs, a client may prefer to use an ITG which is situated as close to the final PSTN callee as possible. This would result in the cheapest call between the ITG and the destination, and therefore would minimize the cost passed on to the client. We are assuming that any costs associated with the Internet portion of the connection are independent of the destination.

Cost may not always be the primary concern, however. Instead, a caller may wish to have the best quality for the call, independent of cost. This might be typical for business calls. Due to varying delays and losses on an IP network, the best quality is probably obtained by using an ITG which is closest to the caller, as measured in terms of delay or IP hop count. We call such a selection criteria *proximity-based*.

Cost is not the only reason why a user may prefer to use one ITG over another. In an international calling environment, the set of protocols and billing mechanisms supported by ITG’s can be expected to vary. Some ITG’s may support billing of IP hosts via credit cards or e-cash [155] on a per-usage basis. Others may require the users to set up an account ahead of time. The ITG’s will have to perform speech transcoding to convert the codec used by the IP host to either G.711 [156] (the standard used in the PSTN), or analog. There are many speech coders used by IP clients. These include the 8 kb/s G.729 standard [157], the 5.3/6.3 kb/s G.723 coder [25], the 16 kb/s G.728 LD-CELP coder [158], and any number of proprietary codecs. There are also a whole host of higher rate, high quality speech coders, such as G.722 [159]. Since not all ITGs will support all codecs, and since IP hosts may not all implement the same baseline codec, an IP host may need to select an ITG based on its speech coder support.

In addition to the speech codec, IP hosts may utilize a range of different signaling protocols for initiating and terminating calls. ITG's must be able to recognize these protocols. H.323 [94] developed by the ITU, and SIP [89] are both used for call signaling. Proprietary signaling protocols exist as well. An IP host may need to select an ITG based on which signaling protocols it can understand.

These signaling protocols often provide optional features, such as multi-party conferencing or call transfer. If a user knows that they may wish to invoke such a service during the call, they may wish to select a gateway which supports those call features.

Authentication and encryption are also commonly used in Internet telephony. If an IP host wishes to encrypt the portion of the call between itself and the ITG, the ITG must support the particular encryption algorithm. This, too, becomes another criteria for gateway selection.

Of course, the need to select gateways based on protocol support (billing, speech codec, signaling, and encryption) can be eliminated by universal agreement on baseline protocols for each of these. The likelihood of permanent agreement in this area in an international context is questionable, however. We believe that there will continue to be a need for ITG selection based on protocol and codec compatibility.

Implicit in much of the above discussion is the fact that an ITG need not be run by the same ISP that is used by a client. In fact, it is highly unlikely that a gateway will be run by the same ISP used by the client. ISP's are frequently local, and its customers will probably want to use ITG's to make calls to locations that are not in the ISP's area of coverage. In fact, there is no reason why the administrator of an ITG need be an ISP at all. In an open business environment, it is important for IP hosts to be able to use ITG's from whatever service provider they desire. This in and of itself can then become another criteria for selection. An IP host may prefer to use an ITG administered by some large telecommunications provider, for example.

We are now in a position to state the gateway discovery problem. ITG's are run by possibly independent, widely distributed service providers. These ITG's may be scattered across the world, and may implement a variety of different protocols for billing, speech coding, signaling, and network transport. Usage of a gateway by a caller on an IP host to complete a call to a PSTN endpoint incurs a cost, which will be passed on to the caller. The caller must be able to determine the IP address of a gateway which meets the requirements of the user. These requirements include (but are not limited to) cost, proximity, protocol and feature support.

Its helpful when formulating requirements (and for comparing alternatives) to have a sense of how large a problem this is. There are two dimensions to the scale. One is the number of gateways, and the other is the number of clients trying to access those gateways. Estimating the eventual size of the latter is difficult. In order to minimize costs for its customers, an ITG provider may decide to deploy ITG's such that nearly every PSTN number is reachable by a local call though some ITG. Current ISP's are faced with a similar problem, but in reverse; they must have points of presence (POPs) so that each PSTN number can call a POP as a local call. As of April 2001, America Online had approximately 2562 POPs in the U.S. (with 316 in California alone). To extrapolate to the number of POPs required worldwide, we multiply this figure by the ratio of worldwide to U.S. telephone access lines. As of June 2000, there were 191 million telephone lines in the United States [160]. We could not obtain current data on the number of worldwide telephone subscribers. In 1996 there were 745 million [154], and forecasts predict approximately 1.1 billion by 2002. As a result, we estimate there are approximately 1 billion lines today. This would result in 13,414 POPs worldwide. The number of gateways will be the number of providers per POP, times this number. As of June 2000, there were approximately 3.8 local phone providers in each zip code in the United States [160]. If we use this same factor, we obtain approximately 51,000 gateways. This number is very rough, and we estimate it is below the actual number of gateway clusters which might be deployed (that's because the number of AOL POPs is far below the number of U.S. local calling areas).

Estimating the number of clients accessing those gateways is easier than estimating the number of gateways. Assume that within the U.S., eventually all inter-LATA and international calls are over IP telephony. In 1999, 84.2 billion inter-LATA calls and 4.8 billion US to international calls were made, for a total of 89 billion [160]. This is 2822 calls per second. We multiply this by the ratio of worldwide to U.S. lines, and obtain roughly 15,000 calls per second. If each call requires a gateway to be discovered, the system needs to handle this many discovery requests. If we assume a system where the total bandwidth usage for a gateway discovery protocol is linear with the number of clients (but independent of the number of gateways), with a usage of 1 kB per user per call, the bandwidth needed for the system is roughly 150 Mb/s. We chose 1 kB based on the assumption that client messages would be similar to typical database queries, which are usually not too large. As a point of comparison, the bandwidth required to carry the voice calls for this system (assuming 3 minute call hold times) is much larger, roughly 432 Gb/s.

This basic analysis reveals that the problem is bandwidth (both network and processing) limited, not data limited. Storing fifty thousand entries for gateways is trivial, and can be done in memory for almost no cost. Designing a system that can process fifteen thousand requests per second is more challenging.

Given the problem definition and the sizing analysis, we can define several requirements for a solution to this problem:

Fast resolution: The protocol should be fast. Since finding the ITG is required before a call can be placed, the call setup times are increased by the amount of time required for this protocol to operate. Therefore, rapid operation is important. The discovery process should not take more than a few hundred milliseconds.

No central point of failure: Since the discovery process is a precursor to call establishment, any system failure is unacceptable. An outage is acceptable if it can be recovered within a second or two (so that call setup times still remain acceptable).

Avoid middlemen: There are two principals in the system, the provider of the gateways, and the client. If a middleman is introduced, to act as a broker which provides a global, cross-company service, we have a third principal. The middleman must now be trusted by both the provider and the client. If the system has a single middleman for all providers, the middleman becomes a central point of security compromise. This problem is especially troublesome for gateway discovery, since immediate revenue is derived from usage of the gateway returned by the system.

Efficient: The protocol should not require significant bandwidth. It is not acceptable, for example, for an IP host to query a long list of candidate gateways. This would multiply the 1kB per user per call usage by some multiplicative factor, and significantly impact network bandwidth usage. Querying gateways also imposes resource constraints on the gateways themselves.

Linearly scalable: The bandwidth generated by the system should scale linearly, or better, with the number of clients. Scaling linearly with the number of clients is more important than scaling linearly with the number of gateways, due to the smaller number of gateways. The processing load on the system should also scale in the same way.

Dynamic: The protocol should be dynamic. As new ITG's come into existence, they should become accessible to IP hosts almost immediately, without requiring a human to enter them into a database. Similarly, if an ITG goes down, this information should be propagated in a timely fashion. Changes in billing policies (due to some upcoming holiday or special promotion) should also be distributed rapidly. Here, rapidly refers to timescales much smaller than the duration over which the data is valid. For example, if ITG characteristics change on a daily basis, rapid dissemination implies timescales on the order of minutes.

Secure: Security is important. The protocol must ensure that the information provided about a gateway is authentic. Privacy of the gateway information does not appear to be important in this application. The purpose of the protocol is widespread dissemination of data. However, the phone number dialed by the client may be sensitive. Privacy of client generated data is therefore important. The protocol should also protect against denial of service attacks.

5.2.2 General Services

The problem that has been discussed so far is the discovery of a particular server, an ITG, based on some criteria. Telephony gateways are not the only types of servers which might need to be discovered. In general, the problem of *wide area service discovery* is to allow clients to find servers for a particular service in possibly remote locations on the Internet based on characteristics of the service. Discovering of services by characteristic is substantially different (and harder) than discovering services by name or address. We can extrapolate most of the requirements discussed above to more general ones for this problem:

Multi-criteria: The client desires a service which can be characterized by a number of attributes. The client should be able to express the desired attributes of the service, and get back a server whose service meets the criteria. There should be no arbitrary restriction on the type, values, or number of attributes which can potentially characterize a service. Some restrictions on typing and some convention with respect to the interpretation of values will exist in any wide area service discovery protocol.

Location-independent: The location (in terms of domain or geographic locale) of the desired server is irrelevant and may not be known a priori. That is, the domain name, administrative domain and network address of the desired service is not generally known in advance and in many cases is not pertinent. In some cases, the location of a server may be important, but no more important than any other attribute of the service.

Auto-configured: It should be straightforward to configure new clients to discover a service, and it should be straightforward for servers to make themselves available for discovery. By straightforward, we mean that the configuration should require, at most, the setting of one or two parameters.

Rapid Availability: When a server is first brought on line, it should become visible and accessible to clients rapidly.

Service-based: The attributes provided by the client provide the attributes of the service, not the content provided by that service. For example, an attribute for a web server might be support of the Internet Cache Protocol (ICP) [161, 162], as this is a characteristic of the web service. “Contains web page X” is not an attribute of a web server, but rather a description of the content provided by a web server. This requirement is necessary in order to achieve reasonable scale.

Automated: The service discovery process should be automated, and not rely on interaction with a human to satisfy a reasonable query. However, the protocol should not prevent interactive sessions.

Policy Support: The agent responsible for satisfying the client’s service request should be able to inject its own policy into the process. This policy may disallow various servers from being used by the particular client, for example.

Global: The location of a matching server can be anywhere, as can be the client. The protocol should be internationalized, supporting queries and attributes in different languages.

Scalable: There can be millions of clients, and thousands of servers for a particular service.

Mildly Dynamic: The attributes which characterize the service provided by some server do not change on timescales that are on the same order of magnitude, or smaller, of network round trip times. However, they may change on timescales much larger than this, and this should be handled appropriately.

Secure: Since servers may be sources of revenue, protection against denial of service attacks are key. The solution must not make it possible for a single entity to deny service to a large group of servers. Similarly, the protocol must provide a means of authenticating servers and verifying the integrity of the service attributes provided.

Lightweight: Servers need not be lightweight, but it should be possible to provide lightweight clients. This is especially important for gateway discovery, which may be used by standalone IP telephones with limited memory capacity.

Rapid Queries: Queries to discover servers should be handled rapidly, preferably on the order of tens of milliseconds. This is because service discovery is a precursor to service access. The longer the service discovery takes, the longer the total time it takes to access the service. Many services are time critical for access; an excellent example is IP telephony. Since discovery of gateways is a precursor to call setup, the call setup delay is directly affected by the service discovery time.

This definition eliminates quite a number of problems which might otherwise be deemed wide area service discovery. For example:

Yellow Pages: The "yellow pages (YP)" service is an abstract type of service which allows you to find pizza parlors in San Antonio which deliver, or cleaners in Boise that are open on

Saturday. While this resembles the wide area service discovery problem, its focus differs in many respects. The most important distinction is that location (here, geographic) is a key part of the selection process. This allows for the use of a global, distributed database in each geographic locale. YP is therefore focused on locating regional services. Locating wide area services, on the other hand, is based on multiple attributes, and is global in scope. Furthermore, there is no single attribute on which to hierarchically organize the database.

White Pages: The white pages service allows you to find some person or service with a specific name, associated with some organization. Like the yellow pages service, it is based on strict hierarchies for the names. It lacks the multicriteria selection required of yellow pages, however. Its main difference from wide area service discovery is the assumption of a hierarchy of names. Service attributes are much flatter, by comparison, and change more frequently than names in a white pages database.

Web Pages: The web page location problem is to find a web page (which is on some web server, of course) which contains the word “foo”. A slightly more advanced version of this would be to find the web page that talks about some subject. This location problem is different from the wide area service discovery problem in that the user is looking for a document, not a service. The documents are usually sorted and indexed based on either content or meta-information. Since searching is not precise, it cannot be automated, and progresses based on interactive input and trial from a human user. Furthermore, web indexing is based on “pull” of information by web spiders. Since spiders have no way of knowing when information on a page has changed, the information returned by a search engine is often stale and out of date.

Beyond IP telephony gateway discovery, there are numerous applications of wide area service discovery:

Media servers: Media servers provide streaming media content, such as audio or video. They have applications in video on demand, music sampling (listening to the first 30 seconds of a new song at a music store), and voicemail. Typically, media servers are controlled with protocols such as the Real Time Streaming Protocol [163], and the media they delivered is carried over RTP. Other protocols might be used. For these kinds of servers, it may be important to discover one based on the types of movies it has, the protocols and codecs it supports, or the billing mechanisms it understands.

Conference bridges: Conference bridges are similar to media servers. They support RTP and can send multimedia streams to numerous users. However, their main function is to mix the media (audio or video) provided by participants in some conference, and then transmit that media back to each participant. When a group of users wishes to participate in a conference, they may need to discover a server that has particular media and codec support, has particular audio mixing features (such as silence detection), has particular video mixing features (video follows audio), is close to the participants, and has specific billing support.

Anonymizing server: In order to make an anonymous phone call, a user can make use of anonymizing server. The server re-originates the SIP and RTP traffic from the user, and terminates the SIP and RTP traffic from the callee. It modifies any fields in the SIP and RTCP packets which may reveal information about the user. The discovery of an anonymizer might be based on its codec capabilities, signaling protocols, and call features (such as transfer) that can be supported anonymously.

5.3 Related Work

As we have indicated, discovery of servers on the Internet is not a new problem. Much of this work, however, is in related, but orthogonal areas. We mention this work in order to differentiate the problems in other spaces from the problems we are trying to solve here.

One related problem is that of *server selection*. In the server selection problem, a client (or their proxy), wishes to select amongst a set of servers. The selection process is driven primarily by quality; that is, the user desires a server with the fastest response time, least load, or largest throughput, as the application needs would dictate. Dykes et al. provide an excellent survey of the work in this field, providing a taxonomy of the various solutions [164]. Carter and Crovella [165, 166] have analyzed client-side selection algorithms using probes, hop counts, and geographic estimates. Stemm, et al. propose SPAND [167], a distributed system for collection, storage, and retrieval of network performance information. Sayal, et al. [168] have examined selection of web servers using a system called Web++ that applies the Refresh algorithm of Sayal [169]. This algorithm selects the server with the minimum HTTP request processing time, but refreshes the request processing time estimates periodically.

The server selection problem is different than our service discovery problem, in that it assumes the addresses of the candidate servers are known a priori. In our problem, the addresses are not yet known. The problems are orthogonal as a result; once the service discovery mechanisms have identified the set of appropriate servers, a server selection algorithm can be used to find one. Xu et al [170], however, have attempted to integrate the process of selection and discovery for the specific case of telephony gateways. Their procedure effectively requires a ping of every server advertised by every provider, which clearly does not scale.

Another problem space is the discovery of servers for a specific service type within a specific domain. This problem is usually addressed using DNS mechanisms, specifically, the Service (SRV) DNS resource record [171]. An example is the search procedure defined to discovery SIP servers within a specified domain [126]. This problem differs since the domain of the targeted server is known a priori.

Another related area of work is searching for resources and content in the web. This is an extensively studied area, with countless references [172, 173, 174, 175]. This problem space differs from ours in that it focuses on defining metrics for successful searches and on mechanisms for indexing of web content

5.4 Existing Solutions

With an understanding of the problem at hand, which is discovery of services within an open accessibility model, the next step is to review architectures that have been proposed for this problem. We have observed that the solutions can be categorized by their general architectural approach. We propose a segmentation of the proposed solutions into five categories. These are centralized databases, regionally replicated databases, distributed databases, indexed databases, and multicast push and pull.

Others, such as Schwartz et al. [176], have proposed different taxonomies for service discovery protocols. Ours focuses more on architecture and data location, rather than other aspects of the system, such as data types, as used by Schwartz et al. [176].

In this section, we review existing protocols in each of these four categories, and determine their suitability for solving both the wide area service discovery problem and the more focused gateway discovery problem.

5.4.1 Centralized Databases

In the centralized database approach, there is a single database, which indexes the complete set of servers for a particular service. When a user wishes to locate a service, they connect to one of these servers (which requires a form of server discovery in its own right), and formulate a service query. Generally, these systems allow for dynamic update of the content of the databases in a distributed fashion. Entities providing some service can update the database with information about their service. The systems are generally segmented by service type; that is, separate protocols, servers or database instances exist for different service types.

This model is the premise behind several concrete system implementations for service discovery. An example of such a system is the Service Location Protocol (SLP) [152], which has been engineered for enterprise-wide service discovery, along with its predecessor, the Resource Discovery Protocol (RDP) [177]. Java's Jini [178], is another example. JINI is remarkably similar to SLP, providing protocols for discovery of lookup services, registration with lookup services (referred to as a *join* in the Jini model), keepalives, and queries, in addition to security, transactions, and even notifications. Bluetooth [179], a standard for wireless ad-hoc networks, has a component called the Service Discovery Protocol (SDP), which allows clients to query a server for a set of services matching some criteria. Unlike SLP, however, SDP does not provide means for the clients to discover the servers. Another system is Salutation [180], which is similar to Jini and SLP. It works with any programming language (unlike Jini) and any network transport layer (unlike SLP). Like the others, it defines a server that holds services, called the Salutation Manager (SLM). Clients can query the SLM, or search for services. The specifications mention communications between SLMs, but little information is given on how data is distributed, and when. Salutation also defines the actual protocols for communicating with services, unlike most of the other architectures. This requires definition of components for every service supported by the system. A good summary of several of these systems can be found in work of Golden [181] and Bettstetter [182].

5.4.1.1 Service Location Protocol

We provide additional information on SLP, since it is a good prototypical centralized database solution, and since our own contribution, which we discuss in Section 5.5 is based on SLP.

SLP can be used to discover services in one of two ways. First, a client (known as a User Agent (UA)) can send a request for a service to a well-known multicast address. Servers (known as Service Agents (SA)) listen for queries on this address. The request contains an expression describing the service required. Any servers matching the query respond to the client. In recognition of the fact that this can cause significant bandwidth consumption, a second method for service location is available. First, the client discovers its Directory Agent (DA). Instead of multicasting its queries to a multicast group, the client unicasts its queries to the DA. All servers register their services, using unicast, with the DA. If there are multiple DAs, the SA registers with each. This allows each DA to build up an identical service database. These registrations are periodically retransmitted to protect against network loss. The DA checks its database against the query, and returns a list of servers to the client.

As an alternative to having each SA register its services with each DA, peering relationships can be established between DAs [183]. This results in a fully meshed connection between the DAs. An SA need only register with a single DA, and its registration is propagated through the mesh to the other DAs.

The use of a DA requires both SAs and UAs to discover the address of the DA handling queries for a particular service. This is accomplished in several different ways. First, DA's multicast advertisements periodically to a well-known address. SAs and UAs can subscribe to this group and learn the address of the DA. An alternate approach is based on the recognition that a DA is itself a server, and the multicast discovery procedures of SLP can be used to discover the DA. Of course, the address of the DA can always be configured statically or through auto-configuration protocols such as DHCP [153].

SLP introduces the concept of *scope*. Each service is associated with some scope, which is just an arbitrary text string. Clients can request services that lie within a particular scope, and DA's can be configured to only accept registrations from servers that have a particular scope. A typical scope might be the string "math-department", so that clients in the math department of a university will only have access to services run by the department.

When a client wishes to discover a service, it formulates a Service Request (SrvRqst) containing the desired service type (be it "printer", "media server" or "telephony-gateway"), and an LDAPv3 predicate describing the desired attributes. The response (SrvRply) contains a set of service URLs. These URLs identify the services which matched the query. For example, a query for printers supporting A4 paper might return the service URL:

```
service:printer:lpr://foo.com:515/draft
```

Clients can also query for the set of service types within a scope, using the Service Type Request (SrvTypeRqst). This is useful for "browsing" the services on the network. Finally, clients can query for the attributes for a particular service, identified by its service URL.

In SLP, service types can either be concrete, or abstract. A concrete type, such as printer or media server, has well defined and fully enumerated attributes. An abstract type, like an abstract class in object oriented programming, has an incomplete template or set of attributes. A concrete type that extends this abstract type would define the complete template or set of attributes. An example of an abstract service type might be "sip server", with concrete types "proxy server" and "redirect server".

5.4.1.2 Discussion

Most of the centralized systems have been used for local area or enterprise-wide service discovery. The most straightforward way to apply them for wide area service discovery is to have a very large server, run by a third-party provider, which can be queried for services. Service providers send updates to the third-party provider.

Scale is the primary issue for these architectures. However, centralized databases can scale up enough to handle gateway discovery. Even with today's technology, a single server could, in principle, handle the bandwidth (150 Mb/s), the query load (2822 queries per second), and the storage (51,000). Of all these figures, the query load is the most problematic, but it is within the realm of performance figures for high end systems.

An important drawback is that it introduces a single middleman into the system. As we discussed in Section 5.2, these introduce a central point of compromise and a central point of failure.

5.4.2 Replicated Databases

In this architecture, there is still a single, third-party service provider acting as a middleman. However, instead of using a single, centralized database, a set of replicated databases are used. These databases may be distributed around the network, so that clients can query a local one.

This model is used by modern web searching services, such as Google [174], Harvest [184, 185], and Alta Vista. The underlying architecture is one of a farm of servers that handle queries. These databases are massive in their size and data storage capacities; Brin [174] reported that in 1997 they had indexed 24 million web pages, requiring 53 GB of storage for the compressed repository alone. As of May 2001, Google had indexed 1.34 billion web pages.

Perkins [186] has proposed to apply both the underlying architecture and the actual search engine service to the service location problem. Gateways (or any server), are described in web pages using some kind of service metadata. Crawlers pick up and index this information. When a user wishes to access the service, they perform a search, and the results are used to select a gateway.

We do not believe this solution is truly viable for automated service location. It is based only on pull, so that service information may be stale for extended periods of time. Web search engines are also not amenable to automated usage, which is critical for service discovery.

It is viable, however, to apply the architecture underlying web search engines for service discovery. Using a set of replicated servers has many advantages. It scales up well. This is particularly true for gateway discovery. Since the query rate is the primary bottleneck, replication can improve the amount of queries any particular server needs to handle. With 200 servers, each one would need to handle only 10 queries per second, on average (assuming uniform load distribution).

As with pure centralized databases, there is a middleman problem. Interestingly, Brin documents cases where search engines have been compromised for profit [174]. One search engine, OpenText, was selling the right to be the first match for a particular keyword search. Brin warns that more insidious compromises are possible. We believe that Internet telephony gateway discovery will be an even more attractive target for compromise, since their is a direct link between revenue and being returned in a query.

5.4.3 Distributed Databases

The solutions in this section implement distributed databases. The information about services are scattered across databases owned by different providers. Each provider indexes the servers it is responsible for. However, distributed databases require that the data be distributed according to some kind of namespace partitioning. This partitioning is effective only when a client query can be satisfied by servers from a small number of providers. This means that the set of servers run by a provider (which are also the ones it indexes) need to be within a contiguous portion of the namespace, with minimal overlap with portions indexed by other providers. For the general service discovery case, we would argue that it is not possible to partition the namespace in such a convenient fashion.

However, such a partitioning is more realistic for Internet telephony gateways. Generally, the queries for gateways are going to be for one that is capable of terminating calls to some PSTN number for the minimum cost. Since the terminating number is effectively the primary key for all queries, we can distribute the namespace in a hierarchical fashion with that key.

Such distributed databases for IP telephony gateway discovery have already been proposed. Both DNS and X.500/LDAP-based distributed databases have been proposed in the literature. In the sections below, we scrutinize those solutions and show why they are not adequate.

5.4.3.1 DNS

The Domain Name System [119] is used to map host names to IP addresses. It has also been used for mapping a service in a domain to a set of servers which can provide that service [171].

Most relevant to the gateway discovery problem is the tpc.int subdomain [187, 188]. The tpc.int subdomain allows a host to register a fax machine as providing fax service to a certain set of telephone numbers. An IP host wishing to send a fax constructs a domain name based on the fax number, and can then find the IP address of an email-to-fax gateway. The construction of the domain name from telephone number is done by assigning each digit to a subdomain, in reverse order. For example, a fax gateway in the 415 area code in the U.S. (country code 1) would construct its domain name as 5.1.4.1.tpc.int. If there are multiple gateways servicing any particular telephone area, the DNS server will contain multiple records, one for each gateway.

More recently, DNS has been proposed as a way to resolve telephone numbers to resources (represented as URLs) on the Internet, using the e164.arpa root domain [189]. This begs the question – can these gateways be resources, represented as SIP URLs? A gateway provider would place entries into the DNS for each gateway. These entries would be the minimal set of prefixes, the union of which represents the set of numbers the gateway can to terminate calls to. For example, a gateway willing to terminate calls to the 415 area code in the U.S., in addition to a particular exchange (453) in the 408 area code, would place two entries into DNS - 3.5.4.8.0.4.1.e164.arpa and 5.1.4.1.e164.arpa. To discover a gateway, a client queries DNS with the number, and will get back the longest prefix matching entry.

There are numerous difficulties with the DNS solution. The most troubling one is related to administering the DNS records. Note that a gateway must create entries for all those country codes, area codes and exchanges it is willing to terminate calls to. Since gateways make money (presumably) by terminating calls, it is in the interest of a gateway to indicate willingness to terminate calls to anywhere in the world (of course, there may be high costs associated with certain numbers). This means that if there were no constraints, a gateway vendor would probably create entries for every exchange, area code and country code in the hopes of obtaining more business. The result is that any query to DNS would return every gateway, and that doesn't scale linearly in bandwidth usage, as we have discussed in Section 5.2.

To solve this problem, some restrictions would need to be placed on the records a gateway can insert into the DNS. These restrictions will fundamentally limit the business a gateway can obtain. Who is to decide which ITG's are to be located in the different sections of the database?

Any solution is fundamentally limiting. In fact, the original proposal for the tpc.int subdomain focuses on the cooperative, non-competitive nature of the system, recognizing that it is inappropriate for competitive business practice [188].

DNS also has security problems, primarily due to the fact that the administrator of a gateway need not be the owner of the DNS server that contains the zone with that gateway entry. Unless each server is run and administered by a truly neutral third party, the possibility of tampering and unfair business practice will exist, and is not solvable with cryptographic mechanisms.

DNS procedures have been described that can allow selection of a server in close proximity to the querier [190]. These approaches, used frequently in content distribution networks, can allow for selection of a gateway based on proximity to the caller.

5.4.3.2 LDAP and X.500

LDAP [191] is the Lightweight Directory Access Protocol. It was originally designed to query X.500 databases, but is now independent of X.500. X.500 is a large, distributed database which can be used to store information about nearly anything. LDAPv3 [192] is a new version of LDAP which adds support for improved security, extensibility, and server referrals. LDAP has been explicitly proposed as a solution to the gateway discovery problem [193].

The architecture of a distributed X.500 database is much like DNS. Its main difference is that DNS only supports lookup of records, whilst LDAP supports searches. LDAP can also be used for lookup; in this case its operation, strengths and weaknesses for solving the gateway discovery problem are identical to DNS. The remainder of this discussion assumes we are trying to use LDAPs search capabilities to find a single gateway based on the desired criteria.

Like DNS, portions of the database are distributed to different servers. It makes sense to organize the database using the same hierarchy for assigning names in e164.arpa. Each digit in the number would form an additional level in the distinguished name, just like the DNS solution. The entry for a particular DN would be run by a trusted party. Service providers would request that their gateways be entered into the database for a particular DN, if the gateway services numbers within that prefix. They would also provide the trusted third party with attributes which characterize the gateway. This is no different than the DNS solution, which also requires a trusted third party to manage the zone for a particular prefix.

The LDAP solution is far more flexible in terms of how the query is formulated. The client can ask for a gateway, starting at the root of all gateways, with particular attributes. Since LDAP does not support complex formulae computations in queries, asking for the “cheapest” gateway is not always feasible (price may be a function of the time-of-day, for example). However, querying for gateways with specific codec, security, and capacity attributes is feasible. The response of the query would yield potential gateways that match on attributes. The client can then go through this list to determine ones that are cheapest or closest to the dialed number. As an alternative, the search can be made starting at a node deeper in the tree; for example, if a call is to be made to +1 212 555-1212, the client can query starting from 2.1.2.1.e164.arpa, guaranteeing the match is a gateway that serves the 212 area code in the U.S. The client does not need to make the query based on knowledge of the numbering plan; it can simply choose increasing numbers of digits as part of the RDN to further narrow the set of gateways returned by the query.

As a result, the query is much more flexible than DNS. It can be based on a variety of query parameters, which is not possible with DNS.

Unfortunately, this approach still suffers from many of the same drawbacks as DNS. The issue of ownership and administration of the database, and restricting how many entries a gateway can have, still exists, and we believe these are fundamental barriers. This problem, in fact, is present in all distributed database solutions.

From a practical perspective, LDAP requires a great deal of regularity in syntax and semantics in order to function properly. All cooperating databases and clients must use exactly the same schema. This will make it difficult for gateways to add attributes that describe unique capabilities they possess.

5.4.4 Indexed Databases

In an indexed database, the data is distributed across multiple servers. However, unlike distributed databases, where the partitioning of data across servers is based on data hierarchy, the partitioning of data across indexing servers can be arbitrary. Each server can hold a portion of the data, and it somehow advertises indices to other databases that provide a summary of the contents of the database. This summary information can be used to overlay a routing network of sorts among the databases. A query can be made to any one of the databases, and this query can be routed to one or more other databases which may have the actual service that is desired. Fundamental to the operation of these systems is an efficient way to compute this summary information, especially doing so in a way which reduces the size of the summary compared to the actual data. Of course, this implies loss of information, which means that a possibility is introduced of either false positive or false negative matches of a query against a summary index.

Examples of such systems are whois++ [194, 195], the Secure Discovery Service (SDS) [196], the Intentional Naming System (INS) [197], and for IP telephony specifically, Telephony Routing over IP (TRIP) [198, 199].

Whois++ is the oldest amongst these systems. The summaries passed between database servers are called *centroids*. The centroid lists, for each template record and attribute in that template, the union of all values for that attribute. Attribute-value pairs are often identical across many records (for example, employees in a company database of FOOBAR INC. will have their DEPARTMENT attribute equal to SALES, MARKETING, or ENGINEERING. The centroid of all these records will then contain just those three attributes.). This means that the centroid can occupy much less space than the sum of the sizes all the records. When a server receives a query it cannot satisfy, it checks the centroids it has to see if other servers might be able to satisfy the query. In this case, the client is referred to those servers. The indexing mechanism in whois++ was tied to whois++, even though it had more general applicability to distributed databases. As a result, the Common Indexing Protocol (CIP) [200, 201] was developed as a general purpose indexing protocol. CIP uses MIME for the transfer of indexes, and allows a variety of different index types beyond centroids to be specified.

