**Loughborough universit y**

Due Date: Friday 1st May (Semester 2)

Authored by:

Nicholas Davies

Kwaku Ankobia (Code)

Bartek Biernacki (Code)

|  |
| --- |
|  |
| 14ELC004 – Computer Networks |
|  |
| Internet chatting in the TCP/IP network |

14ELC004 – Computer Networks

Internet chatting in the TCP/IP network

Introduction

The objective in this coursework was to create an internet based chatting client over TCP/IP and UDP protocols. The two main aspects of this project are the server and client. The code for the server is already written but can be edited if required whereas the client needs modifying to fulfill the project objectives shown below. The programs will be developed using Python 2.7 due to the simplistic interfacing it provides with the networking protocols.

Project Objectives

* Clients must be able to communicate with other clients
* Each client must display a friends List
* Friends List will display the username of each person connected
* Friends List must reflect the online/offline status of each person connected
* Chatlog will display sent messages sent to different clients
* Usernames must be requested via the use of UDP between client
* Group Chatting by allowing multiple users to be selected in the friends list
* Messages will then be sent to all of the select users.

Python Background

Python was first developed about 20 years ago by a Dutch programmer called Gudio van Rossum. It was initially developed as a successor to the ABC programming language but able to interface with the Amoeba operating system. Python’s popularity boomed in 2000 with the release of 2.0 this was aided by the increased usefulness of scripting languages with python being the main alternative to Perl, Ruby on rails was still finding its feet at this point. Python’s easy to read and more relaxed nature to common programming procedures such as initializing and defining variables made it popular with individuals new to programming. The version used in this project is 2.7, released in 2010. Although python is being used to create the Graphical user interface in this project it’s often used instead of PhP to create the functional backend of websites while being fronted by HTML and CSS. In the past 10 Years Google has invested a lot in the language for this particular purpose. The Popular video sharing website “YouTube” currently runs on python.

Current Online Chatting Tools

# Skype

Skype is the most popular voice/text communication service currently in use with over 660 million users at the end of 2010. Recently acquired by Microsoft Skype has replaced MSN their official messenger. Skype uses a proprietary technology called the “Skype Protocol” based on a peer to peer architecture. This is fundamentally different to this project because it doesn’t require a server to communicate with other individuals, however A TCP connection is required to login.



Applications utilizing a peer to peer architecture have a number of advantages, there is no server to maintain meaning the service will not go offline if maintenance is required. One disadvantage of peer to peer networking is that security is more difficult to maintain. Skype utilizes an RC4 Encryption to encrypt the TCP stream.

# Facebook messenger

Facebook Messenger is Facebook’s web implementation of their own built-in chat system. The first iteration was implemented on Mobile and Windows PC in mid-2011. Today over 600 million users utilize the system.



Facebook Messenger uses a lightweight open protocol called MQTT. This stands for Message Quere Telemetry Transport and it used as an extra layer to TCP/IP. This protocol is designed and excels in situations where bandwidth is limited. The protocol uses SSL(Secure Socket Layer) Encryption which is a form of RSA encryption.

# AOL Instant Messegner

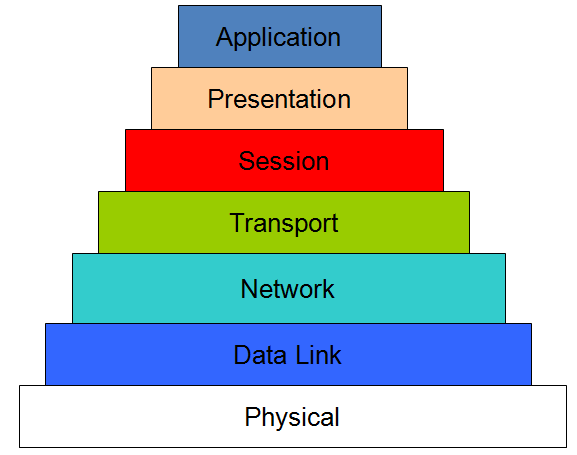
AOL instant messenger also known as aim was originally released in 1997 by AOL. Although very basic in comparison to the services in use today, it’s still adimierd today due to it being one of the first dedicated online chat services. It peaked in 2006 with a market share of 52%.



