Final Project Report

1. Summary

Our project is called SFA-Cast, and it works to display one computer’s monitor onto multiple computers monitors. It uses UDP Multicast to project to all the machines connected to a given switch. We chose this based off of the old program Tight-Receiver and Tight-Projector so that students could see the teacher’s screen on their own devices.

SFA-Cast started out as a TCP “multicast” and we attempted to convert it to UDP Multicast. We decided to start with TCP so that we could test the differences. However, we fell short on getting the UDP Multicast to work across the network onto other computers because we were using the wrong IP address. Although the UDP protocol did not work, the TCP worked great and that is what we presented to the class.

1. Documentation

For our own documentation, we used GitHub.

We did an entire documentation document and I have attached that.

1. Network Application Protocol

We developed a server program that “screen shares” by taking a capture of their screen, converting it into bytes, and sending it as a packet to the client or clients. The client then reads the bytes back into the image and displays this image using pygame. Threads and loops keep the feed fast enough that there is not too much noticeable lagging.

* 1. TCP

We started out with TCP because it was simpler to implement, and we wanted to test the differences between TCP and UDP. The TCP “multicast” that we created was fairly fast, worked without much lagging and took more time than we had anticipated.

How it works:

When the server accepts a new connection, it sends the server broadcast resolution to the client. Then the retrieve frame function is called on a new thread, so the server can handle numerous clients. The retrieve\_frame function will grab each frame and send it to the client. First it sends the length of the pixel array for the frame, then it sends the pixel array itself. An advantage of TCP is being able to pass something large to the socket.sendall() method and the whole thing will be sent without any extra work whereas with UDP we had to break the pixels array apart and send each individual part.

* 1. UDP Multicast

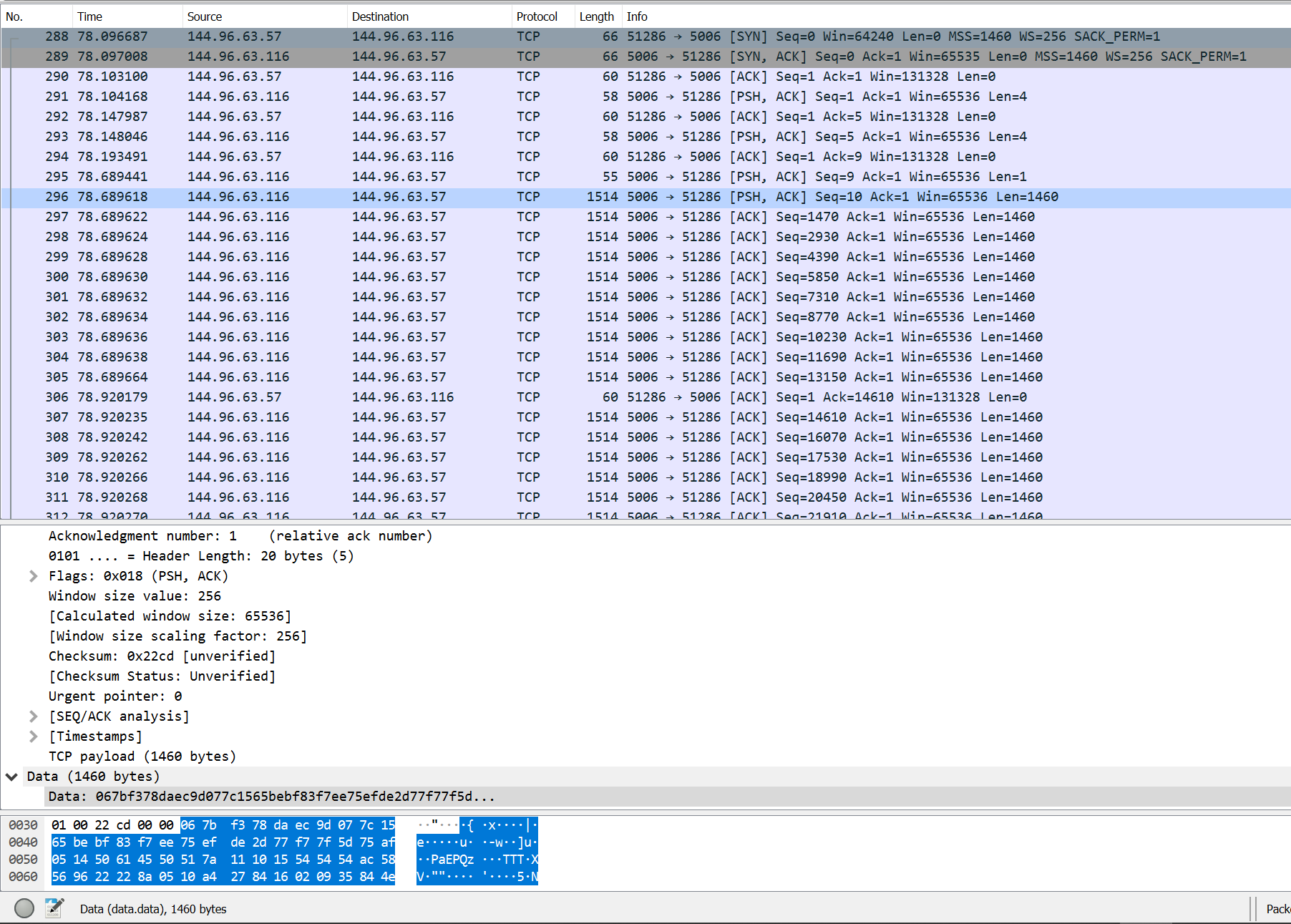
After we got the TCP to work, we moved onto the UDP Multicast socket. The sending, receiving, and socket options are all very different from the TCP protocol. We got it to Unicast, but never to Multicast. This could have been because of IP or Network issues since we never tested off the school’s network.

