Centralized Conferencing using SIP

Kundan Singh, Gautam Nair and Henning Schulzrinne
Columbia University
{kns10,gnair,hgs}@cs.columbia.edu

Abstract— **Multiparty conferencing is an important telephony service, provided in the PSTN by conference bridges. Internet telephony can enhance this basic service by video conferencing and collaborative work. The Session Initiation Protocol (SIP) can support many different conferencing architectures, including the centralized conferencing server model.**

We describe design issues and challenges in implementing a SIP-based centralized conferencing server and discuss the architecture and performance of our implementation, sipconf.

Keywords— **Centralized conference server; dial-in conference bridge; SIP; RTP mixer; packet audio; packet video; Internet telephony; sipconf**

I. Introduction

The Session Initiation Protocol (SIP) [1] defines how to establish, maintain and terminate Internet sessions including multimedia conferences. SIP supports various multi-party conferencing models [2], ranging from mixing in end systems to multicast conferences. When multicast is not available, centralized mixing, transcoding and filtering of media can be used to create multiparty conferences. In centralized mixing, a server receives media streams from all the participants in a conference, mixes or filters these based on pre-defined policy and distributes the streams to the participants. Different types of media streams need to be handled differently, for example, audio streams are typically summed, while video streams are selected, e.g., to present only the active speaker.

The main functions of a conference server is the mixing and redistribution of media streams. Typically, Internet audio streams are added ("mixed"), while video streams and other media are simply replicated. However, a video mixer can also create a new composite video image [3]. For audio, the server needs to ensure that a participant does not receive a copy of his own media in the mixed stream. RTP [4] allows a sender to indicate which sources have been combined in a single media packet. When summing, the server should absorb the jitter in packet arrival times while introducing minimum delay ("playout buffer").

For replication, the server should not need to be aware of the media formats. The RTP SSRC indication [5] ensures that the receiver can distinguish different sources addressed to the same network destination.

For either summing or replication, it is desirable if each participant can use different media types and packetization intervals, to accommodate heterogeneity of end systems and access bandwidths. Implementations need to scale to large numbers of conferences as well as large numbers of participants per conference.

A media mixing module with a SIP interface can act as a conferencing server component in the distributed application server (AS) component architecture [6]. Advanced system can bundle this functionality with other services, such as interactive voice response (IVR) and a web-based user interface.

This paper explores the centralized conference server design issues in detail and describes challenges in implementing such a system. We also explore advanced usage scenarios of the conferencing system in a real-world Internet telephony environment. We are currently collecting performance data on our implementation of the SIP based conference server, sipconf[1], and present some initial results of our experiments.

A. Outline of the rest of the paper

Section II explains the role of SIP in centralized conferencing systems. Section III discusses and compares different conferencing models. Design issues are described in Section IV. We provide an overview of our implementation and performance figures in Section V. The usage scenarios in various Internet telephony and multimedia communication applications are discussed in Section VI. Section VII lists some of the related work. Finally, we summarize and point to future work in Section VIII.

II. Background

Many PSTN carriers offer conference bridges which allow users to take part in a voice conference by dialing a telephone number and possibly access code. We can use the same concept for Internet-based conferencing: The conference can be identified by a destination address, and participants can join the conference by making a call to that address, thus requiring no modifications in end systems. There are currently two Internet telephony signaling protocols, IETF's SIP and ITU-T's H.323 [7]. SIP identifies the destination via a SIP URI of the form *sip:user@domain*, while H.323 uses AliasAddress data structures, which can assume many forms, including URLs.

There are two different aspects of Internet based conferencing, signaling and media. Either SIP or H.323 can be used as a signaling protocol for taking part in a conference. Both SIP and H.323 use the Real-time Transport Protocol (RTP [5]) for carrying real-time media traffic, such as audio and video. H.323 defines a multi-point control unit (MCU) for handling multiparty conferences. An MCU consists of a multi-point controller (MC), which can also be part of a terminal, to handle signaling and control exchanges with every participant in the conference. An optional component, the multi-point processor (MP), handles mixing and filtering of different media streams. SIP does not define any conferencing entity as such, as these entities are easily modeled as SIP user agents. The core SIP specification supports a variety of conferencing models [2]. In the server-based models, RTP media streams are mixed or filtered by the server and distributed to the participants. There is a standard point-to-point signaling relationship between each participant and the conferencing server.