The protocol in use this service was called “Open System for Communication in Realtime” or OSCAR for short. It uses a proprietary container c called FLAP around every package that has information about that the packet size, channel and its number in the sequence. The technology itself was licensed to Apple™ in 2002 to be used in their iChat application

Computer Networks

For computers to communicate effectively they must be incredible organized. A number of different aspects must be considered in order to send a message from one to another. In modern computing a conceptual model has been produced to describe the different parts of this communication. The OSI model or Open Systems Interconnection model is split up into 7 different layers shown below. This model ensures that different parts of the communication are kept separate and organized. Each layer will have its own set of protocols to achieve its particular task.



This coursework focus on the transport layer of the OSI model. The main objective this layer has to achieve is program to program communications. It’s this layer which actually provides the means to send data from one computer to another. This is done by the TCP and UDP protocols.

# UDP – User Datagram protocol

UDP is a connectionless based protocol meaning no pre-established link is required before hand. Due to the fact no handshake is required before the data is transferred results in UDP being of low latency. The main problem with UDP that it doesn’t produce reliable communication. Once a data packet has been sent there is no way of checking it’s been received at its destination. This project will use this protocol to actually send the messages between the clients as this information is considered expendable. Also if the client was modified in the future to allow the transfer of multimedia such as video between clients UDP is preferable.

# TCP – Transmission Control Protocol

TCP requires a link to be established between the host and destination before the data is sent. The main advantage TCP has over UDP is reliable communication meaning if a transmission is lost the host is told and can re-transmit the data. TCP connection is one to one due to it only being able to connect endpoints. In this project TCP is used to connect clients to the server so they can request the client list.

Systems Design

# Flowchart System Diagram

# 

Background Process’s And Methodology

Actually implementing the function to allow users to send messages to other clients was relatively easy. The main challenge was updating the friends list with the relevant usernames. Originally the friends list was displaying the IP and ports of each of the clients. This did allow clients to exchange messages but there was no way of identifying who was who because this information isn’t provided in the client list sent by the server.

Solving this problem required each client to send a “Username request” using UDP to each of the other client who in turn would return their username in the same manner. This raised a new issue though, how would the client differentiate between “chat” messages which needed to be pasted into the chatlog and “Username Request” messages. It was setup so that special messages began with X\_ so they could be filtered out. This was done by simply searching the string for “X\_”. Another filter was then created to differentiate between “Username Requests and “Received Usernames”. If the Special message contained “GET\_NAME\_RESEP” (Username request response) the username that was packaged is extracted and tied to that particular port. The Friends list is then updated. If the special message contained “GET\_NAME\_MSG” (Username Request) then a message was sent back packaged with their username.

One fundamental problem with the request and response methodology was that when a client receives a UDP message it contains two parameters, the actual message and the IP of the client sending it. One piece of information it didn’t include though was the port of the client sending on which was required when sending a response. To circumvent this problem all messages are prepended with the port of the person sending it. The port is then extracted and used sending a response.

