Kevin Yin and Devansh Panirwala
CS 422 - Computer Networks
Final Project Report

Title

Correlation between frame rates, data compression and resolution for UDP/TCP screen sharing

Problem Definition:

Screen sharing software uses many different methods to allow sharing a screen remotely from a server or user acting as server with other user(s) as client(s) who may be able to provide inputs as feedback. For this project we will be exploring the technology behind Screen Sharing application and attempt to build a prototype version of the application. Our goal is to study the different aspects of this application like TCP/UDP, correlation between data compression and frame rates.

Motivation:

In our world, technology is used in many different ways. One particular way is to allow multiple individuals to communicate with each other from a distance over a network. Currently, throughout the world, the majority of the population is under lockdown within their homes due to the pandemic, COVID-19. Universities, companies and schools are forced to be closed to help prevent further acceleration of the spread of the virus. However, activities that are done within these organizations are not hindered due to the force shutdown. This is strictly because of online remote communication. Online remote communication is currently an essential part of what is keeping the world on track. Schools and universities are currently resorting to online screen sharing methods that allow teachers or professors to teach in an environment such that students are able to learn and engage freely outside of the class.

Related Work:

Link:
https://www.researchgate.net/publication/335866573_Real-time_Screen_Sharing_Using_Web_Socket_for_Presenting_Without_Projector
Summary: In this paper, three researchers created a screen sharing application using the WebSocket protocol to share one device's screen to multiple other devices through a computer network (e.g. phone, tablet, laptops and desktops). The motivation of the researchers was somewhat similar to our motivation. They stated that using a projector was nice for an in classroom environment; however, there are times when the projector may not function properly

or be broken. Hence, the person conducting the presentation is directly reliant on the project. Having to move to a remote sharing system is better and voids the dependency on the projector. The researchers presented a diagram of their design of the screen sharing application. What will be different from their experiment to our project is that we will be using a UDP connection instead of a TCP connection that WebSocket uses. We are more focused on having a real time screen sharing platform instead of having a reliable connection but slower screen sharing platform.

Our Work:

For this project, we decided that we wanted to implement two variants of screen sharing applications. The first variant is focused on a TCP connection screen sharing application between one host and multiple users. The second variant is an UDP connection screen sharing application. Our study of this project was to analyze how the two protocols affect the frame rates from a host and each client. Also, we analyzed how compressing data at different levels will affect the frame rates of each screen share.

We used Python 3.5+ to develop our screen sharing application. We utilized several python modules: cv2, numpy, pickle, time, socket, zlib and mss. Cv2 or OpenCV for python was used to help display and resize the screen captures. Numpy was used to convert a PIL image to an array of pixels, so that cv2 can display the screen capture. Pickle was used to help serialize and deserialize objects, or in the case the pixel array, into a sequence of bytes so we can send the pixel array over a network. Time was used to help calculate the frame rates for each client. Socket was used to establish a connection between one (host) to many endpoints (clients) and send data from the host to the clients. Zlib was used to help compress a sequence of bytes to reduce the amount of bytes needed to be sent to all the clients. Mss library was used to capture the screen as a PIL image that will be used to be sent across all the clients. The other benefit of mss library is that it allows faster screen capturing and also allows numerous features like auto-scaling for resolution.

TCP implementation

For the TCP variant of the project, there were two specific implementations that were done. One of the implementations was to have a single thread working in the background to continuously send frames to a list of connected users. Everytime a client makes a connection to the host who is screen sharing, that client's socket connection on the server side is appended to a list of sockets of connected users. The reason we chose to create an implementation like this was because for the other implementation, which will be thoroughly covered later, we decided to create a thread for each connected client; when we did that, there was a hiccup between all the clients frame rates except for one. The second implementation was to create a thread that executes a generic

function that continuously sends frames for each client. We simply did not keep track of the threads created whenever a client joins the screen sharing session; however, whenever the connection socket is closed, we simply kill the active thread.

UDP Implementation

For the UDP variant, a completely different approach was taken due to unavailability of reliable packet transfer. As a User Datagram packet can only contain around 65536 bytes (2^16) of data, the frames which have much more data have to be fragmented before being sent to the client. To achieve this, an algorithm was implemented to divide images into equal size of data segments. This can be seen by observing the code below:

```
IMG_DATA_SIZE_MAX = 65472
img_compress = cv.imencode('.jpg', img)[1]    #compress the image data
data = img_compress.tostring()                #convert to string
size_of_img = len(data)                       #get size of data
count = math.ceil(size_of_img/(self.IMG_DATA_SIZE_MAX))  #divide into segs
```

The IMG_DATA_SIZE_MAX is purposely reduced to 65472 to allow space for headers and in case there is overflow of data.

