COMPUTER NETWORK LAB

PROJECT REPORT : REMOTE SCREEN CONTROL

Registration No: 140905125 Jitesh Kumar Jha
Registration No: 140905134 Vrushank Upadhyay
Registration No: 140905324 Siddharth Kothiyal
Registration No: 140905248 Ishan Handa

Github : https://github.com/jiteshjha/remote-screen-control/tree/sidkothiyal-patch-Final

Manipal Institute of Technology

Department of Computer Science & Engineering

Introduction :

Remote Screen Control let's you control Linux systems remotely, in a easy and fast way over local area networks. The transport layer protocol leveraged is TCP (Transmission Control Protocol) protocol.

Background :

In Remote Screen Control, the client can view the server's screencast, and the client can control the server system with keyboard and mouse events.

The server system captures it's screen frames in real-time, compresses the image on JPEG format for reduced size and easy transmission over TCP/IP, and
each compressed frame is send over the network to the client system. The client system is responsible for assembling each frame, and attaches each received frame to a GUI window. Thus, the frames received per second in real-time creates an illusion of an screencast operation.

While the server side is transmitting the captured screen frames, the keyboard and mouse events are also registered by the client side, sent over the network to the server side, and the server executes the keyboard and mouse events to simulate an control operation directed from client to server.

Implementation:

Broadcasting : Server side sends a broadcast message on the network by a port. If the client is listening it will recieve the address of the server and then make a broadcast itself, with the message as IP of the server. When the server receives the message, it compares its IP with that sent as message. If it matches, the client and server bind at the IP and they start interacting.

Capturing the frame : Client side sends the dimensions of its display for the Tkinter GUI window dimensions(height and window) dimensions to be sent corresponding to that of the client side. This is done on a seperate port at the time of initial setting up.

Transmission of frame, keyboard and mouse events:
Frames and the objects representing keyboard and mouse events are sent over the network using a connection-oriented transport protocol, TCP for complete transmission as byte stream.

Assembly and display of frame: The client is sending the dimensions of its display to the server. The server is setting up a tkinter window accordingly to receive upcoming frames. After sending the display data, the client takes a screenshot and converts the frame from RGB format to BGR format and decreases the quality of the image to be sent, so that we get the max frame rate at the server side and screen updates are as close to real-time as possible. The image is then sent as a byte array to the server. The server after receiving the byte array, it converts it into an image and converts the image back into RGB format. The image is resized to fit into the tkinter window and displayed at the server side. The Tkinter window also contains buttons to close the tkinter window, send command to open a terminal at client side and a FTP button to receive a source file from the client. This keeps happening in a continuously, so that any updates on client side screen is visible almost immediately on the server side.

Capturing the keyboard and mouse events: Using pyxhook at

the server side to capture key events of mouse and keyboard. Using hookMan class to make the keep it running in an infinite loop to read key strokes and logs them. Tkinter GUI window offers support to bind and mouse click event to the window. This capability is used to log left and right mouse clicks. These logs are sent to the server over a port which is different from the port used to send the frames. This is done to have a seperate control port and a seperate port for frame sending, so that one doesn't effect the working and data flow the other. The process is forked and the child process is used to send control events, while the parent process is used to

Executing the keyboard and mouse event:
Key events and mouse clicks were captured using the event listeners and they were executed using PyAutoGUI which provides us with system level calls click() and press() to execute these commands.

FTP:
FTP is made to send basic source files from the server to the client. The FTP only supports source files (non-executable files), which are sent over the control port as byte array.

Future Work :

- More user-centric user interface will be included in the

package
- Establishment of a secure connection before any remote control operation will be incorporated. Like a login/ logout screen.
- FTP has better support for data files.

References and Requirements:

(Python 2.7 : https://www.python.org/)
(Tkinter : https://wiki.python.org/moin/**TkInter**)
(Stack overflow : https://wiki..**stackoverflow**.com/)
(Python-TCP: https://wiki.**python**.org/moin/**Tcp**Communication)
(PyHook: https://github.com/JeffHoogland/**pyxhook**)
(PyAutoGUI: https://pyautogui.readthedocs.io/en/latest/)
(Pillow : https://python-pillow.org/)
(OpenCV 2.4 : http://opencv.org/)
(Numpy : http://www.numpy.org/)
(PyScreenshot : https://pypi.python.org/pypi/pyscreenshot)