[1] More information at http://www.cs.columbia.edu/~kns10/software/sipconf

The conference is identified by the SIP URI, e.g., `sip:discuss@server.com`. The standard user location and routing mechanisms in SIP forward all calls to the appropriate conference server at `server.com` without requiring any extension to the protocol. The SIP message routing entities (SIP proxies) need not be aware that the request URI corresponds to a conference and not to an individual person.

The Session Description Protocol (SDP [8]) is used to indicate media capabilities and media transport addresses. The participant sends the information about his media capabilities (PCMU) and the transport address where he wishes to receive RTP packets. In the message body of the 200 success response, the server sends the transport address to which the participant should send his PCMU RTP packets. More advanced scenarios can be accomplished using the SIP REFER method. For example, an existing participant can invite another user to join the conference. These conferencing models can be found in [2].

SIP-based authentication can be used to prevent unauthorized participants to join a conference. The server can support both pre-arranged conferences as well as ad-hoc conferences by assigning special meaning to the user field in the request URI. For example, participants who wish to join `sip:ietf.arranged@office.com` will need to set up the conference before hand, while those who wish to join `sip:library-discuss.adhoc@office.com` do not need to setup the conference in advance. The conference state is maintained as long as at least one participant is part of the conference. Participants find out about the conference URL via external means, such as email or a web page.

III. Conferencing models

Conference models can be distinguished based on the topology of signaling and media relationships. Conferences with a central server are easier to handle for end systems and simplify keeping track of the conference participants. On the other hand, network-layer multicast is more scalable for large-scale media distribution and allows a "loose" model of conference membership [9], where each member has only an approximate view of the group roster.

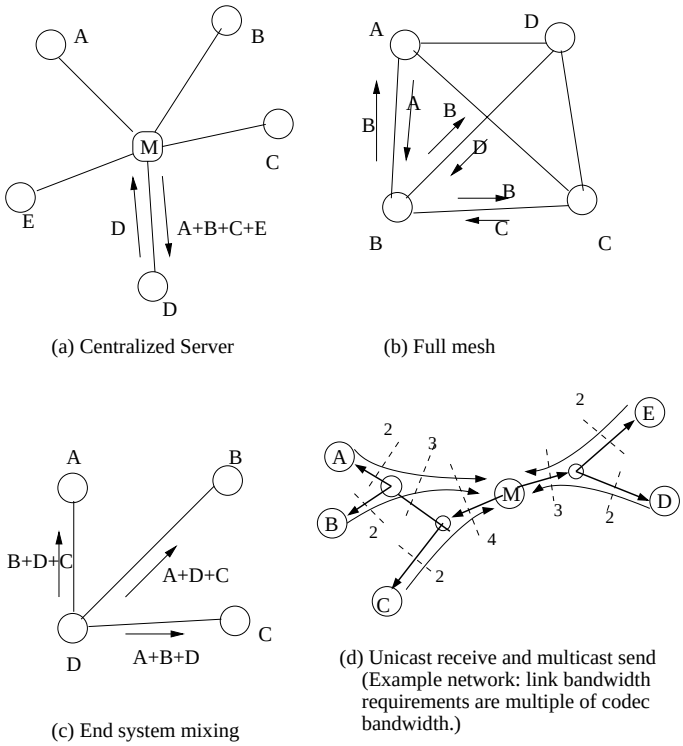
Table I summarizes the different types of *media distribution models* in multimedia conferencing. The table compares the scaling properties, depending on the the number of active senders, M, and the total number of participants, N. Given that M is almost always one for typical audio conferences, most of these models scale similarly in terms of processing and bandwidth requirements. Note that the centralized model performs better with higher M if inputs are summed.

Centralized: In the centralized model, a server receives media streams from all participants, mixes them if needed, and redistributes the appropriate media stream back to the participants (See Fig. 1). Since senders would have difficulty subtracting out their own contribution, the server needs to create a customized stream for each of the currently active M senders and a common stream for all $N - M$ listeners, assuming that they can all support the same media format. The server needs to decode audio streams before mixing, as mixing can only be performed on uncompressed audio. Decoding M and encoding $M + 1$ streams limits the amount of active sources or conferences, while the number of participants limits the total conference membership to the available outbound network bandwidth.

