Michael Evans, Felix Tiet, Brian Xu

CS 4283 Networks Group 5 Final Report

Introduction

Our project is a socket-based video streaming server and client. It takes a video file on a server, splits it up into a set of video frames and audio frames, and sends those to a client. The client then reconstructs the frames into a video with synchronized audio. The client functions as a typical video player, providing useful features such as pause, rewind, and fast forward.

We wanted to choose a project that expanded on the topics we learned in class. Because video streaming is such an integral part of everyday life, we chose this project to better understand the intricacies and obstacles surrounding video streaming. With the popularity of things like YouTube, we thought it would be interesting to explore a very commonly used and practical example of sockets and networking in general.

Background

We approached the problem using Python’s TCP sockets. This allowed us to ensure data was reliably transmitted and received. We used OpenCV to split the video into frames and to play it back, and we used PyAudio along with standard Python WAV support to play our audio. Our architecture is structured as such:

Video

Client

Server

Audio

We open up two sockets between our server and our client. One socket sends video frames, and the other sends audio frames. The server is single-threaded; for every frame of video there is a frame of audio sent.

The client, however, receives the data in two separate threads, one for video and one for audio. Each thread utilizes an individual TCP socket. These threads constantly receive the data and append it into buffers for video and audio, allowing for us to start the video quickly and load it as we go along.

In the main thread, after we being to receive data, we wait for the buffer to build up a bit before starting. We then activate our two playback threads: once again, one for video and one for audio. They’re synchronized so that the audio can’t get ahead of the video. If the playback catches up to the buffer in either, we wait for more data to load in before continuing.

Results

It’s hard to describe our results with a screenshot, since the project is based on video and audio. Here is an instance of our video player running.



Overall, we successfully transmitted and played back synchronized audio and video. The majority of our work revolved around synchronizing audio and video. At first, we attempted to have an array of arrays to implement the synchronization of audio and video. After this idea failed, we instead moved on to the idea of utilizing two sockets between our client and server. This way, it was easier to synchronize the audio and video. We also implemented a buffer to help with synchronization. Without the buffer, it would be possible for either audio or video to get behind the other while its waiting to transmit and receive the TCP packets.

After running the server and connecting to it with the client, the sample music video plays. The client also supports our desired features with key presses – `q` for quit, `p` for pause, `a` for rewind, and `d` for fast forward, as shown below.

Double click the images to run the gifs.

Example of normal running instance:



Quit:



Pause:



Rewind:



Fast forward:



We also found that hardware speed had a significant impact on our design. We ran the video player on three separate machines with the same exact python instance. Each one ran at severely different speeds. So, the video playback depends on the quality of hardware, where computers with more computing power tended to play the video too quickly, while older computers with dated hardware played the video too slowly. Packets are transmitted and played back more quickly with better hardware.

Team Contributions

We didn’t divide work very cleanly, so almost every part was done as a group. The only notable examples of individual work were Felix’s work on packet padding (that ended up needing to be cut for a different approach) and Michael’s work on the initial research on OpenCV. Michael worked on most of the base of the code, while all three of us worked to implement the desired functionality and fix errors in the code.

Pending Status

While we implemented most of our desired features, there are a few things we would have liked to include. In regards to the TCP socket, we currently send data reliably, but no explicit retransmission occurs if there is data corruption or dropped packets. While this was not a huge issue running on local computers, this could be big problem running on a larger network.

We would also like for the client to be able to specify a specific file, and connect to a server other than localhost. This functionality would likely require user input on the client side. We weren’t able to fully complete this service, but it likely would have been minimal additional code.

Finally, some additional features that would be nice to add in future work would be allowing for variable playback speed (0.5x, 2.0x, etc.) and skipping to a specific time in the video.

Conclusion

Overall, we learned about the complexity of sending and receiving data through sockets, and how to encode and decode the data. We ran into some issues with data corruption, and had to explore how strings and integers were converted into bytes. We also had to revisit our architecture design numerous times, shifting from a basic client and server model with a single socket to a multithreaded client with two sockets.

Despite running into numerous obstacles, we were able to playback synchronized audio and video with support for a few basic video player features.