Bandwidth-Conserving Multicast VoIP Teleconference System

Teck-Kuen Chua and David C. Pheanis
(Corresponding author: Teck-Kuen Chua)

Computer Science and Engineering, Ira A. Fulton School of Engineering
Arizona State University, Tempe, Arizona 85287-8809, USA (Email: TeckKuen.Chua@gmail.com)

(Received July 15, 2006; revised and accepted Oct. 6, 2006)

Abstract

Teleconferencing is an essential feature in any business telephone system. A teleconference allows associates to engage in a group discussion by conducting a virtual meeting while remaining at geographically dispersed locations. Teleconferencing increases productivity while reducing travel costs and saving travel time. In a VoIP telephone system, we face the significant challenge of providing a teleconference feature that can support a large-scale teleconference without using excessive bandwidth. This paper presents a new, bandwidth-efficient way of implementing a real-time VoIP teleconference system. This new method provides all of the features that existing teleconference systems provide, but this new approach consumes considerably less data bandwidth than existing systems require. The new system allows a network with a given capacity to accommodate almost double the number of conference participants that an existing system would allow.

Keywords: Audio mixing, bandwidth conservation, business telephone system, IP *Multicast, teleconference system,* VoIP

1 Teleconference Background

An audio teleconference feature with current technology sums or mixes the audio inputs from all of the conference participants to produce a mixed audio stream that contains the audio from everyone in the conference. Hearing a delayed echo of your own speech, though, is unsettling, so the system removes the audio input of an individual participant from the mixed audio to produce the audio that this particular person will hear. The teleconference system transmits that audio to that particular participant. For example, suppose we have a conference with four participants, A, B, C, and D. Let's refer to the audio input stream from A as a, B as b, C as c, and D as d. The conference system generates four different audio streams, one for each participant. Participant A receives audio stream $(b+c+d)$, B receives $(a+c+d)$, C receives $(a+b+d)$, and D receives $(a+b+c)$. The conference system transmits these four different audio streams separately to four different participants.

In a teleconference server application, the server commonly supports a large number of participants in a single conference, and having a large number of participants in a conference introduces an additional challenge. The teleconference server can no longer simply sum the audio of all the participants. Summing the audio of a large number of participants could cause *overflow* in the audio, thus distorting the audio and degrading the audio quality. Even if most of the participants are silent, summing the background noise of a large number of participants produces a loud background noise in the mixed audio.

To solve this problem, a teleconference server that supports large-scale conferences typically incorporates some mechanism to select the audio from only a few *active* (i.e., talking) participants for the mixing process. For example, suppose we have 26 participants, A, B, C, $\cdots$, and Z. Let's use the same naming scheme that we used earlier to name the audio of each participant. If the teleconference server selects participants A, B, and C as active participants for audio mixing, the teleconference system generates four different audio streams, $(b+c)$ for A, $(a+c)$ for B, $(a+b)$ for C, and $(a+b+c)$ for all of the idle (i.e., *listening* but nontalking) participants. The conference system transmits these audio streams separately (i.e., 26 transmissions) to the 26 participants.

This paper presents a new, bandwidth-efficient way of implementing a real-time VoIP teleconference system[1]. This new method provides all of the features that existing teleconference systems provide, but this new approach consumes considerably less data bandwidth than existing systems require. The new system allows a network with a given capacity to accommodate almost *double* the number of conference participants that an existing system would allow.

We start by describing several existing techniques for

[1] Patent pending

implementing teleconferences, and then we explain our new method. We discuss the functions that the conference server performs with our new approach, and we also describe the tasks that the endpoints perform. We explain details such as our method for handling mixed audio when the server modifies the source audio. Finally, we summarize the advantages that our new method provides.

2 Existing Techniques

Teleconferencing is a common and very useful feature, so various designers have implemented this feature using different approaches [1, 2, 3, 8, 9, 10, 11, 12, 13, 14]. However, none of the existing methods can support a large-scale VoIP teleconference in a bandwidth-efficient manner. In this section we explain existing techniques.

