Real-Time Multiple-Description Coding of Speech Signals

Jan Weil and **Kai Clüver** and **Thomas Sikora**

Communication Systems Group, Technische Universität Berlin
Einsteinufer 17
10587 Berlin
Germany
{weil,cluever,sikora}@nue.tu-berlin.de

Abstract

When sending speech data over lossy networks like the internet, multiple-description (MD) coding is a means to improve the perceived quality by dividing the data into multiple descriptions which are then sent as separate packets. In doing so the speech signal can still be decoded even if only parts of these descriptions are received. The present paper describes the structure of a software which demonstrates the benefits of this coding scheme using a client-server architecture.

Keywords

Multiple-Description Coding, Speech Coding, Real-Time Coding

1 Introduction

When transmitting real-time speech data over the internet to multiple receivers (multicast or broadcast), the quality on the receiving side depends on how many packets are lost on their way. Various receivers may experience greatly differing speech quality due to varying network conditions. Since in the multicast scenario delivery monitoring for all of the subscribed receivers is not feasible, action is needed on the receiving side.

To ensure graceful degradation of the perceived quality with increasing packet loss, MD coding can be applied, i. e. the data which is to be transmitted is divided into two or more descriptions. Even if not all of them are received, the signal can still be decoded, albeit with lower quality.

In the course of our project, two different MD speech codecs were developed. The first one is based on logarithmic pulse code modulation (PCM). The second one is a variation of the G.729 annex A codec, which is based on code excited linear prediction (CELP) and defined by the International Telecommunication Union (ITU). To show the improvement due to MD coding over lossy channels, a demonstrator application has been developed. The structure of this application is described in the present paper.

The rest of this paper is organized as follows: In Section 2 the principles of MD coding are explained. The actual implementation of the demonstration application, which is the main subject of this paper, is described in section 3. After that the usage of the program is illustrated in Section 4. Section 5 contains some concluding remarks.

2 Multiple-Description Coding

MD coding [1] provides a transmission link with diversity in order to improve robustness to channel breakdown. The coded signal is split into two or more descriptions which are transmitted over the same number of different channels. These channels may indeed consist of different physical links, or of different packets transmitted through networks like the internet.

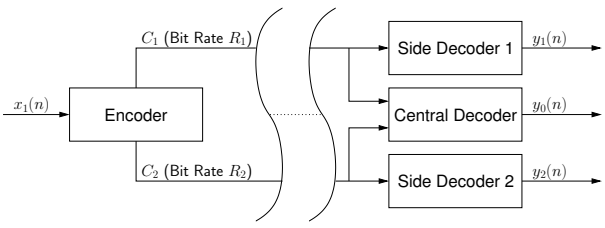

Figure 1: MD coding scheme with two descriptions

The principle of a two-channel MD coded transmission is shown in fig. 1. From the input signal, $x(n)$, the encoder generates two descriptions C_1 and C_2 to be sent over two lossy channels. If no loss occurs, both descriptions will be used by the central decoder to reconstruct the signal $y_0(n)$ with high quality. If one of the descriptions is lost, the received part of the code will enable its corresponding side decoder to yield a reduced-quality version of the output signal, $y_1(n)$ or $y_2(n)$. The transmission will be interrupted only when both descriptions

are lost.

The design of MD coders is subject to conflicting requirements [1]. If the side decoders were optimized for high signal quality, given the bit rates R_1 and R_2 for C_1 and C_2, little would be gained by combining both descriptions in the central decoder, which would then yield a similarly high quality, but at a considerably increased bit rate of $R = R_1 + R_2$. If, on the other hand, the central decoder were designed for minimum distortion at a bit rate of R, any splitting of the code would result in poor performance of the side decoders. Therefore, the usual objective is to find a compromise for central and side decoder qualities.

Many designs aim at balanced descriptions, i. e. equal bit rates ($R_2 = R_1$) and equal distortions of the side decoders. For more than two MD channels, balanced descriptions will yield decoding distortions which do not depend on the individual subset of descriptions but only on the number of descriptions received. The decoded quality will then degrade gracefully with increasing channel failure ratio.