The central server model has the advantage that clients do not need to be modified and do not have to perform media summing. In addition, it is relatively easy to support heterogeneous media clients, with the server performing the transcoding. For example, this allows a conference consisting of participants connected through high-bandwidth networks and modems, each receiving the best possible quality. At the cost of increased inbound bandwidth, silence detection can be delegated from clients to the server. This is helpful as many current IP telephones do not support silence suppression.

Also, the server can enforce floor control policies and can control the distribution of video based on audio activity. Compared to a distributed model, a central server can readily provide a consistent view of the complete conference membership.

Full mesh: In a full mesh, each active participant sends a copy of its media stream to all participants via unicast, without a central server. End systems sum the incoming audio streams; since most of the time, only one speaker will be active, the CPU overhead is modest as long as silence suppression is implemented everywhere, but it fails if the access bandwidth of some participants is just large enough for a single stream. For video, full mesh does not scale unless, for example, only currently active speakers send video. In a full mesh, each pair of participants must share a common codec.



(a) Centralized Server

(b) Full mesh

(c) End system mixing

(d) Unicast receive and multicast send (Example network: link bandwidth requirements are multiple of codec bandwidth.)

Fig. 1. Types of media distribution model

Properties	centralized	full mesh	multicast	unicast rx, multicast tx	end mixing
Topology	Star	full mesh	m-cast tree	star and m-cast tree	ad-hoc
Server processing	O(M+N)	n/a	n/a	O(M+N)	n/a
Endpoint processing	O(1)	O(M)	O(M)	O(1)	variable
Server bandwidth	O(M+N)	n/a	n/a	O(M) based on m-cast tree	n/a
Endpoint bandwidth	O(1)	O(M)	O(1)	O(1)	variable
Scaling	medium	medium	large	large	medium
Heterogeneous endpoints	yes	yes	no	no	yes (partially)
Get back your media	no	no	no	yes	no

TABLE I

TYPES OF CONFERENCES; M IS THE NUMBER OF ACTIVE SENDERS AND N THE TOTAL NUMBER OF PARTICIPANTS

Multicast: Network-layer multicast is ideally suited for large-scale conferences. A multicast address is allocated for each media stream, and every participant sends to that address. As in the full mesh, participants receive packets on the same address from all other participants, and need to sum or select streams. While the incoming bandwidth is the same as in a full mesh, each system only needs to generate one copy of the media stream.

Unfortunately, native multicast is not widely available outside network testbeds such as Internet2. Also, all receivers must share a common set of codecs.

Unicast receive and multicast send: This scheme combines some of the benefits of the server and multicast models. Participants send their media stream using unicast to the conferencing server, which sums them and sends them out on a pre-established multicast address. Thus, unlike pure multicast, end systems do not have to filter or mix media streams. Every participant receives the mixed stream, which includes his own stream. Unless a sender maintain a buffer of the data sent and there is a means of aligning time scales, it will have difficulty removing its own audio content from the mixed stream. The gain in bandwidth efficiency is largest if the number of simultaneous senders is small compared to the total group size. This approach lends itself well to single-source multicast [10], [11].

Endpoint mixing: Instead of in a server, mixing can take place in one of the participating end systems. For example, if A and B are in a call, A can also invite C. A sends the sum of A and B to C, and the sum of A and C to B. B and C do not need to be aware of the service performed by A, but can in turn mix other participants.

Cascading mixers increases the delay on some of the media paths. Another problem is that the conference dissolves when the participant who is acting as a mixer leaves the conference. This model is likely to be suitable only for small conferences of three or four parties.

Besides these, one can imagine a replication model, where the server sends a copy of each incoming media stream to all the participants using unicast. The mixing is done at each end system. This might be useful for media path authentication as every end system exchanges media packets only with the server's IP address. The CPU overhead is modest as long as silence suppression is implemented. The server however is less loaded than in the case of the centralized conference since it is now freed from the task of mixing audio streams. This is the model used in the case of video and text based conferences, since there is inherently no mixing required.

Media and signaling can use different models in the same conference. For example, one could combine centralized signaling with multicast media distribution, where the server maintains a one-to-one signaling relationship with each of the participants. Unfortunately, this requires cooperation from the end system. The server can indicate a multicast address in its SIP success response, causing the end system to send media streams via multicast, but the end system will still expect to receive media via unicast. More sophisticated session description formats may address this issue.

Also, different media streams can use different models. For example, audio could be mixed by a central server and redistributed, while video can be sent point-to-point between every pair of participants as in full mesh.

Thus, as long as multicast is not widely available, server-based conferences will continue to be the only viable model for mid-size conferences of tens to hundreds of participants.