2.1 Peer-to-Peer Unicast

In an ad-hoc peer-to-peer teleconference implementation, each participant uses unicast to transmit his or her own audio to all other members of the conference. In an n-party conference, every member generates $(n-1)$ unicast transmissions to send audio to $(n–1)$ other members. Each participant receives $(n–1)$ unicast audio streams from the other $(n-1)$ members, and each participant uses some well-known method to mix the received audio streams before playing back the mixed audio. Since people do not send their own audio streams to themselves, there is no feedback or echo problem with this implementation.

Unfortunately, this approach is very expensive in terms of bandwidth. Each participant establishes a bidirectional unicast link with each other participant, so an n-party conference requires $[(n^2 - n)/2]$ bidirectional links for a total of $(n^2 - n)$ data streams. In a ten-party conference, for example, there are 90 unidirectional data streams. If another person joins the ten-party conference, the new participant has to establish ten new bidirectional unicast links, making 20 new data streams, to communicate with all of the existing members.

Besides consuming a high degree of bandwidth, this method requires a significant amount of CPU processing capability to process and decode the audio streams from a large number of participants in a large-scale conference. Generally, telecommunication endpoints in business telecommunication systems are embedded systems with limited resources and are not capable of intense processing tasks. An endpoint would require processing capability equivalent to the power of a server in order to handle a large number of audio streams simultaneously, and this requirement would make the implementation too expensive to be a practical business solution.

Peer-to-peer unicast is certainly feasible for small teleconferences involving only three or four participants, but the technique does not scale well and quickly becomes impractical as the number of participants grows.

2.2 Peer-to-Peer Multicast

RFC-3550 recommends an ad-hoc peer-to-peer teleconference implementation that exploits IP multicast [4]. Instead of establishing a unicast link with each of the other members of the conference, the participant uses only one multicast transmission to deliver his or her audio to all other members of the conference. Every member of the conference listens to the multicast transmissions of all other participants. An n-party conference requires only n multicast transmissions, so this approach significantly reduces bandwidth consumption in comparison to the peer-to-peer unicast method.

Since each incoming audio stream arrives as an individual audio stream, a participant can eliminate the problem of audio feedback or echo by simply not processing the participant's own audio stream. However, the participant must process and decode all other incoming audio streams. In a large-scale conference, each participating endpoint would require a substantial amount of processing power to process and handle a large number of incoming audio streams. Since an endpoint in a business telecommunication system is normally an inexpensive embedded system with limited resources, peer-to-peer multicast is impractical for large conferences.

2.3 Server-based Unicast

A designer can use a server-based teleconference system to overcome the limitation on the size of a conference. A server-based system has the processing capacity to handle and process a large number of audio streams simultaneously. The teleconference server receives a unicast audio stream from each participant and mixes the audio streams. The server removes the audio of an active talker from the mixed audio before sending the mixed audio to that particular talker, so there are several different versions of mixed audio. The server uses unicast to transmit these mixed audio streams to the appropriate participants.

This approach requires only $2n$ unidirectional unicast links for an n-party conference, so it uses considerably less network bandwidth than the peer-to-peer unicast technique, especially for a large conference. However, this method still consumes a significant amount of bandwidth when the conference size gets large. The server cannot multicast a single mixed audio stream that contains the audio from all of the active participants to everyone in the conference because the talking participants would hear their own audio. This audio reflection would create an annoying echo effect that would be highly disruptive to the talking participants. Furthermore, the server cannot send a multicast audio stream even to all of the idle (i.e., listening but nontalking) participants because the talkers in the conference change dynamically from moment to moment.

3 Our Method

Our new method is an improvement of the server-based unicast technique. The teleconference server in our system still receives one input audio stream from each participant just as the server does with the server-based unicast method. As with a server-based unicast, the teleconference server in our system can use any known selection mechanism to pick active participants when mixing a large number of participants in a conference. However, the teleconference server with our new method generates only *one* mixed audio output, and the server transmits that single audio stream in one multicast transmission to distribute the same mixed output audio to all of the participants. We therefore use a *single multicast* transmission to replace the n unicast transmissions of the server-based unicast.