Username Request Format

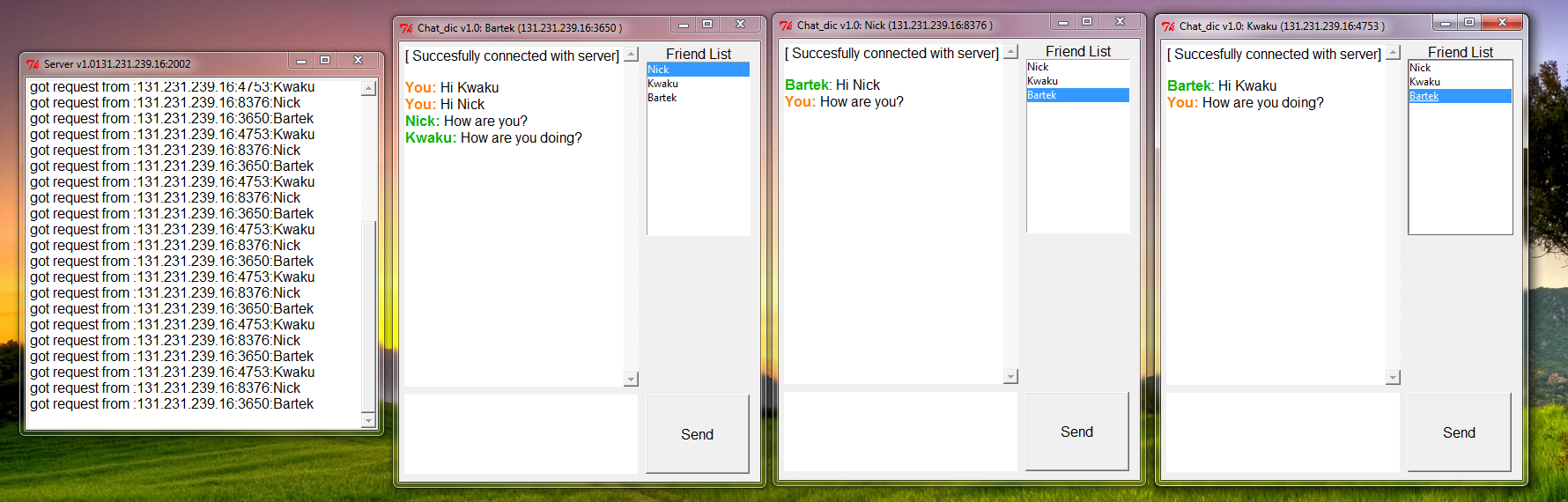