The next problem faced during implementing UDP protocol for live transmission is the unordered packets and extra garbage buffer. To solve this problem, align_buffer() algorithm is implemented to empty the buffer and wait for the first packet of the next frame of the image. Also the header is attached to the packet to track the sequence number and the size of frame to be received at clients end.

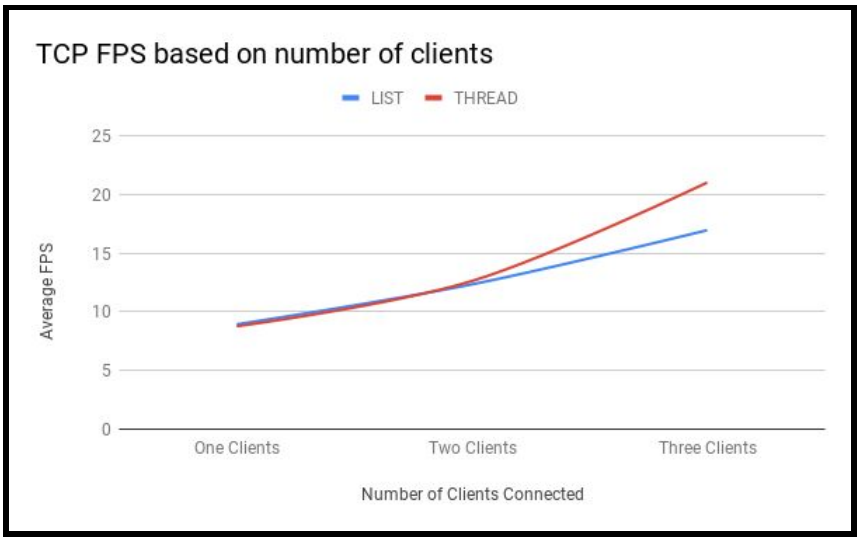
A completely different approach towards implementing multi-client features is taken. As UDP sockets do not allow hand shakes and the use of connect() function, use of methods like “list” and “threads” was impractical. Instead of connecting the client to the server, the server is allowed to broadcast the transmission using the “socket.SO_BROADCAST” flag. This allows multi-casting data to the client without having to spend time in processing of threads.

Conclusion:

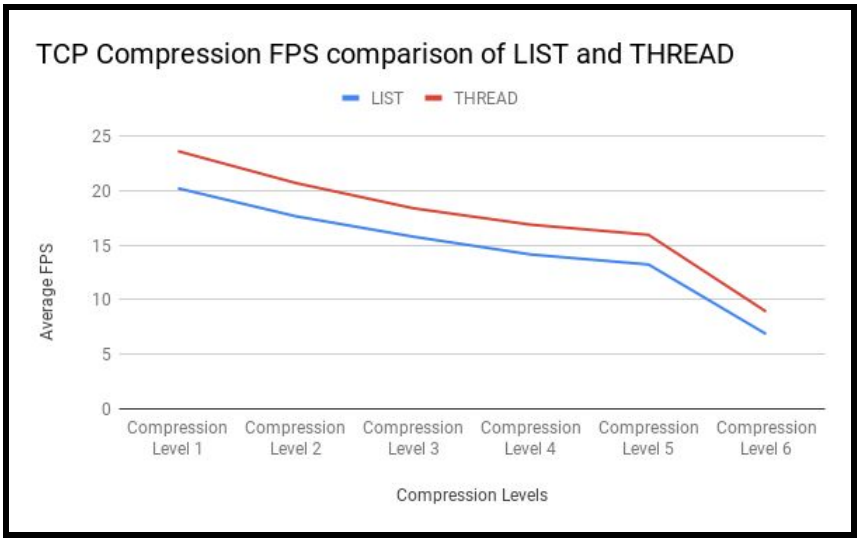
Before concluding on any results, we must say that the data that we produce from this project is strictly dependent on the hardware that is used. There were two different computers used for the data collection. One computer was a 2019 Macbook Pro with 4 cores and the other computer was a Windows 10 desktop with 4 cores.

TCP results

What we found regarding how frame rates are affected on each TCP implementation based on how many clients is connected is that as more clients connect to a host sharing his or her screen, we can see that there is a growing gap between the list and threading implementation.