Our new method requires the cooperation of the endpoints in the conference. Along with the mixed audio output in the multicast transmission, the teleconference server includes auxiliary information. This added information allows each participating endpoint to process the mixed audio output, if necessary, to make the mixed audio suitable for playback at that particular endpoint. The endpoint stores critical data, including a history of the audio that the endpoint has recently transmitted to the server, and the endpoint later uses this data to adjust the mixed audio from the server for playback. In simplified terms, each active endpoint removes its own audio from the mixed audio to eliminate echo.

Figure 1 illustrates our multicast teleconference system with three active participants, A, B, and C, and one passive participant, D. The server mixes the audio streams from the active participants and sends the resulting mixed audio to everyone with a single multicast output, M.

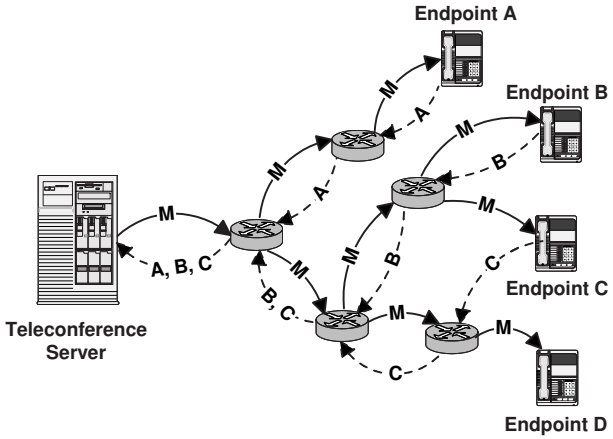

Figure 1: IP multicast teleconference system

4 Teleconference Server

Figure 2 illustrates the implementation and various components of the teleconference server for our approach with three participants in the conference.

With our new technique, the teleconference server transfers some responsibilities to the participating endpoints but also assumes some new responsibilities of its own. Generally, a traditional teleconference server performs the individual participant's audio removal from the mixed audio and generates multiple mixed audio streams that are ready for playback at each of the participating endpoints. With our approach, the server simply mixes the audio sources from all of the endpoints that the server selects as active contributors, and the server generates just one mixed audio stream for all participants. This approach obviously reduces the workload on the server in addition to reducing the consumption of network bandwidth.

The server assumes some new responsibilities that are necessary to let each of the active participating endpoints remove its own audio from the mixed audio. The following sections detail these new server responsibilities.

4.1 Disclosing Media Information

Since each active endpoint must remove its own audio from the mixed audio with our system, the server must disclose information that was not available with previous protocols and techniques. If the server uses a selection mechanism to choose a few active participants as contributors to the mixed audio, the server must reveal the identities of the participants who are involved in the mixing process. Since different participants might be active in different segments of the mixed audio, the mixer has to disclose information regarding the active participants in every segment of the mixed audio. This information tells each participating endpoint whether its own audio is part of the mix, thus allowing the endpoint to remove its own audio from the mixed audio — only if appropriate — before playback.

Note that the server may make modifications (e.g., scaling to avoid overflow) while mixing inputs. If the server modifies or replaces the source audio used in the mixing process or modifies the mixed audio output, the server must convey this information to the endpoints. An active endpoint needs this information so the endpoint can modify or replace its own stored audio history and thereby use the correct audio data for properly removing its own audio stream from the mixed audio. Without this modification information, an active endpoint could not accurately remove its own audio data from the mixed audio.