IV. DESIGN OF A CONFERENCE SERVER

A conferencing server consists of a signaling and a media mixing module. The signaling module receives SIP or H.323 requests to join and leave conferences, while the media mixing module receives and sends RTP media streams from and to participants. Replicating video packets is straightforward; below we describe the operations needed for mixing audio.

A. Audio mixing

Fig. 2 shows how an audio mixing module can be implemented. Participant A support G.711, B DVI ADPCM and C both GSM and G.711. Participants list the codecs they support in their INVITE requests. The server selects an intersection of the algorithms supported by the participant as well as by the server. This selection is returned in the signaling success response to the participant. These algorithms are listed in order of preference in the SDP of the INVITE or its response.

The mixing algorithm follows a *decode-mix-encode* sequence. When an audio packet arrives at the mixing module, it is decoded into 16-bit linear samples and appended to the per-

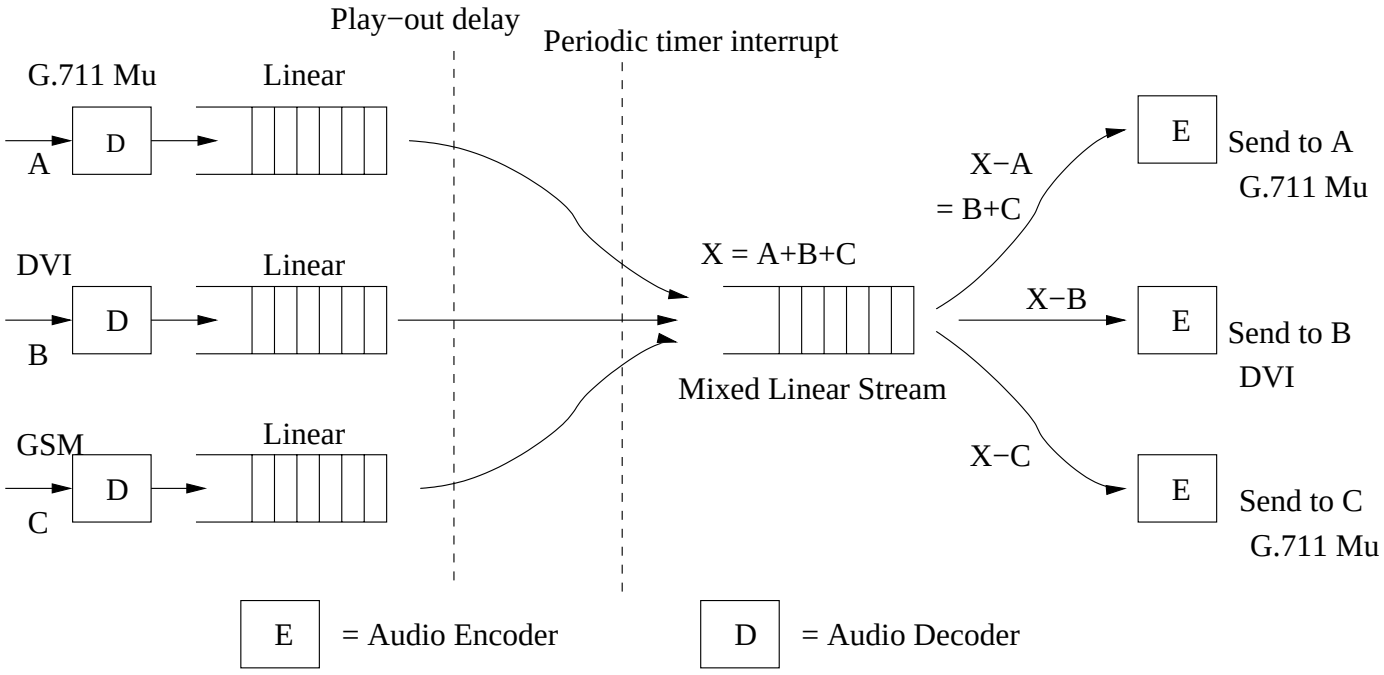


Fig. 2. Audio mixing

participant audio buffer queue. Each buffer is labeled with the corresponding RTP timestamp. The jitter in packet arrivals is absorbed by a play-out delay algorithm. Every outbound packetization interval, a timer triggers a routine that mixes a range of the samples from one of more input buffers from each active participant into a combined packets by simple addition of the sample values. The timer intervals are shortened and lengthened to account for earlier slight variations in timer invocation times and processing delay. (A simple operating system timer that fires after a delay would yield an output rate that is typically slightly below the desired rate.)