A receiver for L descriptions consists of $2^L - 1$ decoders (including the central decoder). Consequently, the MD decoder will be extremely complex for high values of L if explicit side decoders are employed. This problem can be avoided by using a hierarchical (layered) speech coder together with forward error correction (FEC) codes for the construction of multiple descriptions [2]. The approach consists of applying unequal loss protection to L code layers and re-grouping the symbols of the resulting code words into L descriptions. With $k < L$ descriptions received, the MD decoder is able to decode the basic k layers of the coded speech. The side decoders are constructed implicitly by FEC decoding.

The FEC coding scheme causes high gross bit rates compared to the original codecs. It is possible, though, to trade robustness for bit rate savings if the coarse base layer is made up of more than one description.

3 Real-Time Implementation

3.1 Overview

An overview of the system as a whole is given in fig. 2. It allows multiple client applications to be served concurrently. On the sending side a server waits for incoming requests. Clients connect to the server and request speech data streams. For every requested stream, a new sender process is started by the server which connects to one of the available speech sources, encodes the data frame by frame, and sends the coded frames to the client by which the stream has been requested. It is possible to set up several streams to enable the user to compare different configurations.

3.2 Serving side

In fig. 2 every square-cornered box represents a separate process. Technically there is no need to start a separate process for every newly requested stream. Since, however, in the course of this project a single sender application had already been developed, it was easier to add a simple server and start the senders as subprocesses. This server is written in Python [3], which is a dynamic object-oriented programming language and even provides a `TcpServer` class as part of the standard library.

For demonstration purposes the source of the speech data was supposed to be selectable so that different speech sources could be offered. Using the Jack Audio Connection Kit (JACK) [4] this can easily be achieved. Part of it is `jackd`, a low-latency audio server which allows several different applications to share audio data among them. It is typically used for professional audio processing applications, which means that running `jackd` at a sampling rate of 8 kHz, as we did, is quite unusual. Nevertheless, JACK has proven absolutely appropriate to our needs.

In our case the selectable sources are provided by Ecasound [5]. Ecasound is a software package designed for multitrack audio processing, which, among other things, can be used to play back audio files in loops. For each loop a JACK port is registered so that our senders can connect to these ports.

3.3 Transport protocol

The transmission of audio and video data over the internet is often done using the Real-time Transport Protocol (RTP) which is usually built on top of the User Datagram Protocol (UDP). Compared to pure UDP, RTP additionally provides sequence numbering, time stamping, payload-type identification, and delivery monitoring. Because sequence numbering is the only feature that was needed in our case, we decided not to use RTP. Instead we added a 16-bit header containing a sequence number which is incremented for each packet. To make sure that an overflow of the sequence number does not