4.2 Relaying Media Information

In addition to disclosing new information regarding the mixed audio, the server must also relay back to the endpoints certain media information that the server receives from the endpoints. This information is readily available at the endpoints when they transmit data to the server, and each active endpoint must have this information later

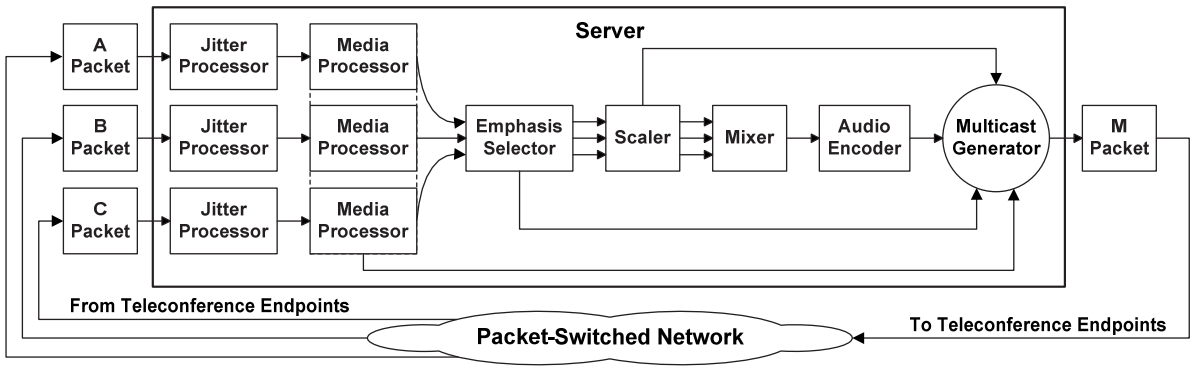

Figure 2: Server components

when the endpoint removes its own audio from the mixed audio. Upon receiving this information from the endpoints, the server simply sends the information back to the endpoints along with the mixed audio and the other information required for the own-audio removal process. One example of such information is the ID *tag* that identifies each individual segment of audio history. This tag tells an active endpoint which segment of the endpoint's own audio history to remove from the incoming mixed audio.

5 Teleconference Endpoint

In a traditional system, the server removes each active endpoint's audio from the mixed audio to create multiple mixed-audio output streams. With our new technique, however, each participating endpoint is responsible for the removal of its own audio from the single mixed-audio output that the server broadcasts to everyone. This new responsibility for own-audio removal leads to new tasks for the endpoint. For example, the endpoint must buffer its own audio history and media information. Each participating endpoint must implement a mechanism to tag its history records so the endpoint can retrieve the appropriate record when the endpoint needs the record in the removal process. Figure 3 illustrates the various components of the endpoint implementation. Typical endpoints with inexpensive DSP chips normally have plenty of processing power and memory to accommodate this implementation.

5.1 Buffering and Tagging the History

Existing VoIP endpoints do not store histories of transmitted audio and media information, so current endpoints cannot support own-audio removal. We must upgrade endpoints to provide this important feature. In particular, an endpoint must maintain a history buffer of its transmitted audio along with the media information corresponding to that audio. This history gives the endpoint part of the information the endpoint must have for removing its own audio from the mixed audio.

Since an endpoint does not know exactly when its transmitted audio will return to the endpoint as part of the mixed audio, we need to label each segment in the history buffer. The endpoint attaches a *tag* to each segment of audio that it sends to the server. If the server uses a segment of audio in the mixing process, the server returns that segment's tag to the endpoint along with the mixed audio and other media information. Using the returned tag, the endpoint can identify and retrieve the appropriate segment of audio and media information.

A segment in the stored history of an endpoint is no longer useful after the playback time of the mixed audio that could contain that segment of audio history, of course. The endpoint can release or re-use the memory that contains a history record that is no longer useful, so a circular buffer works nicely for the history records.

In general, an endpoint stores only enough history to overcome the round-trip delay and delay jitter in the network, so the memory requirements for the buffering are extremely modest. If, for example, we used the high bit rate of the G.711 Pulse Code Modulation (PCM) codec [6] and allowed for an intolerable maximum delay of 500 milliseconds, we would need only 4,000 bytes of storage, a reasonable amount even for a small, embedded processor. With the low bit rate of a highly compressing codec such as the ITU-T (International Telecommunication Union Standardization Sector) standard G.729A codec [7], we would need as little as only 500 bytes of buffer space.