To allow input and output packets to have different packetization intervals, the mixer routine can grab samples from one or more input buffers. (Using a chained list of buffers saves memory compared to a circular buffer and makes it easier to detect when a particular source is silent.) Then, for each of the participants, the linear sample values from the per-participant queue (e.g., A) is subtracted from the mixed data (X) and the resulting data ($X - A$) is encoded using the preferred audio compression algorithm. The encoded data is packetized and sent to the participant. If there are M participants, then both mixing and redistribution will take M additions and M subtractions. Note that the receive and transmit audio algorithms need not be same for each participant.

While the *decode-mix-encode* sequence is the most straightforward approach to implementing an audio mixer, there are alternative approaches. For instance, one can build an addition or subtraction table for G.711 samples, so that conversion to linear is not required to do mixing. This only works for G.711, not for codecs with cross-sample dependencies such as G.723.1 or GSM.

Also, instead of subtraction, one could create $M + 1$ different streams directly, one for each talker and one for the listeners. However, that requires M^2 additions.

B. Playout delay algorithm

Playout delay algorithms help absorb the jitter in network packet arrival due to network congestion. Adaptive playout delay further allows an application to adapt to changes in the amount of jitter, thus giving minimum delay in the audio stream. Playout delay compensation takes place before mixing, stretching or shrinking silence periods between talkspurts to adjust the time between arrival and mixing [12], [13]. (In the absence of silence periods, time stretching or companding can be used, albeit at much greater computational cost.) We have used Algorithm 1 from [12], with $\alpha = 0.95$, for our implementation. The algorithm is basically a linear recursive filter. The adapted delay at any instant depends on the measured delay (using RTP timestamps) plus the previous adapted delay, with a weighting factor α. The playout delay depends on both the adapted delay and the variation in the adapted delay.

V. Implementation

We have implemented a simple SIP conference server based on the above design. It can support some of the common audio algorithms, including G.711 A and μ-law, DVI ADPCM, and GSM. The Columbia SIP C++ library is used for all the SIP and SDP related functionality in the conference server. When a SIP user agent connects to the server the signaling is managed by the routines in this library. The mixer module forms the core of the conference server. We use Columbia RTP Library for implementing the RTP functionality. It sets up and manages the audio transport with the participants. There is a thread for receiving packets from every participant. Another thread (per conference) takes care of sending the mixed stream to the participants.

Below, we discuss design and implementation issues and

present initial performance data.

A. Design issues

Packetization interval: Although RTP implementations are supposed to handle a wide range of packetization intervals, we found 20 ms to be the only one that worked across a range of media clients such as rat [14] or Microsoft NetMeeting. End systems permitting, it may be useful to dynamically change the packetization interval for outgoing packets, as smaller packetization intervals decrease delay, but increase network bandwidth and computational effort.

Scaling: For large conferences, scalability is limited primarily by outbound bandwidth, copying of data between buffers and encoding. If many smaller conferences are to be supported, scaling depends as much on inbound bandwidth and decoding. While simple codecs like G.711 require very little encoding and decoding effort, they impose a heavier burden on buffer copying and bandwidth.

To scale to very large conferences using conferencing servers, a network of servers can be deployed (Section VI). To scale to a large number of smaller conferences, a SIP proxy server can act as a load-distribution system and direct incoming requests for new ad-hoc conferences to different servers. Alternatively, the conference server itself can redirect a request to an alternate server.

Instead of using general-purpose computers, one could also build DSP-based customized hardware at lower per-port cost. However, in many environments, there are enough idle cycles on workstations and servers that can be drafted into service for occasional large conferences.

Inactivity detection: The system should be able to detect if a particular participant becomes inactive, e.g., due to user agent failure. Failures can be detected by observing ICMP errors or sudden discontinuation of RTCP reports.

B. Performance measurements

We are currently measuring performance of our software on a range of platforms. Initial results are below. We characterize server load by processor and memory utilization. As discussed above, both the number of conferences and the total number of participants affect load, assuming that the average number of active senders per conference is one.

