CS 396 - Final Project

8th May 2021

Param Kapur

This project uses the Real-Time Transport Protocol (RTP) and the Real-Time Streaming Protocol (RTSP) to create a server that sends video data to a client that digests and controls that data. **RTP** is used for the actual transmission of video files and "is a network protocol for delivering audio and video over IP networks". It is outlined in RFC 3350. On the other hand **RTSP** is a network control protocol designed for use in entertainment and communications systems to control streaming media servers. The protocol is used for establishing and controlling media sessions between endpoints.

I developed this project by first considering the function of the server. The server needed to do two things. First, it needed to be able to take a video file, split it up frame-by-frame, convert that frame into usable data format (make it a send-able payload), wrap the payload in a RTP packet and send it to the client. Second, it needed to follow RSTP to communicate control instructions (play, pause, start, etc.) with the client.

I started by chipping away at these requirements somewhat from the middle. I looked up what an RTP packet should look like from the RFC and started to write a RTP Packet class. This class essentially serializers the headers and payload into a binary format that can be sent over the wire, and deserializers this binary format on the client end to get the data back. An interesting challenge here was that I could not simply encode a string and send it across, because the payload was obviously not in UTF-8, so I had to get into the weeds of formatting and working with binary data.

Once this class was built and tested with random data I went on to write a class, VideoStream (strictly speaking there is no need for this to be a class, however I wanted to stick the nice OOP I'd written). For this one I had to understand how to move through MJPEG data frame-by-frame. I was inspired by multiple tutorials and code stubs on the web. Notably,

> https://stackoverflow.com/questions/16119266/grab-one-image-frame-from-mjpeg-stream?rq=1
>
> https://gist.github.com/TimSC/6546281
>
> https://github.com/janakj/py-mjpeg

Now I am able to take a MJPEG file and encode it into frame-by-frame packets. Then decode that binary data into usable header and payload information. It is at this time that I started thinking about how the client would access this data.

I was pretty convinced on using Tkinter for this project before I started because I've used it before and its a quick way to get a GUI going. Once I had the packet data I focused on unpacking one packet and rendering the first frame on the GUI. I used PIL to convert the data back into an image and Tkinter to render it on the canvas.

This is where I hit a wall and had to implement the RTSP side of things before moving forward. I wrote the client side interface to send messages to the server on button clicks and wrote the server side interface to digest these messages and perform actions. This is where after a ton of digging around I saw that it was possible to start the RTP "process" in a thread where I would be able to easily start and stop it without having to mess around with control variables etc.

From here I was able to write the send and get RTP methods to continuously send the RTP packets in a loop to the client from on the server side and digest and render them on the client side.

This is the homestretch! I added some simulated jitter and packet loss to the situation to ensure that David couldn't immediately call it out for not being to robust (at least not initially).

References:

https://datatracker.ietf.org/doc/html/rfc3550

https://stackoverflow.com/questions/16119266/grab-one-image-frame-from-mjpeg-stream?rq=1

https://gist.github.com/TimSC/6546281

https://github.com/janakj/py-mjpeg

https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol

https://nickvsnetworking.com/pyrtp-simple-rtp-library-for-python/

https://www.geeksforgeeks.org/start-and-stop-a-thread-in-python/