5.2 Removal of Own Audio

When an endpoint receives the mixed audio from the server, the endpoint checks to see if its own audio is in the mixed audio. If its own audio is not in the mixed audio, the endpoint can simply use the mixed audio for playback without change. However, if the mixed audio contains the audio from the endpoint, the endpoint must

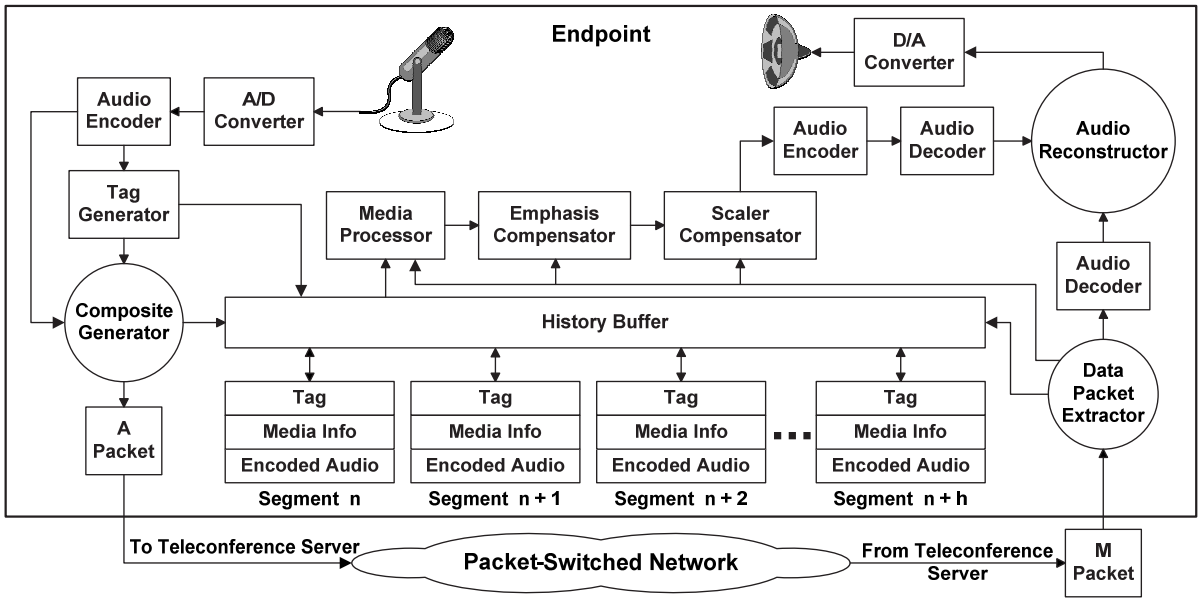

Figure 3: Endpoint components

remove its own audio from the mixed audio before playback.

The endpoint uses the tag that it sent to the server and received back from the server so the endpoint can retrieve the appropriate segment of audio history and media information from the endpoint's history buffer. The endpoint employs its own codec to *encode and decode* the audio data from the history buffer so the endpoint can obtain the same slightly distorted audio that the server used. In practice, the endpoint typically saves the encoded version of its transmitted audio in its history buffer to conserve memory, so the encode step that we describe here actually occurred at the original transmission time.

The endpoint next uses the media information disclosed by the server to see if the endpoint must modify its own segment of audio history. If appropriate, the endpoint modifies (e.g., scales) its own encoded-and-decoded audio to match the modification, if any, that the server made.

Then the endpoint *encodes and decodes* its own (possibly modified) audio data again to produce audio data that is suitable for removal from the mixed audio that the teleconference server transmitted. The endpoint encodes and decodes its audio data with the same audio codec that the server used to encode the multicast mixed audio. This second encode-decode step is necessary to make the audio data suitable for removal from the mixed audio because the mixed audio goes through that same encode-decode step with encoding by the server and decoding by the endpoint. The output audio from the compression and decompression of the coder and decoder is seldom an exact match for the input audio, but applying the same encode-decode step to the endpoint's audio produces a result that closely matches the corresponding portion of the mixed audio from the server. Note that the endpoint encodes and decodes its own audio *twice* to match the transformations that occurred for the endpoint's contribution to the mixed audio. Finally, the endpoint simply subtracts its own audio from the mixed audio to produce modified mixed audio that is suitable for playback.