Table II summarizes the server load depending on how many simultaneous participants are present in a single conference. There was no optimization done at compilation time. There were only one or two active speakers and all others were listeners. The server was running on a Sun SPARC Ultra 10 with 256 MB RAM and a 360 MHz CPU. All participants were in the same 100 Mb/s LAN as the server and used G.711 with a 40 ms packetization interval from server to participant and 20 ms from participant to server. The bandwidth includes IP, UDP and RTP headers, and for a typical 100 Mb/s LAN, is not a limiting factor.

Load figures are obtained using the Unix command `top`. Memory is the amount of resident memory (RES). The audio quality was good up to 80 participants in the single conference, tolerable with 100 participants and very poor for 120 participants.

Participants	CPU (%)	memory (MB)	bandwidth (Mb/s) inbound	outbound
2	< 0.1	2.7	0.08	0.07
20	< 1	6.0	0.08	1.37
40	2-3	9.6	0.08	2.81
60	5	13	0.08	4.25
80	10-15	17	0.08	5.69
100	35-50	22	0.08	7.13
120	50-70	26	0.08	8.59

TABLE II
SERVER LOAD AS FUNCTION OF NUMBER OF PARTICIPANTS IN SINGLE CONFERENCE

Table III summarizes the server behavior depending on the number of simultaneous three-party conferences where every participant is an active speaker. All other parameters are the same as before. Audio quality was good up to 15 three-party conferences, but deteriorated to poor with 18 conferences.

Conferences	participants	CPU (%)	memory (MB)	bandwidth (Mb/s) inbound	outbound
3	9	< 0.4	4.1	0.72	0.65
6	18	< 2.0	5.7	1.44	1.30
9	27	7-13	7.3	2.16	1.94
12	36	15-20	9	2.88	2.60
15	45	25	10	3.60	3.24
18	54	30	12	4.32	3.89

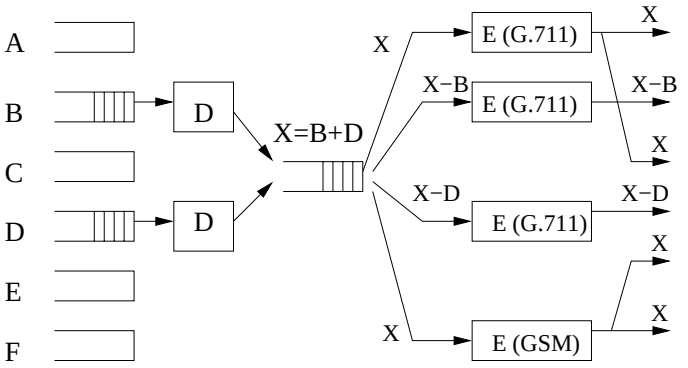
TABLE III
SERVER LOAD AS FUNCTION OF NUMBER OF THREE-PARTY CONFERENCES

The memory requirement depends on the number of participants and seems to increase linearly. For instance, memory requirements for 15 three-party conferences (45 participants) is almost the same as that for 40 participants in a single conference. Secondly, the CPU utilization starts increasing drastically at about 30-40 participants.

CPU load is the primary bottleneck in our test environment. The other factors: memory and bandwidth do not seem to cause problem for a hundred participants. Various factors contribute to the CPU load, e.g., thread switching and list traversal. This bottleneck can be removed by using "Multi-state conferences" (Section VI) or by dedicated hardware for encoding and mixing, for example.

It may be possible to optimize the mixing logic. One such scheme is shown in Fig. 3, combining the encoding step for the output streams that have same mixed audio data and use the same encoding algorithm. For all the participants who did not speak in the last timer interval and who have a common subset of supported receive audio algorithm, we can call the encoder only once. However, if a stream stops being active, it will receive the general listener packet stream rather than its own version, so that the predictor will be wrong. It is not clear how much this would matter in practice.

Scaling may also be limited by the available number of



A−D support G.711; E and F support GSM.

Fig. 3. Possible optimization in decode-mix-encode sequence

threads. Our implementation allocates a thread for every conference as well as for every participant. With 1000 threads allowed per process, the server can support 250 three party conferences (with 750 participants), for example.

VI. Conferencing as part of an overall VoIP architecture

This section describes enhancements to the simple centralized conference system and how it can fit into a more complex Internet telephony and multimedia communication environment.

A. Multi-protocol conference server

A simple enhancement (Fig. 4) is to use a SIP-H.323 gateway and SIP-PSTN gateway to provide a unified conferencing server which can be contacted from any of the SIP, H.323 or PSTN networks. To integrate PSTN users, some form of interactive voice response (IVR) is required, e.g., to prompt for pass codes.