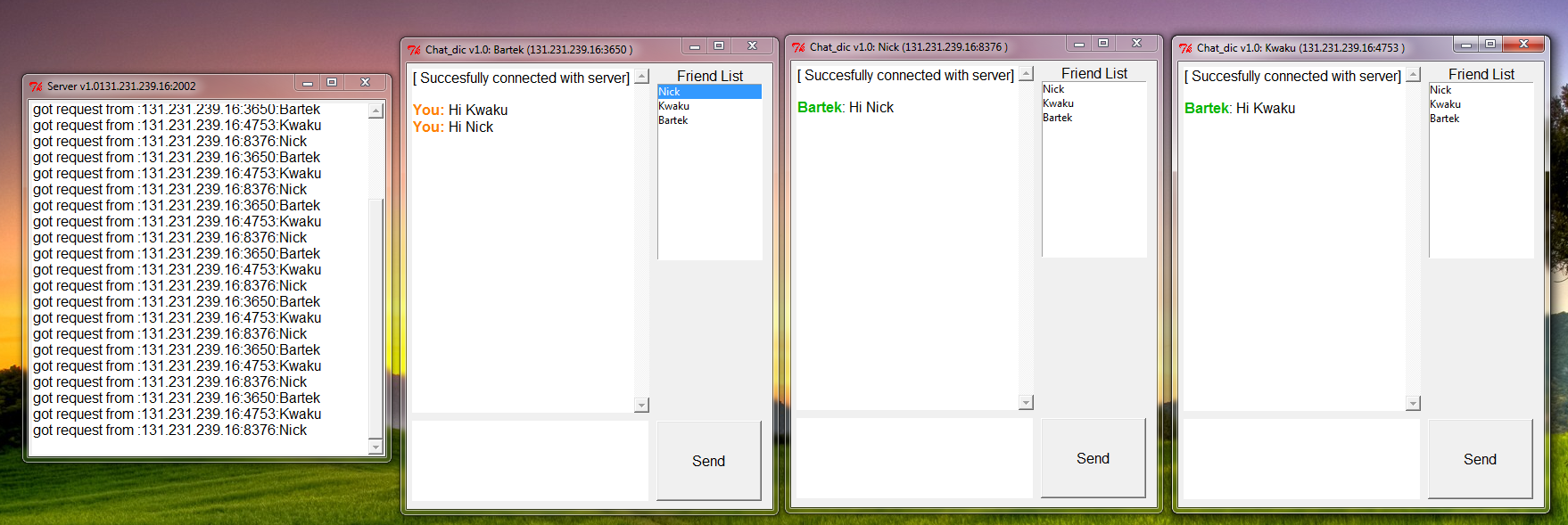
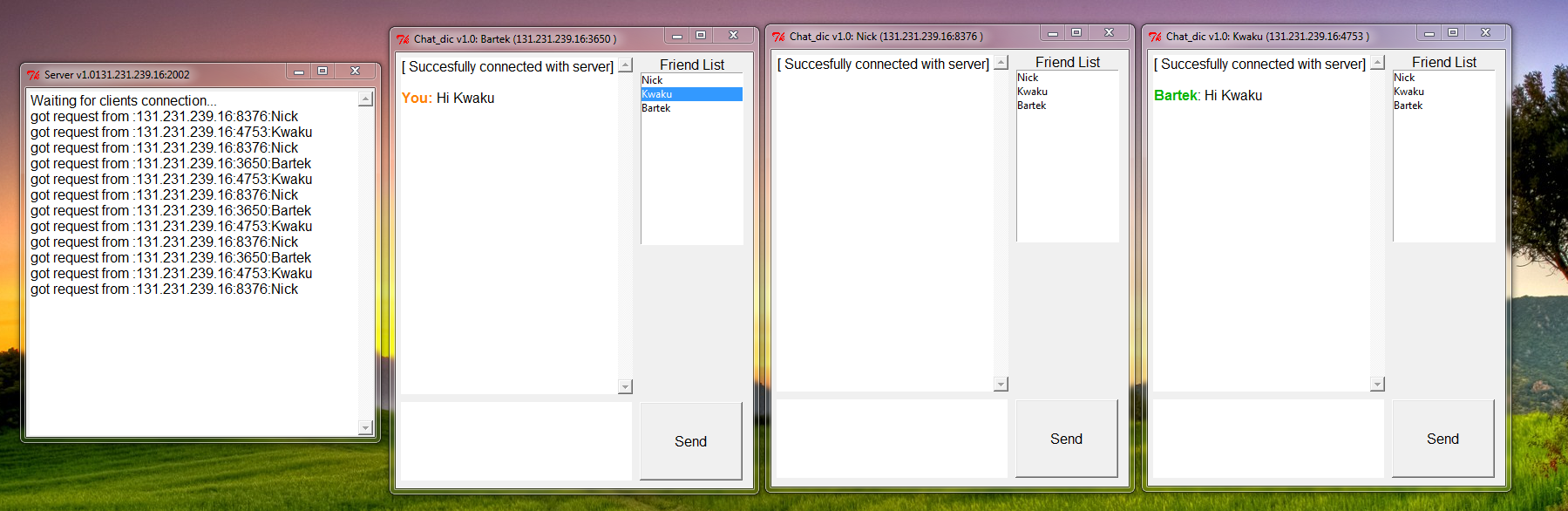
“X\_GET\_NAME | PORT”

