Ruben van der Ham

Arno Bakker

Systems programming

17 October 2018

WAV audio streaming over UDP in C

Requirements

1. Both server and client have to be robust enough to cope with some packet loss when both applications enter the streaming procedure,
2. The server has to work for multiple sequential streams,
3. Server and Client need to communicate via a protocol

Implementation of requirements

Packet loss detection and re-sending via 'messages'

To cope with possible packet loss both server and client use so called "messages" to communicate instead of packets. A message can be seen as a TCP-like transaction of UDP packets. For example, sending a message consists of the following steps:

1. Send the packet containing the message
2. Receive the an acknowledgement packet containing "ACK"
3. Send another "ACK" packet back.

When packets are lost during the transaction, both parties know packet(s) are lost. The default time-out time for receiving packets (e.g RST_ACK, ACK or other messages) is 500 milliseconds. Thus, when implemented correctly, neither the server nor client can be at another state in regard with the protocol.

Multiple sequential stream support

The server is designed to loop until a SIGTERM event has been invoked. In this loop the server will wait continuously for connections with clients. When a client successfully connects, the server will start the streaming setup procedure and wait for request properties and starts streaming. When the stream has ended, either successfully or unsuccessfully, the server continues to wait for new connections.

Protocol Specification

The protocol consists of multiple stages of connection.

Connection Setup

Clients that want to connect open a connection with the server by sending a "HELO" message. The message will be replied by another "HELO" from the server.

Request handling

The client has to send a control packet containing the *filename* and the *library name* of request.

On successful requests, when the server loads both the library and the audiofile, the server will reply with a "CTRL_ACK" message first. The second message, after the "CTRL_ACK " has to be a message containing the audiofile's properties.

On invalid requests, when the audiofile and/or the library could not be loaded, the server will send an error message.

Audio streaming

The server will send the contents of the WAV audio file to the client in packets of at least 1024 bytes each. The server will start sending as soon as it can.

optional: The server will stop when it receives a "RQST_STOP" packet. When an "RQST" packet has been received the transfer will start again.

When the entire content of the WAV file has been sent, the server will send an "RST" packet.

RST procedure/packets

RST packets are packets with the contents "RST".

The server will send an RST packet when the entire content of the file has been streamed. Also both client and server can send RST's at any time to notify the other party that the connection has to be reset/closed. When an RST packet has been received, the receiving party has to reply with "RST_ACK" in order to let the other party know the RST packet has been properly received.

Protocol implementation

Connection setup

Client will try to send a HELO message to the server. On failure it will retry this procedure for another 2 times before quitting.
Server will wait for the HELO message and reply with a "HELO_ACK".

Request handling

The client sends a control packet containing the the request properties in the following format "datafile,[audio filename],libfile,[library file name]".
The server parses the properties of the control packet and tries to load the properties of the stream request. The server sends the audio file's properties back, if applicable, in the following format "[sample size],[sample rate],[nr of channels]".

Audio streaming

The server will send the contents of the WAV audio file to the client in packets of 4096 bytes each. RQST and RQST_STOP messages are not implemented.
Because no buffering is used the amount of packets that are received have to be matched with the sample rate of the audio device on the client. Therefore the client will wait the sample rate* the amount of channels (NOTE: each buffer is 4096 bytes) microseconds, in order to match the amount of data written to the audio device with the sample rate.
When the entire content of the WAV file has been sent, the server will send a RST packet. When an RST packet is received instead of binary data, the client will send a RST_ACK and quit the streaming state.

RST procedure/packets

Both send- and receive message methods contain checks to detect RST and RST_ACK packets. The RST packet is a single packet opposed to RST_ACK which is a TCP-like message (with 2 ACKs). This approach makes implementation easier. However, RST packets can get lost. Therefore RST packets could be sent for at least 3 times before the RST initialization procedure fails completely.

Known limitations and issues

- When quitting the program as client, the server will reach a timeout before the RST has been received. This is most likely due to an anomaly in the send_message function.
- Because buffering is non-existent audio might be silent when multiple sequential messages are lost.
- The tokenization implementation of the control messages relies on strsep(); a function which is not POSIX compliant[1], and therefore it has portability issues.
- The TCP-over-UDP implementation has serious performance issues, since only one packet can be received properly every 1.5 RTT. This is not a huge problem when running both the server and client locally. Though intercontinental connections (300ms RTT) would have a horrible bandwidth of only ~4KB per second with this approach.

References

1. https://www.gnu.org/software/libc/manual/html_node/Finding-Tokens-in-a-String.html