Aatish Varma, Hans Shin

Final Project Report

Original Proposal:

We plan on implementing a video sharing application where a sender will send a video file to the receiver, which will then display the video to the user upon receiving the entire file. For this project, we will be writing a sender program and a receiver program. Both will be written in python. Our implementation of the sender will probably involve a "sender socket" connecting to a receiver's port (`sender.connect()`) and then sending the data. The receiver will receive all of the data (`receiver.recv()`) and then display it to the user (we don't know how this displaying functionality will be implemented at the moment). We'll be using the OpenCV python library to record video data and send and receive that data using the socket library.

Implementation:

Sender.py:

We pass the video file to a `VideoCapture` object, which allows a video to be read frame-by-frame, using code like in the line `(isFrame, frame) = video_capture.read()`. We check if each frame is read properly by making sure the `isFrame` value is equal to `True`. If there are no more frames to send, then `isFrame` will equal `False` and we stop sending frames. We then interface with the socket by first serializing each frame (because TCP uses bytes) using the `pickle` library, and then creating a header for the resulting byte stream using the `struct` library. We then send the header and byte stream through the socket. Once we are finishing sending all of the frames of the video, we close the `VideoCapture` object and close the socket.

Receiver.py:

We parse through the stream of data that our receiver socket receives by dividing the byte stream into discrete chunks using the information stored in the header of each serialized frame byte stream.

```
header = bytes_buffer[:header_length]

bytes_buffer = bytes_buffer[header_length]


frame_length = struct.unpack("L", header)[0]
```

The header then tells us the length of the frame's byte stream, and so, using that information, we parse the stream to get just that frame.

```
frame_data = bytes_buffer[:frame_length]
```

We then unpickle the frame_data and add it to the frames list we created. Once we've gathered all the unpickled frame data, we play the video by using OpenCV's `imshow(..)` function, using the `waitKey(..)` function to control frame rate. Every 33ms is equivalent to 30 frames per second. After playing through the entire received video, the program ends.

Running the Program:

1. Run receiver with the command python3 receiver.py [port number]

   E.g. `python3 receiver.py 9001`

2. Run sender with the command python3 sender.py [port number] [path to video file]

   E.g. `python3 sender.py 9001 citizenkane.mp4`