Username Response Format

“X\_NAME | USERNAME | PORT

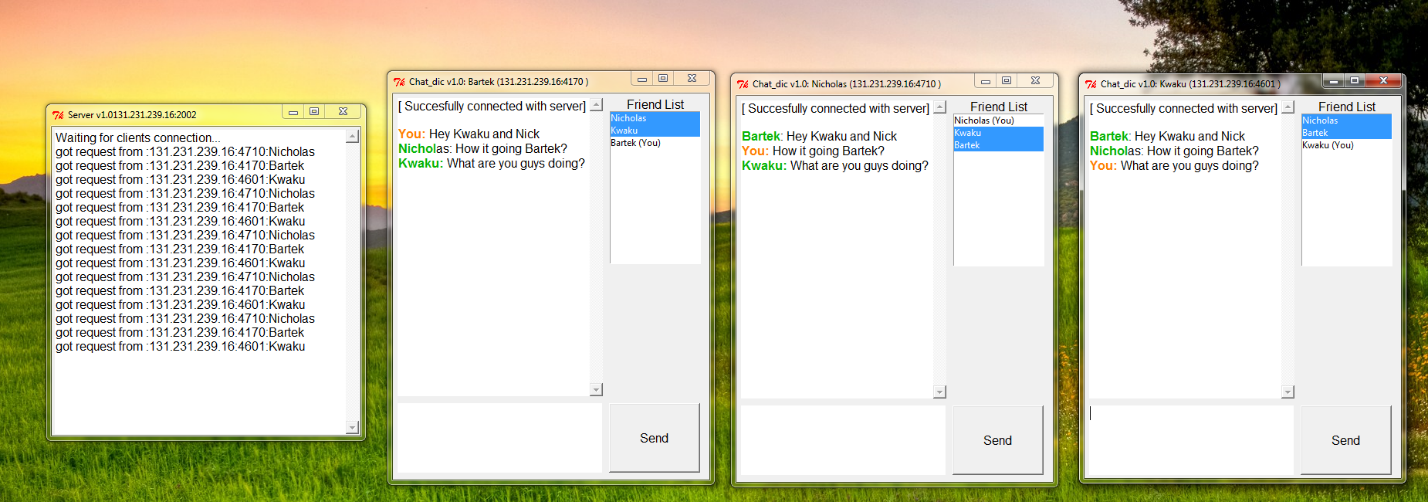
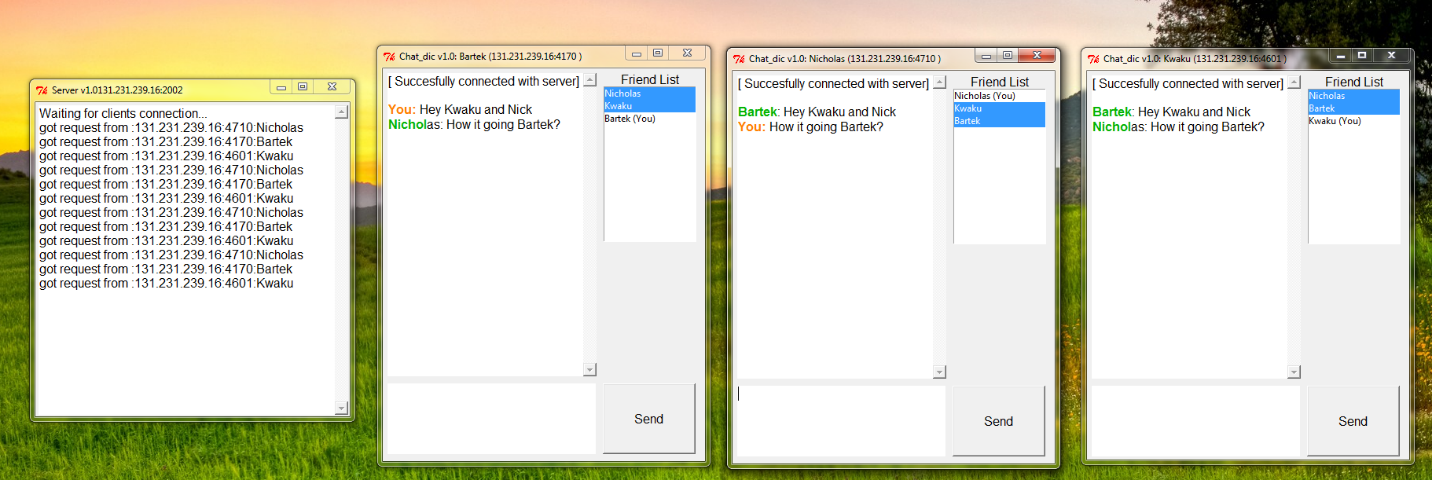
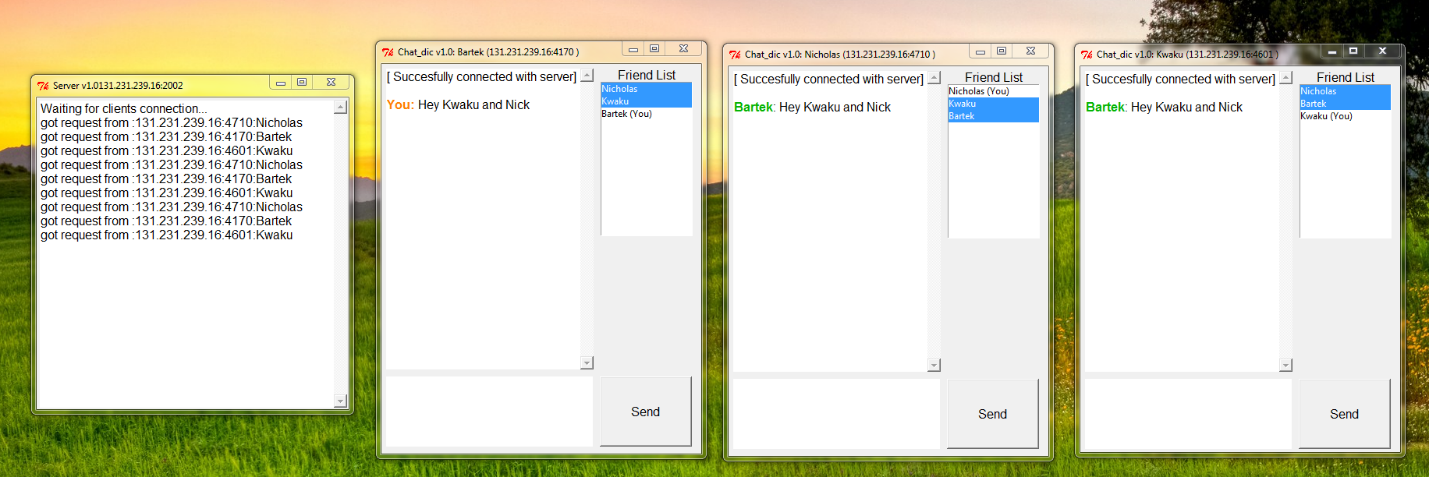
System performance