INS is much like whois++ in its design. Databases (referred to as Intentional Name Resolvers, or INRs), are organized into an arbitrary topology. Each INR publishes a summary of its contents in the form of a Name Tree, which is similar in structure to the centroids of whois++. Unlike whois++, INS allows for queries to be forwarded to a single potential server with the data, all potential servers with the data, or for the INR to return the list of potential servers directly to the querier.

SDS uses a more creative mechanism for its summary data. Instead of propagating the data itself, it propagates sets of hashes of queries which may be successful against the database. These hashes are represented as an array. Assume some query Q has a hash of $H(Q)$, and the array is denoted with I. If Q yields a match in the database, $I(H(Q))$ is set to one, else zero. These large arrays can be represented in compact forms, and used as indices. Like centroids, the summary data elements never have false negatives, but may have false positive results.

5.4.4.1 Telephony Routing over IP (TRIP)

TRIP can be considered as an application of indexed databases to the specific case of databases of Internet telephony gateways. Each indexing database server is known as a *Location Server* (LS). LS's are run by different service providers. These service providers run their own gateways, and the information about those gateways is placed into the LS. The LS communicates with peer providers, exchanging "indices". In this case, the indices are always based on a primary key, which is the phone number prefix. Other attribute information is present as well. The protocol specifies aggregation rules that define if and how two indices (which are referred to as routes) can be combined. TRIP is actually based on BGP [202], since its problem space is the inter-domain exchange of routing information. This information just happens to be for phone number prefixes, not IP address prefixes. However, much of BGP can still be used. An important benefit of TRIP is that the aggregation rules guarantee that there are never false positives if the query is only based on the destination phone number.

The architecture of TRIP is shown in Figure 5.1, which is identical to Figure 4 in RFC 2871 [199].

The network is composed of a set of Internet Telephony Administrative Domains (ITADs). These ITADs own or wholesale gateways to other ITADs. Each ITAD has one or more LSs which are used to exchange information on the telephone prefixes they are willing to terminate traffic to, along with attributes that describe those routes. ITADs may also have End Users (EU), which are people or machines that establish calls that are routed using the data collected through these inter-domain exchanges.

The TRIP framework allows clients to query the system for a route to a particular gateway

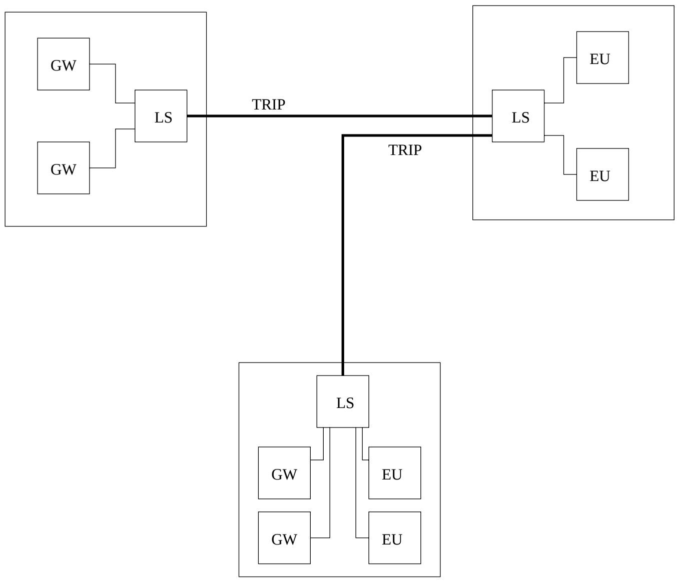


Figure 5.1: Architecture for TRIP

[199]. As a more attractive alternative, the client can send its INVITE request to a proxy which is co-resident with an LS. The proxy looks up the destination number in its database, and finds a matching index (route). Since there cannot be false positives, the proxy routes the call to the peer proxy/LS, which performs an identical operation. The result is that the call is effectively setup in parallel with the query process.

An implication of performing the call routing in parallel with the query process is that the end user need not have a relationship with the provider of the gateway. Instead, there is a chain of bilateral trust relationships, starting with the caller and its proxy, continuing between pairs of LS's along the query/call establishment path, and ending at the terminating gateway. The terminating gateways bill their bilateral peer, and each LS passes on the cost (possibly after adding some kind of brokering fee) to its peer. Finally, the administrator of the first LS bills the caller. Effectively, the exchange of a route is an offer to sell termination of calls to the numbers in the route. Peering relationships go hand-in-hand with business relationships between providers.

Consider an example, shown in Figure 5.2. User X is a customer of provider A. User X only has a relationship with provider A. User X makes a call, which uses a gateway run by provider D. The services offered by this gateway are sold to provider C, who has resold them to

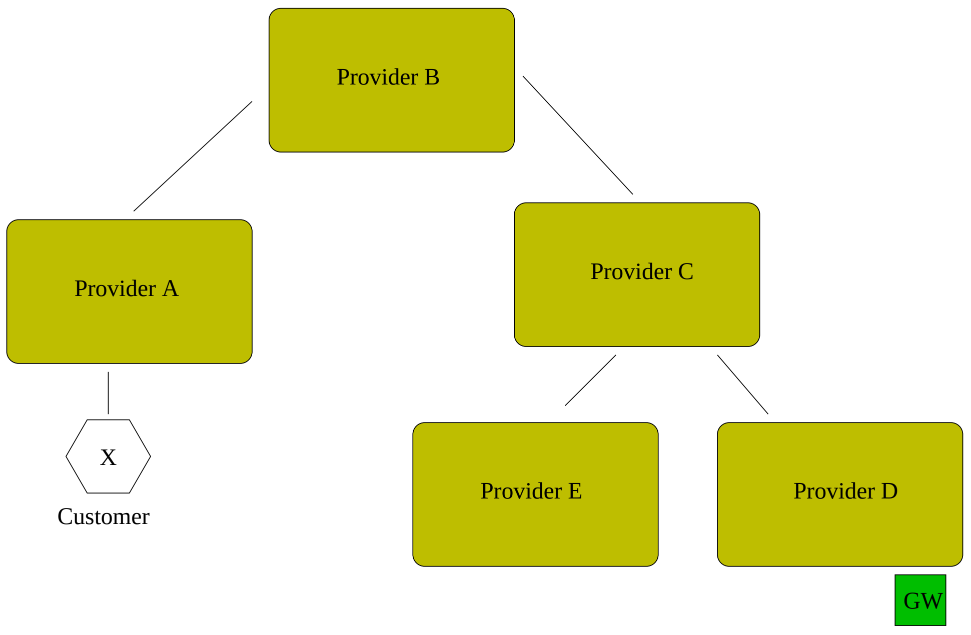


Figure 5.2: Customer X uses gateway through bilateral provider relationships

provider B, who has resold them in turn to provider A. Provider D doesn't know, and doesn't care, about the identity of user X. Provider D bills provider C for the call. Provider C bills provider B, provider B bills provider A, and provider A bills user X.

The advantage of this model to the consumer is that their service provider (who owns the proxy they sent their INVITE to) is ultimately liable for the service provided. The service provider is responsible to determining what other service providers are worthy of peering. Should some kind of fraud occur, the service provider for the user is responsible; they may, in turn, pursue their peer for compensation. Another advantage is that the complexities of multiple providers is completely hidden from the end user. The end user only needs one relationship with a single provider that takes care of all the details. Since a direct relationship exists between the end user and the service provider, shared secret security mechanisms can be used to authenticate the user to the provider. Similarly, shared secret mechanisms can be used between peer providers. This enables the system to work without a PKI.

The drawbacks to this approach are that the user does not have as much flexibility. The servers they ultimately used are based more on the policy decisions of their service provider than the needs of the customer. Cost to the consumer is likely to be higher. New service providers will need to go through extensive negotiations with other providers to establish peering. These peering relationships might even be regulated by government organizations. The model, not surprisingly, is similar to how the telephone network operates.

5.4.4.2 Discussion

Indexed databases eliminate the central point of compromise that is present in the centralized databases, regionally replicated databases, and distributed databases. Instead, there can be a number of alternate paths between callers and gateways, so that the compromise of a single provider does not necessarily compromise the system (although it still may, depending on the topology).

Unfortunately, this benefit comes at the cost of increased query times. The increase comes from the need to iterate or recurse on a query to hit all the database servers where the query might be satisfied. The amount of time needed to complete a query depends on the number of servers which must be queried before reaching a "terminal" database that contains the actual data, not a summary. This depends on the structure of the database interconnection graph. With N servers, the worst case is a simple line, shown in Figure 5.3 (each dot is a server, and each line is an interconnection between servers), and the best case is a tree. Assuming each server holds both data and indices, with data distributed uniformly across the servers, the result is average referral chains of anywhere from $N/2$ servers (in the case of a line) to $1/2 \log_2 N$ in the case of a binary tree. This assumes perfect indexing. False positive matches can increase these numbers (although there won't be false positives for TRIP). How many hops might there be, in practice? In the PSTN, there are at most six switches between any two points. Assuming the numbers are roughly the same for Internet telephony, and assuming approximate query and processing times of 100 ms, the increases in call setup will be approximately 300 ms on average, which is noticeable but not unacceptable.

Indexed databases are also very complex. They require computation of indices (which are often difficult to update incrementally as new services come and go), and most importantly, require the construction of some kind of interconnection topology. These topologies must handle

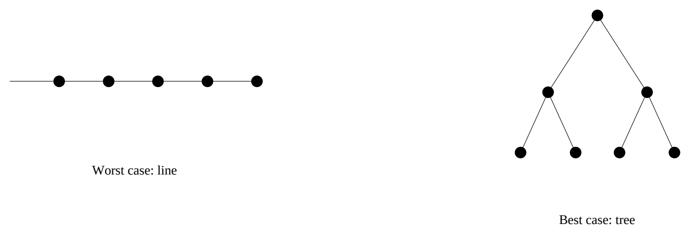


Figure 5.3: Worst and best case topologies for indexed database query times

failures, new servers, load balancing, failover, and administrative policy. To be done correctly, this requires complex routing protocols, such as TRIP, and policy protocols, which, to our knowledge, have yet to be addressed in the literature.

5.4.5 Multicast Push and Pull

An alternate architecture for discovery of services is through multicast. Generally, protocols for multicast discovery fit into either a *pull* model, where the client interested in the service multicasts a query, and responses are sent back (using either multicast or unicast) from service providers, or into a *push* model, where service descriptions are multicast out to a group, and clients can listen to them.

Multicast pull models are the basis for a variety of bootstrapping services, such as the Service Location Protocol (SLP) [152] which contains a mechanism for discovery of a local service through a multicast search, and the Multicast Address Dynamic Client Allocation Protocol (MADCAP) [203], which contains a component that allows a client to discover its MADCAP server through multicast. Multicast pull has also been used as a tool for content discovery in mobile ad-hoc networks [204].

Multicast push works by having servers periodically advertise their services to the network on a well known multicast group. SLP provides such a mechanism. It has also been used in wide area networks for advertisement of media sessions, as part of the Session Announcement Protocol (SAP) [68].

Multicast push and pull were used in conjunction as part of the Simple Service Discovery Protocol (SSDP) [205]. SSDP builds upon HTTP with extensions to support transport over UDP and multicast. Otherwise, it works in a similar fashion to SLP, but without a DA.

The primary benefit of multicast push and pull protocols is the absence of any third party, which differs from all of the other architectures we have discussed. However, there are drawbacks.

Multicast pull protocols don't scale to the wide area. The total query traffic for gateway discovery, 150 Mb/s, will actually be multicast to all gateways which listen for the queries. Of course, since revenue is derived by accepting calls (and thus answering queries), gateway providers are incented to listen for, and answer, all queries. This means the total network traffic could be 150 Mb/s to 51,000 gateways across the wide area Internet, a substantial number.

Multicast push protocols can scale much better; servers can be programmed to rate limit their advertisements (this is done in SAP, for example, using the reconsideration algorithm we described in Chapter 3.) However, the convergence time for clients listening to these advertisements is simply too long. If we assume clients are connected to the network through 56 kb/s modems, the bandwidth for distributing service advertisements is limited to that. With 51,000 servers, each with a description approximately 1kB in size, it will take slightly over two hours to learn about all gateways. It is not acceptable for a client to need to wait two hours before making a phone call. Caching of the data, or some kind of pull from a local server, can help. Indeed, this is the basis for the Wide Area Service Discovery Protocol (WASRV) that we propose below.

5.4.6 Summary of Existing Architectures

Our conclusion is that each of the five architectures has some strengths and weaknesses, but none alone seems sufficient. Centralized databases have substantial query loads and a middleman trust problem. Regionally replicated protocols eliminate the query loads, but still retain the middleman problem. Distributed database protocols are difficult to apply to the general service discovery problem because of the lack of a key upon which to distribute the data. They also have fundamental administrative difficulties for the database administrator that limit their deployment. Indexed databases provide excellent scalability, at the cost of increased query times, and don't suffer as much from the middleman problems, though they do exist. Multicast pull mechanisms don't scale. Multicast push can scale, and it eliminates the need for a middleman. But, it can result in an undue burden on clients for caching.

As a result, we have developed a protocol called the *Wide Area Service Discovery Pro-*

tocol, or WASRV, which combines the best components of all three architectures [154]. We describe this protocol in the next section.

5.5 Wide Area Service Discovery Protocol

We have proposed the Wide Area Service Discovery Protocol (WASRV) as an extension to the Service Location Protocol (SLP). It combines SLPs strengths in query flexibility, auto-discovery and rapid query resolution with wide area multicast dissemination mechanisms, similar in concept to SAP. We make use of multicast to massively replicate the contents of the service databases in each domain to other domains. Our use of multicast congestion control, similar to the operation of RTCP, allows the multicast dissemination component to limit wide area data traffic to a controlled rate independent of the number of database entries being replicated. By limiting the scope of queries to within a domain only, data traffic is further limited and the response times for a query become fast. Using multicast for data dissemination also means that replication within a domain is trivial. This allows many servers to share the query load within a domain. Since data is transmitted directly from domains that own services to domains that wish to use them, issues of administration and security are simplified. By basing our protocol on SLP, we inherit its security features, which primarily allow for secure authentication of the source of the data.

We also eliminate the complexities associated with indexed databases. The function of building interconnection topologies is already solved for us by using the underlying layer-3 multicast routing architecture. This architecture provides for scalability, healing, and addition and deletion of nodes. Rather than reinvent this at the query routing layer, we make use of it directly at the IP layer.

The cost of our mechanism, and its primary drawback, is in the storage space and search times required at a particular server. Because directory servers will contain large portions (but not all) of the global server database, the storage requirements, and the time to search the database, may be high for general services (although no worse than the centralized or regionally replicated architectures). Furthermore, we can enhance the scalability by applying the indexing mechanisms used in CIP and SDS locally within a domain, allowing us to partition data amongst servers while avoiding the administrative burdens of wide area interconnection.

However, for gateway discovery, our mechanism works very well. The entire global server database is quite small (51 thousand entries), and easily searchable with today's technologies. The query load required (2822 queries per second) is now widely distributed across providers, so that the load on any one system is very small. As of June 2000, in the U.S. there were 160 Incumbent Local Exchange Carriers (ILECs) and 76 Competitive Local Exchange Carriers (CLECs). Assuming roughly that many service providers for Internet telephony, with users uniformly distributed amongst providers, each provider would need to handle $1/236$ of the query load, which is approximately 12 queries per second. This is also trivial.

5.5.1 Terms

For clarity, we first define some terms in addition to those specified in SLP:

Service Location Protocol Domain (SLPD): An SLPD is a collection of UA's, SA's, and DA's under the administration of a single authority. DA's within one SLPD provide service only to those UA's within the SLPD. SA's within one SLPD may register their services using SLP mechanisms only to DA's within their SLPD. An SLPD is equivalent to a provider in the discussions above.

Brokering Agent (BA): A brokering agent collects advertisements learned from AA's in remote SLPD's. A BA is much like a DA, except that it may not collect information about all service types. In a sense, a BA provides "brokering services" for a specific set of service types that it knows about. Such a device is useful since the scope and range of services on a wide area network can be large. To reduce storage requirements, and allow for optimized processing and software, a BA only worries about one (or several) service types.

Advertising Agent (AA): An advertising agent is responsible for advertising information about services within an SLPD. An SA, DA, or BA may act as an AA for some subset of the services it knows about.

SLP Version 3: . We refer to WASRV as Version 3 of SLP.

5.5.2 Basic Operation

The multicast extensions allow for both wide area service discovery, and multicast-based registrations within an SLPD. Its architecture is shown in Figure 5.4

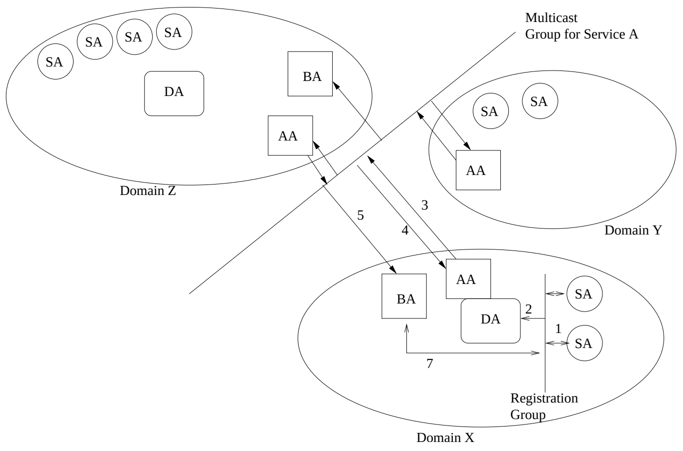


Figure 5.4: WASRV architecture

Within a domain, multicast can be used to allow SA's to register themselves with SA's and/or DA's. Multicast within an SLPD further enhances its scalability for very large domains (like `aol.com`), where there may be a multitude of DA's, each of which shares the load from various SA's.

The operation of the multicast registration within an SLPD is simple. The domain is configured with a single, well-known, administratively scoped multicast group that is used for multicast registrations. Call this the "registration group". Note that this group address is different from the ones currently used in SLP to discover services and send out DAAdverts. A version 3 DA listens to this group (line 2 in Figure 5.4), and version 3 SA's send to the group (arrow 1). Optionally, BA's can listen to the group as well.

Within a domain, there may be a mix of Version 3 aware SAs and DAs. If there is no version 3 DA and no BA, version 3 SAs would be wasting bandwidth by transmitting their registrations over the registration group. To remedy this, we mandate that all BAs and DAs send advertisements on the registration group. If an SA hears an announcement from any device on the registration group, it begins using multicast for its registrations. If there are no announcements on the registration group, the SA will unicast its registration to the DA. When there are a mix of unicast and multicast capable DA's, the SA will unicast its registrations to those non-multicast capable DA's, and multicast the registration as well. This allows an administrator to gradually upgrade an SLPD from unicast only operation to multicast without complex administration or configuration. The same mechanisms are used for Deregistration.

Some subset of the services available within the SLPD can be advertised onto the wide area network to other SLPD's. This advertisement is accomplished by means of an advertising agent (AA). An SA, DA, or BA within an SLPD can be configured to act as an AA. When the AA is colocated with the SA, it must be explicitly configured to advertise some subset of the services known to the SA. When the AA is colocated with a BA or DA, scopes are used to determine which services are to be advertised externally. An administrator assigns an SA to a scope if it wishes its service to be advertised externally. An AA is given the private keys for the scope, and will advertise any services it learns within that scope. This maintains the decentralized model of service configuration, yet allows for administrative controls over which services are advertised.

For each service type, there is a wide area multicast group used for advertisements of servers for that service type. The AA's send these services to the wide area multicast group (arrow 3). The AA's must also join the multicast group (arrow 4). This allows them to count the number of other AA's which are advertising, and then scale the rate of their advertisements proportionally in order to control multicast bandwidth usage.

The BA's within an SLPD join the wide area multicast group corresponding to the service types they wish to broker for (arrow 5). As a result, they will build up a service database of services located in remote SLPD's. An SLPD may have multiple brokers for a particular service type. Furthermore, these brokers may not retain all of the advertisements they receive, dependent on some local policy or policing operations.

BA's also generate service advertisements that they send to the DA's within their SLPD. To do this, the BA acts as an SA. The service type it provides is the abstract type *broker,* and the concrete type is the one corresponding to the service type they are brokering. The service advertisement contains the service attributes of the brokering services that are provided. The BA, acting as an SA, can register itself with the DA using the unicast mechanisms of SLP, or the multicast approaches described here (arrow 7). In this fashion, the DA's within an SLPD know about all of the brokers within the SLPD. The DA's can use this information to decide which BA to contact in order to resolve a query from a UA.

Based on this description, a client can locate a service as follows. Using current SLP methods, the client locates a DA in its domain. It sends out a request for a particular service to the DA. The DA may be able to resolve the query. This may be possible if the required service is within the domain. Otherwise, the DA cannot resolve the service request. However, a DA will always know about the BA's in each SLPD, and know which service types they are providing broker service for. The DA can then contact the BA directly with the query, and then forward the resolution back to the client. In this fashion, client behavior is unchanged between SLPv2 and SLPv3.

Clients may also obtain information about BA's in remote SLPD's through some out-of-band means, such as an advertisement on a web page, and then contact them directly with queries. This allows for competitive broker services (For example, a BA with the largest collection of service announcements from a media servers, with the fastest search engines, could offer its services to UA's outside its SLPD, and possibly charge for the brokering services provided).

5.5.3 BA URL's and Attributes

Broker agents are both repositories for service information, and service providers themselves. Since they listen to particular multicast groups to build up a service database for particular service types, BA's can be considered as providing broker services for those particular service types.

Consider a BA which collects service advertisements about the service type remote-fax. This BA can then be considered as providing broker services for remote-fax; the BA can direct users to the right remote-fax based on the attributes of the request. Since this brokering is a service itself, it can be described by a service URL as well. If a broker provides brokering for some service srvtype, then the URL for the broker service is [206]:

```
"service:broker:" srvtype "://" addr-spec
```

where the addr-spec is the address of the broker.

Furthermore, like other services, the broker service is characterized by attributes. These attributes are always a superset of the attributes which characterize the service being brokered. When a broker has a particular attribute and value pair which are also a valid attribute value pair for the service, it means that the broker collects service registrations from servers which have that value for the attribute.

5.5.4 Message Formats

SLPv3 defines no new messages or message syntax beyond what is described in SLPv2 [150].

However, some of the fields have a slightly different semantic. The differences are:

Version: This protocol document defines version 3 of the Service Location protocol. All multicast registrations from SA's, and advertisements from AA's should have the version number set to 3.

Bitfields: These have the same definition as in SLPv2. For multicast registrations and AA advertisements, the multicast bit is set. The overflow bit may not be set since all multicast registrations are UDP.

XID: The XID field is used to distinguish updated registrations from unchanged registrations. A registration message is considered unchanged if the attributes contained within are the same as the ones in a registration which was previously sent. The XID field is set to 0 for the first registration for a particular service URL. If the service attributes for that URL are unchanged the next time the registration is sent, the XID field is not incremented, else it is incremented by 1. This field helps BA's and DA's to decide whether or not to process a message depending on whether they have seen it or not.

5.5.5 SA Behavior

Service Agents can operate in one of three modes:

Unicast registration: Registrations are unicast to the DA('s) using only those mechanisms described in SLPv2. This happens when all DA's in the SLPD are v2, or the SA is v2.

Multicast registration: Registrations are multicast to the registration group. This happens when the SA is v3, and ALL DA's are v3.

Hybrid registration: Registrations are unicast and multicast. This happens when there is a mix of v3 and v2 DA's, and the SA is v3.

To determine which mode to operate in, the SA listens to the registration group. If, after WAIT_TIMER seconds after joining this group, the SA does not receive any DAAdvert messages or Multicast Service Registrations from a BA which is brokering for the service type provided by the SA, the SA decides to operate in unicast mode.

In unicast registration mode, the SA registers its services using the unicast mechanisms described in SLPv2. The SLPv2 mechanisms will provide the SA with a list of DA's, which we call the unicast DA set, to which it must register. However, the SA continues to listen to the registration group. If, at any time, the SA receives a DAAdvert message or Multicast ServiceReg from a relevant broker, it must switch modes to hybrid.

If, when listening to the registration multicast address, the SA in unicast mode receives either a DAAdvert or Multicast ServiceReg message from a relevant broker, it switches to hybrid mode. In this mode, the SA begins to multicast its service registrations, based on the rules described in Section 5.5.9. The SA also builds up a list of DAAdverts received from the registration group. The set of DA's which are learned through multicast is called the multicast DA set. This set is dynamic. Any DA which has not sent a DAAdvert for more than five times the current transmission interval (this interval is the period between messages from an entity (BA, SA or DA) to the registration group; see Section 5.5.9) is removed from the set. The SA continues to unicast its registrations to any DA which is in the unicast DA set and not in the multicast DA set. The SA also maintains a partial list of brokers. This list contains those brokers which provide broker service for a service type advertised by the SA. Any broker which has not sent a Multicast ServiceReg message for five times the current transmission interval is removed from the list.

If, at any time, the multicast DA set becomes a superset of the unicast set, the SA switches to multicast mode. Furthermore, if the multicast DA set and broker list should become empty, the SA reverts to unicast registration mode.

In multicast registration mode, the SA registers its services using the multicast mechanisms described here. In this mode, all services supported by an SA are registered using multicast. If the multicast DA set should ever cease being a superset of the unicast DA set, the SA reverts to hybrid mode.

This process is depicted in the state machine of Figure 5.5.

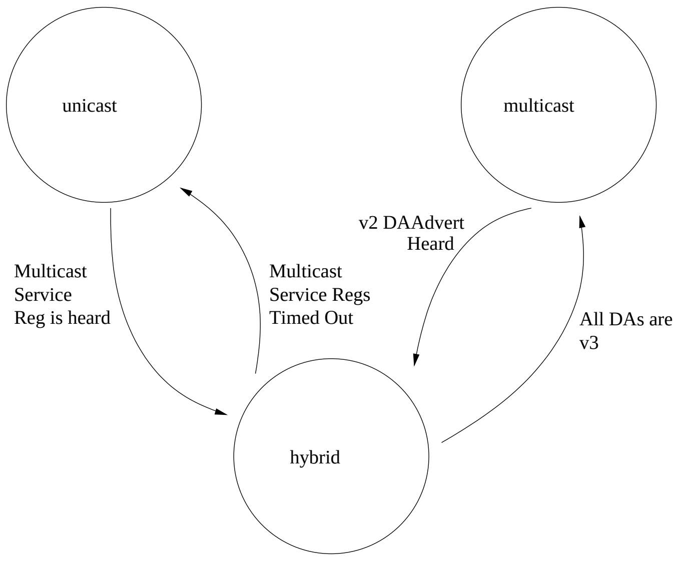

unicast
multicast
v2 DAAdvert Heard
Multicast Service Reg is heard
Multicast Service Regs Timed Out
All DAs are v3
hybrid

Figure 5.5: SA state machine

These basic rules allow an SLPD to be gradually upgraded from v2-only operation to v3 operation with no configuration. Of course, an administrator can force an SA to always unicast its registrations to some set of DA's if desired, even though the DA's may be v3 capable.

5.5.6 DA Behavior

The behavior of a DA is nearly identical to that of SLPv2, with three major differences. The first is that a DA may need to contact a BA to resolve the request. Secondly, a DA must listen to the registration group to collect advertisements, and third, a DA must multicast its DAAdvert messages on the registration group as well as the DA discovery group defined in SLPv2.

5.5.6.1 Multicast Listening

Both BA's and SA's may use multicast to register themselves with the DA. It is therefore necessary for the DA to listen to the registration group used within the SLPD. By joining this group, the DA will receive multicast service registrations from BA's and SA's.

A DA must keep all service registrations received from brokers. A DA may drop registrations received from SA's, but only if the DA knows of a BA in the SLPD which is providing broker services for that SA. Otherwise, the DA must keep all service registrations received from SA's (unless forbidden by some special administrative policy).

This allows a DA to cease storing registrations from SA's as soon as there is a BA which is storing them. Since the DA always knows about BA's, the DA will still be able to satisfy client queries for local services by contacting the BA.

5.5.6.2 Contacting BA's

In SLPv2, it is generally assumed that a DA knows about all services (at least within its scope). However, in SLPv3, there are many reasons why a DA will not know about a service:

- The service is local to the SLPD. However, the SLPD is using the multicast mechanisms for registering some of the SA's. This means that there are BA's within the SLPD, and that the DA may therefore not store all service registrations it receives.

- The service is not local to the SLPD, and the BA's are not colocated with the DA. This means that information collected via multicast advertisements from remote SLPD's are not known to the DA.

Even though a DA may not know about a service, it will always know about the BA's within the SLPD which broker for that service type. This is because all brokers must register themselves with all DA's. This registration describes the service types which are being brokered (via the URL), and the characteristics of that brokering service. These attributes describe, among other things, the subset of the services which are brokered (for example, only those SA's providing remote-fax service with FILE-TYPE of TIFF). In that case, the URL path of all service:broker

URL's should include the predicate describing the restriction. For example, the aforementioned remote-fax broker would have a service URL:

```
service:broker:remote-fax://b3.brkr.com/FILE-TYPE=TIFF
```

When a DA gets a service request which it cannot resolve, since it doesn't know about the service at all, or one which it can resolve, but for which it has only partial information (since there are brokers in the SLPD for that service), the DA should consult a broker. To determine which brokers to consult, the DA eliminates those brokers which do not broker for the service type and service which was requested. Such an elimination can happen if the service type is not brokered, or because the service type is brokered, but the attributes of the broker indicate that it does not broker for that particular service. The resulting set of brokers can potentially resolve the request. The DA then further filters the list based on any local policy (for example, do not consult brokers which require payment for their services).

The DA then acts as a UA, and sends a ServiceRequest message to each broker. It may send a message to each broker in parallel, or in series. Each BA will answer the request with a ServiceReply, containing a list of URL's for servers which match the query requirements. The DA may then send an AttributeRequest message to each SA to obtain additional attribute information which may be required to resolve the original request. The DA should not cache the resulting URL's and attribute information. The final match is then determined, and a list of URL's is returned to the UA in a ServiceReply message.

Consider the following example. A UA formulates a query as follows:

```
(&(service-type=media-server)
  (movie = eraser)
  (cost<=*))
```

This query asks for a media server which has the movie "Eraser", for which the cost of viewing the movie is the cheapest. This query is then sent to the DA. The DA doesn't know about the media-server service, but it knows about two BA's in the SLPD providing the media-server-broker service. It then sends the query to each of the two servers. Both respond with one URL matching the query (presumably the cheapest media server each knows about; this should be the same, but may not be since there are no enforced database synchronization rules). However, if the URL's are different, the DA must determine which is actually cheapest. To that end, it sends an AttributeRequest message to each SA containing the URL for the service. This yields a set of attribute value pairs, including cost. The DA then selects the cheapest of the two, and returns the resulting URL to the UA.

Note that if the query did not contain the operator <=* (the minimum operator) or the operator >=* (the maximum operator), the AttributeRequest query would not have been needed. This is because the minimum and maximum operators are relative; their results depend on the values of attributes of other servers. If the operators are all absolute, the response to the ServiceRequest from any BA will satisfy the query independently of the responses from any other BA.

In order to improve scalability, the DA must not query the SA's with an AttributeRequest, as above, unless the UA request contains either the <=* or >=* operator on an attribute. If the original request does not contain the MIN or MAX operator, the DA may return the entire list of URL's obtained from the BA's, or it may return some subset as it sees fit.