B. Network of conference servers

For larger conferences, it is possible to create a tree of conference servers, where each server appears as a participant in the server at the aggregation level above it (Fig 5). In the figure, S2, S3, and S4 act as participants for the conference server S1. Such a tree adds packetization and playout delay, but can approximate the bandwidth scaling benefits of network-layer multicast if participants select the closest server. Since it is common that corporate conferences consist of a large number of participants spread across a relatively small number of facilities, having a server in each LAN is likely to be a common mode of operation.

C. Integration with other services

A conferencing server can be integrated with a text-to-speech and speech recognition system to allow text-only participants in an audio session. A conference server could also include an RTSP client that can stream media to a recording server.

VII. Related work

Most of the conference servers in the market today are based on H.323. These include MeetingPoint from CUseeMe Networks, Sametime from Lotus and Microsoft Exchange 2000

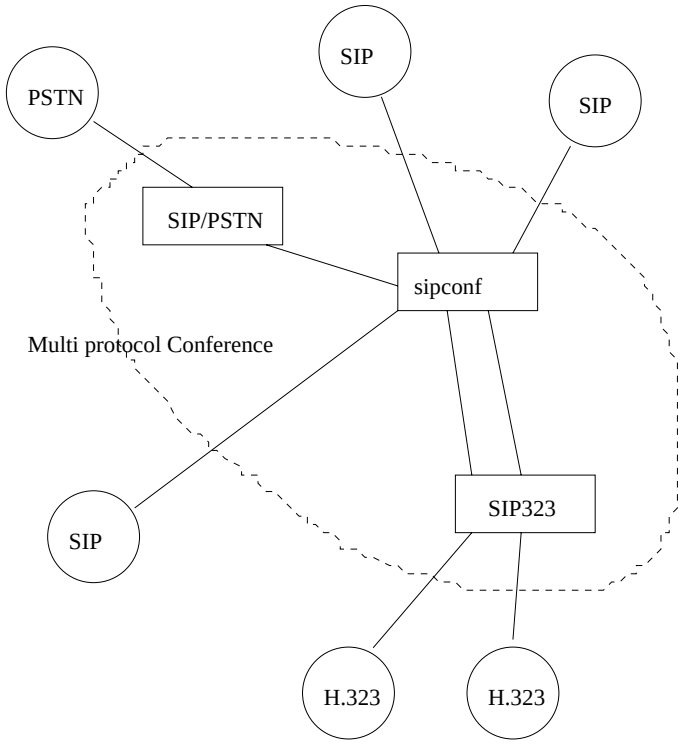


Fig. 4. Multi-protocol conference server

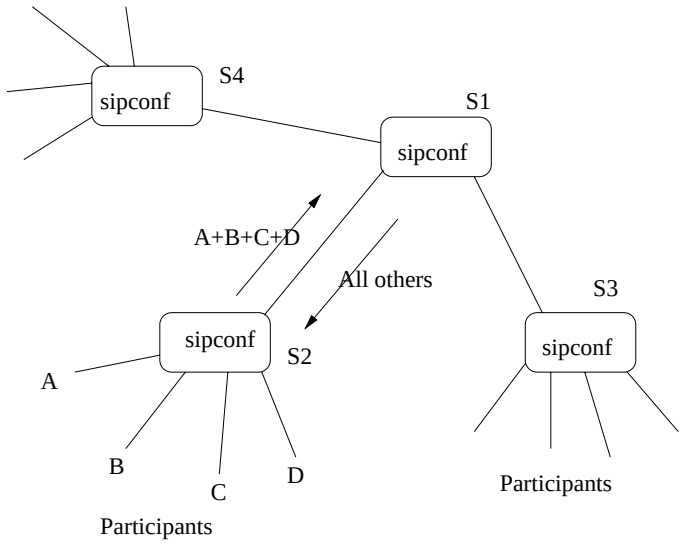


Fig. 5. Multi-stage conference servers

Conferencing Server. These support T.120 for application sharing and whiteboards. MeetingPoint has mechanisms to link servers together so that conferences can be shared and load-balancing can be done.

VideoTalks [15] by AT&T Labs is a comprehensive multimedia conferencing system intended to provide a variety of Internet services such as video conferencing and low cost video-on-demand. It is not based on SIP.

