Safiyy Kanjiyani

Development Diary

To start off this project, I looked for python modules that could make my life easier by being able to read video files and displaying it while I dealt with streaming over the network. I found openCV which fulfilled my first requirement of reading video files. I also didn't want to deal with making my own UI so I thought of streaming through HTML so it can just be displayed over a browser (which all modern browsers should be able to display a video). However, rereading the requirements for the project, it seems like I can't stream to a browser as it would probably auto-implement pause and play functionality. Looking for another UI program now! I might use Tkinter to program a UI.
Looking further into openCV, I found that it has a display UI for videos WITHOUT play pause controls, so I went with that.

I was deciding between UDP and TCP and I realized that UDP would be better for “real” live streams (like twitch) rather than video streams so I went with TCP for reliability. Later on, I found out that many modern streaming platforms like Netflix and Disney Plus use TCP but buffer the stream so that the retransmissions are unbeknownst to the viewer.

OpenCV converted a video into frame objects. However, you cannot transmit these directly in Python over sockets. Pickle solves that problem by turning objects in Python into a byte stream format which can be sent over sockets. I used pickle on a previous Python project because I made an entire object class called a packet that contained ack, sequence numbers, and data. I didn’t want to program an entire class again so I decided to go with something simpler. I wanted to transmit the size (in bytes) for the frame so I used a package named struct to act as a header in which I can put the data size. The client reads the first 4 bytes (the header with the data size). It then receives until it has received a single frame. The client then uses pickle to convert from a byte stream into a frame. It then uses openCV to display the frame.

Now that the videostream was working, I had to program pause/play functionality. In the openCV UI, it can grab key inputs. So I sent it to stall receiving whenever it gets a spacebar input and continue when it gets another spacebar input. Also, I added q input for quit since quitting by hitting the close on the window wasn’t working.

I didn’t get time to work on extensions but this is how I would go about implementing them:
For encryption, I would use the byte stream and apply an encryption algorithm to it before sending it. The client would get the byte stream and apply a decryption algorithm to it, then turn it into a frame object. I read somewhere that I can’t directly encrypt a frame file (since it’s kind of like an image) because they’re subject to lossy compression and I would need to download and use a video codec to go about encrypting frames.

For the backwards and forwards extension, I would use openCV to read like 15 seconds worth of frames before and after. The client would send a request to the server to go back 15 seconds of frames.