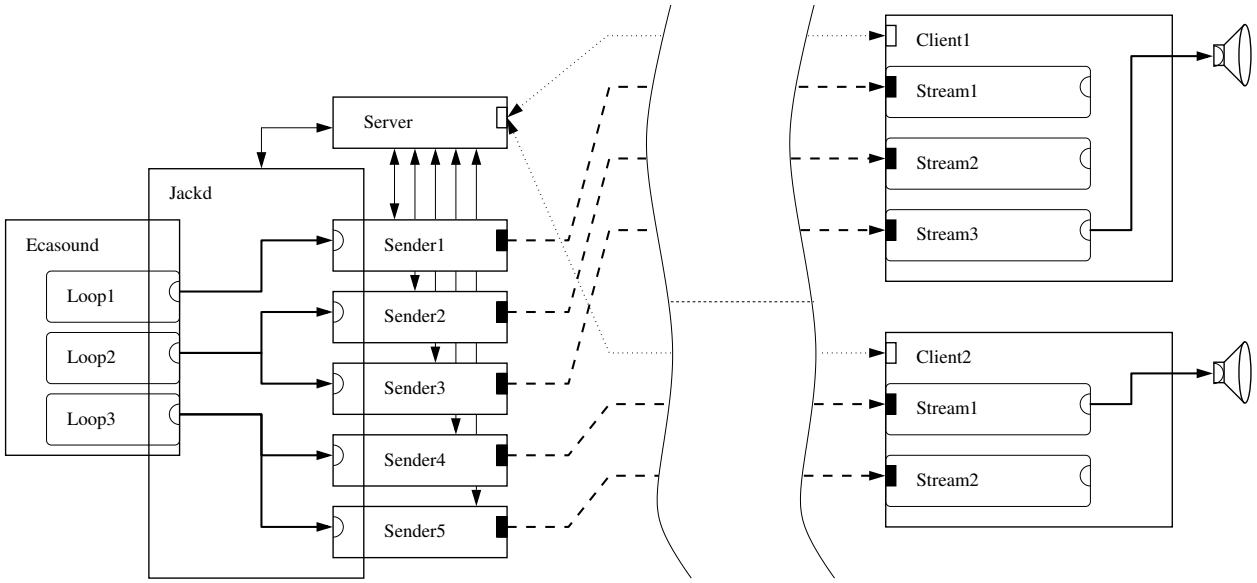

Figure 2: Structure of the system. Clients connect to the server's control port and request streams. For each stream, the server starts a new sender process which encodes the audio data delivered by JACK and sends it to the requesting client.

disturb the order of descriptions, the sequence number range is limited to a multiple of L.

The UDP port number on the receiving side is limited to the range of 55550 to 55569, which means that, on a distinct host, at most 20 separate ports can be served simultaneously.

3.4 Control protocol

In order to control the transmission of data a simple plain-text protocol has been designed. The transport protocol used for this purpose is the Transmission Control Protocol (TCP). The server waits for requests coming in on TCP port 55555. A request consists of a special command word in capital letters, possibly a list of arguments separated by spaces, and the closing two special characters carriage return (CR) and line feed (LF). After the request has been processed the server returns an answer which consists of a return string in capital letters (`OK` on success, otherwise `ERROR`), possibly an additional return string, and once more the concluding sequence CR-LF. There are four known commands:

`SOURCES` This command is used to retrieve the list of available speech sources. It does not allow any arguments. On success a comma-separated list of readable JACK ports is returned.

`PORT` Before a stream is requested, the client asks the server to allocate a port number. The next time the client issues the `OPEN` command this port will be used. On success the port number is returned by the server. If the client is located behind a firewall, incoming UDP traffic is usually blocked. If, however, the client sends an initial packet to the newly allocated port on the serving side, many firewalls will accept the incoming stream as a response to this packet. This concept is known as UDP hole punching.

`OPEN` Exactly six arguments are expected to open a new stream: the number of the receiving UDP port, one of the formerly listed sources, the codec to use (either `pcm` or `celp`), the number of descriptions to use, the number of descriptions which make up the base layer, and an additional argument which is either the segment length (PCM) or the number of CELP segments per MD frame. If a new sender process has been started successfully the server returns the corresponding process ID.

`CLOSE` This command needs the formerly transmitted process ID of a sender as the only argument. By receiving this command the server sends a signal (`SIGINT`) to the appropriate process. To ensure that no other processes are killed maliciously, this is only done if the transmitted ID actually represents one of the formerly spawned child processes.

3.5 Client side

On startup every client must connect to the server. If this connection fails, the client exits with an error message. A client which is connected to the server asks for the available audio sources first. After that it may request up to 20 separate streams. For each stream a new UDP socket is opened on the client side. The audio processing driver has again been implemented based on JACK. For cross-platform purposes audio drivers based on RtAudio [6] and PortAudio [7] have also been added.

