HAT: A High-quality Audio Conferencing Tool using MP3 Codec*

Sooyeon Kim, JeongKeun Lee, Tae Wan You, KyoungAe Kim, Yanghee Choi
School of Computer Science and Engineering, Seoul National University, Korea
Email: {sykim, jklee, twyou, drizzle, yhchoi}@mmlab.snu.ac.kr

Abstract

We introduce an internet protocol version 6 (IPv6) audio conferencing tool implemented using MPEG-1 layer 3 (MP3) codec. In this paper, we present the software architecture, hurdles and tactics while developing the application based on existing modules like public MP3 encoder/decoder and real-time transport protocol (RTP)/ real-time transport control protocol (RTCP) implementation. And some experimental results including end-to-end latency show problemetic points to be faced when we deploy MP3 codec to build an interactive voice communication application.

Keywords: *Audio conferencing tool, MPEG-1 layer 3 (MP3), interactive voice communication.*

1. Introduction

Recently advances in voice encoding technology and high bandwidth availability in access networks have encouraged voice communication over Internet. However, most of voice communication applications do not provide high fidelity, since the bandwidth fluctuation of Internet prohibits continuous provision of fine quality audio. This insists that voice codecs of high compression ratio should be adopted in voice communication tool implementations in order to provide high-quality audio communication over the networks like Internet where end-to-end quality of service (QoS) is not guaranteed.

*This work was supported in part by Electronics and Telecommunications Research Institute (ETRI), in part by the Brain Korea 21 project of Ministry of Education, and in part by the National Research Laboratory project of Ministry of Science and Technology, 2001, Korea.

MPEG-1 layer 3 (also called MP3) is a prevalent standard that provides high compression ratio while maintaining compact disc (CD) quality. Another virtue of MP3 is that a number of public codecs like LAME[2] and mpg123[4] are available, which makes it easy to adopt MP3 encoding into voice application implementations. However, it would be a common concern that MP3 might introduce substantial end-to-end latency which makes the deployment inappropriate for interactive voice communication applications, due to its nature intended to deal with stored audio data rather than real-time ones.

Therefore, in this paper, we examine the applicability of MP3 codec to real-time voice communications, presenting our audio conference tool implementation and its experimental results. In Section 2 introduces related work. Then in Section 3 we present the software design. Inter-node communication scenario and some implementation techniques will be given in Section 4 and Section 5. Section 6 shows experimental results, and finally we conclude this paper in Section 7.

2. Related Work

The real-time transport protocol (RTP) and real-time transport control protocol (RTCP)[5] is the most widely-used protocol suite that supports simple end-to-end network transport functions suitable for applications transmitting real-time data, such as audio or video over multicast or unicast network services. Among a large number of RTP/RTCP implementations available, we chose University College of London's Common project[8] as a baseline, since its functionality and reliability are guaranteed by Multicast backbone (MBONE) applications such as robust audio tool (RAT) developed by the same group.

Lame ain't an MP3 encoder (LAME)[2] is a well-known public MP3 encoding software that also opens its implementation. Its open-source feature and the reliability proven by a large number of applications which adopted Lame as their MP3 encoder are very useful for implementing applications that need to be powered by an MP3 encoder and understanding/optimization research of MP3 encoding. Lame reads MP3 or pulse code modulation (PCM) files as input and encodes them into MP3 format with specified options like bitrate, sampling rate, and variable bitrate (VBR)/constant bitrate (CBR) characteristics.

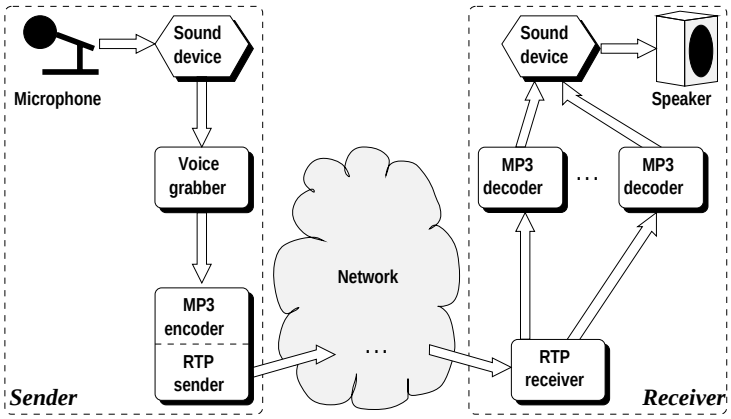
Mpg123[4] is also a famous MP3 software that decodes MP3 file inputs and then either produces a WAV file or play the decoded data through an underlying sound device. Originally Mpg123 is intended to be run on Linux and UNIX systems but there is a third-party patch[3] to enable compilation and execution on Microsoft Windows. Mpg123, for its input, not only accepts MP3 files stored in a local storage like a hard disk, but also downloads from a remote host using hypertext transfer protocol (HTTP).