5.5.6.3 Multicasting DAAdverts

As in SLPv2, the DA's will multicast DAAdverts to make themselves known. However, in SLPv3, DA's will also multicast DAAdverts to the registration group. This effectively announces its ability to receive multicast registrations. The rules for how to transmit the DAAdvert into this group are described in Section 5.5.9.

5.5.7 AA Behavior

An Advertising Agent (AA), is responsible for advertising attributes about the services within its SLPD to other SLPD's. An SA, DA or BA may act as an AA for some subset of services which it knows about. When the SA is acting as an AA, it is administratively configured with the set of services to advertise. Other services which the SA administrator would like to have advertised, but not by the SA itself, should be registered to the DA (and possibly BA) with a different scope. When the BA or DA is acting as an AA, they advertise those services in scopes with which they have been configured to advertise and provided private keys for.

AA's send Multicast Service Registration messages to a wide-area multicast group (called an advertising group). The rules for choosing the address of this group, and for when to send to the group, are described below in Section 5.5.9.

5.5.8 BA Behavior

A BA is responsible for collecting multicast advertisements heard about a particular service.

5.5.8.1 Receiving Advertisements

A BA is administratively configured to broker for some set of service types, S. To do this, it determines the multicast groups to listen to for each service in S (see the section below on multicast group selection), and then joins them. This will cause the BA to receive advertisements. Since the mapping of services to groups is many-to-one, two services may both share a multicast group. The BA must drop all service registrations received on a multicast group for which it is not brokering.

5.5.8.2 Policy

Once the BA has received a registration for a service that it is brokering, it still has the option of dropping this registration. It is within the rights of an administrator to configure a BA to drop advertisements based on any criteria which can be programmed into the BA. This allows administrators to implement policy, in much the same way BGP policies allow routers to choose which routes to accept. Some examples of policies include:

1. The BA may drop all registrations received from a set of IP addresses.
2. The BA may drop all registrations which have an attribute set to a particular value.
3. The BA may drop all registrations which do not contain authentication blocks for the URL's.
4. The BA may only accept registrations with an attribute equal to a particular value.

5. The BA may only hold the most recently received 100 registrations.

This list is not meant to be exhaustive; it is only to illustrate that there is a wide range of possible policies, limited only by the imagination of the administrator.

Note that these policies apply to services learned about from the wide area multicast group. Services learned from the registration group inside an SLPD should not be dropped.

5.5.8.3 Policing

An optional mode of operation for a BA is “policing”. In this mode, the BA will discard advertisements from AA’s who are sending messages faster than is allowed by the protocol operation described below. This will hopefully deter AA’s from flooding advertisements to wide area group, which is a form of a denial of service attack.

For each distinct AA which sends an advertisement, each BA will maintain a few pieces of information. This information includes the last time an advertisement was received from that AA, and a violation counter, which is initialized at zero. At all points in time, BA’s maintain the current minimum transmission interval (described below), which reflects the smallest allowed interval between transmission of a packet from any AA. When a packet is received from an AA, the BA computes the difference between the current time, and the last time an advertisement was received from that AA. If the difference is less than the minimum transmission interval, the counter is incremented. In either case, the value for the last time an advertisement was received is updated to the current time.

If the violation counter hits three, the BA should discard any further advertisements from this AA. The BA may eventually reset the counter, and begin accepting advertisements again, after some suitably long interval, at the discretion of the administrator.

This policing mechanism is just a variant on traditional leaky bucket policing algorithms [207].

5.5.9 Sending Multicast Advertisements

Both within an SLPD, and across the wide area network, entities will periodically transmit multicast packets. This section discusses the rules by which these packets are sent.

5.5.10 Scheduling Transmission of Advertisements

The transmitting entity is assumed to have some set of packets, S, which it wishes to send to a multicast group. This situation arises when:

- An SA within an SLPD wishes to multicast its service registrations to the registration group.
- A DA within an SLPD wishes to multicast its DAAdverts to the registration group.
- A BA within an SLPD wishes to multicast its service registrations to the registration group.
- An AA wishes to multicast some of its services to the wide area network.

The rules for transmission in all of these cases are identical:

An entity wishing to send to some multicast group must be a member of the group. Call the time it first joins the group time t_0. There must be only one entity per IP address that is sending to a particular multicast group. This allows the IP address to be used as a unique identifier for that entity. A multi-homed host should choose one interface for sending its messages.

After joining the group, the entity must continually maintain a list of all of the source IP addresses seen in packets sent to the group. At any point in time, this list is used to determine the group size estimate, L, which is a count of the number of other entities transmitting to the group. This estimate is most easily obtained by counting the number of IP addresses seen. If storage is limited, the entity may use statistical sampling to reduce the size of the list, and obtain a statistical estimate of the group size estimate [86].

A fixed amount of bandwidth, B, is assumed to be available for packet transmissions to the multicast group. By default, B is arbitrarily 8 kb/s, but it can be changed through configuration. In this way, if there are 1000 entities sending to the group, the period between messages from an entity is around 25 minutes on average. For ten-thousand entities it is a little over four hours.

The entity selects an element (message) from S for transmission. If this element differs from the information in the element last time it was transmitted, the XID of the message is incremented, and the age of the message is set to zero. If the element is not different, the age of the element is incremented.

Let t_p represent the last time any message from the entity was transmitted. The time of transmission for the next message, t_n, is set to:

$$t_n = t_p + R(\frac{1}{2}) \max(T_{FAST} 2^{\min(16,age)}, \frac{LK}{B}) \tag{5.1}$$

$R(1/2)$ is a random number uniformly distributed between 1/2 and 3/2. K is the size of the message in bits. T_{FAST} is 1 second. This formula ensures that the total transmission rate of packets to the multicast group never exceeds B, by evenly dividing this rate among all of the L senders in the group. Furthermore, when group sizes are small, the packet rate is limited depending on the age of the message. A new message is transmitted with a small period (T_{FAST}, and the period increases as the age of the message grows. Eventually, the period increases to about once a day.

When time t_n arrives, the entity does not send the packet. It recomputes the above formula, yielding a new time, t_n'. If t_n' is less than t_n, the packet is transmitted, t_p is set to t_n, and the entity selects another message to transmit, computes its transmission time tn as above, and the process repeats. If t_n' is more than t_n, transmission of the message is further delayed until time t_n. This algorithm is identical to the unconditional reconsideration algorithm described in Chapter 3.

5.5.10.1 Timing Out Senders

In addition to maintaining a list of IP addresses of senders to the group, each entity also maintains the last time a packet was received from that sender. Furthermore, each entity maintains a timeout interval, T_o, which is equal to

$$T_o = 5 \max(T_{FAST} * 65536, \frac{LK'}{B}), \tag{5.2}$$

where K' is the MTU for the interface the entity is connected to. Any entity whose last transmission time is earlier than the current time minus T_o is assumed to no longer be active, and is therefore timed out. Its address is removed from the list of senders, and the group size estimate is decremented by 1. An entity which sends a multicast deregistration is also removed from the list, and the group size is decremented by 1.

5.5.10.2 Minimum Transmission Interval

For the policing operation, each entity may maintain an estimate of the minimum transmission interval possible for compliant senders, which is

$$T_{min} = \frac{1}{2}\max(T_{FAST}, \frac{128L}{B}), \tag{5.3}$$

This is nothing more than the minimum value for $t_n - t_p$ from Equation 5.1. 128 is chosen as the minimum value of K, the size in bits for an SLP packet.

5.5.11 Multicast Groups

There are two different cases for choosing the multicast group to send to. In the first case, the entity is an SA, DA, or BA, advertising its availability within an SLPD. In this case, all messages go to a single multicast group, called the registration group. This group is administratively scoped [208].

For AA's, advertisements are sent to wide area multicast groups. Each service is mapped to at least one multicast group. This multicast group is obtained by applying a hash function to the string which describes the service type (remote-fax, for example). The resulting value indicates an index of a multicast group to use.

Furthermore, AA's may also advertise onto private multicast addresses. This is useful when a set of SLPD's wish to share information about services, but only among themselves.

5.5.12 Reducing the Storage Requirements of a BA

If a BA chooses to act as a broker for a specific service type, it will end up storing all service advertisements from all servers of this type. In order to reduce this problem, we can apply the indexing and query routing techniques described in RFC 1913 [195], the Secure Discovery Service (SDS) [196] and the Intentional Naming System (INS) [197], but do so locally, within each SLPD.

To do this, each BA in an SLPD chooses a subset of the services to store. This subset can be chosen in any arbitrary way, and can even overlap amongst BAs. Then, when each BA advertises itself to the registration group, the service advertisement can either contain the filtering rule applied to determine which subset of the service space is stored, or, if no such rule can be constructed, an index or centroid can be sent instead.

The result is that we can reduce the storage requirements in a BA, but avoid the interconnection complexities of protocols like SDS by using the multicast registration techniques of SLP.

5.6 Conclusion

In this chapter, we have introduced the problem of gateway discovery, and shown how it is a subset of a more general problem of wide area service discovery. With a well-formulated set of requirements for wide area service discovery, we investigated existing architectures for service discovery to determine their strengths and weaknesses. We were able to classify existing solutions as either centralized databases, regionally replicated databases, distributed databases, indexed databases, or multicast push or pull. We found each to have strengths and weaknesses, so that none was optimal.

We then presented our own protocol for service discovery, called the Wide Area Service Location Protocol (WASRV), which we structured as an extension to the existing Service Location Protocol. We were able to construct WASRV as a backwards compatible extension to SLP, and demonstrate an easy migration path. We argued how WASRV compares favorably to existing solutions, and combines the best aspects of all of them. We found it to be especially well suited for the gateway discovery problem, although it does require deployment of multicast, which is still in its early stages.

Chapter 6

Application Architecture

6.1 Introduction

As we outlined in Chapter 4, the primary role of signaling protocols are name translation, call state modification, media stream negotiation, and participant management. Another important function of signaling protocols are the invocation and delivery of features, applications and services.

A *feature* is defined as a package of incrementally added functionality providing services to subscribers or the telephone administration [209]. Examples of telephony features include call forwarding, call hold, call transfer, Interactive Voice Response (IVR) and multiparty conferencing. The term *application* is often used to describe a more complex feature. Examples of applications include scheduled conference bridges, interactive voice response systems, and personal attendants. The line between application and feature is often blurry. We do not attempt to make a distinction here, and we use both terms interchangeably. *Service* is a more general term. It refers to any kind of value added function provided by one entity for another.

There are many problems that need to be addressed in order to provide telecommunications applications. The first of these is the *service architecture problem* – where does the code for features and applications reside? How do the various components in the system interact with each other to provide those features? How do the users interact with the systems to provide input to the feature or application? This problem is different from the well studied, but equally important and related *feature interaction problem* [209, 210] – how do multiple features and applications interact with each other? How can multiple features be applied for a call without requiring each feature to know about every other feature ahead of time?

In this chapter, we consider the service architecture problem for Internet telephony. First, we discuss the requirements imposed on a service architecture, based largely on those aspects of IP networks that are different from traditional telecommunications networks. With the problem properly framed, we proceed to study the vast literature that exists on service architectures. We segment these solutions into several categories, namely centralized architectures, distributed object architectures, distributed component architectures, and mobile code architectures, and show examples of past work in each of these areas. We discuss why the existing approaches in each of these categories cannot meet the requirements for an Internet telephony service architecture. We then proceed to present our own solution, the *application component architecture*, which combines the best facets of existing work. After comparing it to the existing approaches, we conclude and discuss future directions.

6.2 Requirements for an Internet Telephony Service Architecture

As we note in the Chapter 1, Internet telephony differs from traditional telephony in that there exist other IP applications, such as web, email, instant messaging, and presence, which can be used to enhance telecommunications services. We believe that the primary benefit of Internet telephony to end users are the new services it provides. Services which combine other IP applications are principal amongst the new services which can be provided. A service architecture must give consideration to how such applications are provided.

We wish to support a broad range of applications on our architecture. Of particular importance are applications that require user interfaces during the execution of the application. Our architecture needs to support user interfaces based on web input and voice input. Our architecture also needs to support applications that are invoked by placing a call or by clicking on a link on a web page. Our architecture needs to support applications that involve multiple users.

The architecture should enable development and execution of telecommunications applications by any third party. Furthermore, the system should enable an application to be composed of services provided by several providers. For example, a particular application might make use of a conferencing system from provider A, a messaging system from provider B, and a translation system from provider C. Allowing third-party service providers, and component service providers, facilitates rapid development of new applications.

Finally, the architecture should allow for applications to be provided to users connected to the telephone network, and should allow for service components within the telephone network (such as conference servers and IVR servers) to become available as resources to users connected on the IP network. Obviously, the user interface for applications will differ for users connected via the telephone network, compared to users on a PC.

It is difficult to show that an architecture can support *any* application. In order to provide basic validation, we consider four target applications with specific user interfaces. Our goal is to demonstrate that the architecture readily supports these applications. The applications were chosen because they cover a broad range of functionality. Our target applications are:

Pre-paid calling cards: In this application, a user purchases a calling card with a certain number of minutes of usage. To make a phone call, the user dials an access number. A voice response system answers, and prompts the user to enter the PIN number written on the calling card. Once the PIN has been entered (using the dial pad), it is verified by the system. The user is then prompted to enter the destination phone number. The user enters it on the dial pad. The application completes the call to the destination number. If the user uses up their minutes, the application terminates the call.

Click-to-dial: In this application, a user browses the web site of some retail vendor. The user has a question on some product provided by the vendor. The user has the option of clicking on a hyperlink to call the customer service desk. Clicking on the link fetches a form. The form asks the user to enter in their home phone number. Once the user enters in the number, and submits the form, the user's home phone rings. When they pick up, the customer service representative is on the other side.

Auto-conference: This application is used to start conference calls automatically based on a *presence* service that detects the availability of the conference participants. A presence service, defined in RFC 2778 [91], is a system that allows subscribers to request notifications when the status of a desired user changes. Here, status refers to the ability and willingness of a user to be communicated with. In the simplest systems, a user's status is *available* when a user logs in to their computer and connects to the presence service, and *unavailable* when they disconnect. More complex systems may determine that a user is unavailable when they are logged in, but in the middle of a phone call. Furthermore, the status of a user can be much more complex than just "available" or "not available". It can convey numerous means of communication, addresses of communication, and willingness to communicate with those addresses [91].

To use the auto-conference application, a user views a web page containing a form. The form contains the email addresses and phone numbers of the conference participants. Once submitted, the application waits until all participants are available, as determined through the presence service. When all are available, the application sends each participant an instant message. The message asks each participant if they actually wish to join the conference. It contains two hyperlinks, both of which resolve to the controller. One is clicked if the user wishes to join the conference, and the other, if they do not. If all click the hyperlink to join the conference within a specified time interval, the application rings the phone of each participant, and then connects each of them into a conference.

Web form entry for call center: This application is used to facilitate call centers, which provide customer support services to users. When a user has a question regarding a product they purchased, the user calls a customer service number. Before the call completes, the user's web browser automatically fetches a web page containing a form. The form prompts the user to enter in the product number in question, and other relevant customer information. When the user submits the form, the call completes to a customer service representative. The form data is provided to the customer service representative, in their web browser, when the call completes.

Speech-to-text for the hearing impaired: This service allows a hearing impaired user, who can only communicate with text chat, to interact with a non-impaired user on a regular telephone. The non-impaired user calls the number of the hearing impaired user. This causes the text chat tool of the hearing impaired callee to flash, indicating an incoming call. The called party clicks on a button to accept the call. When the call is completed, the callee can type text into their chat tool. All text is converted to speech, using text-to-speech algorithms, and presented to the caller. Anything said by the caller is recognized using speech recognition algorithms. The resulting text is presented to the callee in their chat tool.

6.3 Existing Architectures

In this section, we review existing work on telecommunications service architectures, and show how they do not meet the requirements outlined above.

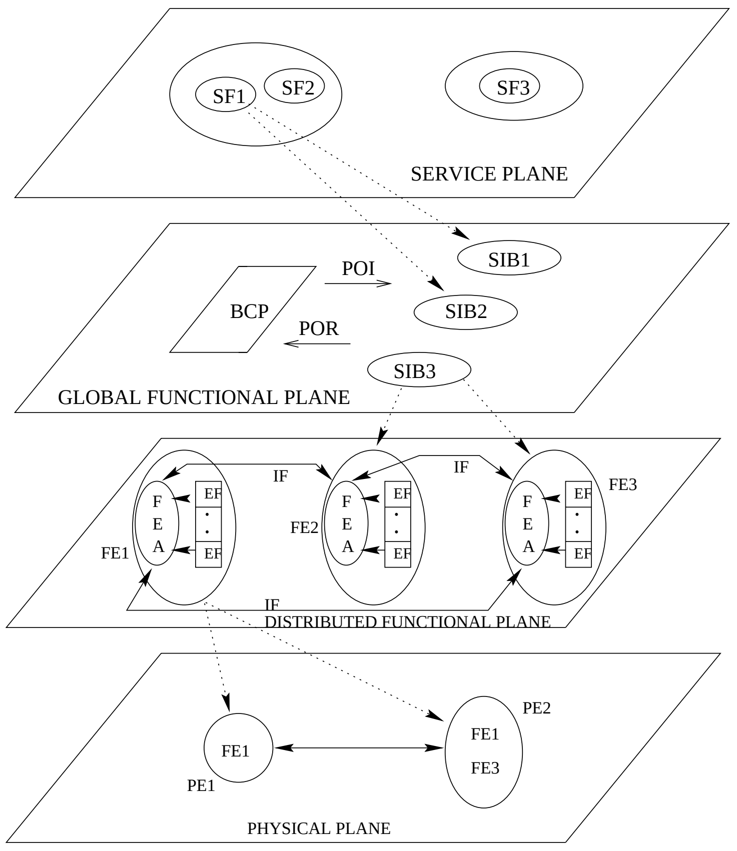
Much work has been done in this area, as we will show. To structure the presentation, we characterize existing work into several general approaches. The first are *centralized architectures*, where a single element is responsible for providing the features and applications for the users it manages. The second are *distributed software architectures*, where distributed object techniques, frequently based on CORBA, are used to design distributed systems. More recently, work has begun to emerge on *distributed component architectures*, where reusable components are defined to provide features. Finally, much work has been done on *mobile code*, where the code for an application is dynamically downloaded into a device (either the client or a server) for provision of the features and applications.

6.3.1 Centralized Architectures

6.3.1.1 Intelligent Network

The most important instance of a centralized architecture is the Intelligent Network (IN), which also happens to be the most widely deployed architecture. The Intelligent Network is a collection of standards codified by the International Telecommunications Union, Telecommunications Sector (ITU-T). It is first introduced in Recommendation Q.1201 [211], which provides the model for IN, duplicated in Figure 6.1. Faynberg [212] provides an excellent overview of IN.

The figure shows that the IN can be modeled as a number of distinct planes. The first



BCP	BASIC CALL PROCESS
EF	ELEMENTARY FUNCTION
FE	FUNCTIONAL ENTITY
FEA	FUNCTIONAL ENTITY ACTION
IF	INFORMATION FLOW
PE	PHYSICAL ENTITY
POI	POINT OF INSERTION
POR	POINT OF RETURN
SF	SERVICE FEATURE
SIB	SERVICE INDEPENDENT BUILDING BLOCK

Figure 6.1: IN conceptual model

is the Service Plane, which contains services (such as call-forward and freephone). The Global Functional Plane (GFP) defines these services in terms of atomic operations, called Service-Independent Building Blocks (SIB's). A SIB can be thought of as a "network level" machine instruction inside a computer. It defines a basic function on a piece of data. Some of the SIBs are *Compare* for comparing two values, *Queue* for queuing a call, and *Screen* for rejecting incoming calls. The GFP also defines the interaction between the telephone switch and the service logic on the general purpose computer, by means of a state machine on the switch (the Basic Call Process, or BCP), and remote procedure calls to the computer. The telephone switch has certain points in the progress of a call where remote procedure calls can be made (Points of Initiation (POI)), and points in the call where the service logic can instruct the switch to return to (Points of Return (POR)). Examples of POI's are *Address Collected* and *Busy*, which occur in the BCP when the digits for the call are collected, and when the remote party signals busy, respectively. Examples of POR's are *Initiate Call* and *Clear Call*.

The next plane is the Distributed Functional Plane (DFP), which maps the abstract GFP into functional blocks, and then defines the required flow of information between them. Finally, the DFP is realized in the Physical Plane, which maps the functional entities into real devices. The IN defines a number of devices, including a Service Switching Point (SSP), which is a telephone switch, a Service Control Point (SCP) which is a general purpose computer that can execute the remote procedure calls from the switch, and a Service Node (SN), which executes service logic, but also has a switch fabric so that it can generate tones, play announcements, and provide additional services. The physical plane also defines the protocols amongst these entities. Most important among these is the IN Application Protocol, or INAP [213], which carries the information for the information flows defined in the DFP. INAP can be thought of as a special-purpose remote procedure call protocol.

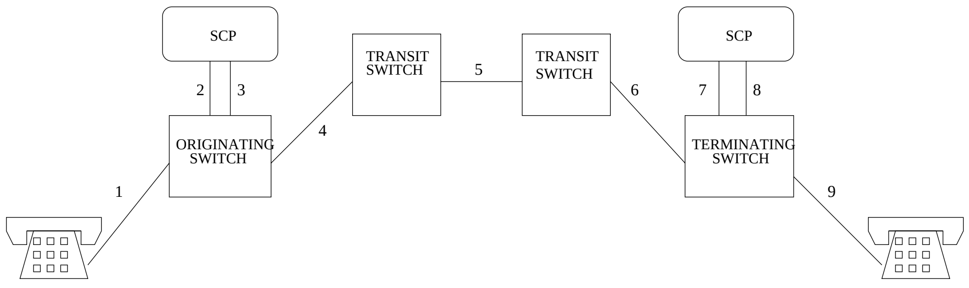


Figure 6.2: Call flow for a phone call

A simplified example of a call flow for a phone call that makes use of IN services is shown in Figure 6.2. In this figure, a call originates at the calling phone, which dials an 800 number. The call signaling arrives at the originating switch (1). Since the switch does not know how to handle the 800 number directly, it asks the SCP for further instructions (2), and the response (3) specifies the further actions the switch should take, including a routing number where the call should be connected to (4). The call passes through another transit switch (5) and eventually arrives at the terminating switch. This switch also consults an SCP for instructions (6), and the response (7) tells it to complete the call to a subscriber line (8).

The IN has been standardized incrementally, starting with a baseline set of services and associated call models and protocols (call IN capability set 1, or CS1). Increasingly more complex services, models, and protocols have been developed as part of capability set 2, and more recently capability set 3. The services enabled with CS1 include freephone, televoting, follow-me, call screening, and call forwarding, among others. IN Capability Set 2 (CS2) supports more powerful features, such as call transfer, call waiting, message store and forward, and conference calling.

Although the conceptual model alludes to distribution of the feature through SIBs, the SIBs for a single application reside entirely within the SCP which provides that application (originating or terminating).

Application of the IN concept to IP telephony is not hard. Instead of switches, proxy servers (or more likely, back-to-back user agents (B2BUA)) are used. The proxy or B2BUA runs a call model. It interacts with some kind of IP-based SCP, which provides instructions on how to proceed with call processing. Significant work has been done on mapping the states of a SIP proxy server to IN call models, in order to facilitate this kind of architecture [214, 215, 216, 217, 218, 219, 220, 221, 222].

However, this approach does not meet the requirements we have outlined in Section 6.2. Specifically, the dependency on a call model as a trigger for services makes it difficult to integrate other IP applications, such as instant messaging and presence, which might also generate events that need to trigger call processing. The centralized nature of the architecture does not facilitate provision of application components by a third-party. Finally, the IN approach does not foster rapid development of services. This is because it is difficult to reuse application code; there are no well defined internal interfaces or techniques for communicating with external entities.

6.3.1.2 MGCP

The Media Gateway Control Protocol (MGCP) [223], and its successor, the MEGACO protocol [224] are IP-based master-slave control protocols. They allow a centralized agent, called the Media Gateway Controller (MGC), to control one ore more slave devices, called Media Gateways (MG). The MG is able to connect together logical voice channels under its control. These logical voice channels can represent circuits on trunks that terminate on the gateway, or they can represent the voice generated from a handset on a telephone. Media events also occur on these channels.

These events include generation of DTMF, hookflash, and other tones. The control primitives in MGCP/MEGACO allow the MGC to subscribe to events on the channels in the MG. This way, the MGC can be alerted when the “pound” key is pressed on a telephone, for example. MGCP/MEGACO also allow the MGC to instruct the MG to connect logical voice channels together in some fashion. This includes taking a channel, compressing it with some encoder, and sending it over an IP network to a specified address, using RTP.

In essence, the MGC can be viewed as an extension of a class 5 switch into the Internet telephony realm. The only difference is that a class 5 switch actually sees the bearer circuits and switches them; an MGC gets notified about events on the bearer circuits and tells the device where to send them. As a result of this, services can be built on top of an MGC in much the same way they can be built on top of a class 5 switch.

For this reason, the same IN principals can be applied to services on an MGC. Even if IN is not used, the MGC can serve as a central point for the provision of services to end users, connected to that MGC. This model has been proposed by Huitema et al. [225], and has been more formally specified by the PacketCable Forum in its Network Control Signaling (NCS) specifications [226].

The MGCP approach has the same issues with meeting our requirements as the IN architecture.

6.3.2 Distributed Software Architectures

In this general class of architectures, the system is specified through the interconnection of distributed software components. Often, the communications between components is based on some object middleware, such as the Common Object Request Broker Architecture (CORBA) [227]. Services are provided by defining APIs on several objects distributed throughout the system. The systems all vary in the types of objects defined, and the interfaces between them. Blum and Molva provide an excellent overview of some of the early distributed software platforms developed for multimedia services [228].

Early work on distributed object systems for telecommunications focused on their application to a specific problem domain: computer supported collaborative work (CSCW). Hofte

[229] describes the implementation of a CSCW application on top of CORBA.

Blum and Molva [228] describe a system for distributed multimedia applications, called the Application Pool and Multimedia Terminal (APMT) architecture. It is based on CORBA, and allows clients to download scripts, usually in Tcl/Tk [230], that define a user interface to an application. The application itself resides on application pools, running on servers in the network. The platform provides primitives for establishing connections between entities in the system.

Arango et al. [231, 232, 233, 234] describe the Touring Machine, one of the earliest platforms for multimedia applications. Originally conceived for circuit networks, the Touring machine was extended to later cover packet networks [235]. The Touring machine was implemented as a set of distributed objects, using a custom distributed object protocol. The objects used to provide features include a *station object*, which models the points of attachment of users to the system, a *session object*, which represent a call or conference (and are created dynamically as new calls arrive into the system), and a name server. The system supports development of new features by exposing APIs on the station objects. The APIs are monolithic and limited in what they can provide [228]. Since the APIs are limited to those used by clients to attach to the system, there is no specific provision for new services provided by third parties.

Another early platform for multimedia applications is Beteus [236], which stands for Broadband Exchange for Trans-European Usage. It was primarily aimed at distributed multimedia classrooms, using an ATM network. It provided a limited API on one of its Site Manager, that enabled interconnection of sites and users. However, it does not scale, does not allow third party applications.

The Medusa system [237] provides an object-oriented distributed system for a variety of multimedia applications. It is more complete than many of the other earlier systems. Modules, which are the primary components of the system, were defined to provide conferencing, speech recognition, and gesture recognition, amongst many others. An application is constructed by chaining together modules, each of which provides different functions. The basics of the communications between modules is specified. The interface (which uses Tcl-DP [238]), allows for subscription and notification of changes in state variables, and the setting and getting of attributes.

The use of distributed object systems as the basis for an Internet telephony service architecture has some advantages. It facilitates distribution of components, allowing for the development of applications by third parties (at least in principle) and reuse of software to enable rapid prototyping of new applications. The systems can provide integration with other Internet applications. For example, since APMT allows for downloadable GUIs into clients, the GUIs can interact with the CORBA system and also with the web and other IP applications.

However, distributed object systems have some drawbacks. In order to truly provide third party development of components, standardized interfaces need to be defined between components. In other words, having a common RPC mechanism, such as CORBA, is not sufficient for interoperability. Unfortunately, each system described above uses its own, proprietary interface between components. Related to this, there is no clear way, in most cases, to access service components on the PSTN. A gateway could be constructed between the distributed system and the PSTN. However, the service interfaces defined on these systems do not map cleanly to the call signaling protocols used in the telephone network.

Another drawback is performance. The generality of the IP Inter-Orb Protocol (IIOP), used as the middleware protocol between components, comes at the expense of performance. Studies have shown that CORBA performance falls far short of more specialized communications protocols [239], however the applicability of these results to telecommunications service architectures is questionable (Gokhale [239] considers large TCP segments, which are unlikely for our application). We believe, however, that protocols developed for a specific application will generally outperform more general purpose protocols, in terms of latency and efficiency.

More significant, however, are the issues related to extensibility and interoperability. Interoperability between a CORBA client and the object implementation requires that both be based on exactly the same Interface Description Language (IDL) description. If we wish to extend the functionality of the system, the only method defined is through interface inheritance. This will allow the server to be improved with new methods, and yet still allow older clients to access its services. However, extensibility is more complex than just this scenario. A common case for network protocols is where the client wishes to request server processing of an optional feature defined in an extension to the protocol. The protocol can define mechanisms whereby optional features are completely ignored by the server if the extension is not supported, but processed if the extension is supported. This extensibility model is not possible with simple interface inheritance. These drawbacks will probably not be problematic in systems constrained to a single enterprise, but when systems are distributed across multiple providers and evolve over many years, which is a requirement we have defined for an Internet telephony service architecture, the limited support for extensibility is problematic.

Finally, CORBA requires that a client have a handle to the object on the server before using it. It can obtain this handle through a naming system, or through a discovery service. This discovery service only works within a domain, as does the naming service. This limits the usefulness of CORBA for interdomain operation. On the other hand, network protocols can be defined which provide this capability.

Therefore, we believe distributed software architectures, generally based on CORBA, meet some, but not all, of our requirements. Specifically, they do not adequately address the ability for third parties, in other domains, to provide components for an application. Their weak support for inter-domain interoperability is problematic for Internet telephony, which is, by definition, based on communications between users in different domains.

6.3.3 Distributed Component Architectures

Component Based Development (CBD) for software engineering has emerged as a new tool for the construction of distributed applications. D'Souza [240] define CBD as "an approach to software development in which all artifacts - from executable code, to interface specifications, architecture, and business models; and scaling from complete applications and systems down to small parts - can be built by assembling, adapting, and wiring together existing components into a variety of configurations". Component architectures are rooted in the premise that applications have components which recur time and time again in the same form. The characteristics of components are outlined by Jung [241]:

- They are decoupled, allowed them to be independently developed and delivered.
- Components have well-specified interfaces for the services they provide.
- Components have explicit and well-specified interfaces for services it expects from other

components.

- Components can be customized and composed with other components without modification of code.

Reusability is a key component of the CBD model, and it is the underlying concept in all of the characteristics defined by Jung [241]. Another key concept in the CBD model is that there is some mechanism to *assemble* the components together. In this context, assembly refers to the temporal ordering of component operation in order to provide an application.

In a distributed CBD model, the components can be physically separated.