3.5.1 Jitter buffer

In the internet, as in any packet-switched network, packets can be lost or delayed, which in real-time applications is the same when a certain delay is exceeded. Due to its best-effort nature, the internet protocol may even result in duplicated packets. Variations in the transmission delay are taken into account by a jitter buffer which collects packets as they come in and delivers them in order and thus ensures continuous playout of the speech data. Our jitter buffer implementation firstly collects the incoming packets in a separate programming thread and, secondly, delivers all available descriptions of the current frame whenever the data of a frame is requested by the decoder. The size of the jitter buffer is initially set to 500 ms. It grows exponentially when a packet is received which cannot be buffered due to its sequence number being too high. Basically, this means a change of size is not supposed to happen more than once. This is, however, also influenced by the clock skew compensation algorithm which is described in section 3.5.3.

3.5.2 Packet loss simulation

To demonstrate the effect of packet loss on the client side, even if actually not a single packet is lost in the internet, a packet loss simulator has been added. Independent random packet losses are simulated. Packet loss is applied after the received packets have been delivered by the jitter buffer.

3.5.3 Clock skew compensation

On the serving side the senders are synchronized by JACK which is driven by a clock as part of the sound card. On the client side another clock is used to drive the audio processing. Since these two clocks are not synchronous it is highly likely that they do not run at exactly the same rate. With typical clocks this skew can amount to up to $\pm 0.5\%$ [8].

Assuming the audio data on the client side is processed faster than on the sending side, the jitter buffer will eventually run empty. If, on the other hand, packets are sent faster than a client plays out the decoded audio data, the jitter buffer will overflow. We tested our client on serveral systems and in some cases we saw the jitter buffer run empty in only 40 seconds. To counter this problem, we added a cubic spline interpolator as suggested in [8]. Depending on the jitter buffer fill level, which was low-pass filtered for this purpose, the playout is accelerated if the level is too high and slowed down if it is too low. Fortunately this interpolation does not degrade the perceived speech quality.

4 Usage

The client side of the demonstrator is implemented as a graphical user interface (GUI), available for GNU/Linux as well as Microsoft Windows operating systems. Fig. 3 shows a screen shot of the demonstrator operating on Linux.

The parameters of a stream that is to be added can be configured in the upper part of the GUI. Beneath this part, the simulated loss rate can be controlled. Every stream is shown as a row in the table on the left side. In this table, several stream parameters are displayed. By selecting one of these rows, the active stream is determined. At any time, there is exactly one active stream of which the audio data is being played out. The history of the active stream regarding all packet loss, both possible losses in the network and simulated losses, is displayed on the right side. The bit rate actually received is drawn in red. The blue graph displays the residual packet loss ratio which counts those frames for which none of the descriptions has been received.

5 Conclusions

Multiple-description coding is a technique to improve the perceived quality when sending multimedia data over lossy packet-switched networks like the internet. A software architecture for real-time transmission of MD coded speech signals over the internet has been developed. The system allows experimental comparison of different configurations of MD speech codecs under varying channel conditions. It forms an extensible framework for further experiments on MD speech and audio coding in general.