How it works:

First the server resolution, width and height, are sent as two separate packets. This is to help control window size on the client end. Then the length of the pixels array (each frame) is send. The reason for sending the pixel array length is because we break the array up into 4KB chunks and the client needs to know the length, so it can tell when each frame has been fully transmitted.

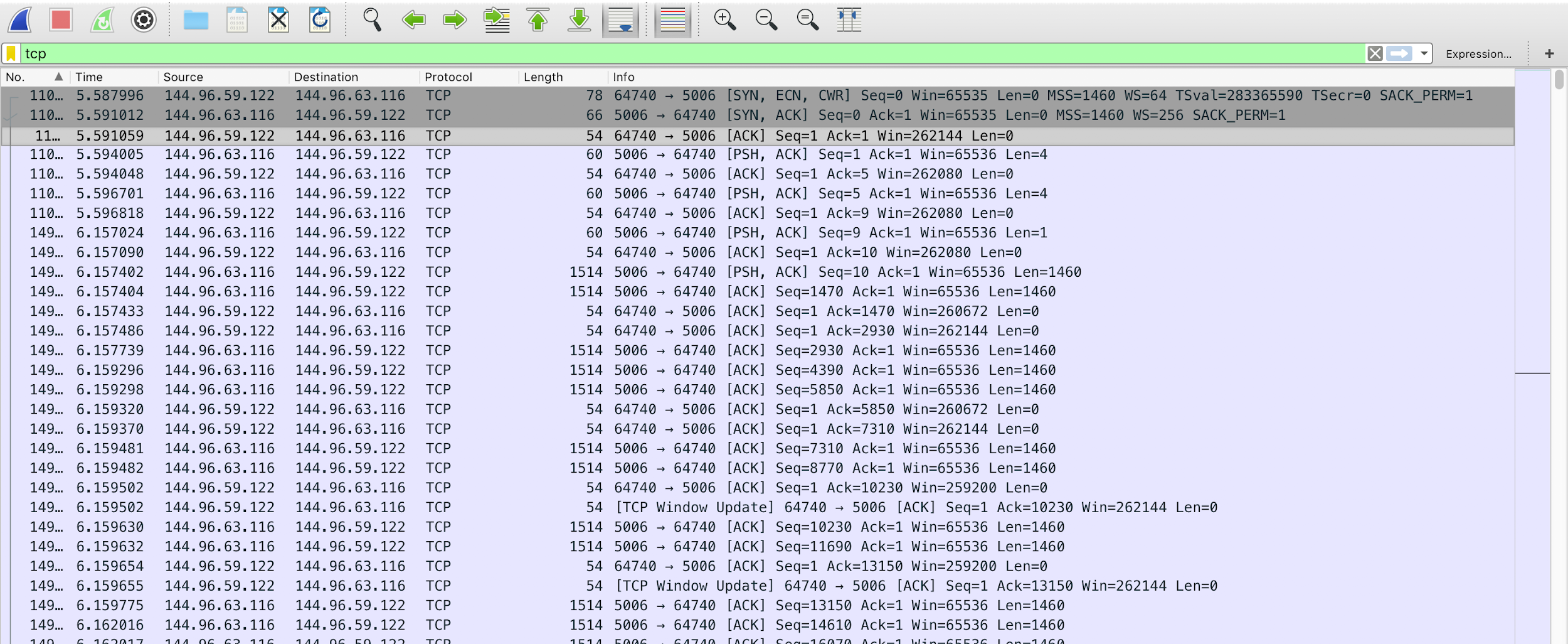
1. Network Usage
   1. TCP Server

We can see the server completes a 3-way handshake on the receiving side and establishes a TCP connection with the client. Then the server is seen sending data and receiving confirmations from the client