3. Software Design

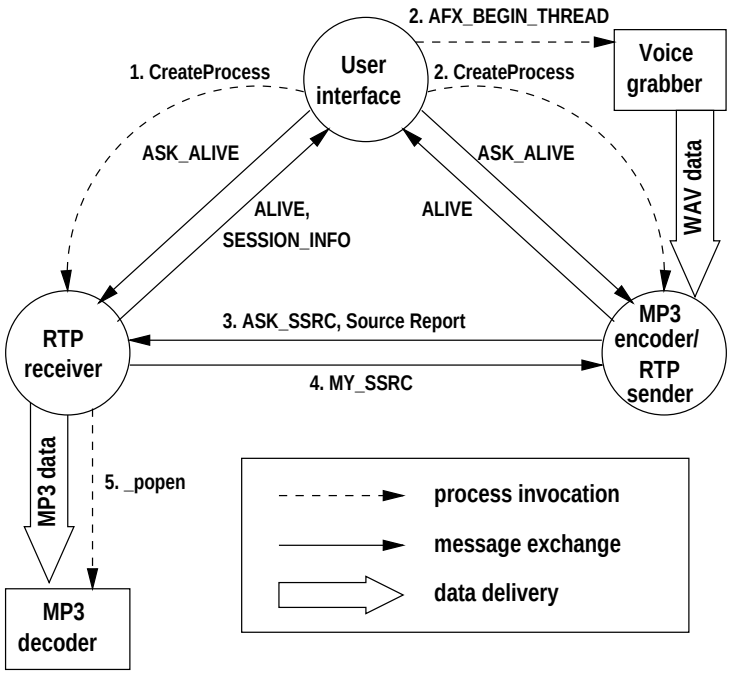
Our MP3 audio conference tool named *high-quality audio tool (HAT)*, as depicted in Figure 1, is comprised of five modules: user interface, voice grabber, MP3 encoder/RTP sender, RTP receiver, and MP3 decoder. *User interface* is the first module to be invoked when HAT is executed, and responsible for launching three other modules: firstly RTP receiver, then voice grabber and MP3 encoder/RTP sender. *Voice grabber* captures the user's voice input in a digitized format, WAV, through a microphone and a sound capture device, and relays the WAV data to MP3 encoder/RTP sender. *MP3 encoder/RTP sender*, as the module name represents, encodes WAV data delivered by voice grabber into MP3 format and transmit the MP3 data as RTP payloads. *RTP receiver* receives all RTP and RTCP packets sent to the multicast address of the currently participating conference session, which includes packets sent by the RTP sender process of the same HAT instance. RTP receiver separates RTP packets that are generated and sent by different sources but mingled in the network, deserts self-generated packets, and then when a new source is found, invokes an instance of MP3 decoder. How to manage multiple MP3 decoders will be explained later. And it is also in charge of dispatching MP3 data to the appropriate MP3 decoder among possibly two or more MP3 decoder instances. Another essential role of RTP receiver is RTCP data provision and management. Keeping track of statistics like the number of received packets from each participant and the packet loss rate, RTP receiver reports them periodically by an interval calculated as specified in [5]. If the node itself is an *active source* (i.e., the node keeps multicasting MP3 data), RTP receiver reports self-statistics (*source report* that appears in Section 4) like how many packets it has transmitted and the current timestamp value. Also the user's information (e.g., canonical name, phone number) should be periodically furnished to allow other participants to browse it.