A number of tools (e.g., RAT and NeVoT) support multicast "light-weight" conferencing, without explicit signaling support [9]. Etherphone [16] is probably one of the earliest systems supporting multimedia conferencing.

Most of these work talk about conferencing in general or a specific implementation of a conferencing system. Here, we compare different models and present performance numbers for a real implementation.

VIII. Conclusion and future work

Based on our implementation, SIP provides a suitable multimedia conferencing platform that allows advanced scenarios and services without requiring that end systems are conferencing-aware. It is possible to build medium scale conferencing servers in software. Our implementation supports up to 100 participants in a single conference using G.711 audio and only one active speaker on a Sun Sparc Ultra-10 platform. It can also support up to 15 three party conferences where all participants are speaking simultaneously. Scalability can be improved by ensuring that the clients support silence suppression at their end.

In addition to audio and video conferences, various other services can be provided at the conference server such as whiteboard applications and multi-user games. This may lead to a distributed conferencing server architecture with different components handling different services. Participants can also join conferences from the PSTN if they use SIP to PSTN gateways. SIP-H.323 gateways [17] exist that can permit the participation of H.323 clients in the conference.

We plan to enhance our prototype in a number of ways:

- We need to gather performance data for different codecs in a heterogeneous conferencing environment, on different computing platforms, including those with multiple processors. We also plan to measure the delay and jitter at the client side as the server load increases.
- Video support will be added to the server. This involves defining policies on which video stream to distribute and possibly merging streams into screen quadrants. (With windows-based end systems, this is not required as the end system can arrange multiple video windows.)
- We plan to add additional codecs, beyond the G.711 μ-law, G.711 A-law, GSM and DVI ADPCM currently supported.
- Additional SIP features such as dial-out and authentication can be added to the server. This allows the server to invite participants to the conference and keep unauthorized participants out. Conferences could also be bounded in duration; however, since the resource consumption of inactive conferences is very small as long as media streams are muted, it is quite feasible to set up permanent conferences in work groups, for hoot-and-holler applications. The transition from centralized conferences to full-mesh and multicast conferences, as well as hybrid solutions, need to be supported.

IX. Acknowledgments

We would like to thank Jonathan Rosenberg and Weibin Zhao for their valuable and insightful comments.

References

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[2] J. Rosenberg and H. Schulzrinne, "Models for multi party conferencing in SIP," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[3] Z.-Y. Shae and M.-S. Chen, "Mixing and playback of JPEG compressed packet videos," in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, (Orlando, Florida), pp. 245–249 (08B.03), IEEE, Dec. 1992.

[4] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," Request for Comments 1890, Internet Engineering Task Force, Jan. 1996.

[5] 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.

[6] J. Rosenberg, P. Mataga, and H. Schulzrinne, "An application server component architecture for SIP," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.

[7] International Telecommunication Union, "Packet based multimedia communication systems," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[8] M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments 2327, Internet Engineering Task Force, Apr. 1998.

[9] M. Handley, J. Crowcroft, C. Bormann, and J. Ott, "The internet multimedia conferencing architecture," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[10] S. Bhattacharyya *et al.*, "A framework for source-specific IP multicast deployment," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[11] H. W. Holbrook and D. R. Cheriton, "Ip multicast channels: EXPRESS support for large-scale single-source applications," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), August/September 1999.

[12] 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.

[13] 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.

[14] A. Sasse, V. Hardman, I. Kouvelas, C. Perkins, O. Hodson, A. Watson, M. Handley, J. Crowcroft, D. Harris, A. Bouch, M. Iken, K. Hasler, S. Varakliotis, and D. Miras, "Rat (robust-audio tool)," 1995.

[15] M. R. Civanlar, G. L. Cash, R. V. Kollarits, B.-B. Paul, C. T. Swain, B. G. Haskell, and D. A. Kapilow, "VideoTalks: A comprehensive multimedia conferencing system," in *Proc. of Packet Video*, (Sardinia, Italy), May 2000.

[16] H. M. Vin, P. T. Zellweger, D. C. Swinehart, and P. V. Rangan, "Multimedia conferencing in the Etherphone environment," *IEEE Computer*, vol. 24, pp. 69–79, Aug. 1991.

[17] K. Singh and H. Schulzrinne, "Interworking between SIP/SDP and H.323," in *Proceedings of the 1st IP-Telephony Workshop (IPtel 2000)*, (Berlin, Germany), Apr. 2000.