6 Modified Source Audio

In order to provide a high-quality teleconference that sounds "natural," a teleconference server may modify the source audio in the mixing process. Sometimes the server must generate audio samples that are totally different from the original source audio for the mixing process. Therefore, the audio histories that the endpoints have stored may not match the actual audio that the server mixes into the mixed audio. The teleconference server must disclose this modification information to the endpoints so they can apply the same modification to their history audio or generate new audio samples to remove their own audio from the mixed audio. In this section, we explain two common scenarios that cause a server to modify the source audio.

6.1 Overflow

Whenever we sum two or more audio samples, there is a possibility of overflow. Overflow is an error that occurs when the sum exceeds either the most-positive or most-negative value that the system can represent correctly. In audio processing, we generally correct the overflow error by saturating the result. *Saturation* converts the result of an arithmetic operation to the most-positive or most-negative value as appropriate. Because of saturation, the

final result does not represent the sum of all of the original data values. In the case of saturation, therefore, a participating endpoint cannot remove its own audio by simply subtracting the original data that the endpoint stored.

Saturation is just one of many ways of handling or preventing overflow in the mixing process. Another approach is to prevent overflow by scaling the audio samples. The teleconference server can apply *attenuation* to the source audio so the sum of the audio samples does not produce overflow. If the teleconference server attenuates the source audio to a weaker signal for mixing, the original audio history that the participating endpoint saved does not match the weaker signal in the mixed audio. Consequently, the participating endpoint cannot simply use the original audio history data in the own-audio removal process but must instead apply the same attenuation to its history data.

6.2 Lost Packet

In VoIP, some audio packets inevitably arrive late due to large network delays or disappear altogether in the data network. In both situations, we consider the packet to be a lost packet, and the receiving device must use a *packet-loss concealment* (PLC) technique to synthesize the lost audio. Often, the synthesized audio is completely different from the original audio history data that the participating endpoint stored. Therefore, the endpoint cannot use the stored audio history data to remove its own audio from the mixed audio. The participating endpoint must know when packet loss has occurred and what PLC technique the server has used so the endpoint can synthesize the same audio data that the server used in the mixing process. Then the endpoint can use that synthesized audio packet to remove its own audio from the mixed audio.

7 Analysis of Bandwidth Savings

To quantify the bandwidth improvement that our system achieves, consider implementations with various codecs. We have computed results for server implementations with the ITU-T (International Telecommunication Union Standardization Sector) standard G.729A codec [7], the Internet Low Bitrate Codec (iLBC) [5], and the G.711 Pulse Code Modulation (PCM) codec [6].

Table 1 shows the bandwidth required for the teleconference server to deliver the mixed audio signals to the endpoints in an Ethernet environment using G.729A (8 kbps) unicast, G.729A (8 kbps) multicast, iLBC (15.2 kbps) multicast, and G.711 (64 kbps) multicast. In this analysis, the server transmits a 20-millisecond segment of mixed audio in every packet, and each packet carries 78 bytes of Ethernet, IPv4, UDP, and RTP headers — including the inter-packet idle time, preamble, and CRC of the Ethernet link-layer header. The analysis shows that our new teleconference technique using G.729A multicast and iLBC multicast (and even G.711 multicast) produces remarkable savings in bandwidth consumption for a conference with as few as only three participants. The savings become much more dramatic, of course, as the number of participants increases.

8 Conclusion

This new teleconference technique reduces the number of server transmissions from multiple unicast transmissions down to a single multicast transmission. The advantage of this new technique in terms of reduced bandwidth consumption increases tremendously when the number of participants in a conference grows. For a 100-participant conference, for example, this new approach requires 100 incoming audio streams and only one outgoing mixed audio stream. An existing teleconference system, on the other hand, would require 100 incoming audio streams and 100 outgoing mixed audio streams. Although the improvement is not as dramatic for a conference with a small number of participants, this new method is still effective in saving data bandwidth even for small conferences. *In general, this new approach reduces the network bandwidth consumption of a conference by a factor of two.* In addition, this technique also reduces the CPU bandwidth utilization at the teleconference server.

