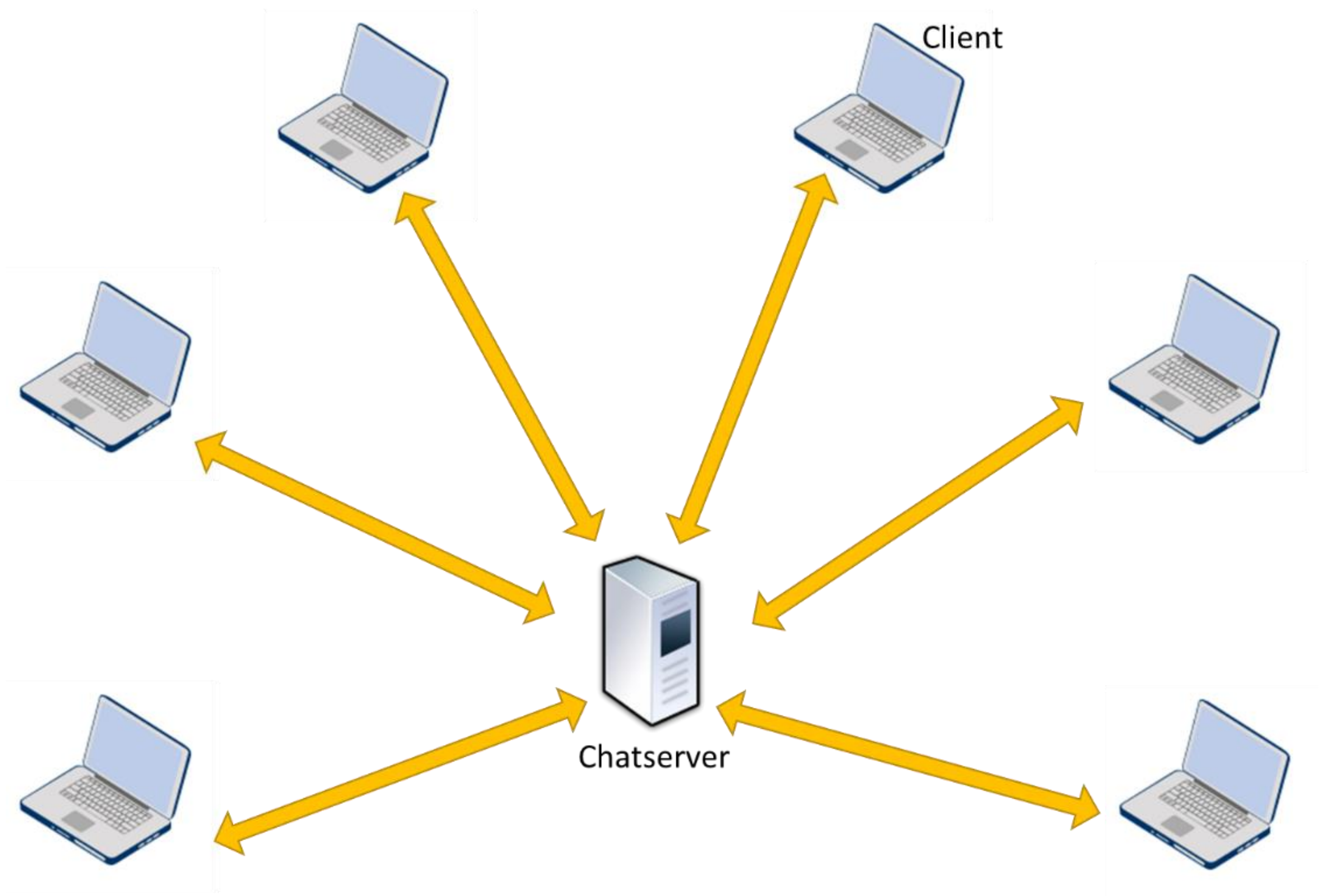
Programming Assignment Multi-user Chat Application

2021-22 COMP3234B & ELEC3443B

Objectives

- An assessment task related to **ILO4 [Implementation]** – "be able to demonstrate knowledge in using Socket Interface to design and implement a network application".
- A learning activity to support ILO1, ILO2, & ILO4.
- The goals of this programming assignment are:
  - to get a solid experience in using Socket functions to implement a real-life protocol;
  - to get a good understanding of how a JSON-based networking protocol works as well as how to implement one.