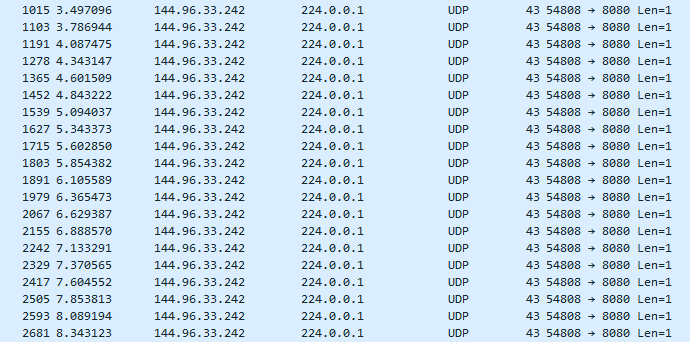
b. TCP Client

The client initiates a 3-way handshake which the server accepts.



c. UDP Multicast Server:

For the UDP Multicast Server, you can see the packets are being sent to port 8080. We were getting roughly 40-60 packets per image, and because of this (after presenting) we decided to segment the packet into bigger chunks so that there would be less packets and also a reduced chance of packet dropping or packets being out of order.



1. Challenges

* UDP Multicast:

UDP Multicast was a huge obstacle for us. I read several different websites, typed several files of code that didn’t work (you can see by my GitHub use that files were pushed for testing then removed), and spent several hours and days on it. I still believe our code is right and that the IP addresses we used were wrong, especially after talking to Luke and James.

I would have to say that this was by far the hardest challenge that we faced. It was also something we fell short on.

* Operating Systems:

Having a Mac and running MacOS was an obstacle for me because of testing the code and resolution differences. Because of this challenge, I added MacOS capabilities to the GUI so that the GUI could run on both Mac and Windows. We also had the Server and Client send their resolutions to each other (TCP) because of my different resolution.

Another issue that we encountered with me being a Mac user was that I had made a GUI function where you could change the screenshot path. Because I did this with my machine in mind, it did not work on the Windows machines and I had to go back and add Windows paths and functionalities to the button functions.

* Screenshot Implementation:

This is something that could also go under Operating Systems. The screenshot function did not work on my MacBook when the screenshot button was F12, however (after presenting) I changed it to the S button and it worked. So, there is some kind of issue with MacOS and the F12 button. I had even tried command + F12 and control + F12. However, this is a challenge that we were able to resolve.

1. Testing

* Libraries:
  1. OpenCV and ImageGrab(PIL)

We tested these libraries to continuously grab frames and until the esc key was pressed. We stored the ImageGrab.grab() as an image, then stored it into an image array. OpenCV was the library we used to display the screen and frame. This was all in a while loop which kept the image refreshing.

However, this method was too slow for our liking, even after we multithreaded it. It looked as though it was “lagging” because of the wait time between new images.

* 1. MSS

Because of the slow speed we were experiencing with ImageGrab and cv2, we tried MSS to test whether it was the libraries that were slow or if the socket was the problem. MSS was faster even before we added threading to it. Instead of displaying with cv2, we used pygame for its update functions. The functions that come along with pygame also made it much easier to implement our screenshot function.

* Transport Layer Protocols:

1. TCP

We originally created a TCP socket to test that everything was being received and the connection was there on both ends. Although we started with TCP, we always had the intentions of swapping it out to UDP Multicast. We thought it would be cool to see the differences in speed, functions, and other factors even if it meant spending more time on the project. We achieved a TCP “multicast” that we tested with 6 clients.

1. UDP Multicast

UDP Multicast was the obvious choice for our program because of the ability to communicate with all the devices connected to a certain switch. Because of this, the teacher could broadcast their screen to each student in the classroom. It is also faster because neither connection nor acknowledgement is needed from the client for the server to send out packets.