Storyboard for Project

by Ray Alfano

Network Computing, Spring 2013

Project title: Synchronous Multi-User Music Production Via the Internet

Project Implementation at Current Stage

- Slides 3-4 briefly discuss the core program code that is runnable right now.
- In this draft submission I have included my documents and the essential server/client code. The statistics-generation and RTCP-derived dynamic configuration code are still in progress.
- Slides 5-14 are written explaining my full project and encompass all completed functionality and what still needs to be done.

Explanation of the Runnable Code for Draft Submission

- To run the RTP version:
  - Within Ubuntu Linux open folder 'rtp'
  - Run pcm-server.py and pcm-client.py
- To run the TCP version:
  - Load from Linux (tested on Ubuntu)
  - From folder newImplementation:
    - Execute command "python main.py"
  - Two servers and two clients will spawn, playing the sample PCM input over HTTP.

Libraries Required for the Project

- I re-implemented most of my project after insurmountable dependencies on Mac OSX.
- My re-implementation on Linux requires a minimum of dependencies, see "dependencies.txt" in the projmain folder for a simple set of commands to acquire them.
- The statistics generation and dynamic portion of my final project will require a few more dependencies that can be easily acquired. Instructions will be included.

Requirements for the Final Project

Input: Provided sample wav files or valid PCM audio stream such as microphone input.

System: Currently only Ubuntu Linux is supported due to Mac OSX config issues.

Users: Two individual users, or simulation of multiple users by supplying a wav file as input for one of the clients.

Connection: TCP for wav file stream test, RTP in the joint performance program.

Program Components at Initialization

Server: pcm-server.py

- Controls program runs.
- Sends raw PCM data from system input.
- One instance required for each user.

Individual clients: pcm-client.py

- Receives raw PCM data and plays back.
- One instance required for each user.
- Provides RTCP response data used for dynamic configuration of the channel.

Program Runtime Part 1: Setup

- Run both pcm-server.py and pcm-client.py.
- In the current implementation pcm-server.py will simply send a fixed test signal to verify the RTP packet transfer.
- In the final version there will be a local test version with fixed WAV input and a dual-input live version for my presentation.
-

Program Runtime Part 2: Execution of Audio Transmission

- Input is broken into frames for sending over the network over RTP. The steps involved are:
    - Receive raw PCM data from user input
    - Send raw PCM over GStreamer RTP service. Send RTCP data as well for stats.
    - Play back incoming audio on remote client. Detect runtime statistics from RTCP data and adjust quality of audio accordingly.
    - Goal is emphasizing real-time audio.

Program Runtime Part 2.1: Description of Sample Rates

- The ability to throttle audio bitrates and sample rates in the conversion from PCM to Opus formats prior to network transfer was removed because it introduced far more latency than it saved.
- This will be discussed in my final report, but it is not a significant portion of my current implementation (and it was not part of my proposal).

Program Runtime Part 3: Transmit Audio Frames to Other Client

- PCM frames are sent in real time over a GStreamer pipeline using RTP.
- RTCP data is communicated at the same time between server and client.
- The RTCP data is used to generate statistics and adjust the PCM sample rate at which the audio data is sent.
- The final version will log latency and complexity data from RTCP and result in conclusions about "best fit" audio quality.

Program Runtime Part 4: Output of the Transmitted Audio Frames

- To avoid echo, a musician does not hear their own input from their local system output.
- The program will directly output PCM data to the local system output, which will introduce a small amount of latency that my program will try to anticipate.

Program Runtime Part 5: Conclusion

- The program currently lacks statistics generation and the adaptive RTCP code that I am reimplementing from my finished Mac code. The resulting code will be much faster and simpler and avoids dependencies.
- The code currently uses CLI to run and takes keyboard input to terminate. The final version will have at least a simple GUI to control host setup, runtime and termination.
- See submission note for questions.