Video Streaming Design Report

Wagner Colodette and Karl Boatright

Problem Definition:

To design a simple network application in Python that will take a video file as input and send it over a socket to a receiver, who will then be able to stream the video as it is being received.

Design Requirements:

The necessary design requirements include one program developed to send a video in small chunks, and another program developed to receive those chunks and display them in the form of a streaming video at the specified endpoint. Ultimately, we decided to use the OpenCV API to develop our solution for streaming video files. OpenCV is a robust API that includes functionality for facial and object recognition, but also provides solutions for a problem such as video streaming. Additionally, we used *imutils* as a solution to manage frames per second rate and resizing of the video on the receiver side. The program *send.py* uses the Python socket library to create a TCP socket and connect to the proper address and port to send video data. We use *VideoCapture* to create a *VideoCapture* object that takes the argument of the video name that we are using for testing purposes, which is named “me_at_the_zoo.mp4”. The VideoCapture object is then read, using the function *read()*, which returns a boolean value, “*True*” if read correctly, “*False*” if otherwise, as well as the frame that has been retrieved. Next the frame is serialized, packed into 64 bits, and sent over the socket to the receiver.

The receiver program, “receive.py” works in such a way that a socket is created, binds to the designated port, and listens for any data being sent to it. Once data is received by the receiver, the receiver will continue to process the data and display the frames that it has received until there are no more left. We resize the image using *imutils*, and display the image, we use the OpenCV function *imshow()*.

The protocol that we are using for this project is TCP protocol. The argument SOCK_STREAM given when creating the sockets in both *send.py* and *receive.py* designates the type of protocol that will be used for that socket. We

chose TCP protocol because we wanted to ensure that all the packets were being received sequentially for the sake of observing such a video streaming solution for academic reasons. More practical implementations may choose to use UDP protocol, because it may be more important for the user to have a video that is showing the latest frame, possibly because they are viewing some type of live event and want their feed to be as close in time to when the actual event is occurring rather than see every single frame. Typically, a dropped frame here and there will not impact the user's experience negatively.

Citations:

[1] Adrian Rosebrook: Faster video fil FPS with cv2.VideoCapture and OpenCV - https://www.pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/
[2] OpenCV 2.4.13.7 Documentation - https://docs.opencv.org/2.4/index.html#
[3] OpenCV-Python Tutorials - https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html
[4] Python3 pickle lib - https://docs.python.org/3/library/pickle.html
[5] Python3 struct lib - https://docs.python.org/3/library/struct.html