Without a delicate floor control that allows only a single participant to have the right to speak, definitely there is a chance that an audio conference session has two or more participants who are willing to speak at the same time. Thus how to mix and play audio data from multiple sources should be considered. We exploit the sound mixing feature supported by most of present-day ordinary sound devices, which means that HAT does not undertake audio mixing but that simply sends out multiple streams of audio data to the sound device in a parallel manner. To enable parallel output to sound devices, MP3 decoder processes should be separately launched for each audio source. Here we claim that there should be provided a way to keep the number of decoder processes appropriate (e.g., same as the number of active sources) so that the application can avoid scalability problems.

The robustness of HAT that maintains the whole set of required process alive is assured by our own messaging protocol between processes as presented in Figure 1(b). User interface process periodically requests liveness reports the two processes, MP3 encoder/RTP sender and RTP receiver, by sending ASK_ALIVE messages to



(a) Audio data delivery scenario.



(b) Interactions and message exchanges between modules.

Figure 1. High-quality audio tool design.

them. When the two processes receive an ASK_ALIVE message, they promptly send an ALIVE message. Thus if there is no ALIVE message from either MP3 encoder/RTP sender or RTP receiver after a sufficient amount of time has passed after sending ASK_ALIVE message, user interface can determine liveness and invokes a new process to replace the dead one. At the same time, when the two processes does not receive any ASK_ALIVE messages during a substantial amount of time, they infer that user interface is terminated and then kill themselves to save system resources like CPU usage.

The main purpose of the other message exchanges is to maintain the consistency of statistical data through the whole application. SESSION_INFO messages provided to user interface by RTP receiver contain user information and networking statistics for each participant in the session. ASK_SSRC message elicits MY_SSRC that encloses the synchronization source (SSRC) number being used by RTP sender to provide identification information of the audio data source, in order to make RTP sender and receiver use the same SSRC.

4. Inter-node Communication

In this section, we introduce how a HAT instance communicates with other instances scattered over the network, using RTP and RTCP. Figure 2 presents the communication sequences between two nodes participating in a conference session that uses the multicast address ff0e::2:fb4b and the port 27764. In this paper we does not explain how to open a conference session, where and how to obtain session information, and how to join an interested session, since we suppose that those kind of procedures follow session announcement protocol (SAP)[7] and session description protocol (SDP)[6] like multicast backbone (MBONE) applications. And notice that Figure 2 shows a situation where only a single node (participant 1 (P1)) distributes audio data, and that the dotted arrows depicts messages sent back to the origin due to IP multicast, RTP messages which include MP3 data are drawn as lighter colored solid arrows, all the other arrows in the figure represent RTCP message transfers.

Joining the session, a node multicasts an RTCP sender report (SR), receiver report (RR)[1], and source description (SDES) message to notify its participation to the other nodes in the session (messaging sequence 1 and 2). Nodes who received an RTCP message from a new source create an entry for that in their databases and save the information (enroll P1/P2). Now suppose that P1 begins talking and accordingly it sends MP3 data as the payload of RTP packets (sequence 3 and 6). The first RTP packet from P1 informs that P1 is an active source and a designated MP3 decoding process should be launched (start playing / P1 := active source). Every active source

[1] RFC 1889[5] imposes that the first RTCP packet must be a report packet for header validation purpose, even if no data has been sent nor received.