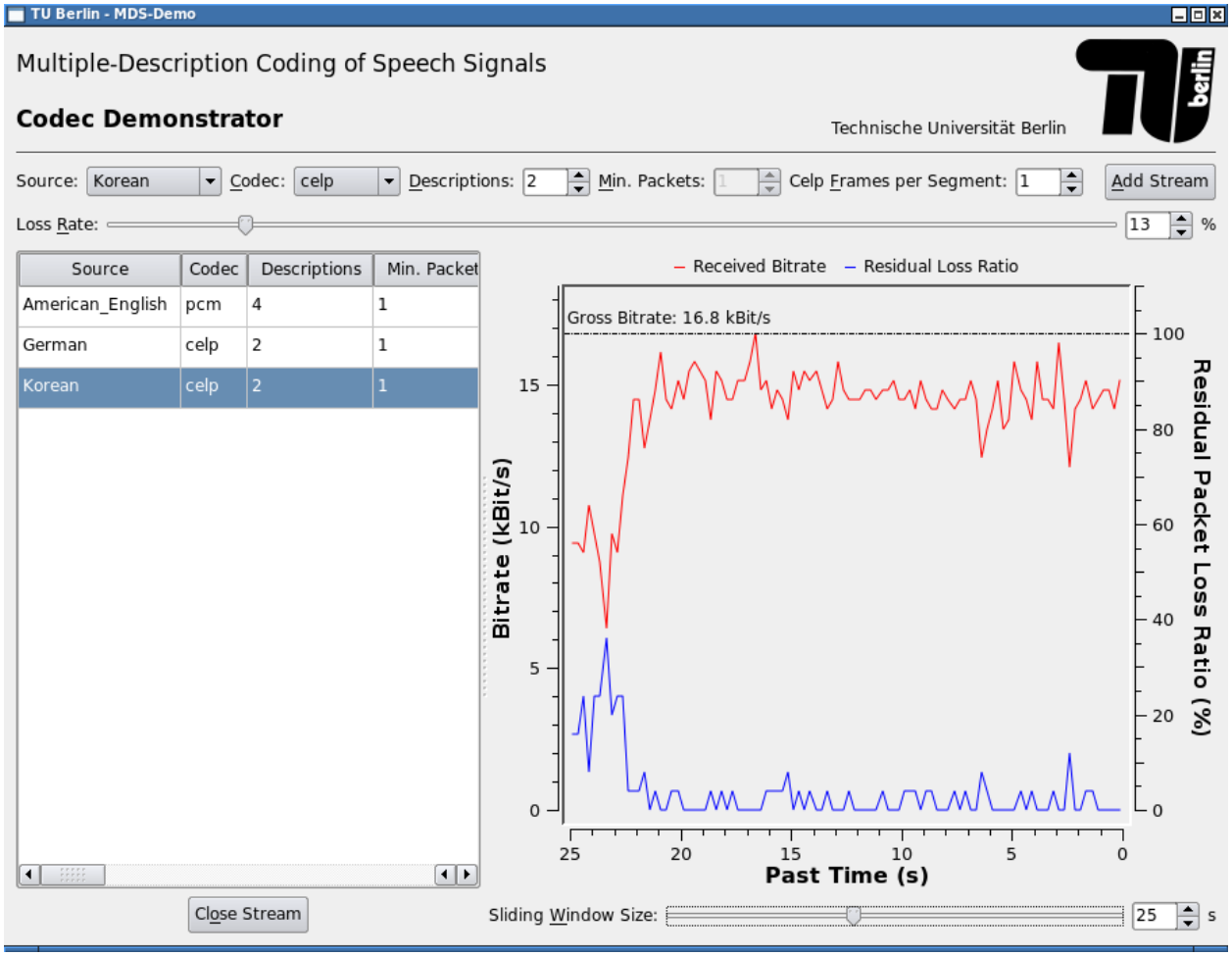

Figure 3: Screen shot of the demonstrator

6 Acknowledgements

The authors would like to thank Rubén Heras Evangelio for providing the initial implementation of the jitter buffer as well as helping with porting the demonstrator to the Windows operating system.

This project was funded by the German Research Foundation (DFG).

References

[1] V. K. Goyal. Multiple description coding: compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, 2001.

[2] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), April 1997.

[3] Python Software Foundation. Python programming language – official website. http://www.python.org. last checked: 05.01.2007.

[4] P. Davis et al. Jack audio connection kit website. http://www.jackaudio.org. last checked: 05.01.2007.

[5] K. Vehmanen. Ecasound website. http://www.eca.cx/ecasound. last checked: 05.01.2007.

[6] G. P. Scavone. Rtaudio website. http://www.music.mcgill.ca/~gary/rtaudio/. last checked: 07.01.2007.

[7] R. Bencina et al. Portaudio website. http://www.portaudio.com/. last checked: 07.01.2007.

[8] T. Trump. Compensation for clock skew in voice over packet networks by speech interpolation. In *Proceedings of the 2004 International Symposium on Circuits and Systems*, Vancouver, Canada, 2004.