System Overview:
Centralized Chat Server



Overview of the Application

ChatApp (Client)

- Control by user
- Three main functions
  - Join the chatroom
  - Send messages to chatroom
  - Leave the chatroom

Communication protocol

- JSON-based
- 5 commands

ChatServer

- Centralized server
- Support multi-users
- Maintain peer list
- Relay messages to target recipient(s)

Task

- Develop two Python3 programs
  - ChatApp.py – Chat client program
  - Chatserver.py – Central server program
- Given
  - ChatApp-UI.py
    - Makes use of the Tkinter module to implement the UI for handling user’s inputs and displaying chat messages
  - Workshop 2 sample implementation
    - A good starting point for the Chatserver.py (if you prefer using select() for the I/O)
  - A set of config files - config.txt, config1.txt, config2.txt, & config3.txt
  - A set of script files - start.ps1, start-linux.sh, start-OSX.sh, & start-OSX-tab.sh.

ChatApp.py program

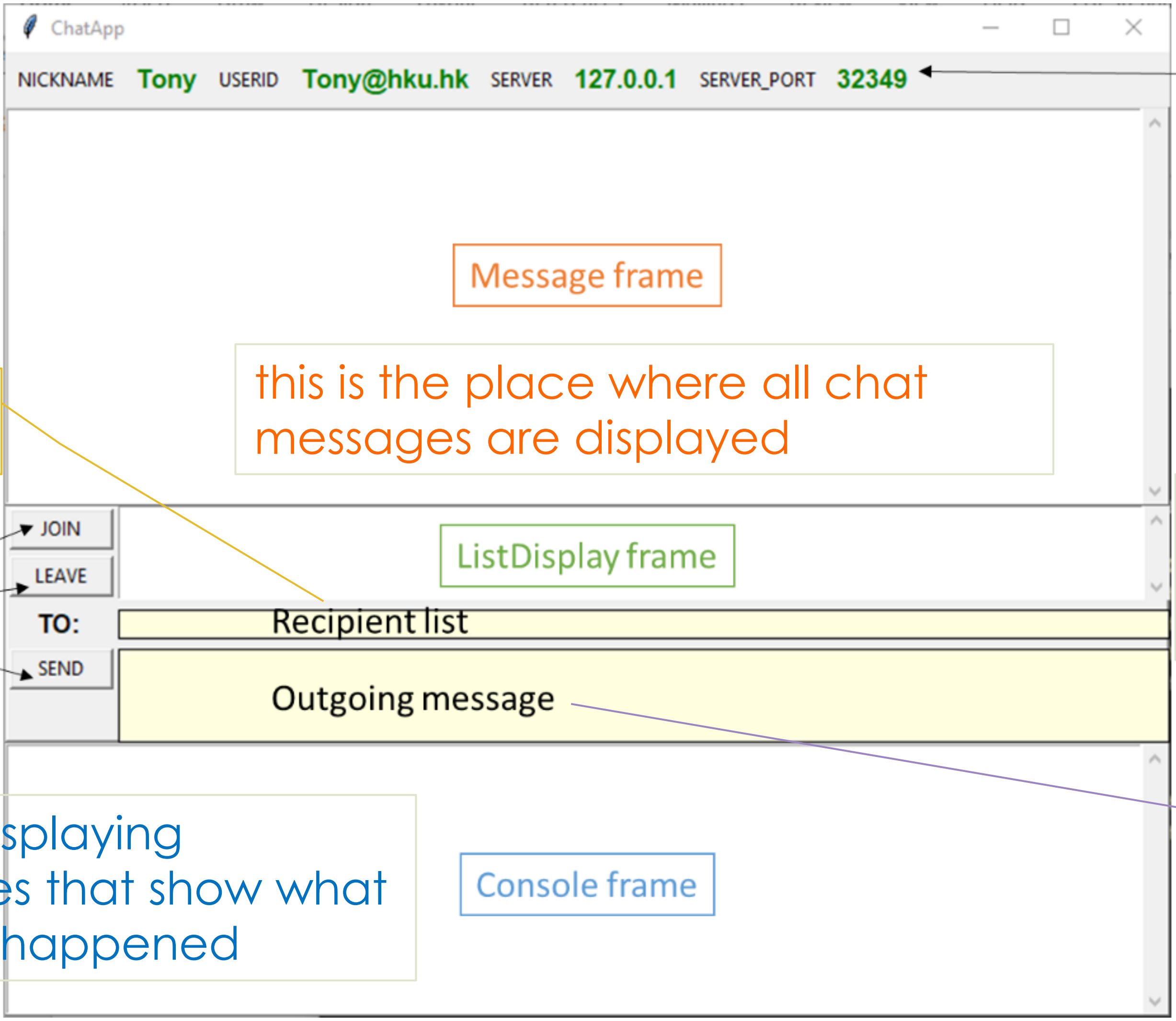
- To run the program,

  python3 ChatApp.py [config file]

- Config file
  - For setting the user's information and Chatserver information
  - Load by ChatApp.py during startup (already implemented)
  - Contains a JSON string:

```
    {
      "USERID": "Tony@hku.hk",
      "NICKNAME": "Tony",
      "SERVER": "127.0.0.1",
      "SERVER_PORT": 32349
    }
```