CBD has traditionally been implemented using object-oriented programming techniques. The JavaBeans specification [242] is perhaps the most widely used object oriented approach for component systems. It specifies that each object (called a bean) must implement methods which allow it to be queried for a listing and description of its interfaces. Beans are also required to conform to specific naming conventions for methods. However, it is important to realize that component architectures are more general than object oriented systems, and can be realized in many ways, as we discuss below.

Work on CBD for telecommunications application architectures is fairly recent. Mennie and Pagurek describe a system of dynamically composeable components [243], building on the pipe and filter model of Zave and Jackson [244]. They use Jini as the primary tool to discover service elements, and then fetch an XML-based service definition template which describes the inputs and outputs of the system. They then make use of JavaBeans to connect and communicate between components. However, they do not detail the methods defined by the beans in their system.

Jung and Biersack [241] consider the problem of statically composable components using JavaBeans. However, they consider a very wide range of components, focusing on those that provide any type of communications system, not just those for telecommunications system.

Gbaguidi et al. [245, 88] have proposed a component architecture specifically aimed at hybrid services, which combine Internet and PSTN services. Their components are based on JavaBeans, and they achieve interaction with the PSTN by assuming an interface between their beans and existing SCP elements in the Intelligent Network. Despite their claims at a component model, they do not clearly specify the set of components in their architecture, why they are reusable, or what the interfaces are.

Mennie and Pagurek [243] made the important observation that their architecture was based on the DFC work of Zave and Jackson [244]. In the DFC architecture, features are provided to users by composing together reusable components called *boxes*. Calls enter and leave the system through *line interface boxes*, and within the system, are connected together through *feature boxes* that provide call filters. The interfaces between boxes use internal calls. The set of feature boxes brought into a particular call are called a *usage*. A typical usage scenario, reproduced from Figure 1 of Jackson and Zave [244], is shown in Figure 6.3.

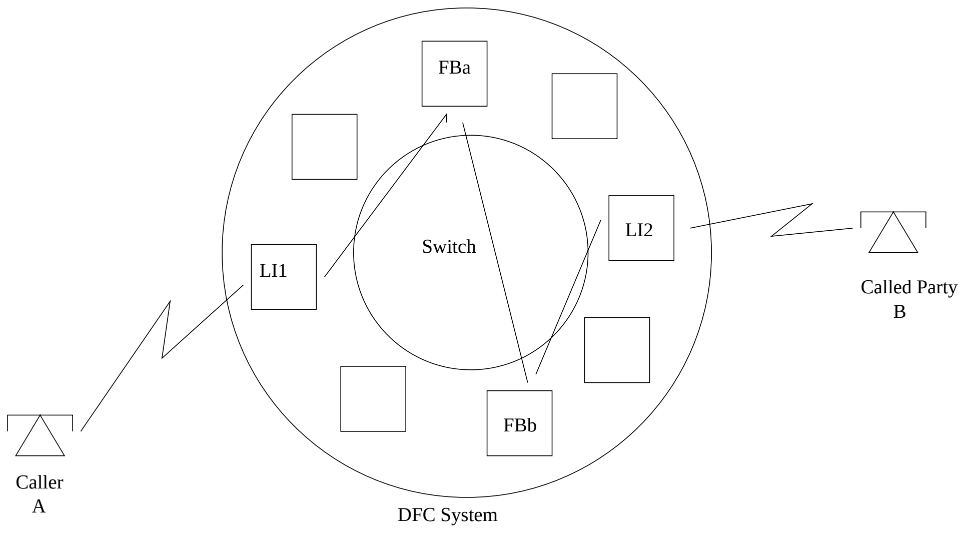


Figure 6.3: A usage in the DFC architecture

In this scenario, user A calls user B. The call enters the system through a line interface box, LIB1. It then is passed to an internal switch, also known as a *feature router*, which determines that the call is to connect to a particular feature. So, it routes the call to the first feature, feature box A, FBa. FBa may do some processing, and then, in this case, forwards the call. The feature router connects the call to a second feature, encapsulated in feature box B. This feature may do some processing, and then it forwards the call as well. Finally the call is connected to another interface box, and then eventually to the callee. Feature boxes may behave *transparently*, in which case they do nothing but forward the call onward to another element. The communications between boxes in DFC uses a call signaling protocol that includes messages such as setup, teardown, and quickbusy (which indicates that the call could not complete because resources were not available).

The DFC work is also effectively a component model for services, although it is not presented that way by the authors. The composition of feature boxes into a usage is effectively a tool for assembly of more complex features; however, the authors view it more as an artifact of feature interaction than as a specific paradigm for constructing services. Interestingly, the implementation of DFC, known as ECLIPSE [246], is not based on Javabeans, but rather on custom protocols between components.

The CBD architectures discussed here have similar drawbacks to the distributed object architectures in Section 6.3.2. Their usage of proprietary JavaBeans interfaces, or proprietary, single application protocols (in the case of DFC), make it difficult to support third party application component development. However, the component model provides a better paradigm for third party service providers. Since the components have well-defined and reusable functions, they can be used to support many different applications. This is beneficial to the provider of such component services, since it means increased usage, and therefore, increased revenue.

6.3.4 Mobile Agents

Work has taken place on the application of mobile agents to Internet telephony services. Mobile agents are software entities that move around the network in order to perform specific tasks. We consider two different types of mobile agent architectures, general purpose languages and domain specific languages.

6.3.4.1 General Purpose Languages

With general purpose languages, the code that is transported through the network is written in a Turing complete language, such as Java.

Pagurek, et al. [247] propose a scheme for constructing services in H.323 using mobile agents. Their work focuses primarily on how available services are discovered, subscribed to, customized and loaded into systems.

Gessler et al [248] promote a service architecture based on mobile Java agents. Complex services are constructed through the communications between three components, one stationary, and two mobile. The stationary component resides on a server in the provider network, and executes the core components of the service. One of the mobile components is used to define trigger conditions at the callee's site, and the other is used for service invocation. For example, the latter component might be a series of buttons on the GUI of the caller's tool. Clicking a button causes communications with the server in the provider domain to execute the service. Gessler does not provide much detail on the actual Corba interfaces they use.

Rizzetto et al. also define an architecture for downloading agents into H.323 gatekeeper platforms. These agents represent the participants involved in the call. The agents communicate and negotiate in order to deliver service [249].

Mobile agent technology has significant drawbacks for meeting our requirements. There remain significant interoperability, reliability, and security issues raised by transmitting code across the network. These issues imply that mobile agents cannot meet requirements to support third party application component providers.

6.3.4.2 Domain Specific Languages

The mobile agent problem becomes somewhat more tractable when, instead of transporting code for Turing complete languages, code for domain specific languages is transported. Domain specific languages can be designed with limited functionality, eliminating many of the security and interoperability issues. As an example, the Call Processing Language (CPL) [144, 250, 251] is a domain specific call control language, specified in XML, that users can upload into signaling servers in order to provide services [129].

Another example of a domain specific language applied to mobile agents is TOPS. Anerousis et al. have defined a complete architecture for Internet telephony, called Telephony over Packet Networks (TOPS) [252, 253]. Their architecture includes a signaling protocol, database components, media transport, conferencing servers, and QoS mechanisms. To make a call, the calling party queries the database server of the callee, and obtains a Call Handling Profile (CHP), which are a set of directives used by the caller to contact the callee. The CHP contains a list of prioritized addresses, representing the call appearances where the caller can be reached. Services, such as call forward no answer, are provided by including additional directives in the CHP.

Effectively, the CHP represents a form of mobile code that is downloaded into the client at call setup time. Since the "code" is not a real program, the mechanism does not suffer from the security and reliability problems of downloadable Java and related approaches.

The drawback of the domain specific language approaches are that they cannot support complex applications. Both TOPS and CPL are good for call routing, screening, and logging applications, but not much beyond that. None of the target applications outlined in Section 6.2 are enabled by either TOPS or CPL.

6.4 Application Component Architecture

Our solution is a new service architecture based on the CBD paradigm, which we call the *Application Component Architecture*, or ACA. They key ideas underlying ACA are:

Coarse granularity: We define our components as coarse grained elements. Much of the existing work has defined very fine-grained components. For example, Gbaguidi et al. [245, 88] define a component for a call forward feature, which is very fine-grained. Using coarser-grained components brings greater value to each particular component, and increases the likelihood that a third party provider would be willing to construct and offer that component as a stand-alone service.

Restricted problem space: Unlike Jung and Biersack [241], who attempted to describe components for arbitrary communications systems, we focus on components that can be used to construct a class of Internet telephony applications. This class includes those applications with conferencing, dialogs, web interfaces, presence, translation, speech-to-text, and text-to-speech. By focusing on a specific class of applications, we can do a good job in specifying well defined components, something we have found lacking in prior work.

Use industry standard protocols: The interfaces between our components are not based on Corba IDLs or Javabeans. Rather, they are industry standard protocols for call control, data transfer, and dialog scripting. By using industry standard protocols, which already provide inter-domain operation, scalability, performance, and security, we can realistically enable third party creation of application components and services.

Unify user-to-component and inter-component interfaces: The interface between users of a service and our platform are the same as the interfaces between components within our platform. This allows components themselves to be further decomposed into components. The result is what D'Souza [240] calls a "fractal" topology for components, where each component can be broken up into smaller and smaller components. This is a key concept in the Catalysis paradigm for component architectures [240], and it improves the ability to more rapidly construct complex applications.

The component assembly is the application: In our model, the set of components in the service, way the components are assembled - their ordering, their inputs and outputs - are what define the application, and are what differentiate applications from each other. Therefore, we define a specific component, the *controller*, whose sole job is to dynamically assemble components together to drive the application, as it executes. The controller, which is described in Section 6.4.7, assembles components by using third party call control.

Components can be run by third parties: The components in our model can not only be implemented by third parties, they are actually run and executed by third parties as component services. This requires that the system be capable of dynamically discovering and communicating with entities whose location is unknown. We enable this dynamic composition of components by using industry standard call control protocols, SIP in particular, as a key piece of the interface between components.

The ACA architecture is shown in Figure 6.4. The system has users, which represent people or machines that wish to invoke some service. The users are connected to an IP network, either directly (using a IP phone), or through some form of gateway from the PSTN. Users make and receive calls using SIP. Sometimes, users have web browsers, which they can use for control of applications. In that case, the users are capable of initiating HTTP requests. Users are also capable of sending and receiving media traffic (audio and video) over RTP.

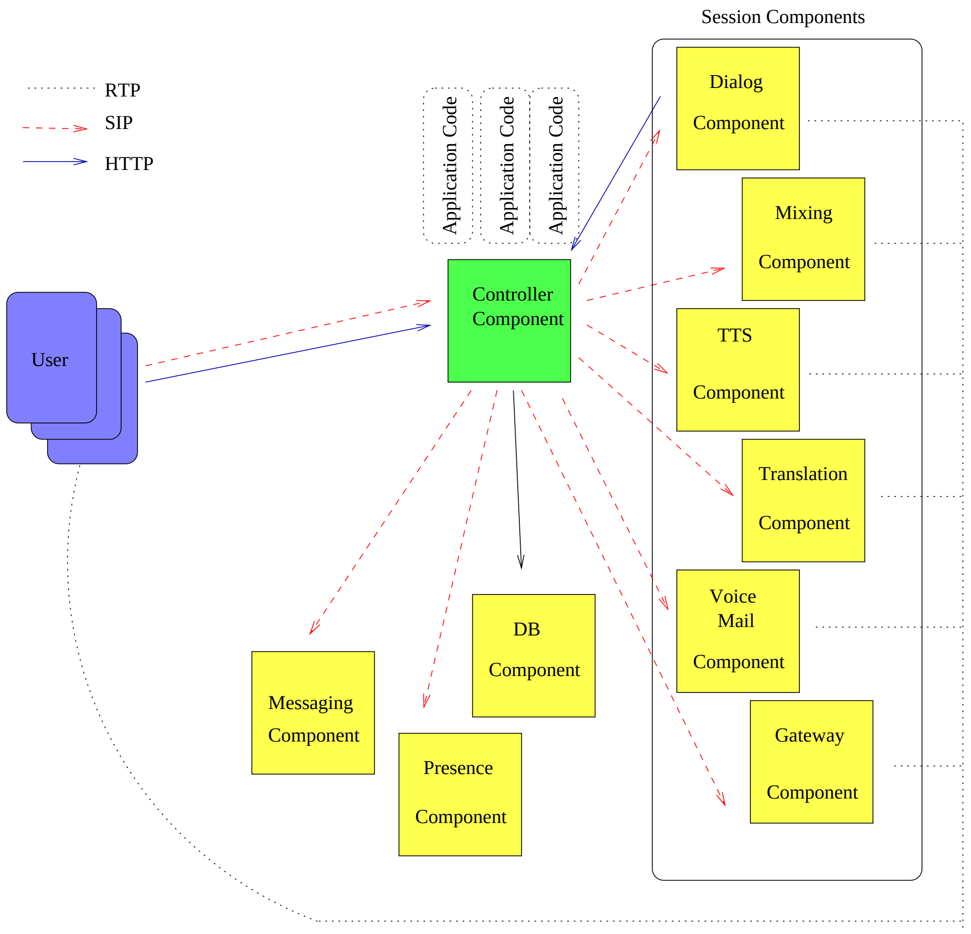


Figure 6.4: Application component architecture

We recognized that a particular theme was common in our target applications. There were pieces of the applications where a user needed to connect their media stream to some device, which would perform a media related function, and then, at some point later, the user disconnects from the device. For example, in the pre-paid calling card application, when the user runs out of money, they are played a prompt and asked to enter in their credit card number. Once thats done, the credit card is verified and the call is reconnected. This piece of the application requires the user to connect their media stream to a *dialog component*, which interacts with the user over the media stream, and collects the credit card. When the data is collected, they are disconnected from the dialog component, and their media stream is reconnected to the callee. This dialog component arose in other applications, such as auto-conference, as well. As a result, we define a general set of components that we refer to as *session components*. These are components that originate and terminate media streams. There is a distinct process of connecting to, and disconnecting from, these components. They generally provide a well-defined and reusable media processing function.

We recognize that there were many other types of session components that can be involved in an application. We can classify session components into three general categories, which are format translation, storage and playback, and semantic translation. A *mixing component* provides media mixing services, bridging together the media streams from some associated group of users. It is an example of a component that provides format translation. A *translation component* provides language translation services. It takes the media it receives, recognizes the speech, translates it, synthesizes text, and generates speech back out. It is an example of a semantic translation component. A *voice mail component* receives calls, and allows the caller to leave a message or hear messages left for them. It is an example of a storage and playback component. A text-to-speech, or *TTS component*, receives text as a continuous media stream, and converts it to speech, reflecting it back. It is an example of a semantic translation component.

All of the session components use SIP and RTP as their interfaces. Generally, the session components are just User Agent Servers (UAS), as they do not initiate calls. Interaction with the component begins when the component itself is “called”, by sending it a properly formatted SIP INVITE request. By properly formatted, we mean that the session component is addressed in a particular fashion. In SIP, users and devices are addressed using the Request-URI, which defines the recipient of the request. In typical SIP usage, this is an email-like identifier for a person. However, in this case, the Request-URI identifies the component, which is a piece of software, and defines the exact service that is to be provided by the component. As a simple example, a SIP INVITE sent to `sip:dialog23@dialogservers.com` might indicate that a specific named dialog, dialog 23 (which perhaps prompts the user for a PIN number) is to be executed. Once the INVITE is accepted by the session component, a media session is established between it and the client which has called it. The client sends media to the component, and it is processed according to the functional specifications of the component. Media is returned to the caller. The content of that media stream also depends on the functionality provided by the component.

Session components send RTP between themselves and users. Session components can also send RTP between each other. An example would be a conference that has its contents recorded. In this case, a dialog server would be one the entities receiving the mixed media stream from the conference server.

We also define several non-session components. These include a presence component, instant messaging component, and a database component. These are discussed in Section 6.4.6.

The central element of the architecture is the *controller*. It is the first point of contact for initiating an application. Applications are triggered by receiving a call at the controller, or through a click on a web page, delivering an HTTP request to the controller. The controller coordinates the involvement of several other components in the completion of the application, assembling them together to form a complete application. It assembles them by accessing them directly, or by connecting media streams from another device to them. The controller is the one and only point of processing of user input to the system. It can receive user input through SIP signaling (when the user hangs up a call, for example), direct HTTP requests (when the user submits a form), or indirect HTTP requests (when the dialog server interprets speech or DTMF in the voice stream, and informs the controller by sending it an HTTP request). Effectively, stimulus signaling is delivered to the controller through HTTP, and functional signaling through SIP. The controller never receives or sends media. It is purely in the signaling and control plane.

Most importantly, the controller is the one and only place in the architecture where application specific code resides. All of the other components are general purpose, and need not run specialized code in order to provide applications. This eliminates any need for mobile code, and the security problems it causes. It also simplifies management and feature interaction, since the code for an application resides in one place, even if the components are distributed around the network.

In the subsections below, we detail the interfaces and operations of these components. Specifically, any component architecture is obliged to define how its components are discovered, what the inputs and outputs are, and what the nature of the service provided is [243].

As a matter of terminology, we refer to the term *client* as the entity that requests service from a component. Normally, the client is the controller. The client is not the end user in most cases.

6.4.1 Dialog Component

The primary function of a dialog component (which we also refer to as a dialog server) is to use voice prompts and menus to interact with a user (which is not necessarily the client which has invoked the services of the component), for the purposes of conveying and collecting information through voice. Information is collected from the user by means of either speech recognition, or DTMF input (e.g., “press ’1’ for sales”), and information is presented to the user in the form of voice, generated either through stored voice files, or through text-to-speech. Dialog servers can also record speech that they receive. There is an underlying state machine that governs the operation of the dialog server. The voice prompts played by the dialog server will depend on what data has been collected so far from that user. We refer to the *dialog* as the temporally ordered set of speech generated by the dialog server, and speech sent to the dialog server by the user.

As an example, consider a dialog between a user and a dialog server, for the purposes of asking the user whether they would like to join a conference call. In the notation below, DS refers to speech generated by the dialog server, and U refers to speech generated by the user.

```
DS: Welcome to the conference service.
You have been invited by Joe to
join this conference. Say yes, or press 2,
to join. Say no, or press 1, to decline.
Say help, or press
pound, for the operator.
U: what
DS: I’m sorry, that is not a valid command.
Say yes, or press 2, to
join. Say no, or press 1, to decline.
Say help, or press pound, for the
operator.
U: yes
```

```
DS: Thank you. Joining the conference now.
```

Generally speaking, the purpose of a dialog server is to collect some form of information from a user, and then present that information, in computer understandable format, to an interested entity (which may be the client itself). In the dialog above, the information collected is effectively a boolean variable, with a value of 1 if the user joined, and 0 if they asked for the operator. Generalizing this, the information collected by the dialog server is a sequence of variable values, where the set of variables represents the data to be obtained through the dialog.

In the IN architecture, the Intelligent Peripheral plays a similar role to the dialog server.

From a software component perspective, the dialog server can be represented abstractly through the following Java interface:

```
public interface DialogServer {
 String [] executeDialog(DialogDefinition dialogToExecute,
                         MediaConnection connectionToClient);
}
```

A `MediaConnection` describes the IP addresses, port numbers, and codecs used for media flow in both directions. It is passed to the component containing the IP address and port information for one end of the stream. The component fills in its own IP address and port for receiving media.

It is important to note that the dialog to execute is an input to the dialog server. As an alternative, the interface to the dialog server could consist of the set of media primitives to execute, such as play a tone, play an announcement, and collect a digit. In this case, the dialog is not passed to the server. Rather, it is interpreted by the client, and the client instructs the dialog server to execute fine-grained, specific media processing functions. By passing the dialog definition as input to the component, rather than defining an interface function for every function specified in the dialog definition, we cleanly separate the user interface (the dialog definition), from the flow of the application.

From the above interface definition, the problem is clear. How are the the two inputs to the dialog component (the dialog to execute, and the media connection) passed to the component, and how is the output of the dialog, the set of variable values, passed back to the client which has invoked the dialog?

Fortunately, standards-based protocols and technologies exist to address all three problems.

Passing the media connection to the server is simple. As we have mentioned above, the media connection is established by having the client call the dialog server by sending it a SIP INVITE. The normal SIP call establishment process will generate a media connection between the dialog server, and the entity (the user) whose IP address and port where provided in the SDP in the INVITE. This need not be the client which sends the INVITE to the dialog server (and won't be, in most cases.).

To define the dialog, we make use of VoiceXML [254]. VoiceXML is an XML-based domain specific language that specifies a dialog between a user and a dialog server. It supports speech recognition, DTMF detection, text-to-speech, recording of speech files, and playing of prompts from recorded speech files.

The VoiceXML document is passed to the dialog server by embedding an HTTP URL in the request-URI of the SIP INVITE. This HTTP URL is used by the dialog server to fetch the VoiceXML script. An example SIP INVITE request line might look like:

```
INVITE sip:http%3acontroller.com%2fvxml4@dialog.com SIP/2.0
```

We have URL-encoded the HTTP URL, and placed it into the user portion of the SIP URL.

A VoiceXML document actually contains a set of dialogs. Each dialog is either a *form*, or a *menu*. Forms are used to collect data from the user, and menus are used to provide choices. A form has a series of fields, which represent the data values to be collected through the dialog. Form dialogs also contain control elements that help control the gathering of the form's fields. VoiceXML has the notion of a grammar, which defines the set of things the user can say (or press, using DTMF) when filling out fields of the form. Forms also contain event handlers and *filled* items, which are commands that get executed when the form is filled in. The most common action is to generate an HTTP request (either GET or POST), with the values collected through the form. The use of HTTP is natural here, since the model is that of a user filling out a voice form, in much the same way they would fill out, and then submit (using HTTP GET or POST), the values of an HTML form. Just as a normal web server can return another form in response to a form POST or GET, the HTTP POST or GET generated by the VoiceXML script can return another VoiceXML script, defining further interactions that are to take place with the user.

The use of HTTP form POST or GET provides the final piece of the interface to the dialog server. It defines the way by which the output of the dialog server, the data that has been collected, is passed back to the client.

When the client has completed its interaction with the component, it terminates the connection with a SIP BYE.

As an example, consider the pre-paid calling card application. When the user makes the call, they are prompted with a greeting, asking them to enter in their PIN number. This PIN number is the desired output from the dialog server. The client can then verify the PIN, and if its correct, connect the call to the callee.

The VoiceXML script for this dialog is:

```
<vxml version="1.0">
<form id="get_pin">
 <block> Welcome to prepaid calling.</block>
 <!-- The grammar for type="digits" is built in. -->
  <field name="pin" type="digits">
   <prompt>What is your pin?</prompt>
   <filled>
     <submit next="http://controller.com/p.asp"
             namelist="pin"/>
   </filled>
 </block>
</form>
</vxml>
```

The message flow for interacting with the dialog component is shown in Figure 6.5.

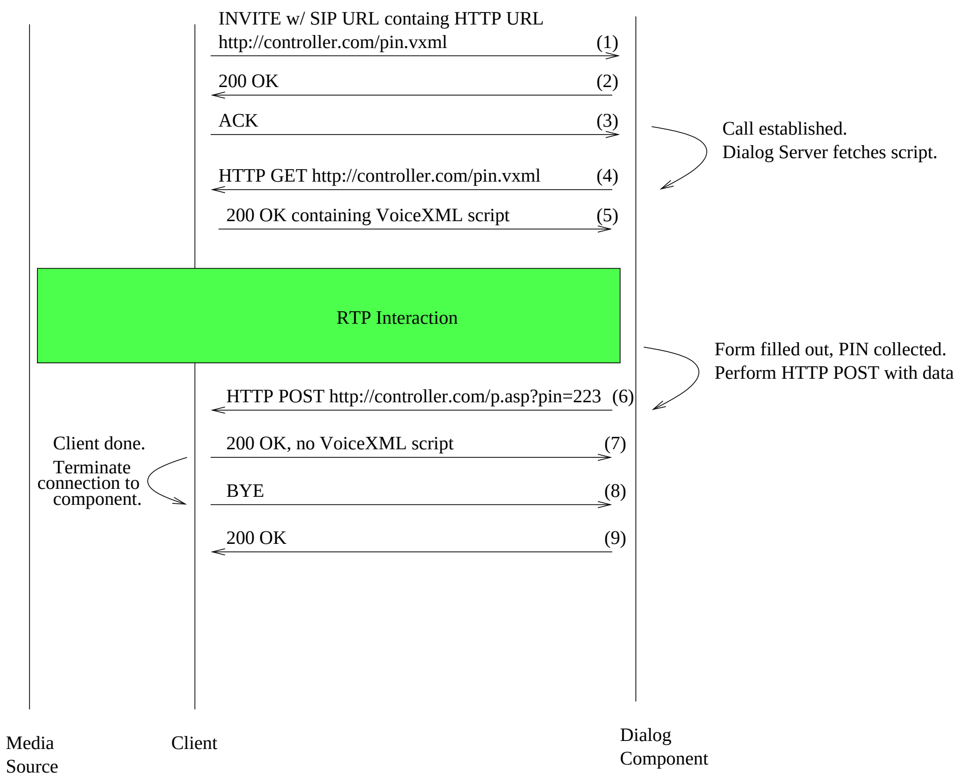

INVITE w/ SIP URL containg HTTP URL
http://controller.com/pin.vxml
(1)
200 OK
(2)
ACK
(3)
Call established.
Dialog Server fetches script.
HTTP GET http://controller.com/pin.vxml
(4)
200 OK containing VoiceXML script
(5)
RTP Interaction
Form filled out, PIN collected.
Perform HTTP POST with data
HTTP POST http://controller.com/p.asp?pin=223
(6)
Client done.
Terminate connection to component.
200 OK, no VoiceXML script
(7)
BYE
(8)
200 OK
(9)
Media Source
Client
Dialog Component

Figure 6.5: Client interaction with dialog server for PIN collection

In message (1), the controller (which is a process running at `controller.com`), acting as a client of the dialog server, sends a SIP INVITE request to the dialog component. The user portion of the request URI contains the http URL of a dialog to execute. In this case, that HTTP URL references back to the client. The call is accepted (2,3). The dialog server then takes the HTTP URL from the INVITE, and fetches it (4). The client receives the HTTP request, and returns the VoiceXML script to execute (5). The dialog server then interacts with the media source, which is the entity whose IP address was listed in the SDP in the INVITE. The PIN is collected using the dialog described above. The dialog server then uses an HTTP GET, using the URL provided in the `submit` tag in the VoiceXML script (6). The HTTP request contains the field value for the variable "pin" as a URL parameter (this is the standard convention for web forms). This request also goes to the client, at `controller.com`. The client now determines that it has completed its interaction with the dialog component, and it hangs up, using a SIP BYE request (8).

In summary, we define a dialog server as a component of our system. The input to this component is the media session used for the dialog, which is established with a SIP INVITE. Another input to the component is the dialog itself, represented with a VoiceXML script. The output of the component is the form data collected through the dialog. It is passed using an HTTP request from the dialog server back to the client. The dialog component is highly reusable, since it can work for any dialog that can be expressed in VoiceXML. This includes one-way dialogs from the dialog server to the user (where the server simply plays some announcement without collecting data) and one-way dialogs from the user to the dialog server (where the dialog server waits for the pound key to be pressed). Discovery of the dialog server component is simple, and is based on the call routing and user discovery capabilities provided by SIP, and described in Chapter 4.

6.4.2 Mixing Component

The primary function of the mixing component is to take a group of incoming media streams (usually voice, but the concepts can be applied to video), and perform a "mix-minus" bridging operation on them, generating a mixed media stream for each incoming media stream. The mixed stream for input stream i is the sum of all the other streams except i (this is the definition of mix-minus). We refer to this group of media streams as a *mixing context*. Traditionally, this group is known as a conference, but we avoid this term due to its overloaded usage in the literature.

The mixing component allows new streams to be added to the mixing context at any time. Similarly, a stream can be removed from the mixing context at any time.

These mixing capabilities are standard features on traditional circuit switched and packet switched conferencing systems. However, we add an additional capability. The mixing context does not need to be established ahead of time. Mixing contexts are given unique identifiers. If the client of the component requests that a stream be added to a specific mixing context with a given identifier, and that identifier does not correspond to an existing mixing context, a new context is created, and the stream is added to the context. The context remains in existence within the mixing component so long as there is at least one media stream within the context. When the last media stream is removed from the context, the context can be destroyed. We refer to this capability as *spontaneous mixing contexts*.

Traditional conferencing systems do not support spontaneous mixing contexts. Usually, conferences need to be established ahead of time, and their associated mixing contexts have well defined start and stop times.

The motivation for adding this capability is to promote component reuse. Policies regarding how long a conference call should last, and how many participants can be present, differ from application to application. By extracting this from the component, leaving just a pure mixing function, we are left with a component which can work in any conferencing system with any defined policies. Furthermore, the component can be used for non-conferencing applications. As an example, the component can be used to support a privacy service. When a user makes a call, their signaling is routed to a controller providing an anonymizing service. The controller modifies the addresses in the message to hide the identity of the caller, and forwards the call request to the callee. It also modifies the SDP in the INVITE and the 200 OK response, so that media from both caller and callee are sent to the mixing component. Using the third party call control techniques we describe below, it establishes a spontaneous mixing context containing the two streams. Effectively, the mixing component plays the role of an RTP translator, by using a mixing context with two streams. These mixing contexts are not set up ahead of time; they need to be instantly created when a call is made by a user of the service.

From a software engineering perspective, we can define the interface to this component very simply:

```
public Interface MixingComponent {
  public void addMediaConnection(MediaConnection aConnection,
                                 MixingContextID id);
  public void removeMediaConnection(MediaConnection aConnection,
                                    MixingContextID id);
}
```

SIP can readily be used to provide this interface. To add a media connection to a mixing context, the client of the component sends an INVITE to the mixing component. The request

URI contains the identifier for the mixing context, encoded in the user portion of the URI. The mixing component accepts the call, establishing a media stream. As discussed above, the mixing component creates a mixing context if none exists. The media stream established by the INVITE is then added to the mixing context. A mix-minus operation is provided across all the media streams in the mixing context.

Removing a media connection is easily provided through a SIP BYE. The client sends a BYE for the call leg established with the INVITE. This removes that media stream from the mixing context.

6.4.3 Text-To-Speech Component

Another example of a session level component is a continuous text-to-speech (TTS) converter. This kind of service allows a real time text stream ,encapsulated in RTP using the RTP payload format for text [255]to be received, which is then converted to speech and returned as an audio stream encoded using a traditional speech codec, such as G.723.1 or G.711.

Text-to-speech functionality can also be providing using a dialog server. Since VoiceXML provides text-to-speech capability, the client can generate a VoiceXML script with the text in it. The dialog server will speak the text over the media connection. When a dialog server is used, however, the text stream to be converted is provided by the client of the component. With a continuous text-to-speech component, the text stream to be converted is provided in a media connection, from the user.

It is likely that the text-to-speech conversion process differs significantly depending on the language. As such, the language needs to be provided as input to the component.

As with the other session level components, we need to be able to initiate the conversion process by establishing a media connection, and we need to be able to terminate it.

Once more, using Java notation, we can describe this interface as:

```
public Interface TextToSpeechComponent {
  public void beginTranslation(MediaConnection streamToTranslate,
                               Language lang);
  public void stopTranslation(MediaConnection streamToTranslate);
}
```

As with the other session level components, the mapping to SIP is simple. The translation process is initiated with a SIP INVITE, which provides the media connection. The request URI identifies the text-to-speech component, and also conveys the language. We adopt the convention `sip:server_specific_name-language_tag@domain`, where the language tags are selected from the set defined in RFC 1766 [256]. For example, the URL for contacting a component that performs text-to-speech in English might be `sip:ttscomponent-en@foo.com`.