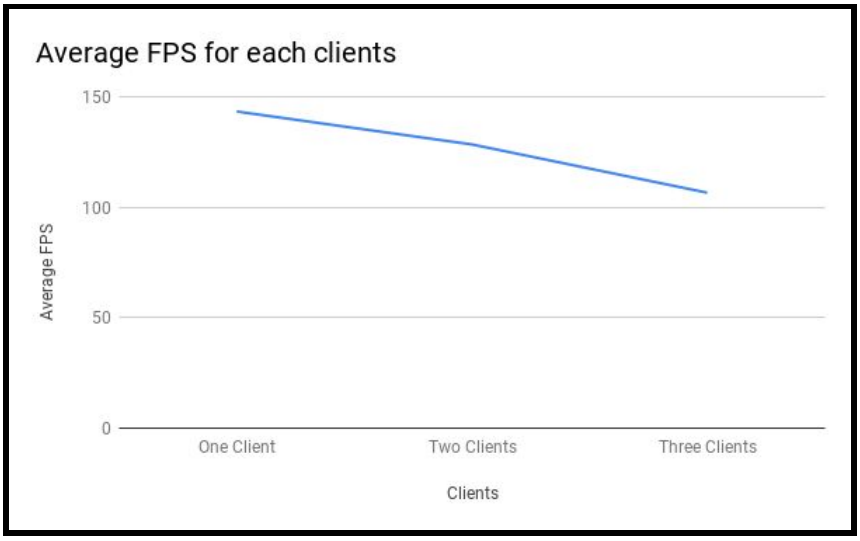


What we found regarding how frame rates are affected on each TCP implementation based on compression levels is that as we increase the compression level, the frame rates for a user drops significantly. The compression levels we used ranges from 1 to 6 where 1 has no compression and 6 has more compression. There is also an overhead in compressing the data that is to be sent; this means that higher compression levels result in longer times needed to compress data.

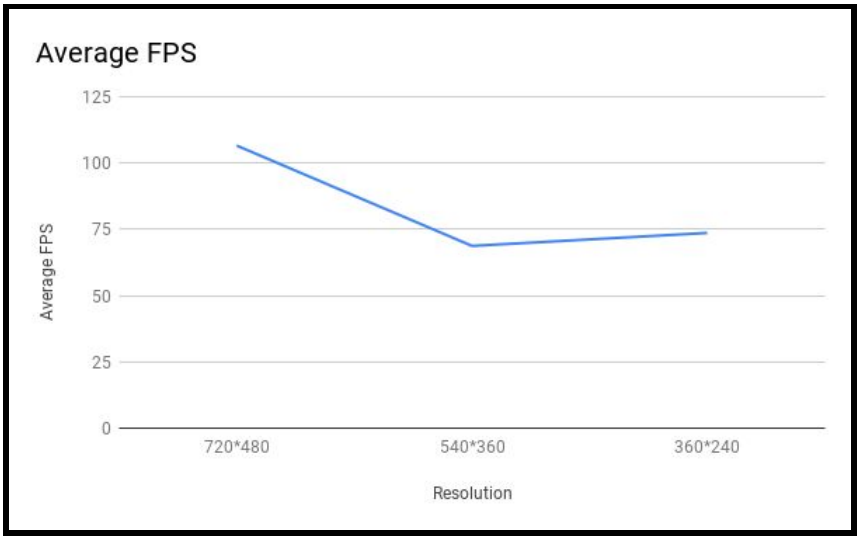


UDP Results

Testing the UDP implementation gave results that confirmed our biases about the speed of transfer of each frame. As UDP protocol does not require to check the integrity of data, the number of frames transferred were drastically increased as seen in the figure below. Also as the number of clients increases, the serve has to transfer more frames to satisfy the data demand which decreases the fps proportional to the number of clients.



The next result that we were interested to see was the relation between resolution size and fps number. We expected that fps would be inversely proportional to the resolution size as higher the resolution size would require more data for each frame. But the result after testing the fps with resolution scale of 720*480, 540*320 and 360*240 was completely opposite as seen in the figure below. The reason for this could be the unreliability of UDP protocol and how data might be lost during transmission of the packets.



Another surprising result we found was that we expected the fps to improve when compressed with zlib as the data size would be smaller for transfer, but in reality the fps increased when the data was sent uncompressed in just string format. This result can be attributed to the fact that compression takes much more processing time and thus increasing the duration of frame process/transfer.

Contribution:

Kevin Yin -

- Worked on the TCP implementation of the source code
- Gathered all the TCP data and created a script to analyze it.
- Reported on TCP data and create the graphs
- Reported on TCP implementation and discussed libraries/modules used.
- Referenced a related work
- Wrote the motivation

Devansh Panirwala

- Worked on the UDP implementation of the source code
- Gathered all the UDP data and created a script to analyze it.
- Reported on UDP data and create the graphs
- Reported on UDP implementation and discussed libraries/modules used.
- Referenced a related work
- Assisted in project purpose and documenting code