UI of ChatApp.py program



Chatserver info

this is the place where all chat messages are displayed

User enters recipients' nicknames

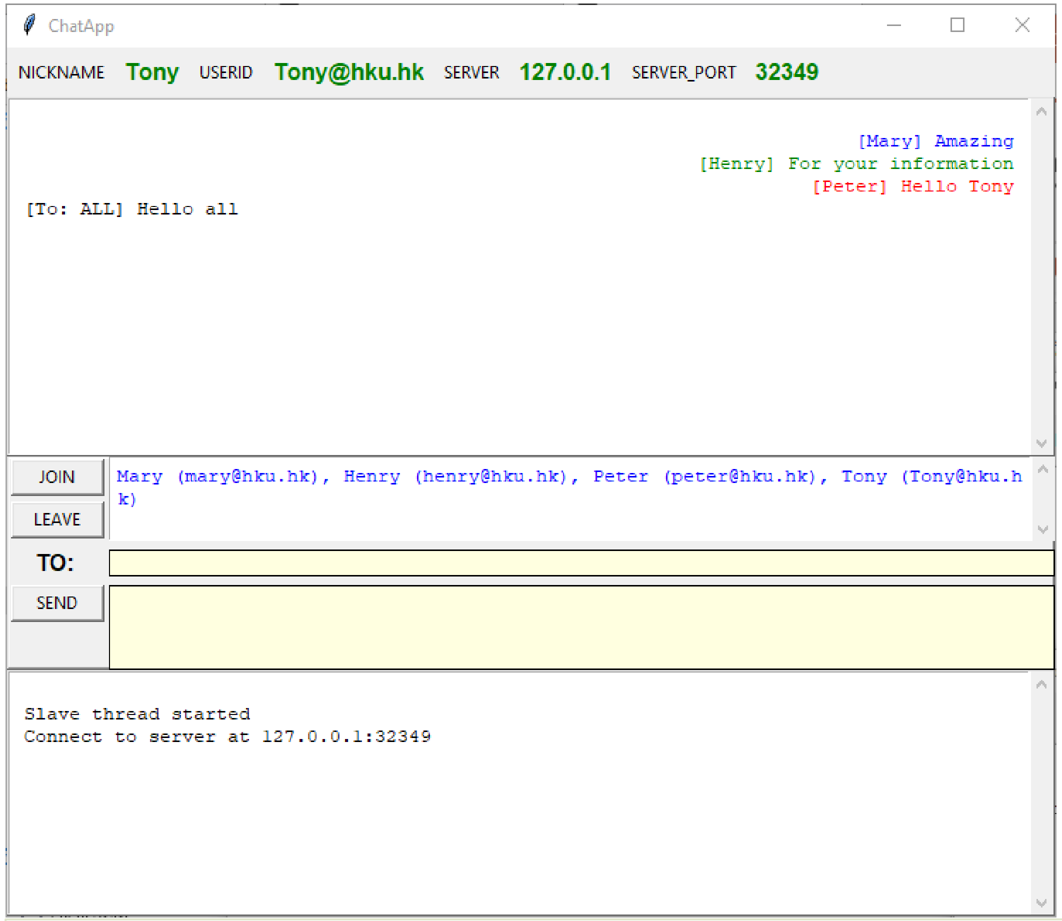
This is the place for displaying the current peer list

Buttons

User enters the message

This is the place for displaying system/error messages that show what actions/events have happened

Display Chat messages to Message frame



NICKNAME Tony USERID Tony@hku.hk SERVER 127.0.0.1 SERVER_PORT 32349

```
                                                   [Mary] Amazing
                                     [Henry] For your information
                                                [Peter] Hello Tony
[To: ALL] Hello all
```

```
Mary (mary@hku.hk), Henry (henry@hku.hk), Peter (peter@hku.hk), Tony (Tony@hku.h
k)
```

```
Slave thread started
Connect to server at 127.0.0.1:32349
```

- User's outgoing message
  - **chat_print**("[To: ALL] Hello all")
- Received messages
  - Private message
    - chat_print("[Peter] Hello Tony", 'redmsg')
  - Group message
    - chat_print("[Henry] For your information", 'greenmsg')
  - Broadcast message
    - chat_print("[Mary] Amazing", 'bluemsg')

chat_print() - a method provided by the UI to add the message to **the top** of the Message frame

Console frame & ListDisplay frame

Console frame

- Displays system or error messages
- console_print() – a method provided by the UI to add it to the top of the console frame
- No specific requirement or format
- Suggestion - print any error/log message for significant event to the console frame

ListDisplay frame

- Displays current peer list
  - Chatserver will broadcast the updated list whenever there are changes to the list
- list_print() – a method provided by the UI to update the peer list
- Format
  - Nickname (USERID)
  - Separated by commas