One of the unfortunate limitations of SDP is that it is not currently possible for a single media stream to be one media type (such as audio, video, or text) in one direction, and a different media type in the other direction. Text over RTP is considered a text media type. As a result, two media streams are needed for this service. One is a unidirectional audio stream from the user to the component, and the other is a unidirectional text stream from the component back to the user.

First, the client INVITEs the server. The SDP indicates two media streams. One stream is of type audio. It contains the set of audio codecs acceptable to the client. The stream is marked as receive-only. The other stream is of type text. It contains a single codec, which is a dynamic payload number bound to text/t140. The stream is marked as send-only. The SIP 200 OK response from the TTS server that accepts the call has SDP with a two media lines, one of type audio, and one of type text, in the same order the streams appeared in the INVITE, as mandated by RFC2543. The audio stream contains a subset of the codecs listed in the audio stream in the INVITE. The audio stream is marked as send-only. The text stream contains a single codec, which is a dynamic payload type number bound to text. The stream is marked as receive-only.

The client then ACKs the request. The TTS server converts all text received on the incoming text stream to speech, and return the resulting speech on the outgoing audio stream.

6.4.4 Additional Session Components

A variety of other session components can be defined. One possible component is a translation component. It translates speech from one language to another language. It works almost identically to the text-to-speech component. An INVITE is used to establish a media stream with the server. The request URI indicates that a translation service is desired, and it carries the language tags of the input and output languages. Like all of the other session components, the translation component is terminated by a BYE.

Another session component is a voice mail service. Voicemail service can be broken down into a number of sub-components, each of which represents a particular type of interaction with the voicemail service. The interactions can be broken into two separate categories - message drop, where the caller leaves a message, and message retrieval, where the user retrieves their messages. Within each category, there are several different interactions that might take place with a caller. For example, a message might be left because the caller attempt to contact a user, and the user was busy, resulting in a connection to the voicemail system. The prompts played by the system for a call-forward-busy drop might be different than for other message drop cases.

Campbell and Sparks [257] detail the SIP interfaces to a voicemail system component. They fit within the general framework of using the Request-URI as a service identifier, which we have defined here.

It is useful to note that the voicemail component can be implemented using a series of dialog servers assembled together in a particular fashion. In this case, the voicemail interface is a facade pattern, hiding a more complex conglomeration of components. This facilitates reusability, as we have discussed.

Another session level component is a PSTN gateway component. The service provided by the gateway is to take calls received by the client, and connect those calls to a number in the telephone network. The number to connect to is provided in the user portion of the request URI of the SIP INVITE. To terminate the call, either the client, or the gateway, sends a SIP BYE. This basic interface is exactly the one provided by existing Internet telephony gateways, as described both in the literature [258, 259, 105, 260, 261], and in current commercially available products. Viewing these devices as reusable components, in a CBD design paradigm, is helpful towards understanding how they fit into our overall architecture.

6.4.5 Presence Component

We defined the presence service in Section 6.2 as a system that allowed entities to subscribe to, and be notified of, changes in the status of other users. Presence systems have been described extensively in the literature. Examples include the Montage system [262] and the MIT Zephyr system [92].

Presence is often associated with *instant messaging*, which allows users to send electronic notes to each other. Unlike email, where messages are stored on a server, and then retrieved by the client at a later time, instant messages are displayed immediately on the screen of the recipient without being stored in a server. Some systems do store instant messages when the recipient is not actually online, and others do not. Because of its focus on immediate delivery, as opposed to storage and message management, instant messaging is only useful when the recipient is logged in. This is what instant messaging is often provided in conjunction with presence (Zephyr provides both, for example).

However, none of the systems documented in the literature operate well in a wide area network. They lacked adequate addressing, routing, and security mechanisms.

In 1998, the IETF began consideration on standardizing protocols for presence services across wide area networks. The generated a requirements document, RFC 2779 [263], and a framework and model document [91]. We proposed the usage of SIP to support presence [264] and instant messaging [265].

In our proposal, a *presence server* is a network entity that receives subscription requests (in the form of a SIP SUBSCRIBE message) for a particular *presentity* served by that server. A presentity, defined in RFC 2778 [91] is a presence entity, which is the entity that publishes the presence information (available, not available, or any other status) to the system on behalf of a user. When the status of the presentity changes, the presence server generates notifications (in the form of SIP NOTIFY requests), conveying the new status of the presentity.

A basic presence exchange using SIP is depicted in Figure 6.6. The client, which can be any entity interested in receiving presence information, constructs a SIP SUBSCRIBE request. The Request-URI of this request identifies the presentity. In Figure 6.6, the presentity is identified as `sip:presentity@server.com`. This request is routed through a SIP network of proxy servers (not shown), arriving at the presence server (1). If the subscription is allowed, and is accepted by the server, a 200 OK response is sent (2). At some point later, the status of the presentity changes. This change can be known to the presence server in many ways. One mech-

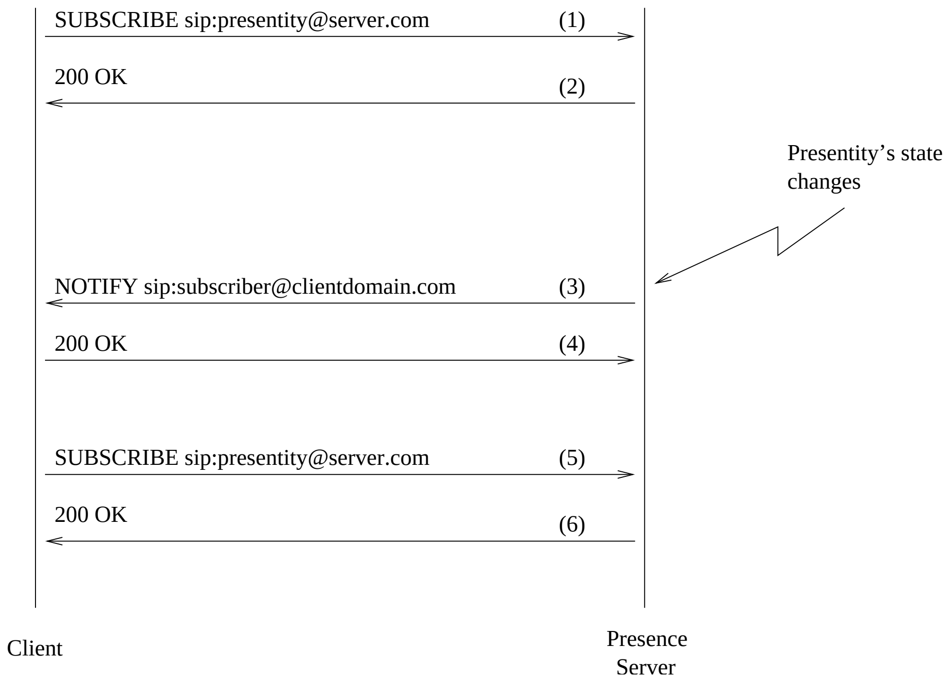

SUBSCRIBE sip:presentity@server.com
(1)
200 OK
(2)
Presentity's state changes
NOTIFY sip:subscriber@clientdomain.com
(3)
200 OK
(4)
SUBSCRIBE sip:presentity@server.com
(5)
200 OK
(6)
Client
Presence Server

Figure 6.6: Message exchange for basic presence service

anism described in our proposal [264] is through SIP registrations, which effectively convey the status of the user. As a result, presence servers are ideally co-located with registrars.

As a result of this change in status, the presence server generates a SIP NOTIFY request (3). This request is addressed to the subscriber. The body of the request contains a presence document, describing the current status of the presentity. When the NOTIFY is received, the subscriber sends a 200 OK response (4), confirming its arrival. In our proposal, subscriptions are soft state, and must be refreshed. This refresh, which works identically to the original subscription exchange, is performed with messages (5) and (6).

In traditional applications of presence, such as Montage, it is the end user software that generates the subscriptions which access presence services, in order to learn the status of another user for rendering on a graphical interface. In our service architecture, we model the presence server as a component, which provides a generic presence service that can be accessed by other components, such as the controller, to facilitate the delivery of complex applications. The interface to the component is through the SIP SUBSCRIBE and NOTIFY requests, and their responses, as specified by Rosenberg [264]. The service provided by the component is also well defined, and specified by Rosenberg [264]. Most importantly, the presence component is highly reusable. It lacks any application specific interfaces or functional semantics, and can therefore be used as a key component of any application that requires presence information.

6.4.6 Additional Components

The ACA architecture, as shown in Figure 6.4, also shows a messaging component and a database (DB) component.

The messaging component supports instant messaging, as defined in RFC 2778 [91] and discussed in Section 6.4.5. The component provides a well-defined, and simple service. It takes instant messages it receives, and delivers them to the desired recipient. The interface to this component is defined in our instant messaging specification [265]. The client formulates a SIP MESSAGE request, and sends it to the messaging server. The server delivers the message to the recipient identified by the Request-URI in the SIP request. If the message is successfully delivered, a 200 OK response is provided. The call flow for this interface is shown in Figure 6.7.

In SIP terms, the messaging server is nothing more than a SIP proxy server, which routes the message to the recipient identified in the request URI.

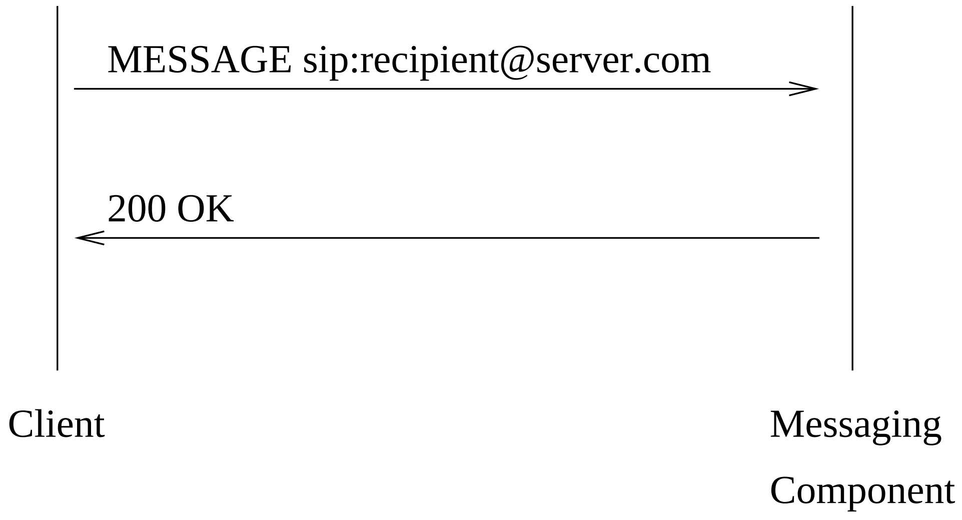


Figure 6.7: Call flow for interface to message component

Another component is a database. To be a reusable component, the database must be application independent. This means that the schema for the data storage must be general purpose and usable for any application. The only realistic schema that can fit such a need is a simple name-value mapping service. Clients can store any object, specifying a name when the object is placed into the database. It can then be retrieved later on by fetching it with the same name. Many existing database protocols can be used for such an interface, including the Lightweight Directory Access Protocol (LDAP) [191].

6.4.7 Controller

In our architecture, the controller is responsible for "assembling" the various other components together, as needed, to execute the application. This means that it is responsible for determining the ordering of components to invoke based on the requirements of the application. As an example, the pre-paid calling card application begins with a call that arrives at the controller. The controller then invokes the dialog component, and connects the user to it in order to obtain the PIN number. Once done, the controller invokes a database component, to validate the PIN number. Assuming the PIN is validated, the controller next invokes the gateway component, connecting the user to the PSTN.

The controller is analagous to the feature router in DFC [244], and to the GUI tools used to assemble JavaBeans. In our architecture, the controller is the one and only place where application-specific code resides.

Since the controller is a component itself, it has a well defined input and definition of the service it provides. The controller is contacted by the end user who wishes to use the application. This contact can occur in one of two ways. In the first approach, the end user initiates a SIP call, which is routed to the controller. The reception of a call is what triggers the application. As with other components we have defined here, the user portion of the Request-URI of the INVITE identifies the application.

In SIP terms, the controller can be a proxy or User Agent Server (UAS) when receiving a call to initiate an application. In the processing of that call, it can additionally act as a UAC (making it a B2BUA).

Another way to initiate a service is using HTTP. A user can click on a web page, initiating some kind of application. In this case, the HTTP request is routed to the controller, and the request URI of the HTTP request defines the application to execute. Effectively, the controller is an HTTP server.

The application is constructed by invoking the services of the other components. These components are invoked in some sequence, which is dependent on the application to be delivered, and the outputs of the components invoked in the processing of the application.

Session components are invoked by sending them a SIP INVITE to establish a session with the component. The media stream that connects to the session component is not generated by the controller. Rather, the controller uses third party call control (discussed below), to connect media streams from users or other components to the session component being invoked. Once the controller is finished with the session component, it disconnects with a BYE request. In the case of a dialog component, there may be an HTTP request passed from the component to the controller, informing it of data collected from the dialog. The interaction between a controller and a session component is shown in Figure 6.8.

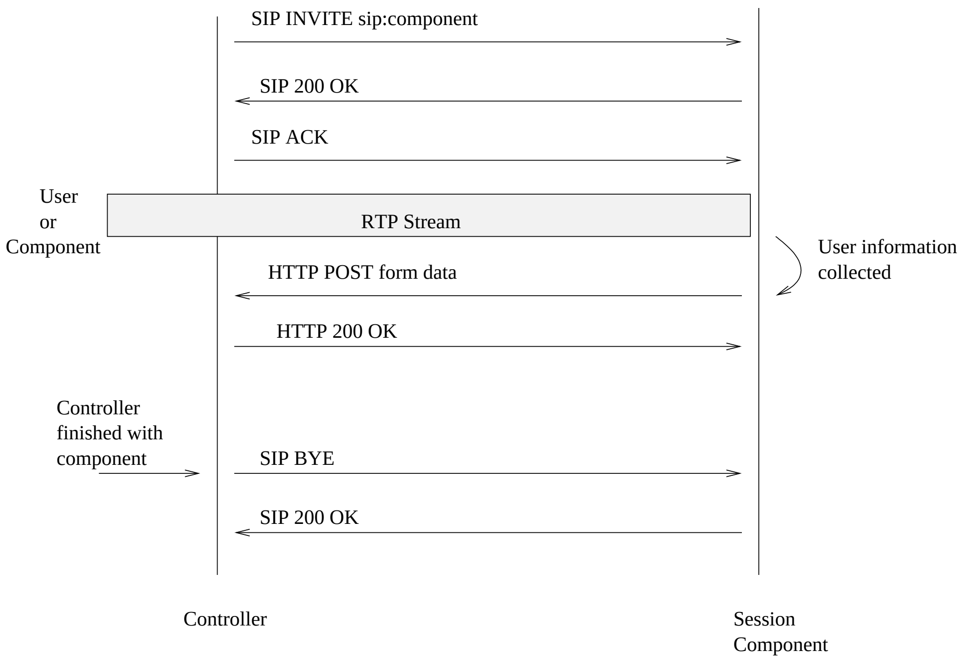


Figure 6.8: Interface between the controller and session components

6.4.8 Third Party Call Control

The means for invoking the services provided by the components in our architecture are detailed above. However, one piece is missing. Although the controller will be the client of most of those services, it is not the one generating the media stream that interfaces to the session level resources. The media streams used as input to the session components will ususally come from the end users in the application, or from other session level components. Therefore, another required capability is for the controller to connect the media streams from end users and session components, to other end users and session components. This capability is known as third party call control. Third party call control allows for signaling relationships to exist between the user and controller, and the controller and component, but for the media streams to flow directly between the user and component. This is shown in Figure 6.9. We have developed usage scenarios for SIP that allow the controller to invoke third party call control services (3pcc) [121].

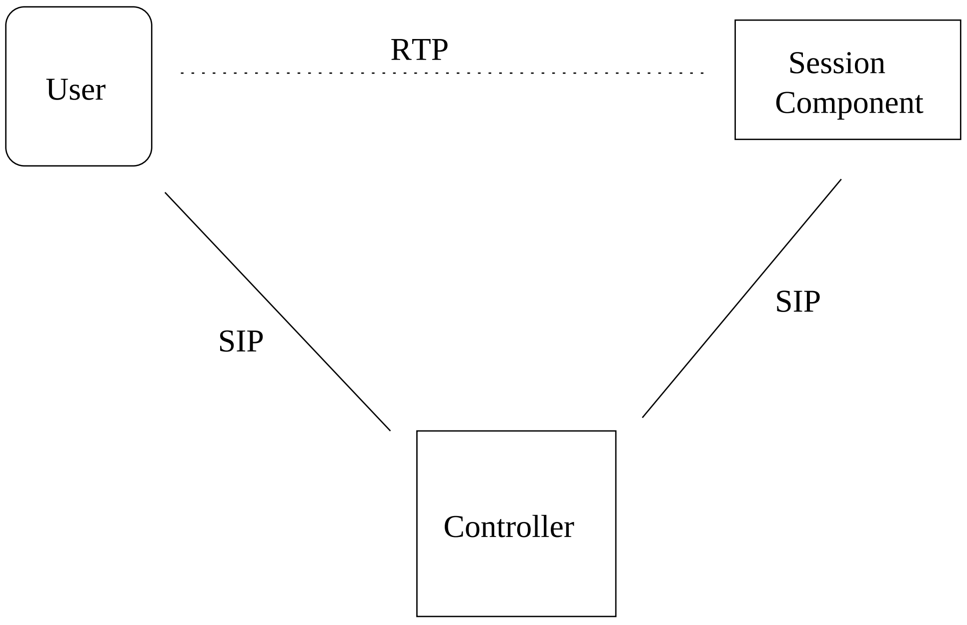


Figure 6.9: Signaling and media relationships in third party call control

There are two usage cases for SIP third party call control. Both have advantages and disadvantages, as we discuss below.

6.4.8.1 Basic Flow

The controller first sends an INVITE to the first entity, A, which is to represent one side of the media connection. The entity may be an end user, but it could also be any of the session components described above. This is a standard INVITE, but it contains no SDP. As we discussed in Chapter 4, this causes A to make the initial offer of SDP for the call in its 200 OK. When this 200 OK arrives, the controller does not yet send an ACK. It generates a second INVITE. This INVITE is addressed to the second entity, B, to be connected in the call. This INVITE contains the SDP as received from the 200 OK from A. When the 200 OK to this second INVITE arrives, the controller ACKs it, takes the SDP, and includes that in the ACK for the first call. A flow diagram for this mechanism is given in Figure 6.10.

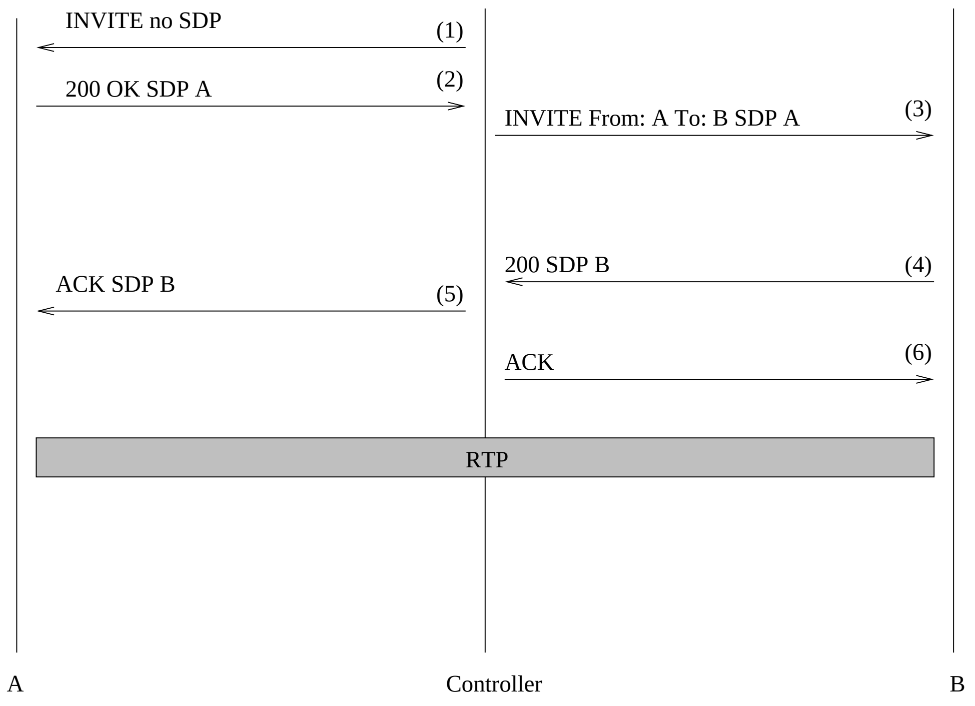


Figure 6.10: 3pcc basic flow

This flow is simple, requires no manipulation of the SDP by the controller, and works for any media types supported by both endpoints. However, it has a serious timeout problem. Entity B may not answer the call immediately. The result is that the controller cannot send the ACK to A. This causes A to retransmit the 200 OK response periodically. In fact, if B does not answer within 32 seconds, the call with A times out [89].

6.4.8.2 Advanced Flow

A more complex flow, which does not suffer the timeout problem described above, is shown in Figure 6.11.

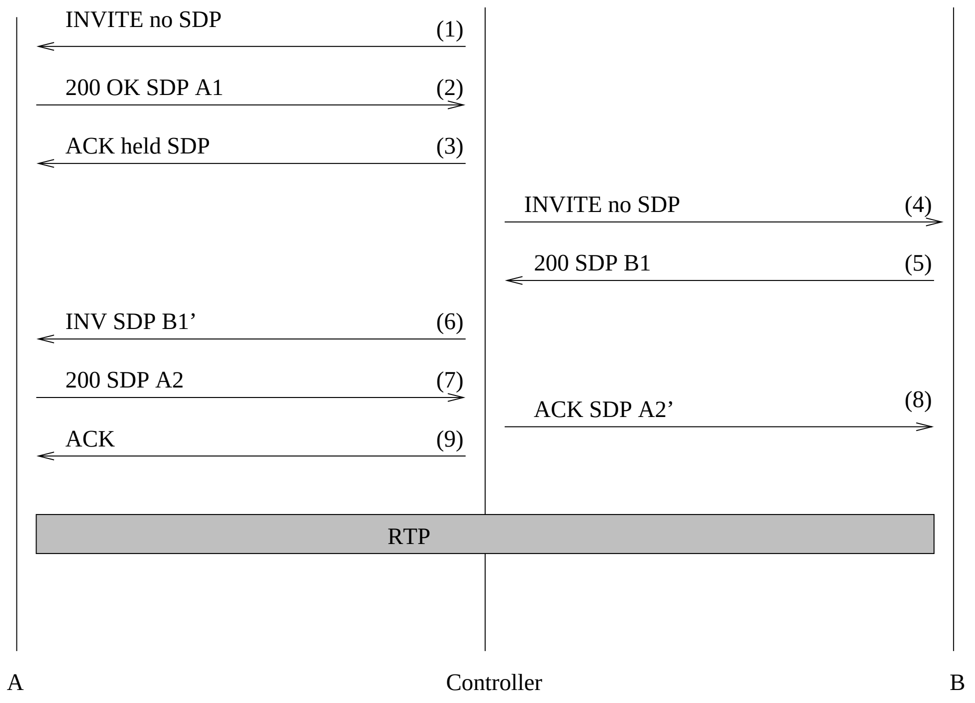


Figure 6.11: 3pcc advanced flow

First, the controller sends an INVITE to the first entity, A, without any SDP. Entity A responds with its SDP, A1, in a 200 OK, which is immediately ACKed with an on-hold SDP generated by the controller (an on-hold SDP has the IP address in the SDP set to 0.0.0.0, which is defined by SIP as media-on-hold).

Next, the controller sends an INVITE to the second entity, B, also without SDP. The SDP in the 200 OK, SDP B1, is used to create a re-INVITE to entity A. That re-INVITE is based on SDP B1, but may need to be reorganized to match up media lines (recall that the media lines in the answer to an offered SDP must align) . We therefore call that SDP B1'. Since this is a re-INVITE, it should complete quickly in the general case. That is important, since entity B is retransmitting their 200 OK, waiting for an ACK. The SDP in the 200 OK from A, SDP A2 (which may be different than A1), is then passed to entity B in the ACK. It may also need reorganization to match up media lines.

This flow has many benefits. First, it will usually operate without any spurious retransmissions or timeouts (although this may still happen if a re-INVITE is not answered quickly). Secondly, it does not require the controller to guess the media that will be used by the participants.

There are some drawbacks. The controller does need to perform SDP manipulations. Specifically, it must take some SDP, and generate another SDP which has the same media composition, but is on hold. Secondly, it may need to reorder an SDP message X, so that its media lines match up with those in some other SDP message, Y. Finally, the flow is far more complicated than the simple and elegant flow in Figure 6.10.

As a result of these drawbacks, it is our recommendation that the basic flow, shown in Figure 6.10 be used if, and only if, the controller knows that entity B is actually an automata that will answer the call immediately. This is the case for all of the session level components described above. Since we expect a great deal of third party call control to be between end users and session components, this simpler flow can be used frequently.

For calls to unknown entities, or to entities known to represent people, it is recommended that the flow in Figure 6.11 be used for third party call control.

6.4.8.3 Continued Processing of Third Party Calls

Once the calls are established, both entities are in a point-to-point SIP call with the controller. However, they are exchanging media directly with each other, rather than with the controller. The result is that the controller has set up a call between both entities.

Since the controller is still a central point for signaling, it now has complete control over the call. If it receives a BYE from one of the participants, it can create a new BYE and hang up with the other entity. This is shown in Figure 6.12.

As an alternative, when the controller receives a BYE from A, it can generate a new INVITE to a third entity, C, and connect B to that entity instead. A call flow for this is shown in

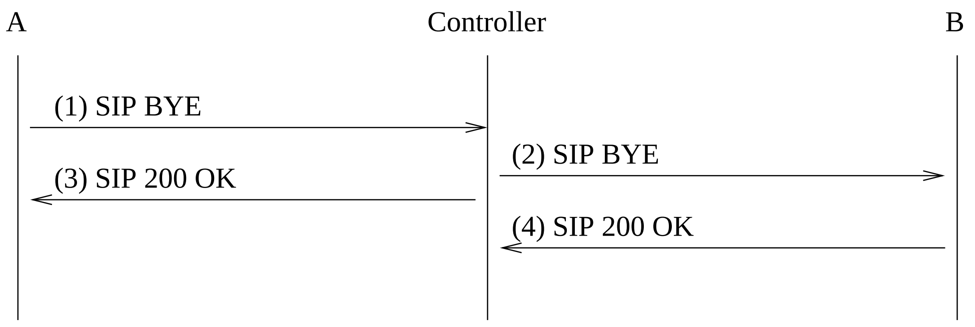


Figure 6.12: Hanging up with 3pcc

Figure 6.13, assuming the case where C represents an end user, not a session level component. Note that it is simply messages 4 through 9 of the basic call flow of Figure 6.11.

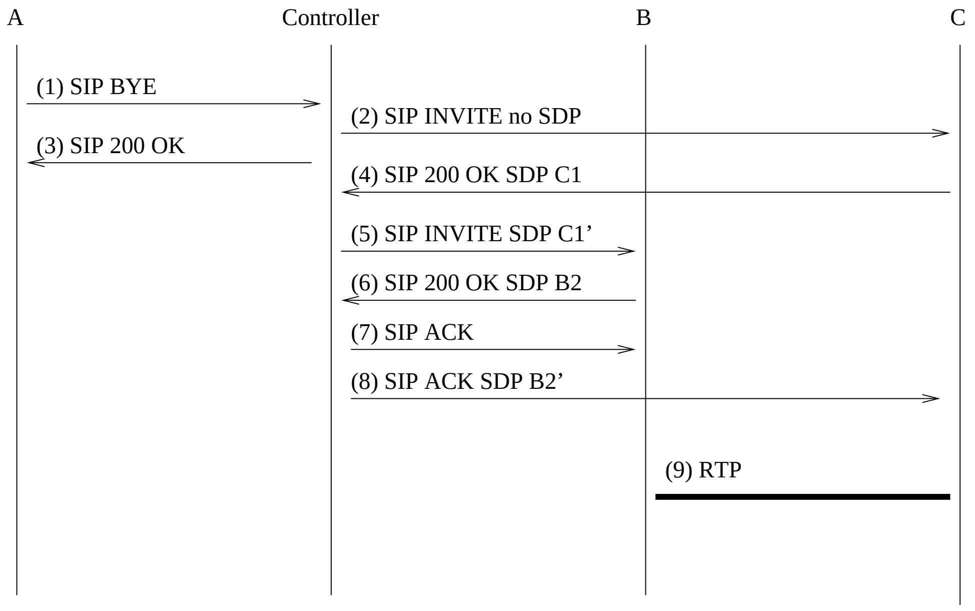


Figure 6.13: Alternative to hangup

From here, new entities can be added, removed, transferred, and so on, as the controller sees fit.

The general idea behind the mechanism is that there is a point-to-point SIP relationship between each entity and the controller. However, by passing the SDP it receives from one participant to another, it can causes users to actually communicate with each other rather than the controller.

6.4.8.4 End User Initiates Call

The call flow in Section 6.4.8.1 assumes that the controller is the entity that initiates calls to the entities that need to be connected. However, a more common case is that the end user initiates a call to the controller, and the controller then needs to connect this call to another entity, and have the two exchange media. We have also developed call flows for this usage scenario, shown in Figure 6.14.

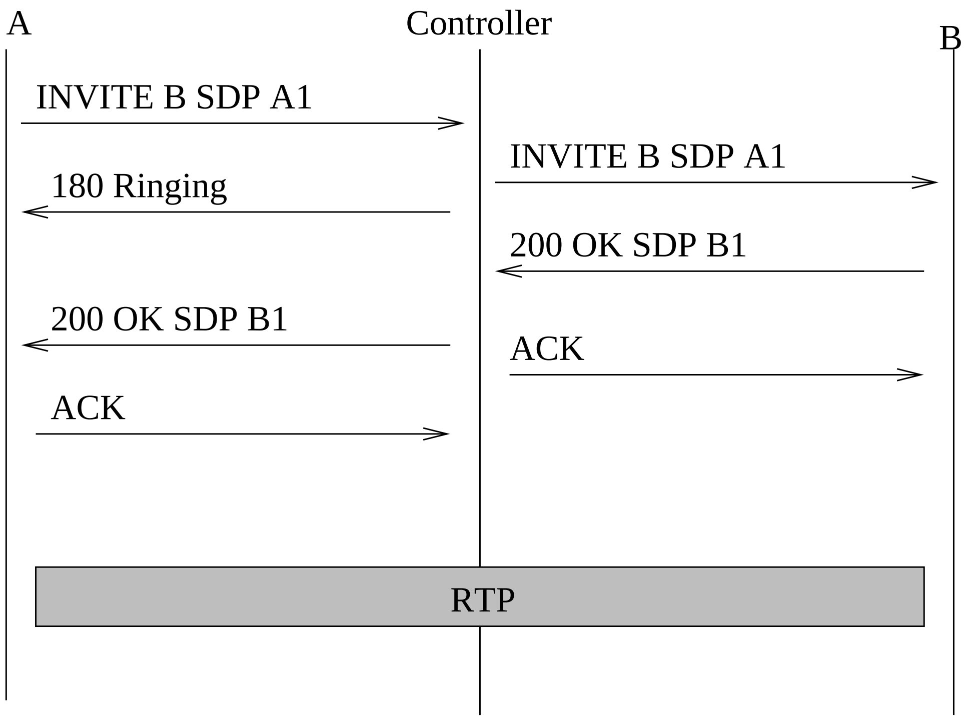


Figure 6.14: 3pcc where the end user initiates

In this call flow, the controller looks deceptively like a SIP proxy, but it is not. The controller acts as a UAS for the INVITE received by A, and then as a UAC when it initiates a call to B. It is this fact which allows the controller to generate its own ringing messages, or to generate an ACK for a 200 OK, both of which are done in this call flow.

