Remote Desktop Control

CS378 Group Project Report

#### Group Members:

Anurag Shirolkar (120050003)

Prateek Chandan (12050042)

Nishant Kumar Singh (120050043)

Aditya Kumar Akash (120050046)

# Introduction

Remote Desktop Application is a software that enables users to control their machine from a remote machine using browser. The User’s machine which is being controlled runs a python server and creates a websocket.

# Implementation

## Web Socket

This protocol has two parts: a handshake and a data transfer

The handshake from the client looks like this:

header - GET /client HTTP/1.1

Host: <IP Addr>

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Origin: <ws://IP\_Addr>

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

The handshake from the server looks as follows:  
 HTTP/1.1 101 Switching Protocols  
 Upgrade: websocket  
 Connection: Upgrade  
 Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=  
 Sec-WebSocket-Protocol: chat

Once the client and server have both sent their handshakes, and if

the handshake was successful, then the data transfer part starts.  
This is a two-way communication channel where each side can,  
independently from the other, send data at will. After a successful handshake, clients and servers transfer data back  
 and forth in conceptual units referred to in this specification as  
 "messages".

Conceptually, WebSocket is really just a layer on top of TCP that does the following:

* adds a web origin-based security model for browsers
* adds an addressing and protocol naming mechanism to support multiple services on one port and multiple host names on one IP address
* layers a framing mechanism on top of TCP to get back to the IP packet mechanism that TCP is built on, but without length limit
* includes an additional closing handshake in-band that is designed to work in the presence of proxies and other intermediaries

The WebSocket Protocol is an independent TCP-based protocol. Its  
 only relationship to HTTP is that its handshake is interpreted by  
 HTTP servers as an Upgrade request.

## Client side keyboard, mouse event handling

Client side events are handled using jQuery and javascript. The locations of mouse clicks with respect to the image and keyboard events are sent via websocket to the server.

An event is either keydown, keyup, mousedown or mouseup. Every packet sent is of the following form

<’,’ separated keyboard events>|<’,’ separated mouse events>|<mouse x, mouse y>

## Server side keyboard, mouse event handling

For handling of mouse a utility tool named **xdotool** is used to press and release the mouse event. For Controlling of the keyboard python’s library pyKeyBoard is used

Screenshot Transfer:

***Issue:*** The size of the screenshot is too large to be sent repeatedly in order to provide real-time rendering of the remote desktop on the client’s computer

To tackle the above issue, we initially thought of taking difference of the pixel arrays of consecutive screenshots and transfer only the difference. This method would definitely reduce the size of the data that needs to be transferred but the processing time at both the server and client is too high for this method to be practical. So no difference is being computed and all screenshots are being transferred to the client at regular intervals

***Issue:*** The system takes some time to capture the screenshot

To solve this we implemented the screenshot using a very low level api, **pygtk**. This ensured that the screenshots can be captured and saved quickly

***Issue:*** Transfer the image in form of a base64 encoded string which is larger than the saved image

Base 64 encoding encodes every three bytes of the byte array corresponding to the image in 4 bytes of the string using a set of 64 characters. So the string size is ~4/3\*image\_size. So instead of transferring the base 64 string we send the **byte stream** of the image

[See project’s code and Readme](https://github.com/prateekchandan/Remote-Desktop-Connection)