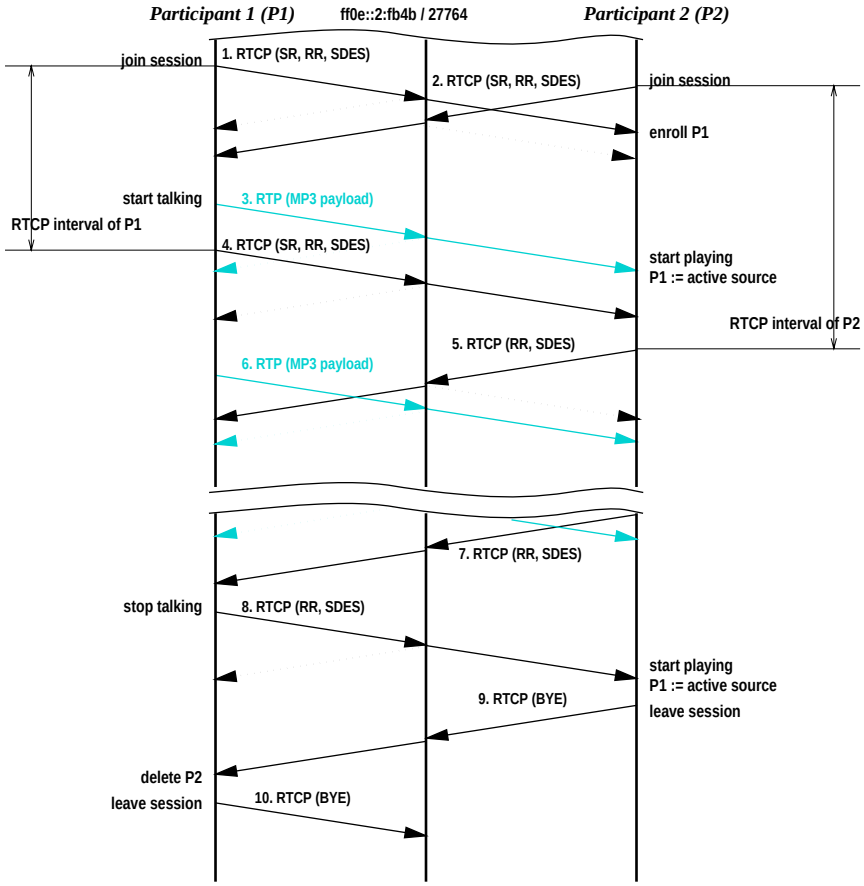


Figure 2. Communication sequences.

Table 1. Module Characteristics

Module name	*Domain*	*Real-time streaming needs*
User interface	window-based	none
Voice grabber	window-based	to MP3 encoder
MP3 encoder/RTP sender	win32 console	from voice grabber
RTP receiver	win32 console	to MP3 decoder(s)
MP3 decoder	win32 console	from RTP receiver

repeats RTCP message transfers that include both of SR and RR. And every inactive source periodically transmits RR with SDES until it leaves the session (sequence 5, 7, 8), which also allows P2 to know that P1 is not an active source any more (sequence 8). In that case, P2's hat instance terminates the decoder process for P1 but only after a certain amount of time (e.g, several times of RTCP intervals) has passed after the status change, since immediate terminations might cause inefficiently frequent re-invocations. When a participant wants to leave the session, the node sends an RTP BYE message, which is followed by the corresponding resource release like database entry elimination and MP3 decoder termination.

5. Real-time Data Streaming between Modules

In order to keep the end-to-end latency low enough and avoid the discontinuity of audio data provision, audio data relay from a module to another should be real-time streaming. In this section, we introduce the hurdles and tactics we have faced while implementing an audio conference tool with software modules already written in different ways, especially regarding the real-time streaming feature provision.

Table 1 summarizes the characteristics of each module. As the third column indicates, there are two spots that requires real-time streaming to be implemented: (1) between voice grabber and MP3 encoder/RTP sender, and (2) RTP receiver and MP3 decoder.

One of the easiest way to enable continuous data feed from one process to another is to use standard input/output supported by operating systems, that is, the data source process simply keeps writing its output to the standard output and then the destination process receives the data just by reading the standard input. However, we had two tricky points to solve the problem in such a manner. The first one is that LAME expect that the type of input data should be a file, and that does not have an option to read the standard input. And the second one is that, for the case of voice grabber and MP3 decoder, the domains are different as window-based versus win32 console, which prevents the two processes from communicating via standard input and output.

The first problem is resolved by modifying LAME. Since the main reason LAME requires a complete file input is that it is intended to understand how to interpret the input data by reading the header of a complete WAV file, we hard-coded the characteristics of voice grabber's WAV format into the beginning of LAME implementation.