Once set up, the controller is exactly in the same state as if it had initiated the call as described in Section 6.4.8.1. The controller can hang up to one side, hang up to both sides, reconnect the users to media servers, and so on.

6.4.9 Obtaining Data from End Users

An important component of any application is how information is collected from the users of the application during its execution. Since the controller is responsible for the overall execution of the application, it is the element responsible for processing user input.

As an example, a pre-paid calling card application requires the user to interact with the application. They need to provide the PIN number for the account in order to make the call. Of course, some applications do not require input to be provided during the execution of the application. An example is a call-forward on no-answer service. This service requires that the user provide input, which is the forwarding number, but it is provided ahead of time, as part of the configuration of the application itself.

Generally, the user can provide input to the system in two different approaches, which are *stimulus signaling* and *functional signaling*. The difference is whether the end user equipment (software or hardware) needs to understand the semantics of the input provided by the end user. In stimulus signaling, the client equipment has no semantic understanding of the input. In functional signaling, it does.

Consider the example of a transfer feature. In this feature, two end users, A and B, are talking. User A wishes to transfer B to some other user, C. One way to do this is for user A to depress the hookflash on their phone. This information is transmitted to the switch, which returns a dialtone to the phone. The user then dials the number to transfer the call to. This information is passed to the switch as well, just as dialed digits normally are. The switch then transfers the call. In this model, the client equipment, the phone, was not aware that the transfer feature was being invoked. The service can be added to existing systems by upgrading the switches, and then informing the user about how to use the service. The client equipment does not have to change.

When functional signaling is used, the transfer feature might be invoked differently. Rather then depressing the hookflash, the user presses the transfer button, and then enters in the number to transfer to. The phone sends a specific protocol message to the switch, requesting the transfer and providing the number to transfer to. In this case, the switch, the user, and the phone all have an understanding of the semantics of the service being invoked.

Stimulus signaling can be provided through many different mechanisms on phones. One is depressing the hookflash (which is only possible on certain phones). More commonly, users press the numbers on the keypad, generating DTMF. Speech recognition is another way to provide stimulus signaling.

It is worthwhile to note that the web usually operates under the model of stimulus signaling. The client equipment, the web browser, is not aware of the semantics of the services invoked by the end user. The end user knows how to use the service (as a result of the user interface provided by the server, through the HTML content downloaded into the browser), and the server knows what service is being requested when a page is fetched or a form is posted. The browser is not aware of the semantics of the application. This is not true for web applications based on the Simple Object Access Protocol (SOAP) [266], which uses HTTP and XML to communicate between automata.

There are strengths and weaknesses to each method. Stimulus signaling only works effectively when there are humans driving the application on the client side. This can be very limiting. Without the presence of downloadable user interfaces (as is the case for the web), the user interfaces for stimulus signaling need to be known by users a-priori, and they are often confusing and difficult to understand. Error conditions are difficult to handle. However, stimulus signaling allows for rapid introduction of new applications and features, since a much smaller set of elements need to be modified to support the application.

Functional signaling is faster, and can be driven by automata, not just human beings. Applications and features provided by functional signaling can have rich user interfaces without the need to download them. However, functional signaling requires that every feature or application have a protocol defined to support it, and then have this protocol deployed in client devices and in the servers.

Both methods have a place in an Internet telephony application architecture. It is our belief that functional signaling is a requirement for the most common features and applications, so that they can be driven by automata (call establishment, call termination, hold, transfer). However, deployment of new applications is hindered by functional signaling. Complex and specialized applications, such as the ones discussed in the introduction, are not likely to have protocols defined just to support them. For these kinds of applications, stimulus signaling is more appropriate.

Our architecture supports both forms of signaling for driving applications. In the next two subsections, we examine how this is done.

6.4.9.1 Stimulus Signaling

The primary role of the dialog component is to collect stimulus input from users, in order to drive applications. The dialog component can support input from users either through DTMF or speech recognition. To collect stimulus input from the user, the controller uses third party call control to connect the user to the dialog component. The dialog server is passed a VoiceXML script that defines the desired data to be collected from the user, and the voice form used to collect it. The data collected is passed to the controller using HTTP.

Another form of stimulus signaling is through web forms. Collection of stimulus input through web forms is also supported by our architecture. When a user makes a call to initiate an application, the call is received by the controller. The controller can return a provisional or final response to the caller, containing a Call-Info header. This header, specified in the updated SIP specification [267], contains an HTTP URL that points to content associated with the call. When returned in a response, it causes the caller to fetch the URL contained in the header. The controller can also place this header in an INVITE it creates or forwards, so that a called user also fetches a URL.

Once this initial HTTP connection is made between the users in the call, and the controller, the controller can use it to collect data directly from the users. A new web page can be "pushed" by the controller, to the users, by placing the Call-Info header in an INFO request, generated at any time during the lifetime of a call leg between the end users and the controller. Pushed web pages can contain HTML forms for gathering stimulus, as needed. This case is shown in Figure 6.15

The end user initiates a call using an INVITE (1) that results in the invocation of an application. The controller responds with a 200 OK (2) and the user ACKs it (3). At some point during the call, the controller wishes to obtain input from the user through a web form. The controller can determine that the caller's access device is web capable based on the presence of

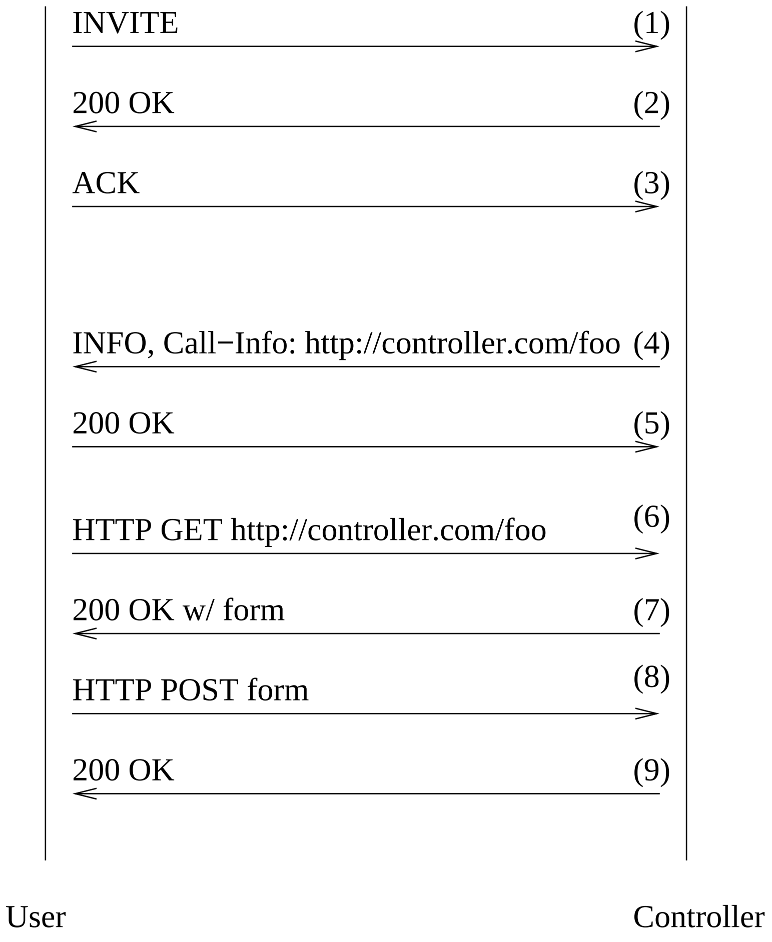


Figure 6.15: Using HTTP as a stimulus protocol

certain headers in the INVITE (specifically, the Accept-Contact and Accept headers, which can indicate support for HTTP URLs and HTML, respectively). Assuming the user's device is web enabled, the controller sends it an INFO request (4). This request contains a Call-Info header, indicating a URL to fetch. In this case, the URL (`http://controller.com/foo`) points back to the controller. After accepting the INFO request (5), the user fetches the URL (6). The controller returns a form (7). The user fills out the form, and POSTs it (8). The response (9) can contain another form, if needed.

6.4.9.2 Functional Signaling

We propose two distinct classes of functional signaling for services. The first are primitives for media stream manipulation, and the second are primitives for call routing. We chose these two as they are the fundamental services provided by SIP.

The basic primitives needed for media stream manipulation are the ability to add a stream between parties or in a multicast group, remove a stream, modify the codec or other characteristics of a stream, modify the transport destination of a stream, and to put a stream on hold. Fortunately, all of these are supported within the baseline SIP specification [89].

The second set of primitives are call routing services. SIP is primarily responsible for call routing. However, the baseline SIP model is that each proxy makes an independent decision on call routing based on its own local policy. There is no provision for other entities involved, like the caller or callee, to guide call routing. Allowing callees to provide input on call routing is the subject of the Call Processing Language [144, 141], and we do not discuss it further. Allowing the caller to have input to the call routing process is fundamentally a functional signaling capability. To support it, we have developed an extension to SIP for *caller preferences and callee capabilities* [268].

The extension defines a set of additional parameters to the Contact header. These parameters specify attributes that define the characteristics of the UA at the address in the header. For example, there is a mobility parameter which indicates whether the UA is fixed or mobile. When a UA registers, it places these parameters in the Contact headers to characterize the URIs it is registering. This allows the proxy to have information about the contact addresses for a user.

The INVITE message, and its response, also contain Contact headers used to route subsequent messaging. This extension allows these headers to contain extension parameters to provide additional information about the type of user agent being used. For example, by including the feature parameter with value “voicemail” in the 200 OK to an INVITE, the UAS can indicate to the UAC that it is a voicemail server. This information is useful for user interfaces, as well as automated call handling.

When a caller sends an INVITE, it can optionally include new headers which request certain handling at a proxy. These preferences fall into two categories. The first category, carried in the Request-Disposition header, describe desired server behavior. This includes whether the caller wishes the server to proxy or redirect, and whether sequential or parallel search is desired. These preferences can be applied at every proxy or redirect server on the call signaling path.

The second category of preferences are carried in both the Accept-Contact and Reject-Contact headers. These preferences contain rules that describe the set of desired URIs that the caller would like the server to proxy or redirect to. These rules are matched against the Contact headers sent in a registration (or through some other configuration means). If a rule in a Reject-Contact header matches a Contact header from the registration, that address is not used. If a rule in a Accept-Contact header matches a Contact header, the q values in the rule are combined with the q values in the Contact header, resulting in a "merged" q value. This merged q value is then used by the proxy to determine the ordering of addresses to use. Note that this second category of preferences can only be applied at a proxy which accesses a registration database.

Details of the syntax and the processing algorithm used at servers are described by Rosenberg [268]. We do not repeat them here for brevity's sake.

Clearly, protocols for signaling additional features are needed. Much work has been done on defining protocols for conference controls [269, 270, 271, 272]. The ITU has standardized a conference control protocol as part of its H.323 series of recommendations [97]. However, no standards-based conference control protocol has been defined for SIP systems.

Additional functional signaling can be done through the REFER method, defined by Sparks et al. [273]. It is used to request the UAS to initiate a request to some other entity. Its primary application is call transfer, but other uses exist. Earlier work has proposed the addition of an Also header that could be included in a SIP request [109]. This header would instruct the recipient of the request to place a call to the URL included in the header. However, this approach was discarded because it combined two separate actions (the request itself and the new call) into single message.

6.5 Target Services

In this section, we describe how our architecture supports the implementation of the target applications defined in Section 6.2.

6.5.1 Pre-Paid Calling Card

The service is readily implemented in our architecture. It requires two session components, a gateway and a dialog server. The basic workflow is straightforward. The first operation is the connection of the caller to the controller. The next is the interaction of the user with a dialog server, so that the controller can obtain the PIN. The next is the validation of the PIN with a database. The next step is to connect the caller with the gateway (assuming the PIN is valid), and to set a timer to expire when time runs out on the card. The final step is to disconnect the user from the gateway when the timer fires. The call flow is shown in Figure 6.16.

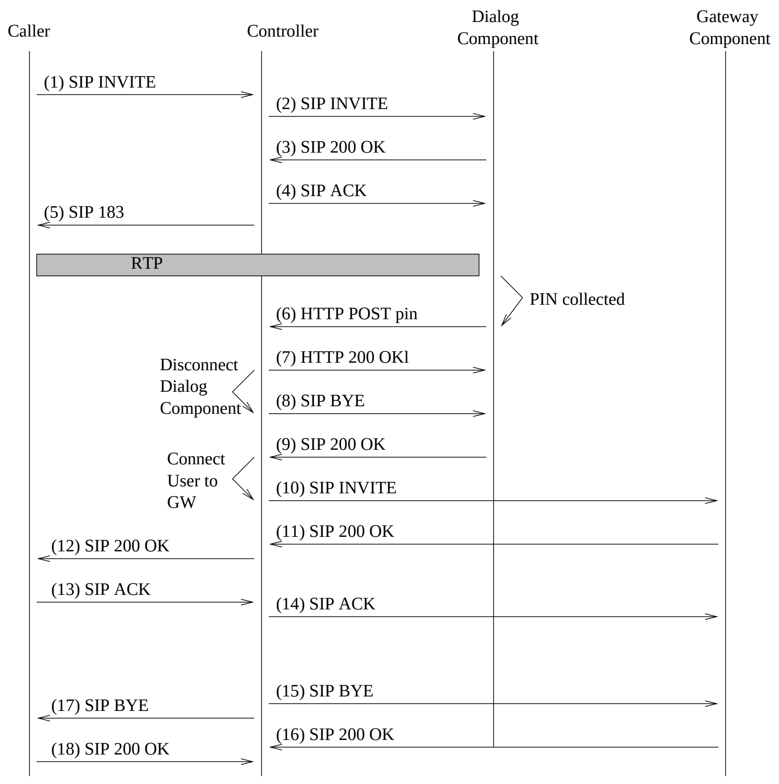


Figure 6.16: Pre-Paid calling card service

The caller begins by constructing a SIP INVITE request addressed to sip:4085551212@prepaid.com, where 4085551212 is the phone number that the caller would like to connect to. This INVITE is sent to the controller (1), which is the server handling SIP requests for `prepaid.com`. The controller recognizes that this is a pre-paid customer. So, it first needs to connect the caller to a dialog component to collect the user's PIN. To do this, it uses third party call control, and sends an INVITE (2) to the dialog component. This INVITE contains the SDP from INVITE (1). The dialog component accepts the call (3), and after the response is acknowledged (4), the SDP is returned to the caller in a provisional 183 response (5). This sets up an "early" media stream from the caller to the dialog component. The dialog component fetches the PIN collection VoiceXML script from the controller (not shown). The media stream is now connected between the caller, and the dialog component. Upon collecting the user's PIN, the dialog component sends an HTTP POST to the controller, containing the PIN collected from the user (6). The controller then terminates the call with the dialog component (8,9). The controller then authorizes the user based on the PIN, and determines the number of minutes left. Typically this is done through access to backend database and authorization systems (not shown). Assuming there are minutes remaining, the INVITE is proxied (10) to the destination user (likely a gateway). The call is accepted (11,12), and acknowledged (13,14).

Some time later, the remaining time on the card expires. The application server therefore decides to terminate the call forcibly. To do this, it once again uses third party call control, and sends a BYE in both directions, hanging up the call (15-18).

6.5.2 Click-to-dial

We want to enable this service in such a way that its easy for the commerce site to deploy without doing any real extra work. This means that the actual click-to-dial service is run by a third party provider. The provider tells the commerce site to include a URL in the page whenever they want a call to the customer service department to be made. The commerce site is responsible for dynamically generating the URL with form parameters containing information on the customer. In particular, it must contain their phone number. For example, the commerce site would be required to embed the following URL in a web page when user Joe is browsing:

```
<a
href=http://controller.c2d.com?
 username=joe&phone=5551212&site=amazon">
Call customer rep</a>
```

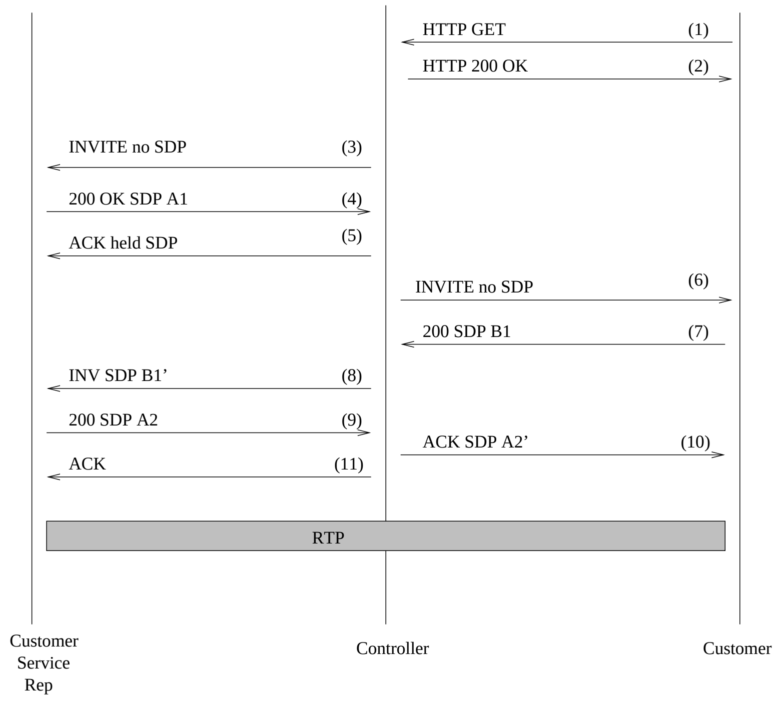


Figure 6.17: Click-to-dial service

A call flow is shown in Figure 6.17. The call flow is nearly identical to that of Figure 6.11. The difference is that the third party controls are executed as a result of the customer clicking on a link. When the user clicks on this hyperlink, it sends a request to the controller providing the click-to-dial service (1). This initiates the application. The code in the controller extracts user's phone number from the HTTP URL. The URL also contains an identification of the commerce site. The controller uses this to determine the phone number of the customer service representative. This can be an Internet telephony terminal, or a traditional PSTN number. The controller returns a web page in the response indicating that the application was invoked (2).

At this point in execution of the service, the controller has the phone numbers or SIP URLs of the two participants (the end user and the customer service representative). The controller determines a gateway to use for connecting calls to the phone numbers (using the mechanisms we describe in Chapter 5). It then begins the third-party call control process based on the advanced flow of Section 6.4.8.2. This starts with an INVITE (3), containing no SDP, targeted at the customer service representative (or the gateway used to connect the to representative). The

200 OK response accepting the call (4) contains the offered SDP, A1. The controller responds with an SDP answer in the ACK (5), indicating that it is on hold. It then contacts the customer using an INVITE with no SDP (6). The end user offers their SDP, B1, in the 200 OK used to accept the call (7). The controller then begins the re-INVITE with the customer service representative. The INVITE request sent to the representative contains a re-ordered version of SDP B1 (8). The 200 OK response (9) contains the answer from the representative, A2. This response is acknowledged (11), and at the same time, a reordered version is passed as the answer in the ACK to the customer (10). Media know flows between the customer and the customer service representative.

6.5.3 Auto-conference

The auto-conference application is described in Section 6.2. The application is initiated by an HTTP request containing the list of participants for the conference, and their presence addresses (these are the SIP URLs to which a SUBSCRIBE request is sent) and phone numbers (or SIP URLs for Internet telephony endpoints). This request is delivered to the controller. The controller begins by accessing the presence component, whose interface is described in Section 6.4.5. This allows the controller to subscribe to the status of the participants in the conference. The presence component will begin generating notifications to the controller. When the controller determines that all are online, it accesses the instant messaging component. It requests that the component deliver an instant message to each participant. Each instant message contains an html document that queries the participant for their willingness to join the conference. When the participants click on either “yes” or “no”, an HTTP request is sent to the controller. The Request URI of the request informs the controller about whether that participant is willing or not to join. Once all participants have responded, the controller initiates the conference call. To do that, it uses third party call control, in conjunction with a mixing component. One by one, it connects each participant to the mixing component, and uses the same Request-URI for each call leg initiated to the mixing component. This causes each participant to be placed into the same mixing context within the mixing component. At this point, the application is complete.

The detailed call flows for this application are shown in Figures 6.18 and 6.19, which we have split apart for readability.

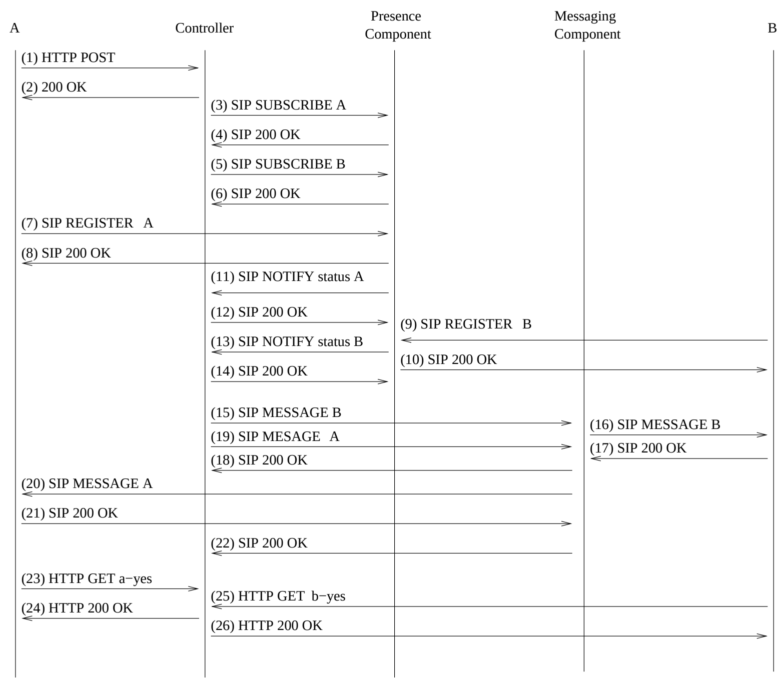


Figure 6.18: First half of auto-conference service

The first half of the call flow for this service is shown in Figure 6.18. We have simplified the flow by assuming that there are only two users in the conference call. First, the user who wishes to schedule the call (in this case, user A, who also wishes to be a participant), fills out a web form and submits it (1). This HTTP request is delivered to the controller, which returns an HTTP 200 OK response containing an HTML document (2). This document indicates that the application is in progress. The controller then accesses the presence component. It first sends a SUBSCRIBE request to subscribe to the status of A (3), and then another to SUBSCRIBE to the status of B (5). Both requests are accepted (4,6). In this example, the presence component knows the status of A and B because it is co-located with a registrar.

Some time later, user A logs into their computer, and starts their SIP phone application. This generates a SIP REGISTER message to be sent to its registrar (7), indicating that user A is now available. The registration is accepted (8). The presence component proceeds to deliver notifications about the change in status to the subscribers. In this case, it generates a NOTIFY request to the controller (11), containing the status of user A. This notification is accepted (12).

Similarly, user B logs in, generating a REGISTER request (9), which is accepted (10). A notification is sent to the controller (13), and it is accepted (14). At this point in the progress of the application, both conference participants are now available.

The next step is to communicate with the message component, to deliver instant messages to each participant. It sends an instant message to B (15) and A (19). The messaging component delivers the messages, and their acceptance responses (16,17,18,20,21,22). As discussed above, each message contains an HTML document that asks the user to click one link to join the conference, and another to decline.

The HTML document delivered to A might look like:

```
<html>
<body>
User A has asked for a conference call to be set up when user A and
user B are both available. Both are available right now. If you
would like to join the call, click <a
href="http://controller.com?sessionid=a9sdyasd&user=a&answer=yes">
here</a>, otherwise, to
decline, click
<a href="http://controller.com?sessionid=a9sdyasd&user=a&answer=no">
here</a>.
</body>
</html>
```

The URLs for the yes and no responses convey the identity of the respondent (A or B) to the controller, along with the answer itself. Both are conveyed as URL parameters, using standard HTTP form conventions. These requests represent stimulus signaling, using HTTP, as discussed in Section 6.4.9.1.

Let us assume both users accept. The result is going to be an HTTP GET request from both (23,25), which is accepted. The responses (24,26) might contain an HTML document confirming initiation of the call.

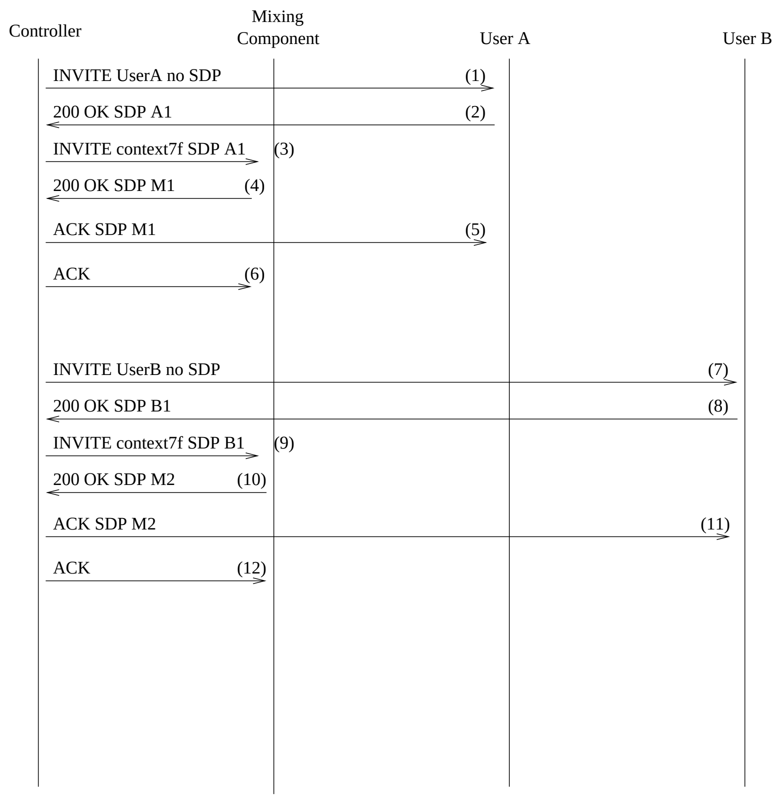


Figure 6.19: Call establishment phase of auto-conference

At this point, the controller must set up the conference call. It does so by using third party call control mechanisms between the users and the mixing component. This process is shown in Figure 6.19. The call flow is basically two invocations of the basic third party call control primitive of Figure 6.10. The first invocation connects user A to the mixing component, in context 7f. The second invocation connects user B to the mixing component, also in context 7f. Both users are now in a conference call.

The example is a bit contrived, since in the case of two users, the controller can directly connect them, rather than using a mixing component. However, we use the mixing component to illustrate what it would look like if there were more than two users.

6.5.4 Web Form Entry for Call Center

The “Web form entry for call center” application is described in Section 6.2. This application is done readily without the services of any components beyond the controller. Like auto-conference, it uses HTTP-based stimulus signaling between the user and the controller.

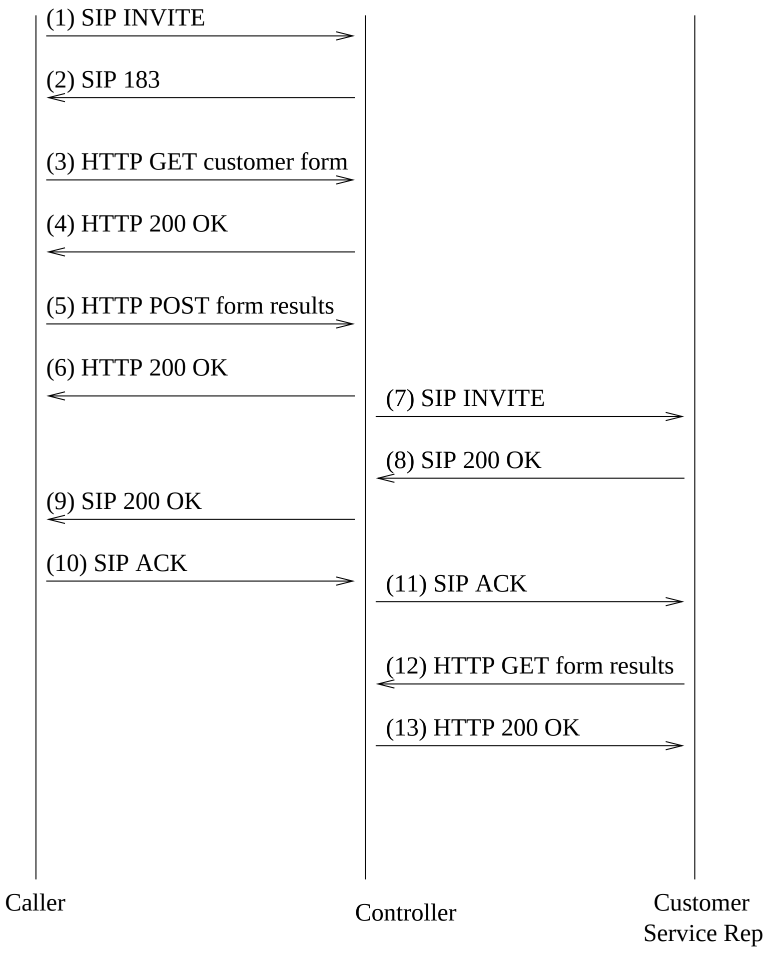


Figure 6.20: Call flow for web form entry for call center

The call flow for this application is shown in Figure 6.20. The caller sends an INVITE to the controller (1). The controller returns a provisional response (2) containing a Call-Info header. This header contains an HTTP URL that resolves to the controller. The client fetches the content referenced by the URL (3). The response (4) contains a form for the caller to fill out. The user fills out the form and posts it back to the controller (5). The posted form is accepted (6). Now that the controller has sufficient information to route the call, it forwards the INVITE (7) to the right call center agent. This request also has a Call-Info header with a URL that resolves to the controller. This URL can be used to fetch the form data (along with any other customer specific data the server would like to provide). The call center agent answers the call (8), and the acceptance is forwarded to the caller (9). The call is ACKed (10,11). Then, the call center agent fetches the URL presented in the Call-Info header in the INVITE (12). This returns a web page with the caller's form information (13).

6.5.5 Speech-to-text for the Hearing Impaired

We have investigated numerous applications enabled by our model which can benefit the hearing impaired [274]. These applications generally involve translations between communications mediums, such as text, speech, and video. One such application, "Speech-to-text for the Hearing Impaired", is described in Section 6.2.

The application is enabled with a pair of translation components. One translation component converts the speech to text, and another, the text to speech. The controller connects the the hearing impaired callee and the non-impaired caller to these translation components. If the caller was connected through the PSTN, a gateway component would be involved as well.

