Performance Analysis of a Bandwidth-Efficient Real-Time VoIP Teleconference System

Teck-Kuen Chua and David C. Pheanis

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 evaluates 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.

Index Terms—audio mixing, bandwidth conservation, business telephone system, IP Multicast, teleconference system, VoIP.

I. TELECONFERENCE BACKGROUND

A real-time 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.

II. EXISTING TECHNIQUES

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, ..., 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 implementing teleconferences, and then we explain

\(^{1}\text{Patent pending}\)
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. Then we present simulation results that demonstrate the effectiveness of our system. Finally, we summarize the advantages that our new method provides.

Teleconferencing is a pervasive and extremely useful telecommunication feature, so various designers have used differing approaches to implement this feature [1]–[10]. However, none of the existing methods can support a large-scale VoIP teleconference in a bandwidth-efficient manner.

Currently, there are three common implementations of VoIP teleconference systems: peer-to-peer unicast, peer-to-peer multicast [11], and server-based unicast. Peer-to-peer unicast is an ad-hoc teleconference implementation. Although the peer-to-peer unicast technique is simple and straightforward, it has a scalability problem and requires extensive bandwidth for a large-scale teleconference: \((n^2 - n)\) unicast links for an \(n\)-party conference. The peer-to-peer multicast approach alleviates the bandwidth issue by using multicast, but this method still does not scale well because a large teleconference with this approach requires more endpoint processing power than endpoints can typically deliver. On the other hand, server-based unicast systems handle the scalability issue well, but these systems still require a considerable amount of bandwidth — \(2n\) unicast links for an \(n\)-party conference.

### III. 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.

### IV. TELECONFERENCE SERVER

Fig. 1 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 follow-
ing sections detail these new server responsibilities.

A. 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.

B. 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 when the endpoint removes its own
audio from the mixed audio. Upon receiving this infor-
mation 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.

V. 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. Fig. 2 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.

A. 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 end-
point part of the information that the endpoint must have
for removing its own audio from the mixed audio.

Since an endpoint does not know 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 saved audio and saved media
information.

A segment in the stored history 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 roundtrip 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 [12] 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 [13], we would need as little as only 500 bytes of buffer space.

B. 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 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 almost suitable for playback.

In most cases, the simple subtraction of the endpoint’s own audio from the mixed audio produces a resultant mixed audio that is suitable for the endpoint to play back without any audible echo. For some codecs, however, the simple subtraction may not remove the endpoint’s own audio entirely, leaving some low-level residue signal in the resultant mixed audio. Therefore, in the final step of the own-audio removal process, the endpoint completely removes any remaining residue signal in the resultant mixed audio to produce a clean mixed audio signal for playback.

VI. TELECONFERENCE SIMULATION RESULTS

To quantify our results, we simulated a three-party conference using our new teleconference technique. The simulation results clearly demonstrate the feasibility of our new teleconference approach. The endpoints are able to remove their own audio and produce a version of mixed audio that is suitable for playback without echo. In our simulation, the endpoints use the ITU-T (i.e., International Telecommunication Union Standardization Sector) standard G.729A CS-ACELP audio codec to encode their audio, and the endpoints send the G.729A-encoded audio to the teleconference server. We experimented with various audio codecs — Internet Low Bitrate Codec (iLBC), ITU-T standard G.729A, G.711 Pulse Code Modulation (PCM), and G.726 Adaptive Differential PCM (ADPCM) audio codecs — to encode the mixed audio that the server multicasts to the endpoints.

Fig. 3 illustrates a mixed audio signal of all three participants. In this particular test, each participant speaks a word at different time. Fig. 3 therefore shows a mixed audio signal with three distinct groups of sound, one spoken word from each participant. The teleconference server transmits this mixed audio to all participants in the teleconference with a single multicast stream.

Fig. 4 illustrates the resultant audio signals for the second participant after the endpoint has removed its own audio signal from the multicast mixed audio when using the iLBC codec. The results for the G.729A, G.711, and G.726 audio codecs are nearly identical to the iLBC results. In each case, the endpoint of the second participant very successfully removes its own audio from the multicast mixed audio. The resulting audio stream has virtually no noise residue of the signal from the second participant, so the audio is entirely suitable for playback to the second participant with no echo. The output that we obtain with our approach is nearly identical to the output that we obtain from a traditional server-based unicast system, but our technique requires only about half as much bandwidth as a server-based unicast system.

VII. 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,
the Internet Low Bitrate Codec (iLBC), and the G.711 Pulse Code Modulation (PCM) codec.

Table I 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 grows.

**VIII. 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.

### Table I

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

**References**