References

[1] L. Aguilar, J. J. G. L. Aceves, D. Moran, E. J. Craighill, and R. Brungardt, "An architecture for a multimedia teleconference system," in *Proceedings of ACM SIGCOMM Conference on Communications Architecture and Protocols*, pp. 126-136, Aug. 1986.

[2] S. R. Ahuja, J. Ensor, and D. Horn, "The rapport multimedia conferencing system," in *Proceedings of Conference on Office Information Systems, COIS 1988*, pp. 1-8, Mar. 1988.

[3] L. Gharai, C. Perkins, R. Riley, and A. Mankin, "Large scale video conferencing: A digital amphitheater", in *Proceedings of* 8^{th} *International Conference on Distributed Multimedia Systems*, Sep. 2002.

[4] IETF RFC-3550, *A Transport Protocol for Real-Time Applications (RTP)*, July 2003.

[5] IETF RFC-3951, *Internet Low Bit Rate Codec (iLBC)*, Dec. 2004.

[6] ITU-T Recommendation G.711, *Pulse Code Modulation (PCM) of Voice Frequencies*, Nov. 1988.

[7] ITU-T Recommendation G.729, *Coding of Speech at 8 kbit/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP)*, Mar. 1996.

[8] K. A. Lantz, "An experiment in integrated multimedia conferencing," in *Proceedings of ACM Computer-Supported Cooperative Work (CSCW'86)*, pp. 267-275, Dec. 1986.

[9] H. Liu and M. E. Zarki, "On the adaptive delay and synchronization control of video conferencing over the Internet," in *Proceedings International Conference on Networking, ICN 2004*, Feb. 2004.

Table 1: Teleconference bandwidth utilization

Number of Participants	G.729A Unicast (kbps)	G.729A Multicast (kbps)	% of Reduction	iLBC Multicast (kbps)	% of Reduction	G.711 Multicast (kbps)	% of Reduction
3	117.6	39.2	66.67	46.4	60.54	95.2	19.05
4	156.8	39.2	75.00	46.4	70.41	95.2	39.29
5	196.0	39.2	80.00	46.4	76.33	95.2	51.43
6	235.2	39.2	83.33	46.4	80.27	95.2	59.52
7	274.4	39.2	85.71	46.4	83.09	95.2	65.31
8	313.6	39.2	87.50	46.4	85.20	95.2	69.64

[10] P. V. Rangan and D. C. Swinehart, "Software architecture for integration of video services in the etherphone environment," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 9, pp. 1395-1404, Dec. 1991.

[11] P. V. Rangan, H. M. Vin, and S. Ramanathan, "Communication architectures and algorithms for media mixing in multimedia conferences," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 20-30, Feb. 1993.

[12] H. M. Vin, P. V. Rangan, and S. Ramanathan, "Hierarchical conferencing architectures for intergroup multimedia collaboration," in *Proceedings of the Conference on Organizational Computing Systems (COCS'91)*, pp. 43-54, Nov. 1991.

[13] Y. Xie, C. Liu, M. Lee, and T. Saadawi, "Adaptive multimedia synchronization in a teleconference system," *Multimedia Systems*, vol. 7, no. 4, pp. 326-337, July 1999.

[14] C. Ziegler, G. Weiss, and E. Friedman, "Implementation mechanisms for packet-switched voice conferencing," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 5, pp. 698-706, June 1989.

Teck-Kuen Chua earned his PhD degree in Computer Science and Engineering at Arizona State University in 2006. He is a senior software design engineer at Inter-Tel, Incorporated, where he works with advanced VoIP technologies, embedded systems, and real-time applications for digital signal processors.

David C. Pheanis is a Professor Emeritus of Computer Science and Engineering at Arizona State University and is the Principal of Western Microsystems. He works with embedded systems and real-time applications of microcontrollers. He earned his PhD degree at Arizona State University in 1974.