Figure 3 summarizes a programming tactic, encouraged by [1] that solves the second problem and successfully builds a real-time streaming framework between two processes of different domains. At first, a pipe tied with the standard input and output should be opened in the window process (voice grabber here), and the write pipe is tentatively disconnected so that the property is not inherited by the child process, which is done by saving the write pipe's handle in a temporary variable and then closing the handle. Secondly, a console process (MP3 encoder/RTP sender) is created with its standard input connected to the read pipe of the window process. Then, in Figure 3(c) the window process backups the handle to the standard output and closes the read pipe to assure that the output from the window process will not be consumed by itself. Finally, the two processes communicate through standard input and output as shown in Figure 3(d). Data sent to the write pipe by the voice grabber, a window process are forwarded to the standard output, and the MP3 encoder, a console process can receive the input data via its standard input that inherited the window process' read pipe handle.

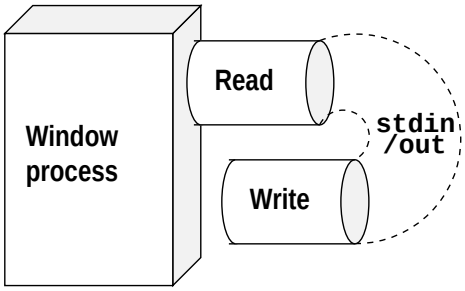
Real-time streaming from RTP receiver to MP3 decoder is straightforward, because both of them are win32 console programs. When RTP receiver is supposed to invoke an MP3 decoder process, it opens a pipe such that the MP3 decoder reads the standard input.

6. Results

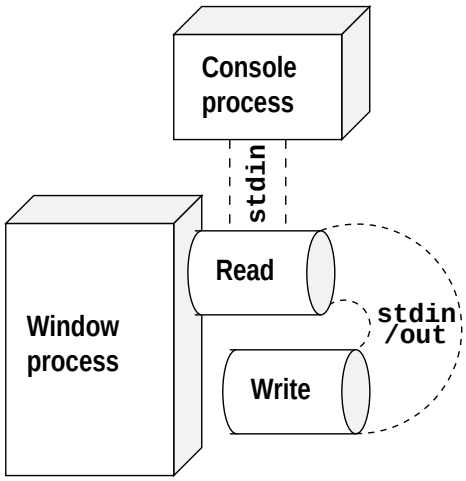
Figure 4 shows our experimental environment, a native IPv6 network that connects Seoul National University and ETRI using PC-based IPv6 routers and ATM switches.

Traditional voice encoders like G.711 uses PCM[10]. More specifically, existing digital audio employs PCM with 16 bits per sample and sampling rates of 32 kHz, 44.1 kHz, 48 kHz[11], and this leads to bandwidth consumption of a few Mbps; For example, with two-channel stereo and 44.1 kHz sampling rate, bandwidth requirement will be $16bits/sample * 44,100samples/sec * 2channels \approx 1.35Mbps$. With MP3 encoding, however, 128 to 256 kbps achieves almost equivalent audio quality. Hence by deploying MP3 codec we can acquire six or twelve times of effectiveness in terms of bandwidth consumption.

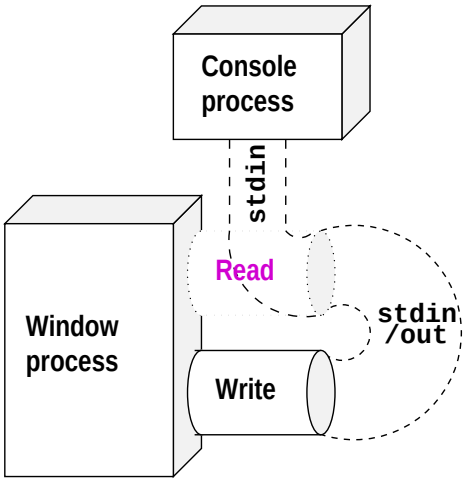
We compare our MP3 audio conference tool HAT with robust audio tool (RAT) which is developed by UCL and supports PCM encoding. Retaining the similar audio quality, we choose PCMU-16K-Stereo codec for the RAT instance, and the sampling rate of both of RAT and HAT is set to 16 kHz.



