# Overview

This time I worked in solo even though this is a group project. I encountered some difficulties during the project but I overcame it by myself so I feel quite accomplished. The project is to implement a P2P app which works as a bare bone Facebook messenger. Each client has a chat window and an input box. The client communicates with his/her peer by typing in the box and transmit data when ENTER is hit. The core functionalities of the project are implemented via Qt which is a framework in c++. Qt takes care of the GUI part as well as UDP protocols. The project is run in a VirtualBox environment in order to visualize the chat window.

# Functionalities

The program is built upon two classes, NetSocket and ChatDialog. Most of the functions are inside ChatDialog. A NetSocket instance is used to initializes the socket and bind the socket with the user port.

ChatDialog class is the meat and bone of the project. I’ll briefly describe the semantic of the implementation of the functions within the class.

* ChatDialog(): it initializes a chat window and also made an instance of NetSocket to get the port number for the sender. It then creates two separate connect() functions. One has an update function gotReturnPressed() when returnPressed() is triggered. The other has an update function readPendingDatagram() when readyRead() is triggered.

Sending a message:

* gotReturnPressed(): it displays the info user typed to the chat window. It also stored the information into a customized data structure I created called Dict\_Message. Dict\_Message is a map like data structure which stored information of each message including Origin, SeqNo, and ChatText. The data structure is defined in the header file as it’ll be reused later when creating rumor messages and status messages. Inside the function, create\_rumorMessage() is called.
* serializeMessage(QVariantMap &myMap): is called by create\_rumorMessage() and create\_statusMessage(). The two functions created QVariantMap and passed in the function. Inside the serializeMessage(), I converted the QMap into a QByteArray by using QDataStream. Based on the port of the user, I called writeDatagram to send out my QByteArray. After sending out the QByteArray, I activated the timer.
* active\_timeout(): sets the timer and called the update function when timeout.

Receiving a message:

* readPendingDatagram(): it is triggered when there’s readyRead(). Within the function, I called deserializeMessage() to see the actual message.
* deserializeMessage(QByteArray datagram): the function deserialized a message and then called receiveText().
* receiveText(QVariantMap &inMap): is a core function when receiving a message. It looked at the input and distinguished either receiveStatusMessage() should be called or receiveRumorMessage()
* receiveStatusMessage(QVariantMap inMap)
* receiveRumorMessage(QVariantMap inMap)
* create\_rumorMessage(QString one\_has\_info, quint16 seqNo\_to\_send)
* create\_statusMessage(QString one\_has\_info, quint16 seqNo\_to\_ask)

# Challenges

1. Get VirtualBox: I haven’t used VirtualBox before so it took me a while to set up the environment. Also, it wasn’t explicit that Qt didn’t work with the newest Ubuntu. I got around it by installing Ubuntu 14.04.
2. Understanding Qt: Qt wasn’t really hard but knowing when to call the right function could be tricky. Especially for writeDatagram(), the parameter is supposed to be a QByteArray. Also, the declaration of QMap took me a while to master. For a status message, we need to have a nested QMap but Qt doesn’t support that. I needed to convert the inner QMap to a QVariant by qvariant\_cast() method.
3. Write in C++: I used C++ when I was in undergrad but I haven’t used it since. I brushed up on the skill by reading through some stack overflow answers. The most challenging part of the project is to figure out when to pass in by reference in C++, other words pointers are hard to master.
4. Rumormongering: A trick is to split them into three cases. When the sequence number is less than that stored in Dict\_Message, when it’s larger, when it’s equal.