# Chat Testing (One to One)

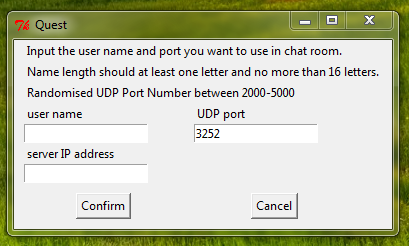


Here the one to one chat was demonstrated

# Chating Testing (Group)(Updated Client)



# Modified Client



The client input form was modified slightly so that the UDP port was randomly generated but the user was allowed to overwrite it if they wanted to. This minimized user input error while still retaining the feature.

How can it be used in practice?

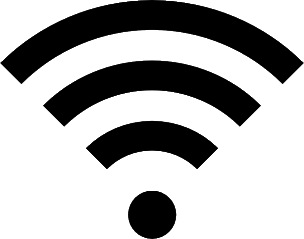
Currently the system would have to undergo a number of changes before it could actually be rolled out in the real world. The major problem with the current system is that no data is encrypted before it is transmitted. This is the case for information send to the server and between clients. The communication to the server is of particular concern because its sending the IP’s and ports of everyone currently connected to it. If an individual with malicious intent managed to gain access to this information the consequences could be severe. There are a number of encryptions algorithms which could be implemented such as SSL. SSL encryption uses RSA which makes use of private and public keys. This is the encryption used by all online services exchanging information over HTTPS.

Although it would require a complete system redesign a P2P system similar to skypes doesn’t actually require a central server may be viable, mostly because of the increased reliability.

The current system does have similarities to current commercial on-line chatting systems in the sense that its utililising TCP/IP and UDP connections however the main difference in this project is that we’re using functions that access them directly whereas a commercial product will use a technology based on top of TCP/IP such as MQTT or the “Skype Protocol”.

Mobile Phone Application Possibilities

Porting this system to an Android or IOS wouldn’t be particularly difficult. The system architecture would have to change where the server and clients would be on different platforms. The always on server would be running on a pc with the client side software being installed on the PC. Developing the actual client side applications for each platform wouldn’t be particular difficult due to both Java and Objective-C having support for TCP/IP connection library’s.



Client

Client

Server

For this to work correctly on an android device an activity would have to be created with some sort of input form to enter the server IP. Once entered the device would create a new socket and attempt to connect. The methodology is exactly the same as the program created in the coursework.

Conclusion

All of the original objectives were fulfilled resulting in a successful project. Any further improvements that could have been implemented such as additional security have been noted earlier on in the report.

Nick

B225546