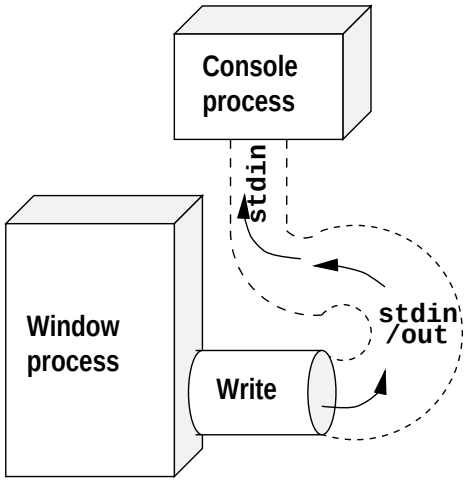
(a) Window process creates a pipe and temporarily closes the write pipe.



(b) Create a child console process whose stdin inherits the parent's read pipe.



(c) Window process closes the read pipe and backups the write pipe.



(d) Data are delivered from window process to console process.

Figure 3. Streaming from window process to console process.

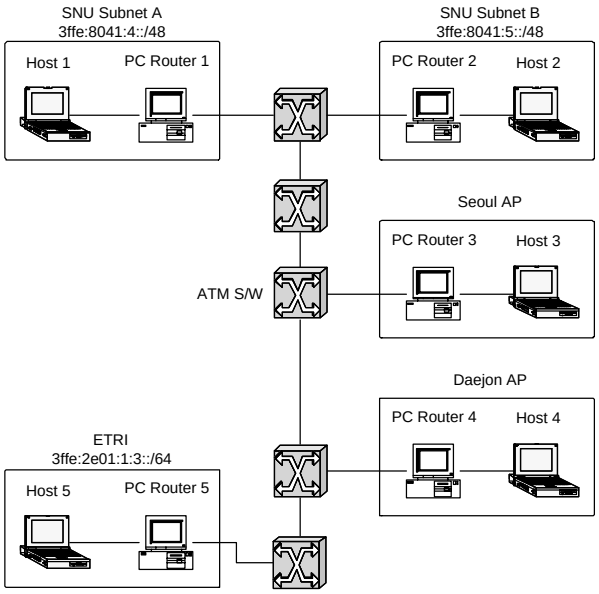
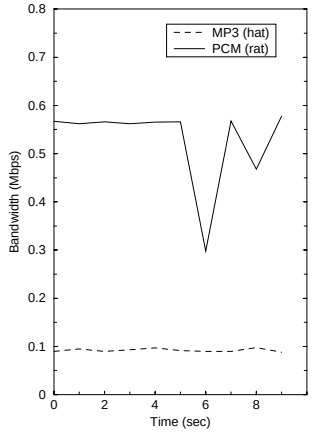


Figure 4. Network configuration.

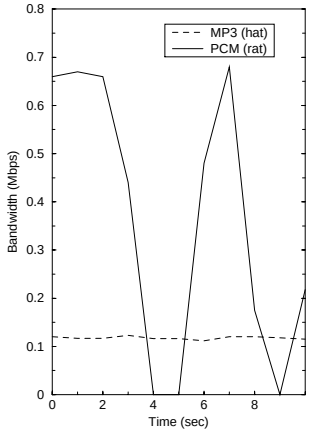
Figure 5 depicts the bandwidth consumption of HAT and RAT, measured by tele traffic tapper (ttt)[9]; Figure 5(a) presents a talkspurt and Figure 5(b) includes silent periods. For both of the cases, we can observe that the bandwidth requirement of our MP3 audio tool is significantly lower than that of RAT with PCM encoding. In Figure 5(b), however, there are two regions where HAT occupies more bandwidth than RAT. It results from the silence detection feature of RAT, that is, RAT does not transmit at all during silent period detected while HAT keeps encoding and sending packets regardless the existence of active voice input. Nevertheless, even in Figure 5(b), in terms of the average bandwidth usage MP3 encoding outdoes PCM encoding, which results in the robustness of perceived audio quality when we increase the network load by generating dummy traffic.

As presented above, MP3 extremely enhances bandwidth efficiency but it is a common concern that the complexity of encoding procedure would cause substantial amount of end-to-end delay that is not appropriate for interactive voice communication where the maximum end-to-end delay of $200 - 300$ ms should be guaranteed[10]. Without loss of generality, assume that the latency caused by data transmission over the network is dependent of audio coding mechanism in use, the main concern would be the latency caused by MP3 encoding and decoding.