A call flow for this application is shown in Figure 6.21. We refer to the text-to-speech component as the TTS component, and the speech-to-text component as the STT component. The caller is a non-impaired user, and the callee is hearing impaired. The called party makes use of a text chat tool.

The caller, who is unaware the callee is using a text chat tool, places a call to the SIP URL of the callee, `sip:calledparty@controller.com`. It formulates a SIP INVITE, and sends it to the controller (1). The SDP in this INVITE is A1, and it indicates a single, audio media stream. The receipt of the INVITE at the controller triggers the application. It sends a provisional response to the caller (2), indicating that translation services are being invoked. This is easily done by using an appropriate reason phrase; i.e., "183 Translation for Hearing Impaired Being Established". Then, using third party call control mechanisms, it sends an INVITE to a text-to-speech (TTS) component (3). This INVITE contains an SDP, A2, constructed by the

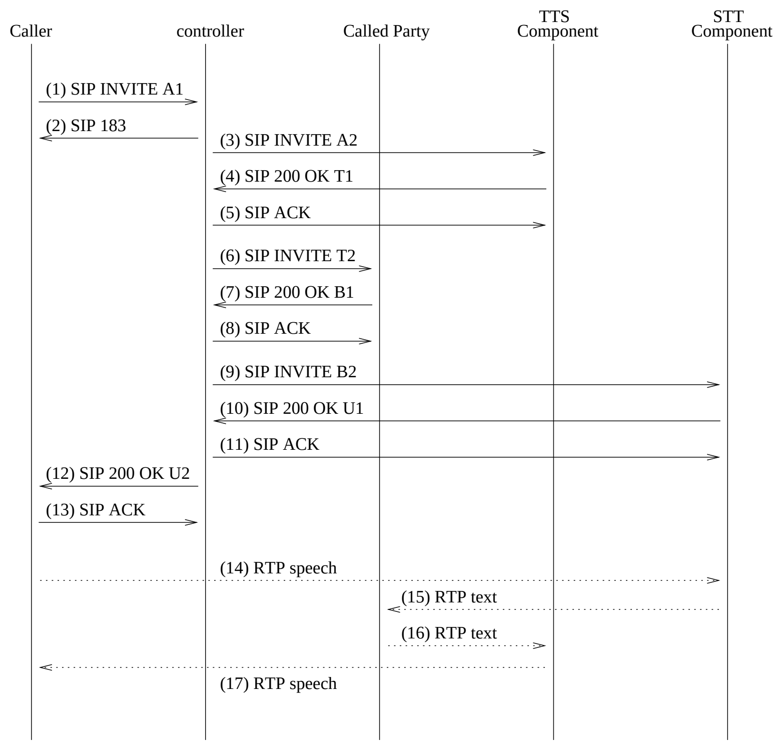


Figure 6.21: Bidirectional translation services for the hearing impaired

controller. This SDP contains two media streams. One is an audio stream, listed as receive-only, and the other is a text stream, listed as send-only. The audio stream contains the IP addresses, ports, and codecs from the media stream definition in SDP A1. This will connect a one way speech stream from the TTS component, to the caller. The TTS component accepts the call and responds with a 200 OK (4). The SDP in this response (T1) also indicates two media streams. One is the audio stream, listed as send-only, and the other is a text stream, listed as receive-only. The receive-only text stream indicates the IP address and port that it expects to receive the text stream. Now, the controller sends an INVITE to the original callee (6), with SDP T2. T2 indicates a single bidirectional text media stream. The stream indicates the IP address and port to use, and these are the IP address and port from SDP T1. This will connect a one way text stream from the callee, to the TTS component. The callee responds with a 200 OK (7), containing SDP B1. B1 indicates a single bidirectional text media stream, containing the IP address and port where the callee would like to receive the text stream.

Now, the controller must introduce the speech-to-text component. It sends an INVITE (9) to the STT component. This INVITE contains SDP B2. B2 indicates two media streams. One is a receive-only text stream, and the other is a send-only audio stream. The IP address and port for the receive-only text stream are copied from SDP B1. This will connect a one way text stream from the STT component, to the callee. The 200 OK response (10) from the STT components contains SDP U1. SDP U1 has two media streams. One is a send-only text stream, and the other is a receive-only audio stream. This response is acknowledged (11). Finally, the controller sends a 200 OK response to the caller (12). The SDP in this response, U2, contains a single bidirectional audio stream. The IP address and port for this stream are copied from the receive-only audio stream in SDP U1. This will connect a one way audio stream from the caller, to the STT component. After the ACK is sent (13), the translation service is established. Speech flows from the caller to the STT component, and then to the callee as text. Text flows from the called party, to the TTS component, and back to the caller as speech.

6.6 Comparison to Existing Architectures

It is difficult to quantify the strengths and weaknesses of our architecture as compared to those that have been described previously in the literature and summarized above in Section 6.3. However, we can evaluate it qualitatively based on the requirements we outlined in Section 6.2.

6.6.1 DFC and ECLIPSE

The ECLIPSE architecture [246], which extends DFC into the IP communications space for an implementation, is most closely related to our architecture, and for this reason, we consider it first. ECLIPSE uses IP protocols between components, allowing them to be widely distributed. It uses a centralized feature router, much like our controller, to compose feature boxes together. It supports applications that integrate other IP applications, including instant messaging and web. It has demonstrated interoperability with PSTN endpoints through gateway feature boxes. Features can be provided for heterogeneous devices, ranges from phones to web enabled Internet telephony PC clients.

However, there are some important differences. The goal of DFC and ECLIPSE is not to describe a single feature as a desired composition of components, rather, it is to manage the interaction of a number of different features, each implemented as a single component. This distinction is important in regards to several of our requirements. First, it means that the DFC architecture has not been proven as a means for building large and complex applications. DFC has not focused on defining modular, highly reusable feature boxes, and it is not known whether the existing set of feature boxes (nearly 100 [246]) are readily composed to construct the applications we describe here. Furthermore, it is unlikely that their precedence rules, which are used to determine the ordering and usages of feature boxes, is sufficient for purposefully composing feature boxes to build a more complex application. Another implication of this distinction is that it is not known whether DFC facilitates rapid construction of new applications, for the same reason it is not known whether it supports construction of modular ones.

On a more practical perspective, a weakness of DFC and ECLIPSE is that they have defined their own, proprietary protocols for interconnection between components. These protocols have not been defined for inter-domain operation. As such, it is not likely that they could construct applications where a single complex application is built from feature boxes provided by separate providers. Furthermore, third party development is not facilitated by the usage of proprietary protocols and APIs.

Note, however, that ECLIPSE has specified line interface boxes that can interface SIP systems to their protocol. However, that does not mean that feature boxes themselves can be interconnect with SIP (whereas we have shown that it can be done between our components), unless it can be demonstrated that their is a one-to-one mapping between the two protocols.

6.6.2 Distributed Software and Component Architectures

In Section 6.3.2, we considered many distributed software and component architectures. We pointed out that these architectures, which generally relied on distributed object middleware (usually CORBA), had difficulties meeting some of our requirements. Specifically, they had difficulty handling third party component providers and third party component service providers. This was largely a result of the non-standard interfaces between components, and the security, and backwards compatibility issues with inter-domain communications.

Our architecture overcomes these limitations. By using industry standardized protocols as the interfaces between components, third party development is enabled, since third parties generally prefer to implement to standardized interfaces.

Usage of industry standard protocols helps overcome the interoperability issues discussed in Section 6.3.2. SIP (and HTTP) have significant support for extensibility. Clients can request new features, and if the server doesn't support them, the requests are ignored. SIP extensions have been defined that let a client indicate what extensions they support, so that the server can apply extensions to the response [275]. These extensibility mechanisms are not possible in CORBA systems.

Domain interoperability issues are also addressed. Inter-domain naming and routing, inter-domain and end-to-end security mechanisms, and discovery features are all provided by SIP, and fully spelled out in its specifications.

Finally, because of the existence of equipment that speaks these standardized protocols (gateways, media servers, clients, and so on), complete systems can be built more rapidly.

To a large degree, usage of distributed object systems, as opposed to designing customized protocols, is one of personal taste and preferences. We believe that customized protocols can always be designed to outperform a general purpose framework for almost any metric that can be defined. Of course, this comes at the expense of additional development time to build these customized protocols, which frequently share functionality with each other in any case. Our architecture presents an intermediate approach. We do use customized protocols, but they are fairly general purpose ones in their own right (which explains why we can construct so many different applications with them). Thus, we inherit some of the reuse benefits that exist in CORBA systems, yet inherit the inter-domain operation, better performance, and better extensibility that is derived from customized protocols.

It is interesting to note that the Simple Object Access Protocol (SOAP) [266] is under development within the W3C as an RPC mechanism for business-to-business transactions based on inter-domain network protocols (HTTP, specifically). The driving factors behind SOAP, the need for secure, robust, inter-domain, application-specific RPC, are the same factors which led us to specify the use of SIP for inter-component communications.

6.6.3 Mobile Agents

We discussed existing mobile agent approaches, based either on Turing complete languages, or domain specific languages, in Section 6.3.4. These approaches based on Turing complete languages have significant security and interoperability problems, while the domain specific approaches are functionally limited.

Our architecture overcomes these limitations. Using industry standardized protocols for interdomain communication allows us to use the security features they provide. There is no risk of malicious code, yet we have the benefit of being able to construct arbitrarily complex applications whose components can span domains.

6.6.4 Centralized Architectures

We outlined a number of problems with centralized architectures in Section 6.3.1. These problems were the inability to support third party development of applications, long development cycles for new applications, difficulty integrating heterogeneous access devices, and inability to support third party components or providers.

Not surprisingly, our architecture rectifies all of these problems. By distributing components, rather than relying on centralized, monolithic code, we improve the time to develop new applications. Our architecture enables third party component providers, which is fundamentally impossible in the centralized service model.

6.7 Conclusions and Future Work

In this chapter, we have examined the problem of building complex telecommunications applications in an Internet telephony system. The end-to-end nature of the Internet enables distributed service architectures, which are advantageous because of their faster development cycles and better scalability. However, our challenge was to enable distributed architectures where the components could be distributed amongst services providers, incorporate non-voice Internet applications, and still work with users connected through the telephone network.

Much of the existing work on distributed service architectures made use of distributed software middleware, such as CORBA or JavaBeans. However, we found that these systems end up being proprietary and do not enable inter-provider communications. Our contribution is the construction of a distributed component architecture that makes use of industry standard, inter-domain call signaling and bulk data transfer protocols between components. We define a series of highly reusable components, and for each, describe their functions and interfaces. We then show how these components can be used to construct complex applications.

We, along with a team of engineers at the employer of the author of this dissertation, have constructed the controller component described in this architecture, along with the presence and messaging components. We obtained commercially available mixing components and dialog components, which were constructed according to the interfaces we define here. The architecture was validated by building and demonstrating the auto-conference application. Our component architecture allowed for substantial reuse. The code at the controller, which was the only code specific to this application, was written in Java, using the SIP and HTTP servlet specifications [276, 277]. It was less than 1000 lines of Java code, including the code for the generation of the web pages. We believe this is an excellent result, and demonstrates that applications can indeed be written rapidly with this architecture.

The continuing challenge with any service architecture is to apply it to more and more applications. Future work involves the investigation of more complex applications, to validate the architecture and to identify additional components needed to enable those applications.

Chapter 7

Conclusion and Future Work

We conclude this dissertation by focusing once more on some of the fundamental differences between IP networks and circuit networks, casting our work in that light and looking at future problems for multimedia IP communications.

The most apparent difference between IP networks and the circuit networks is the nature of the data delivery service. Circuits offer low latency, zero jitter, and low loss, whereas IP networks have highly variable delays, substantial jitter, and variable loss rates. Not surprisingly, this means that best effort IP networks (and indeed, packet networks in general), are not as well suited for synchronous multimedia delivery. Approaches to solving this problem have been fairly well investigated. Our contribution is the realization that these varying solutions interact with each other, and that such interactions need to be considered in order to improve the application to application perception of quality. It is our belief that voice transport problems will ultimately be addressed at the network layer itself, through the usage of application-independent QoS services, such as integrated services, differentiated services, and packet scheduling systems within the network. The challenge, therefore, will be to consider the interaction the existing end-to-end adaptation mechanisms (such as FEC, playout buffer adaptation, packet loss concealment) with these techniques.

Interestingly, the most important difference between IP networks and circuit networks, as far as delivery of multimedia communications services, is not the increase in jitter or packet loss, but rather the end-to-end nature of the Internet as compared with the centralized and segmented nature of the public switched telephone network. The end-to-end nature of the Internet means all hosts, which include end-user participants and network servers, are directly addressable by any other host. This fundamentally changes many aspects of the way telecommunications services are offered.

At the simplest level, end users can now directly communicate with each other without the support of application layer intermediaries. If party A wishes to talk with party B, party A need only be aware of the IP address of party B. Messaging can then take place directly between A and B in order to establish a call. In such a scenario, network switches are no longer present; rather, application unaware routers provide the connectivity. In practice, party A will not know the IP address of party B. Therefore, network servers are required to provide name to address translation and user discovery services. These servers, known as proxies in SIP terminology, play a similar role to the SS7 signaling network elements in the PSTN. Both are responsible for forwarding call establishment messages from caller to callee. However, their role differs in many respects. Although the PSTN supports a separate signaling and circuit transport network, the primary role of the SS7 switches are to establish the circuit path. SIP proxies, on the other hand, have no interaction with media connectivity. They do not need to remain in the call signaling path once the call is established. This implies that they are not responsible for providing many of the services and applications their SS7 equivalents provide.

This difference, in turn, creates some important technical problems that need to be addressed. In the circuit switched network, quality of the media transport is monitored by the switches. Inter-switch signaling protocols provide mechanisms to detect failure conditions and errors, and to switch to alternates if needed. However, in IP networks, the equivalent of the switch, the proxy server, no longer handles media traffic. Therefore, the monitoring of the transport quality happens end-to-end, rather than between switches. This need has driven the development of RTCP, and has introduced scaling issues for large party conferences. Our reconsideration work has addressed these problems and allowed for end-to-end feedback to work for small and large party conferences alike. Additional work is needed to improve the quality of this feedback in the common case of unicast, point-to-point media communications.

The substantially differing roles between a circuit switch and a proxy server also mean that the signaling protocols used for call establishment need to be different. Instead of establishing hop-by-hop circuit connections, end-to-end transport parameters are conveyed. Since the proxies are not participants in the end-to-end media sessions, the information about these sessions can be separated from the call routing aspects of the signaling protocols, which are important for the proxies. This would dictate a signaling protocol that separates the transfer of signaling information from the content of the session established by the signaling. Indeed, this is exactly how SIP operates.

The differences between switches and proxies also mean that features are provided in an entirely different way. Normally, the switch handled the media and signaling for a call. A switch had the one and only point of access for communicating with the end systems in a call. However, in IP networks, these assumptions are no longer true. Proxies do not handle the media, and are not the sole point of communication between the end device and the network. This means that call features can and should no longer be provided in the same way they were in the circuit network. Many features belong in the end devices; typically, these are features which require knowledge about all calls established to that device (such as call waiting). Since the device can establish calls without support from any network intermediaries, the only entity that knows about all calls established to the device is the end device itself. Other features can be distributed to specialized devices scattered about the network. Since the IP network is end-to-end, the end user devices can directly communicate the the network servers providing features, as needed. Our work on the application component architecture for SIP shows how to taxonomize and construct a wide set of applications in this model. However, much additional work is needed on features and applications. The problem of feature interaction, in particular, is much more difficult in IP networks. The distributed nature of features, and the introduction of new applications as part of communications features (such as web and instant messaging), further complicate the feature interaction problem. Furthermore, several telephony features require cooperation from a large set of elements in the circuit network. A example of such a service is emergency calling, which requires support at almost all levels of the telephone network. Providing support for these features in an IP network poses significant technical obstacles. Indeed, it is not even clear what 911 means in a multi-application network. Can a user send an instant message to 911 and expect similar results to making a phone call to 911? What aspects of 911 are application independent, and therefore need to be supported in the underlying IP network itself? Do we require there to be a 911 IP address [278]?

The end-to-end nature of the Internet, and of IP networks, is what allows for an application to be distributed amongst components scattered throughout the network. It also allows end users to directly talk to other network resources, such as telephony gateways, without intervening providers. However, in order to communicate with such resources, they must be discovered. This resource discovery problem for telecommunication services is unique to IP networks. It exists only because it is possible for end users to communicate directly with servers from other providers. Our work on multicast-based discovery of network services, and telephony gateways in particular, provides a scalable solution for this problem. Much more work remains in this area as well. Our techniques are particularly susceptible to packet loss, which can increase substantially the amount of time it takes to discover a new resource. Similarly, the large learning times make it difficult to apply our techniques to resources whose characteristics are highly dynamic. We believe that these problems can be addressed by forms of distributed forward error correction, commonly used in reliable multicast protocols [279].

Further integration of IP-based telecommunications services with the existing circuit switched network will continue to pose new research problems. For example, how can local number portability services be provided to IP endpoints? Can existing PSTN billing systems be used to bill for IP telephony calls? Work also remains to address security for Internet telephony. Specifically, Internet telephony introduces many new opportunities for fraud, denial of service, and theft of service. These security weaknesses must be found and countered.

Bibliography

[1] D. Cohen, "Issues in transnet packetized voice communications," in *Proceedings of the Fifth Data Communications Symposium*, (Snowbird, Utah), pp. 6–10 – 6–13, ACM, IEEE, Sept. 1977.

[2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments 1889, Internet Engineering Task Force, Jan. 1996.

[3] V. Paxson, "End-to-end internet packet dynamics," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cannes, France), Sept. 1997.

[4] V. Paxson, *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California at Berkeley, Berkeley, California, May 1997.

[5] International Telecommunication Union (ITU), "Transmission systems and media, general recommendation on the transmission quality for an entire international telephone connection; one-way transmission time," Recommendation G.114, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[6] H. Sanneck and N. T. L. Le, "Speech property-based FEC for Internet Telephony applications," in *Proceedings of the SPIE/ACM SIGMM Multimedia Computing and Networking Conference (MMCN)*, (San Jose, California), pp. 38–51, Jan. 2000.

[7] N. S. Jayant, "Effects of packet losses on waveform-coded speech," in *Proceedings of the Fifth International Conference on Computer Communications*, (Atlanta, Georgia), pp. 275–280, IEEE, Oct. 1980.

[8] J. Wroclawski, “Specification of the controlled-load network element service,” Request for Comments 2211, Internet Engineering Task Force, Sept. 1997.

[9] S. Shenker, C. Partridge, and R. Guerin, “Specification of guaranteed quality of service,” Request for Comments 2212, Internet Engineering Task Force, Sept. 1997.

[10] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation protocol (RSVP) – version 1 functional specification,” Request for Comments 2205, Internet Engineering Task Force, Sept. 1997.

[11] R. Braden and L. Zhang, “Resource ReSerVation protocol (RSVP) – version 1 message processing rules,” Request for Comments 2209, Internet Engineering Task Force, Sept. 1997.

[12] 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.

[13] 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.

[14] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, “Assured forwarding PHB group,” Request for Comments 2597, Internet Engineering Task Force, June 1999.

[15] V. Jacobson, K. Nichols, and K. Poduri, “An expedited forwarding PHB,” Request for Comments 2598, Internet Engineering Task Force, June 1999.

[16] G. Carle and E. W. Biersack, “Survey of error recovery techniques for IP-based audio-visual multicast applications,” *IEEE Network*, Vol. 11, pp. 24–36, Nov. 1997.

[17] A. Mukherjee, “On the dynamics and significance of low frequency components of Internet load,” *Internetworking: Research and Experience*, Vol. 5, pp. 163–205, Dec. 1994.

[18] J.-C. Bolot, “End-to-end packet delay and loss behavior in the Internet,” in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (San Francisco, California), pp. 289–298, ACM, Sept. 1993. also in *Computer Communication Review* 23 (4), Oct. 1992.

[19] D. L. Mills, “Internet delay experiments,” Request for Comments 889, Internet Engineering Task Force, Dec. 1983.

[20] D. Sanghi, A. K. Agrawala, O. Gudmundson, and B. N. Jain, “Experimental assessment of end-to-end behavior on Internet,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), pp. 867–874 (7d.2), March/April 1993.

[21] M. Yajnik, J. Kurose, and D. Towsley, “Packet loss correlation in the MBone multicast network,” in *Proceedings of Global Internet*, (London, England), Nov. 1996.

[22] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, “Measurement and modelling of the temporal dependence in packet loss,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[23] M. Handley, “An examination of MBone performance,” Technical Report ISI/RR-97-450, ISI, Jan. 1997.

[24] N. F. Maxemchuk and S. Lo, “Measurement and interpretation of voice traffic on the Internet,” in *Conference Record of the International Conference on Communications (ICC)*, (Montreal, Canada), June 1997.

[25] International Telecommunication Union, “Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s,” Recommendation G.723.1, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1996.

[26] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, “Adaptive playout mechanisms for packetized audio applications in wide-area networks,” in *Proceedings of the Conference*

on Computer Communications (IEEE Infocom), (Toronto, Canada), pp. 680–688, IEEE Computer Society Press, Los Alamitos, California, June 1994.

[27] P. T. Brady, “A statistical analysis of on-off patterns in 16 conversations,” *Bell System Technical Journal*, Vol. 47, pp. 73–91, Jan. 1968.

[28] C. Perkins, O. Hodson, and V. Hardman, “A survey of packet loss recovery techniques for streaming audio,” *IEEE Network*, Vol. 12, pp. 40–48, Sept. 1998.

[29] C. Perkins and O. Hodson, “Options for repair of streaming media,” Request for Comments 2354, Internet Engineering Task Force, June 1998.

[30] International Telecommunication Union, “Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear-prediction annex A: Reduced complexity 8 kbit/s CS-ACELP speech codec,” Recommendation G.729A, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1996.

[31] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. C. Bolot, A. Vega-Garcia, and S. Fosse-Parisis, “RTP payload for redundant audio data,” Request for Comments 2198, Internet Engineering Task Force, Sept. 1997.

[32] J.-C. Bolot and A. V. Garcia, “Control mechanisms for packet audio in the Internet,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), Mar. 1996.

[33] V. Hardman, A. Sasse, M. Handley, and A. Watson, “Reliable audio for use over the Internet,” in *Proc. of INET’95*, (Honolulu, Hawaii), June 1995.

[34] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley, “Adaptive FEC-Based error control for interactive audio in the Internet,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[35] N. Shacham and P. McKenney, “Packet recovery in high-speed networks using coding and buffer management,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), pp. 124–131, IEEE, June 1990.

[36] E. W. Biersack, “Performance evaluation of forward error correction in ATM networks,” in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (Baltimore, Maryland), pp. 248–257, ACM, Aug. 1992. in *Computer Communication Review* 22 (4), Oct. 1992.

[37] E. W. Biersack, “Performance evaluation of foreward error correction in an ATM environment,” *IEEE Journal on Selected Areas in Communications*, Vol. SAC-11, no. 4, 1994.

[38] L. Rizzo, “Erasure codes for computer communication protocols,” technical report, Universita di Pisa, Pisa, Italy, Jan. 1997.

[39] J. Nonnenmacher, E. Biersack, and D. Towsley, “Parity-based loss recovery for reliable multicast transmission,” *ACM Computer Communication Review*, Vol. 27, pp. 289–300, Oct. 1997. ACM SIGCOMM’97, Sept. 1997.

[40] D. Rubenstein, S. Kasera, D. Towsley, and J. Kurose, “Improving reliable multicast using active parity encoding services (APES),” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[41] N. Matsuo, M. Yuito, and Y. Tokunaga, “Packet interleaving for reducing speech quality degradation in packet voice communications,” in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (Tokyo, Japan), pp. 1787–1791, IEEE, Nov. 1987.

[42] C. Perkins, “RTP payload format for interleaved media,” Internet Draft, Internet Engineering Task Force, Feb. 1999. Work in progress.

[43] B. J. Dempsey, J. Liebeherr, and A. C. Weaver, “A new error control scheme for packetized voice over high-speed local area networks,” Technical Report CS-93-23, Computer Science Department, University of Virginia, Charlottesville, VA 22903, May 1993.

[44] B. J. Dempsey, J. Liebeherr, and A. C. Weaver, “A delay-sensitive error control scheme for continuous media communications,” Technical Report CS-93-45, Computer Science Department, University of Virginia, Charlottesville, Virginia, Oct. 1993.

[45] H. Sanneck, *Packet Loss Recovery and Control for Voice Transmission over the Internet*. PhD thesis, Technical University of Berlin, Oct. 2000.

[46] J. Rosenberg, L. Qiu, and H. Schulzrinne, "Integrating packet FEC into adaptive voice playout buffer algorithms on the Internet," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[47] S. B. Moon, J. Kurose, and D. Towsley, "Packet audio playout delay adjustment: performance bounds and algorithms," *Multimedia Systems*, Vol. 5, pp. 17–28, Jan. 1998.

[48] G. Barberis and D. Pazzaglia, "Analysis and optimal design of a packet-voice receiver," *IEEE Transactions on Communications*, Vol. COM-28, pp. 217–227, Feb. 1980.

[49] P. M. Gopal, J. W. Wong, and J. C. Majithia, "Analysis of playout strategies for voice transmission using packet switching techniques," *Performance Evaluation*, Vol. 4, pp. 11–18, Feb. 1984.

[50] T. Suda, H. Miyahara, and T. Hasegawa, "Performance evaluation of a packetized voice system – simulation study," *IEEE Transactions on Communications*, Vol. COM-34, pp. 97–102, Jan. 1984.

[51] W. A. Montgomery, "Techniques for packet voice synchronization," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-1, pp. 1022–1028, Dec. 1983.

[52] F. Alvarez-Cuevas, M. Bertran, F. Oller, and J. M. Selga, "Voice synchronization in packet switching networks," *IEEE Network*, Vol. 7, pp. 20–25, Sept. 1993.

[53] W. E. Naylor and L. Kleinrock, "Stream traffic communication in packet switched networks: destination buffering constraints," *IEEE Transactions on Communications*, Vol. COM-30, pp. 2527–2534, Dec. 1982.

[54] A. Campbell and G. Coulson, "QoS adaptive transports: Delivering scalable media to the desk top," *IEEE Network*, Vol. 11, pp. 18–27, Mar. 1997.

[55] K. Rothermel and T. Helbig, "An adaptive stream synchronization protocol," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video*

(NOSSDAV), Lecture Notes in Computer Science, (Durham, New Hampshire), pp. 189–202, Springer, Apr. 1995.

[56] A. Campbell, G. Coulson, F. García, and D. Hutchison, “A continuous media transport and orchestration service,” in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (Baltimore, Maryland), pp. 99–110, ACM, Aug. 1992. in *Computer Communication Review* 22 (4), Oct. 1992.

[57] J. Escobar, D. Deutsch, and C. Partridge, “Flow synchronization protocol,” in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (Orlando, Florida), pp. 1381–1387 (40.04), IEEE, Dec. 1992.

[58] S. Moon, P. Skelly, and D. Towsley, “Estimation and removal of clock skew from network delay measurements,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (New York), Mar. 1999.

[59] G. Liebl *et al.*, “An RTP payload format for erasure-resilient transmission of progressive multimedia streams,” Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[60] S. Casner and V. Jacobson, “Compressing IP/UDP/RTP headers for low-speed serial links,” Request for Comments 2508, Internet Engineering Task Force, Feb. 1999.

[61] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. Bolot, A. Vega-Garcia, and S. Fosse-Parisis, “RTP payload for redundant audio data,” Internet Draft, Internet Engineering Task Force, Aug. 1998. Work in progress.

[62] R. G. Kermode, “Scoped hybrid automatic repeat request with forward error correction (SHARQFEC),” *ACM Computer Communication Review*, Vol. 28, pp. 278–289, Sept. 1998.

[63] D. Rubenstein, J. Kurose, and D. Towsley, “Real-time reliable multicast using proactive forward error correction,” in *Proc. International Workshop on Network and Operating*

System Support for Digital Audio and Video (NOSSDAV), (Cambridge, England), pp. 279–293, July 1998.

[64] J. Rosenberg and H. Schulzrinne, “An RTP payload format for generic forward error correction,” Request for Comments 2733, Internet Engineering Task Force, Dec. 1999.

[65] A. Li *et al.*, “An RTP payload format for generic FEC with uneven level protection,” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[66] K. Almeroth and M. Ammar, “Collection and modeling of the join/leave behavior of multicast group members in the MBone,” in *High Performance Distributed Computing Focus Workshop (HPDC ’96)*, (Syracuse, NY), Aug. 1996.

[67] K. C. Almeroth and M. H. Ammar, “Multicast group behavior in the Internet’s multicast backbone (MBone),” *IEEE Communications Magazine*, Vol. 35, June 1997.

[68] M. Handley, C. Perkins, and E. Whelan, “Session announcement protocol,” Request for Comments 2974, Internet Engineering Task Force, Oct. 2000.

[69] S. Floyd and V. Jacobson, “The synchronization of periodic routing messages,” in *SIGCOMM Symposium on Communications Architectures and Protocols* (D. P. Sidhu, ed.), (San Francisco, California), pp. 33–44, ACM, Sept. 1993. also in [280].

[70] J. Nonnenmacher and E. W. Biersack, “Optimal multicast feedback,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), pp. 964–971, March/April 1998.

[71] 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.

[72] V. Jacobson, “sd, the LBL session directory.” Manual page, Nov. 1992.

[73] M. Handley, “Session directories and scalable internet multicast address allocation,” *ACM Computer Communication Review*, Vol. 28, pp. 105–116, Sept. 1998.

[74] M. Hofmann and M. Rohrmuller, “Impact of virtual group structure on multicast performance,” in *Fourth International COST 237 Workshop*, (Lisbon, Portugal), pp. 165–180, Springer Verlag, Dec. 1997.

[75] J. Chang and N. Maxemchuk, “A broadcast protocol for broadcast networks,” in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, Dec. 1993.

[76] B. Aboba, “Alternatives for enhancing RTCP scalability,” Internet Draft, Internet Engineering Task Force, Jan. 1997. Work in progress.

[77] R. El-Marakby and D. Hutchison, “A scalability scheme for the real-time control protocol,” in *Proc. of IFIP Conference on High Performance Networking (HPN’98)*, (Vienna, Austria), Sept. 1998.

[78] J. Rosenberg and H. Schulzrinne, “Timer reconsideration for enhanced RTP scalability,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), March/April 1998.

[79] D. Waitzman, C. Partridge, and S. E. Deering, “Distance vector multicast routing protocol,” Request for Comments 1075, Internet Engineering Task Force, Nov. 1988.

[80] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, “Protocol independent multicast-sparse mode (PIM-SM): protocol specification,” Request for Comments 2362, Internet Engineering Task Force, June 1998.

[81] M. Handley and V. Jacobson, “SDP: session description protocol,” Request for Comments 2327, Internet Engineering Task Force, Apr. 1998.

[82] International Telecommunication Union, “Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service,” Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.

[83] R. Rivest, “The MD5 message-digest algorithm,” Request for Comments 1321, Internet Engineering Task Force, Apr. 1992.

[84] A. Demers, S. Keshav, and S. Shenker, “Analysis and simulation of a fair queueing algorithm,” in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Austin, Texas), pp. 1–12, ACM, Sept. 1989. also in Computer Communications Review, 19 (4), Sept. 1989.