Figure 6 decomposes the throughout process to let Bob hear Alice talk to him, and for each of the decomposed steps, measured latency is presented in Table 2. Far from our guess that the bigger portion of end-to-end latency would be incurred by using a software encoder instead of hardware one, most (approximately 80%) of the end-to-end latency is introduced by decoding process. One of the reasons that cause the substantial amount of latency is



(a) During a talkspurt.



(b) With silent period.

Figure 5. Bandwidth consumption.

Table 2. Decomposed End-to-End Latency

step	*latency measurement timing*		*latency (in* ms*)*
	from	*to*	
(1)	launch MP3 encoder	start reading encoder buffer	$50 - 60$
(2)	start reading encoder buffer	end encoding	$30 - 40$
(3)	start sending RTP packets	end receiving the first RTP packet	$20 - 30$
(4)	end receiving an RTP packet	launch MP3 decoder	$20 - 80$
(5)	launch MP3 decoder	end receiving the first frame to decode	$440 - 490$
(6)	start decoding	start writing decoded data to sound device	210
total end-to-end latency			$770 - 910$

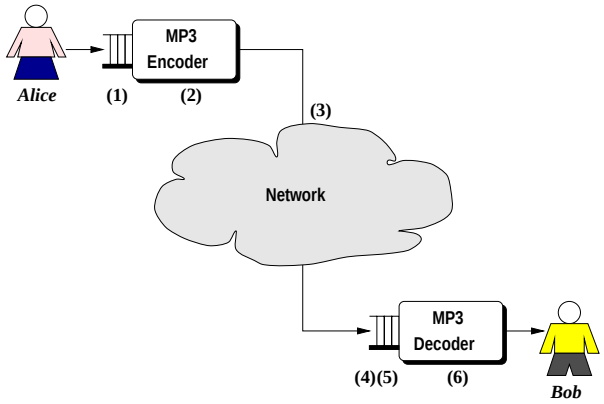


Figure 6. Voice delivery. (1) Launching MP3 encoder (2) MP3 encoding (3) RTP packetization and transmission (4) launching MP3 decoder for a new source (5) reading MP3 data (6) MP3 decoding and writing output buffer.

mpg123 waits for a frame of 2048 bytes, which imposes that the decoding cannot be started without being delayed for a certain amount of time, for example $2048bytes/128kbps = 125ms$ when applying 128 kbps CBR encoding.

7. Conclusion

In this paper we present our audio conference tool which deploys MPEG-1 layer 3 (MP3) compression as the audio data coding. Experimental results assert that high compression ratio can provide robust audio quality, and show the potentialities of MP3 as an encoding format for interactive audio; It is true that the MP3 audio conference tool introduces substantial end-to-end delay around 881 ms on average, which is not suitable for voice communications that requires fast responses. However, as the decomposed latency data suggest, most (about 80%) of the end-to-end delay is induced by the decoder, which is different from our expectation that the complexity of MP3 encoding mechanism would be the main cause and proposes that the delay could be significantly lowered by tuning the decoding process such as buffer size adjustment.

References

[1] Info: Redirection issues on windows 95 ms-dos applications. http://support.microsoft.com/default.aspx?scid=kb;EN-US;q150956.

[2] The LAME project. http://www.mp3dev.org/mp3.

[3] Mpg123 for win32. http://www3.tky.3web.ne.jp/ takuroho/mpg123.html.

[4] Mpg123 homepage. http://www.mpg123.org.

[5] RTP: A transport protocol for real-time applications. RFC 1889.

[6] SDP: Session description protocol. RFC 2327.

[7] Session announcement protocol. RFC 2974.

[8] UCL Common multimedia library. http://www-mice.cs.ucl.ac.uk/multimedia/software/common/.

[9] K. Cho. Tele traffic tapper. http://www.cs1.sony.co.jp/person/kjc/programs.html#ttt.

[10] M. J. Karam and F. A. Tobagi. Analysis of the delay and jitter of voice traffic over the internet. In *Proc. of IEEE INFOCOM 2001*, Anchorage, Alaska, USA, April 2001.

[11] P. Noll. High quality audio for multimedia: Key technologies and MPEG standards. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, Rio de Janeiro, Brazil, November 1999.