[85] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[86] J. Rosenberg and H. Schulzrinne, “Sampling of the group membership in RTP,” Request for Comments 2762, Internet Engineering Task Force, Feb. 2000.

[87] International Telecommunication Union, “Functional description of the ISDN user part of signalling system no. 7.,” Recommendation Q.761, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, 1994.

[88] C. Gbaguidi, J.-P. Hubaux, G. Pacifici, and A. N. Tantawi, “Integration of Internet and telecommunications: An architecture for hybrid services,” *IEEE Journal on Selected Areas in Communications*, Vol. 17, pp. 1563–1578, Sept. 1999.

[89] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, “SIP: session initiation protocol,” Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[90] H. Schulzrinne and J. Rosenberg, “Signaling for Internet telephony,” in *International Conference on Network Protocols (ICNP)*, (Austin, Texas), Oct. 1998.

[91] M. Day, J. Rosenberg, and H. Sugano, “A model for presence and instant messaging,” Request for Comments 2778, Internet Engineering Task Force, Feb. 2000.

[92] C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinsky, J. T. Kohl, and W. E. Sommerfeld, “The Zephyr notification service,” in *USENIX Winter Conference*, (Dallas, Texas), Feb. 1988.

[93] International Telecommunication Union, “Bearer independent call control protocol,” Recommendation Q.1901, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 2000.

[94] International Telecommunication Union, “Packet based multimedia communication systems,” Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[95] International Telecommunication Union, “Packet based multimedia communication systems,” Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 2000.

[96] International Telecommunication Union, “Media stream packetization and synchronization on non-guaranteed quality of service LANs,” Recommendation H.225.0, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1996.

[97] International Telecommunication Union, “Control protocol for multimedia communication,” Recommendation H.245, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[98] International Telecommunication Union, “Security and encryption for H-series (H.323 and other H.245-based) multimedia terminals,” Recommendation H.235, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[99] International Telecommunication Union, “Interworking of H-series multimedia terminals with H-series multimedia terminals and voice/voiceband terminals on GSTN and ISDN,” Recommendation H.246, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[100] International Telecommunication Union, “Generic functional protocol for the support of supplementary services in H.323,” Recommendation H.450.1, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[101] J. Toga and J. Ott, “ITU-T standardization activities for interactive multimedia communications on packet-based networks: H.323 and related recommendations,” *Computer Networks and ISDN Systems*, Vol. 31, pp. 205–223, Feb. 1999.

[102] J. Toga and H. ElGebaly, “Demystifying multimedia conferencing over the Internet using the H.323 set of standards,” *Intel Technology Journal*, 2nd quarter 1998.

[103] H. Schulzrinne and J. Rosenberg, “A comparison of SIP and H.323 for Internet telephony,” in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV),* (Cambridge, England), pp. 83–86, July 1998.

[104] B. Pagurek and T. White, “A quick evaluation of H.323/H.450,” Technical Report SCE-99-02, Carleton University, Ottowa, Canada, Apr. 1999.

[105] C. Agboh, “A study of two main IP telephony signaling protocols: H.323 signaling and SIP; a comparison and a signaling gateway specification,” Master’s thesis, Unversite Libre de Bruxelles (ULB), Facuts des Science, Dpartment Informatique, Brussels, Belgium, 1999. supervised by Eric Manie.

[106] I. Dalgic and H. Fang, “Comparison of H.323 and SIP for IP telephony signaling,” in *Proc. of Photonics East*, (Boston, Massachusetts), SPIE, Sept. 1999.

[107] O. Levin, “SIP requirements for support of multimedia and video,” Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[108] H. Schulzrinne and J. Rosenberg, “The session initiation protocol: Internet-centric signaling,” *IEEE Communications Magazine*, Vol. 38, Oct. 2000.

[109] H. Schulzrinne and J. Rosenberg, “Signaling for Internet telephony,” Technical Report CUCS-005-98, Columbia University, New York, New York, Feb. 1998.

[110] H. Schulzrinne and J. Rosenberg, “The session initiation protocol: Providing advanced telephony services across the Internet,” *Bell Labs Technical Journal*, Vol. 3, pp. 144–160, October-December 1998.

[111] H. Schulzrinne and J. Rosenberg, “Internet telephony: Architecture and protocols – an IETF perspective,” *Computer Networks and ISDN Systems*, Vol. 31, pp. 237–255, Feb. 1999.

[112] A. Johnston, *SIP: Understanding the Session Initiation Protocol*. Artech House, 2001.

[113] J. Postel, “Simple mail transfer protocol,” Request for Comments 821, Internet Engineering Task Force, Aug. 1982.

[114] J. Klensin, “Simple mail transfer protocol,” Internet Draft, Internet Engineering Task Force, Sept. 2000. Work in progress.

[115] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol – HTTP/1.1,” Request for Comments 2616, Internet Engineering Task Force, June 1999.

[116] C. Elliott, “A ’sticky’ conference control protocol,” *Internetworking: Research and Experience*, Vol. 5, pp. 97–119, 1994.

[117] E. Schooler and S. L. Casner, “An architecture for multimedia connection management,” *ACM Computer Communication Review*, Vol. 22, pp. 73–74, Mar. 1992.

[118] H. Schulzrinne, “Personal mobility for multimedia services in the Internet,” in *European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS)*, (Berlin, Germany), Mar. 1996.

[119] P. V. Mockapetris, “Domain names - concepts and facilities,” Request for Comments 1034, Internet Engineering Task Force, Nov. 1987.

[120] P. V. Mockapetris, “Domain names - implementation and specification,” Request for Comments 1035, Internet Engineering Task Force, Nov. 1987.

[121] J. Rosenberg, J. Peterson, H. Schulzrinne, and G. Camarillo, “Third party call control in SIP,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[122] J. Rosenberg and H. Schulzrinne, “Reliability of provisional responses in SIP,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[123] D. Crocker, “Standard for the format of ARPA internet text messages,” Request for Comments 822, Internet Engineering Task Force, Aug. 1982.

[124] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform resource identifiers (URI): generic syntax,” Request for Comments 2396, Internet Engineering Task Force, Aug. 1998.

[125] P. Hoffman, L. Masinter, and J. Zawinski, “The mailto URL scheme,” Request for Comments 2368, Internet Engineering Task Force, July 1998.

[126] H. Schulzrinne and J. Rosenberg, “SIP: Session initiation protocol – locating SIP servers,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[127] S. Donovan, “The SIP INFO method,” Request for Comments 2976, Internet Engineering Task Force, Oct. 2000.

[128] A. Vemuri and J. Peterson, “SIP for telephones (SIP-t): Context and architectures,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[129] J. Lennox and H. Schulzrinne, “Transporting user control information in SIP REGISTER payloads,” Internet Draft, Internet Engineering Task Force, Oct. 2000. Work in progress.

[130] H. Schulzrinne, “RTP profile for audio and video conferences with minimal control,” Request for Comments 1890, Internet Engineering Task Force, Jan. 1996.

[131] N. Borenstein and N. Freed, “MIME (multipurpose internet mail extensions): Mechanisms for specifying and describing the format of internet message bodies,” Request for Comments 1341, Internet Engineering Task Force, June 1992.

[132] P. Hoschka, “Synchronized multimedia integration language (SMIL) 1.0 specification,” W3C Recommendation REC-smil-19980615, World Wide Web Consortium (W3C), June 1998. Available at http://www.w3.org/TR/REC-smil/.

[133] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, “Extensible markup language (XML) 1.0,” W3C Recommendation REC-xml-19980210, World Wide Web Consortium (W3C), Feb. 1998. Available at http://www.w3.org/TR/REC-xml.

[134] D. Kutscher, J. Ott, and C. Bormann, “Session description and capability negotiation,” Internet Draft, Internet Engineering Task Force, Apr. 2001. Work in progress.

[135] N. Freed and N. Borenstein, “Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies,” Request for Comments 2045, Internet Engineering Task Force, Nov. 1996.

[136] B. Ramsdell and Ed, “S/MIME version 3 message specification,” Request for Comments 2633, Internet Engineering Task Force, June 1999.

[137] F. Dawson, “The vcard v3.0 XML DTD,” Internet Draft, Internet Engineering Task Force, June 1999. Work in progress.

[138] S. Thomas, R. Brennan, B. Anton, and D. Oran, “The application/osp-token MIME type,” Internet Draft, Internet Engineering Task Force, Apr. 1999. Work in progress.

[139] M. O’Doherty, “Java enhanced SIP (JES),” Internet Draft, Internet Engineering Task Force, Jan. 2001. Work in progress.

[140] R. Moats, “URN syntax,” Request for Comments 2141, Internet Engineering Task Force, May 1997.

[141] J. Lennox and H. Schulzrinne, “CPL: a language for user control of internet telephony services,” Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[142] J. Lennox, H. Schulzrinne, and J. Rosenberg, “Common gateway interface for SIP,” Request for Comments 3050, Internet Engineering Task Force, Jan. 2001.

[143] “The apache project home page.” http://www.apache.org/.

[144] J. Lennox and H. Schulzrinne, “Call processing language framework and requirements,” Request for Comments 2824, Internet Engineering Task Force, May 2000.

[145] B. Marshall *et al.*, “Integration of resource management and SIP,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[146] J. Rosenberg and H. Schulzrinne, “SIP traversal through residential and enterprise NATs and firewalls,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[147] J. Rosenberg and H. Schulzrinne, “Models for multi party conferencing in SIP,” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[148] J. Rosenberg and H. Schulzrinne, “Guidelines for authors of SIP extensions,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[149] S. Donovan and J. Rosenberg, “SIP session timer,” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[150] A. Vaha-Sipila, “URLs for telephone calls,” Request for Comments 2806, Internet Engineering Task Force, Apr. 2000.

[151] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, “Service location protocol,” Request for Comments 2165, Internet Engineering Task Force, June 1997.

[152] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service location protocol, version 2,” Request for Comments 2608, Internet Engineering Task Force, June 1999.

[153] R. Droms, “Dynamic host configuration protocol,” Request for Comments 1541, Internet Engineering Task Force, Oct. 1993.

[154] J. Rosenberg and H. Schulzrinne, “Internet telephony gateway location,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (San Francisco, California), March/April 1998.

[155] S. Brands, “Electronic cash on the Internet,” in *Proc. of the Internet Society 1995 Symposium on Network and Distributed System Security*, (San Diego, California), Feb. 1995.

[156] International Telecommunication Union, "Pulse code modulation (PCM) of voice frequencies," Recommendation G.711, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1998.

[157] International Telecommunication Union, "Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear-prediction," Recommendation G.729, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1996.

[158] International Telecommunication Union, "Coding of speech at 16 kbit/s using low-delay code excited linear prediction," Recommendation G.728, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Sept. 1992.

[159] International Telecommunication Union, "7 kHz audio coding within 64 kbit/s," Recommendation G.722, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1988.

[160] Federal Communications Commission, "Statistics of communications common carriers." http://www.fcc.gov/Bureaus/Common_Carrier/Reports/FCC-State_Link/socc.html.

[161] D. Wessels and K. Claffy, "Internet cache protocol (ICP), version 2," Request for Comments 2186, Internet Engineering Task Force, Sept. 1997.

[162] D. Wessels and K. Claffy, "Application of internet cache protocol (ICP), version 2," Request for Comments 2187, Internet Engineering Task Force, Sept. 1997.

[163] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," Request for Comments 2326, Internet Engineering Task Force, Apr. 1998.

[164] S. G. Dykes, C. L. Jeffery, and K. A. Robbins, "An empirical evaluation of client-side server selection algorithms," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[165] R. L. Carter and M. E. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Kobe, Japan), p. 1014, Apr. 1997.

[166] M. E. Crovella and R. L. Carter, “Dynamic server selection in the Internet,” in *Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS’95)*, (Mystic, CT), Aug. 1995.

[167] M. Stemm, S. Seshan, and R. H. Katz, “A network measurement architecture for adaptive applications,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[168] R. Vingralek, Y. Breitbart, M. Sayal, and P. Scheuermann, “Web++: A system for fast and reliable web service,” in *1999 USENIX Annual Technical Conference*, (Montery, California, USA), June 1999.

[169] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, “Selection algorithms for replicated web servers,” in *Proc. of on Internet Server Performance (WISP ’98)*, (Madison, Wisconsin), June 1998.

[170] D. Xu, K. Nahrstedt, and D. Wichadakul, “Megadip: a wide-area media gateway discovery protocol,” in *Proceedings of IEEE IPCCC 2000*, Feb. 2000.

[171] 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.

[172] R. Stata, K. Bharat, and F. Maghoul, “The term vector database: fast access to indexing terms for web pages,” *Computer Networks*, Vol. 33, pp. 247–255, June 2000.

[173] J. Shakes, M. Langheinrich, and O. Etzioni, “Dynamic reference sifting: a case study in the homepage domain,” *Computer Networks*, Vol. 29, pp. 1193–1204, Sept. 1997.

[174] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” *Computer Networks*, Vol. 30, pp. 107–117, Apr. 1998.

[175] D. Byers, “Full-text indexing of non-textual resources,” *Computer Networks*, Vol. 30, pp. 141–148, Apr. 1998.

[176] M. F. Schwartz, A. Emtage, B. Kahle, and C. Neuman, “A comparison of Internet resource discovery approaches,” *Computing Systems – The Journal of the Usenix Assocation*, Vol. 5, pp. 461–493, Fall 1992.

[177] C. Perkins and H. Harjono, “Resource discovery protocol for mobile computing,” in *IFIP ’96 14th World Congress*, Sept. 1996.

[178] Sun Microsystems, “Jini techonology core platform specification,” java specifications, Sun Microsystems, Oct. 2000.

[179] Bluetooth, “Specification of the bluetooth system,” bluetooth specification, Bluetooth, Dec. 1999.

[180] Salutation Consortium, “Salutation architecture specification version 2.0c,” salutation specification, Salutation Consortium, June 1999.

[181] G. R. III, “Service advertisement and discovery: Enabling universal device cooperation,” *IEEE Internet Computing*, Vol. 4, Sept. 2000.

[182] C. Bettstetter and C. Renner, “A comparison of service discovery protocols and implementation of the service location protocol,” in *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School, Twente, Netherlands, September 2000.*, Sept. 2000.

[183] W. Zhao, H. Schulzrinne, and E. Guttman, “mSLP - mesh-enhanced service location protocol,” in *International Conference on Computer Communication and Network*, (Las Vegas, Nevada), Oct. 2000.

[184] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, M. F. Schwartz, and D. P. Wessels, “Harvest: A scalable, customizable discovery and access system,” Technical Report CU-CS-732-94, University of Colorado, Boulder, Colorado, Mar. 1995.

[185] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, “The Harvest information discovery and access system,” *Computer Networks and ISDN Systems*, Vol. 28, pp. 119–125, 1995.

[186] C. Perkins, “Wide area service location protocol,” Apr. 1998. http://www.jdrosen.net/iptel/perkins_mar98.ps.

[187] C. Malamud and M. Rose, “Principles of operation for the TPC.INT subdomain: Remote printing – technical procedures,” Request for Comments 1528, Internet Engineering Task Force, Oct. 1993.

[188] C. Malamud and M. Rose, “Principles of operation for the TPC.INT subdomain: General principles and policy,” Request for Comments 1530, Internet Engineering Task Force, Oct. 1993.

[189] P. Faltstrom, “E.164 number and DNS,” Request for Comments 2916, Internet Engineering Task Force, Sept. 2000.

[190] A. Shaikh, R. Tewari, and M. Agrawal, “On the effectiveness of DNS-based server selection,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Anchorage, Alaska), Apr. 2001.

[191] W. Yeong, T. Howes, and S. Kille, “Lightweight directory access protocol,” Request for Comments 1777, Internet Engineering Task Force, Mar. 1995.

[192] M. Wahl, T. Howes, and S. Kille, “Lightweight directory access protocol (v3),” Request for Comments 2251, Internet Engineering Task Force, Dec. 1997.

[193] A. L. Sears, “Directory services for Internet telephony: Creating a spanning layer over the internet and telephone networks,” Master’s thesis, MIT, Cambridge, Massachusetts, Sept. 1997.

[194] P. Deutsch, R. Schoultz, P. Faltstrom, and C. Weider, “Architecture of the WHOIS++ service,” Request for Comments 1835, Internet Engineering Task Force, Aug. 1995.

[195] C. Weider, J. Fullton, and S. Spero, “Architecture of the whois++ index service,” Request for Comments 1913, Internet Engineering Task Force, Feb. 1996.

[196] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, “An architecture for a secure service discovery service,” in *Mobicom*, (Seattle, Washington), pp. 24–35, Aug. 1999.

[197] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, “The design and implementation of an intentional naming system,” in *Proc. ACM Symposium on Operating Systems Principles*, (Charleston, South Carolina), pp. 186–201, Association for Computing Machinery, Dec. 1999.

[198] J. Rosenberg, H. Salama, and M. Squire, “Telephony routing over IP (TRIP),” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[199] J. Rosenberg and H. Schulzrinne, “A framework for telephony routing over IP,” Request for Comments 2871, Internet Engineering Task Force, June 2000.

[200] J. Allen and M. Mealling, “The architecture of the common indexing protocol (CIP),” Request for Comments 2651, Internet Engineering Task Force, Aug. 1999.

[201] J. Allen and M. Mealling, “MIME object definitions for the common indexing protocol (CIP),” Request for Comments 2652, Internet Engineering Task Force, Aug. 1999.

[202] Y. Rekhter and T. Li, “A border gateway protocol 4 (BGP-4),” Request for Comments 1771, Internet Engineering Task Force, Mar. 1995.

[203] S. Hanna, B. Patel, and M. Shah, “Multicast address dynamic client allocation protocol (MADCAP),” Request for Comments 2730, Internet Engineering Task Force, Dec. 1999.

[204] M. Papadopouli and H. Schulzrinne, “Seven degrees of separation in mobile ad hoc networks,” in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (San Francisco, California), Nov. 2000.

[205] Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, “Simple service discovery Protocol/1.0 operationg without an arbiter,” Internet Draft, Internet Engineering Task Force, Nov. 1999. Work in progress.

[206] E. Guttman, C. Perkins, and J. Kempf, “Service templates and service: Schemes,” Request for Comments 2609, Internet Engineering Task Force, June 1999.

[207] W. S. Lai, “The leaky bucket algorithm for throughput control in packet networks.” Bellcore, 1987.

[208] D. Meyer, “Administratively scoped IP multicast,” Request for Comments 2365, Internet Engineering Task Force, July 1998.

[209] F. S. Dworak, T. F. Bowen, C. H. Chow, G. Herman, N. Griffeth, and Y. J. Lin, “Feature interaction problem in telecommunication systems,” in *Proceedings of the Seventh International Conference on Software Engineering for Telecommunication Switching Systems*, pp. 59–62, July 1989.

[210] J. Lennox and H. Schulzrinne, “Feature interaction in Internet telephony,” in *Proc. of Feature Interaction in Telecommunications and Software Systems VI*, (Glasgow, United Kingdom), May 2000.

[211] International Telecommunication Union, “Principles of intelligent network architecture,” Recommendation Q.1201, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, 1992.

[212] I. Faynberg, L. R. Gabuzda, M. P. Kaplan, and N. J. Shah, *Intelligent Network Standards: their Application to Services*. New York: McGraw-Hill, 1997.

[213] International Telecommunication Union, “Intelligent network interfaces,” Recommendation Q.1218, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, 1995.

[214] V. Gurbani, “SIP enabled IN services - an implementation report,” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[215] V. Gurbani and V. Rastogi, “Accessing IN services from SIP networks,” Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[216] D. Lebovits, "SIP/IN interworking," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[217] L. Slutsman, G. Ash, F. Haerens, and V. Gurbani, "Framework and requirements for the internet intelligent networks (IIN)," Internet Draft, Internet Engineering Task Force, Mar. 2000. Work in progress.

[218] H. Schulzrinne, L. Slutsman, I. Faynberg, and H. Lu, "Interworking between SIP and INAP," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[219] L. Slutsman, I. Faynberg, and H. Lu, "IN/Internet interworking in support of software switches," Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[220] S. Kapur and R. Vij, "Approach for services in converged networks," in *Proceedings of the IP Telecom Services Workshop 2000 (IPTS2000)*, (Atlanta, GA), Sept. 2000.

[221] K. Vemuri, "Sphinx: A study in convergent telephony," in *Proceedings of the IP Telecom Services Workshop 2000 (IPTS2000)*, (Atlanta, GA), Sept. 2000.

[222] T.-C. Chiang, J. Douglas, V. Gurbani, W. Montgomery, W. Opdyke, J. Reddy, and K. Vemuri, "IN services for converged (internet) telephony," *IEEE Communications Magazine*, Vol. 38, June 2000.

[223] M. Arango, A. Dugan, I. Elliott, C. Huitema, and S. Pickett, "Media gateway control protocol (MGCP) version 1.0," Request for Comments 2705, Internet Engineering Task Force, Oct. 1999.

[224] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, and J. Segers, "Megaco protocol version 1.0," Request for Comments 3015, Internet Engineering Task Force, Nov. 2000.

[225] C. Huitema, J. Cameron, P. Mouchtaris, and D. Smyk, "An architecture for Internet telephony service for residential customers," *IEEE Network*, Vol. 13, pp. 50–57, May/June 1999.

[226] PacketCable Forum, "Network-based call signaling protocol specification," packetcable forum specification, PacketCable Forum, Dec. 1999.

[227] Object Management Group (OMG), “Common object request broker architecture specification,” omg specification, OMG, Feb. 2001.

[228] C. Blum and R. Molva, “A software platform for distributed multimedia applications,” in *Proceedings of the 1st Workshop on Multimedia Software Development*, (Berlin, Germany), Mar. 1996.

[229] T. Hofte, H. van de Lugt, and H. Bakker, “A CORBA platform for component groupware,” in *Proceedings of the OZCHI96 Workshop on the Next Generation of CSCW Systems*, Nov. 1996.

[230] J. K. Ousterhout, *Tcl and the Tk Toolkit*. Reading, Massachusetts: Addison-Wesley, 1994.

[231] M. Arango, P. Bates, R. Fish, R. Gopal, N. Griffeth, G. Herman, T. Hickey, W. Leland, C. Lovery, V. Mak, J. Patterson, L. Ruston, M. Segal, M. Vecchi, A. Weinrib, and S. Wuu, “Touring Machine: a software platform for distributed multimedia applications,” in *IFIP’92*, (Vancouver, Canada), p. 11, IFIP, May 1992.

[232] V. Mak, M. Arango, and T. Hickey, “The application programming interface to the Touring Machine,” technical report, Bellcore, Morristown, New Jersey, July 1992.

[233] M. Arango, P. Bates, G. Gopal, N. Griffeth, G. Herman, T. Hickey, W. Leland, V. Mak, L. Ruston, M. Segal, M. Vecchi, A. Weinrib, and S.-Y. Wuu, “Touring Machine: a software infrastructure to support multimedia communications,” *ACM Computer Communication Review*, Vol. 22, pp. 53–54, Mar. 1992.

[234] Arango *et al.*, “Touring machine system,” *Communications ACM*, Vol. 36, pp. 68–77, Jan. 1993.

[235] M. Arango, M. Kramer, S. L. Rohall, L. Ruston, and A. Weinrib, “Enhancing the Touring Machine API to support integrated digital transport,” in *Third International Workshop on network and operating system support for digital audio and video*, (San Diego, California), pp. 166–172, IEEE Communications Society, Nov. 1992.

[236] S. Znaty, T. Walter, M. Brunner, J. Hubaux, and B. Plattner, “Multimedia multipoint teleteaching over the European ATM pilot,” in *Proceedings of the 1996 International Zurich Seminar on Digital Communications*, (Zurich, Switzerland), Feb. 1996.

[237] A. Hopper, “The Medusa applications environment,” Technical Report TR 94-12 (video), Olivetti Research Laboratory (ORL), Cambridge, England, 1994. Proceedings of European Computer Support for Collaborative Working, Stockholm, September 1995.

[238] B. Smith, L. Rowe, and S. Yen, “Tcl distributed programming,” in *Tcl/Tk Workshop*, (Berkeley, California), June 1993.

[239] A. Gokhale and D. C. Schmidt, “Measuring the performance of communication middleware on high-speed networks,” *ACM Computer Communication Review*, Vol. 26, pp. 306–317, Oct. 1996.

[240] D. D’Souza and A. Wills, *Objects, Components, and Frameworks With UML: The Catalysis Approach*. Addison Wesley, 1998.

[241] M. Jung and E. W. Biersack, “A component-based architecture for software communication systems,” in *IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS) ECBS*, (Edinburgh, Scotland), Apr. 2000.

[242] Sun Microsystems, “Javabeans,” Java community process specification, Sun Microsystems, July 1997.

[243] D. Mennie and B. Pagurek, “An architecture to support dynamic composition of service components,” in *Proceedings of the 5th International Workshop on Component-Oriented Programming (WCOP 2000)*, (Sophia Antipolis, France), June 2000.

[244] M. Jackson and P. Zave, “Distributed feature composition: A virtual architecture for telecommunications services,” *IEEE Transactions on Software Engineering*, Aug. 1998.

[245] C. Gbaguidi, J.-P. Hubaux, G. Pacifici, and A. N. Tantawi, “An architecture for the integration of Internet and telecommunication services,” in *Proceedings Openarch 99*, (New York, NY), Mar. 1999.

[246] G. Bond, E. Cheung, A. Forrest, M. Jackson, H. Purdy, C. Ramming, and P. Zave, "DFC as the basis for ECLIPSE, an IP communications software platform," in *Proceedings of the IP Telecom Services Workshop 2000 (IPTS2000)*, (Atlanta, GA), Sept. 2000.

[247] B. Pagurek, J. Tang, T. White, and R. Glitho, "Management of advanced services in H.323 Internet protocol telephony," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[248] S. Gessler, O. Haase, and A. Schrader, "A service platform for Internet telephony," in *Proceedings of the 1st IP-Telephony Workshop (IPtel 2000)*, (Berlin, Germany), Apr. 2000.

[249] D. Rizzetto and C. Catania, "A voice over IP service architecture for integrated communications," *IEEE Network*, Vol. 13, pp. 34–41, May/June 1999.

[250] J. Rosenberg, J. Lennox, and H. Schulzrinne, "Programming Internet telephony services," *IEEE Network*, Vol. 13, pp. 42–49, May/June 1999.

[251] J. Rosenberg, J. Lennox, and H. Schulzrinne, "Programming Internet telephony services," Technical Report CUCS-010-99, Columbia University, New York, New York, Mar. 1999.

[252] N. Anerousis, R. Gopalakrishnan, C. R. Kalmanek, A. E. Kaplan, W. T. Marshall, P. P. Mishra, P. Z. Onufryk, K. K. Ramakrishnan, and C. J. Sreenan, "TOPS: an architecture for telephony over packet networks," *IEEE Journal on Selected Areas in Communications*, Vol. 17, pp. 91–108, Jan. 1999.

[253] N. Anerousis, R. Gopalakrishnan, C. R. Kalmanek, A. E. Kaplan, W. T. Marshall, P. P. Mishra, P. Z. Onufryk, K. K. Ramakrishnan, and C. J. Sreenan, "The TOPS architecture for signaling, directory services and transport for packet telephony," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, (Cambridge, England), pp. 41–53, July 1998.

[254] VoiceXML Forum, "Voice extensible markup language (VoiceXML) version 1.00," VoiceXML forum specification, VoiceXML Forum, Mar. 2000.

[255] G. Hellstrom, “RTP payload for text conversation,” Request for Comments 2793, Internet Engineering Task Force, May 2000.

[256] H. Alvestrand, “Tags for the identification of languages,” Request for Comments 1766, Internet Engineering Task Force, Mar. 1995.

[257] B. Campbell and R. Sparks, “Control of service context using SIP Request-URI,” Request for Comments 3087, Internet Engineering Task Force, Apr. 2001.

[258] M. Hamdi, O. Verscheure, I. Dalgi, J.-P. Hubaux, and P.Wang, “Voice service interworking for PSTN and IP networks,” white paper, 3Com, Santa Clara, California, Mar. 2000.

[259] G. Camarillo, “IP telephony gateways,” Master’s thesis, Royal Institute of Technology, Stockholm, Sweden, Nov. 1998.

[260] M. Hamdi, O. Verscheure, J.-P. Hubaux, I. Dalgic, and P. Wang, “Voice service interworking for PSTN and IP networks,” *IEEE Communications Magazine*, Vol. 27, pp. –, May 1999.

[261] S. Donovan and M. Cannon, “A functional description of a SIP-PSTN gateway,” Internet Draft, Internet Engineering Task Force, Nov. 1998. Work in progress.

[262] J. C. Tang, E. A. Isaacs, and M. Rua, “Supporting distributed groups with a montage of lightweight interactions,” in *Proc. of the Conference on Computer-Supported Cooperative Work (CSCW) ‘94*, (Chapel Hill, North Carolina), pp. 23–34, Oct. 1994.

[263] M. Day, S. Aggarwal, G. Mohr, and J. Vincent, “Instant messaging / presence protocol requirements,” Request for Comments 2779, Internet Engineering Task Force, Feb. 2000.

[264] J. Rosenberg *et al.*, “SIP extensions for presence,” Internet Draft, Internet Engineering Task Force, Apr. 2001. Work in progress.

[265] J. Rosenberg *et al.*, “SIP extensions for instant messaging,” Internet Draft, Internet Engineering Task Force, Apr. 2001. Work in progress.

[266] S. Simeonov, “What is this thing called SOAP? here’s the background,” *XML Journal*, Vol. 1, Sept. 2000.

[267] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, “SIP: Session initiation protocol,” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[268] H. Schulzrinne and J. Rosenberg, “SIP caller preferences and callee capabilities,” Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[269] H. J. Wang, A. D. Joseph, and R. H. Katz, “A signaling system using lightweight call sessions,” in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[270] D. Cohen, “The network voice conference protocol (NVCP).” NSC Note 113, Feb. 1976.

[271] N. Kausar and J. Crowcroft, “An architecture of conference control functions,” in *Proc. of Photonics East*, (Boston, Massachusetts), SPIE, Sept. 1999.

[272] E. Schooler and S. L. Casner, “An architecture for multimedia connection management,” in *Proc. of 4th IEEE ComSoc International Workshop on Multimedia Communications*, (Monterey, California), p. 5, Apr. 1992. also as ISI reprint ISI/RS-92-294.

[273] R. Sparks, “SIP call control,” Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[274] J. Rosenberg, H. Schulzrinne, and H. Sinnreich, “SIP enabled services to support the hearing impaired,” Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[275] J. Rosenberg and H. Schulzrinne, “The SIP supported header,” Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[276] A. Kristensen and A. Byttner, “The SIP servlet API,” Internet Draft, Internet Engineering Task Force, Sept. 1999. Work in progress.

[277] Sun Microsystems, “Java servlet API.” Available at http://java.sun.com/products/servlet/.

[278] H. Schulzrinne, “Providing emergency call services for SIP-based internet telephony,” Internet Draft, Internet Engineering Task Force, Mar. 2001. Work in progress.

[279] D. Rubenstein, S. Kasera, D. Towsley, and J. F. Kurose, “Improving reliable multicast using active parity encoding services (APES),” Tech. Rep. 98-79, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1998.

[280] S. Floyd and V. Jacobson, “The synchronization of periodic routing messages,” *IEEE/ACM Transactions on Networking*, Vol. 2, pp. 122–136, Apr. 1994.