Interact with end-user

- Get user's input
  - TO: field
    - **get_tolist**() method
    - Format:
      - Private message – a nickname
      - Group message – a list of nicknames; separated by commas
      - Broadcast message – the keyword "ALL"
  - Outgoing message field
    - **get_sendmsg**() method
- Three buttons
  - JOIN, LEAVE, & SEND
  - Each button has its associated function
    - JOIN – **do_Join**()
    - LEAVE – **do_Leave**()
    - SEND – **do_Send**()
- Your task is to complete those functions

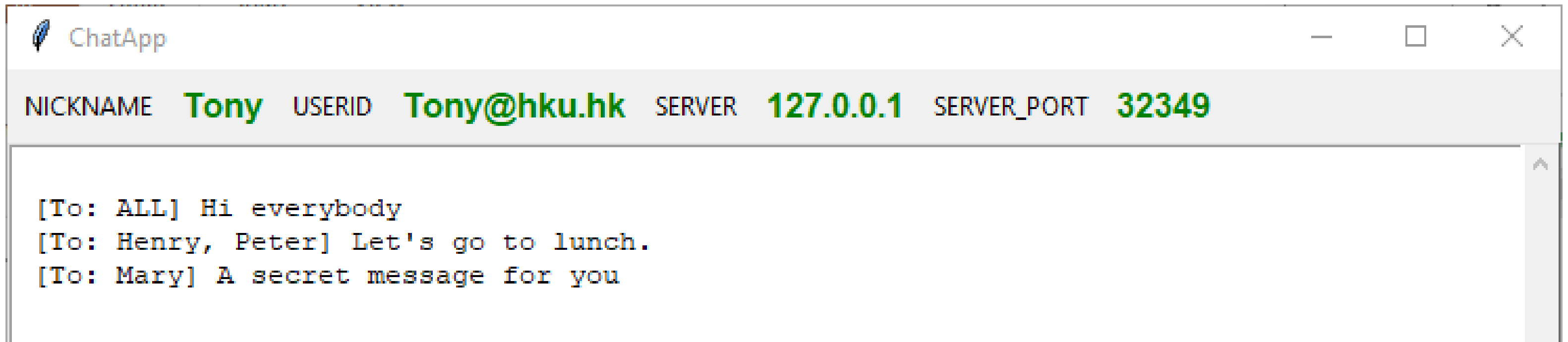
Actions

- do_Join()
  - Connect to Chatserver using TCP if not connected yet
  - Send the **JOIN** command
  - Expect an **ACK** command returned by server
  - If successful, expect to receive a peer list from server
  - Server will push updated peer list whenever it has changes

- do_Leave()
  - Close the TCP connection if is connected
  - Release any resources previously created for the chat session
    - i.e., allows user to re-join the Chatserver

Actions

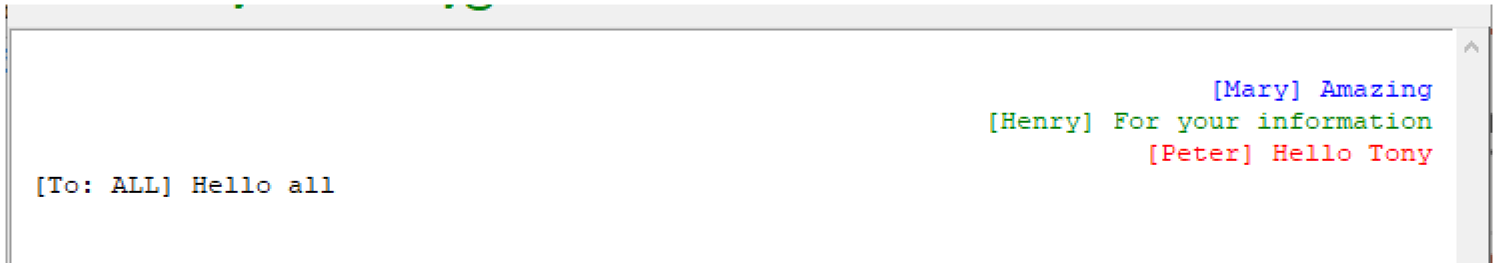
- do_send()
  - For sending private / group / broadcast message if the client has connected to the Chatserver
  - Get the list of recipient(s) from the **TO:** field
  - Get the message from the **Outgoing message** field
  - Send the message using the **SEND** command
  - Display the outgoing message to the Message frame with appropriate header


ChatApp
NICKNAME Tony USERID Tony@hku.hk SERVER 127.0.0.1 SERVER_PORT 32349

```
[To: ALL] Hi everybody
[To: Henry, Peter] Let's go to lunch.
[To: Mary] A secret message for you
```

Actions

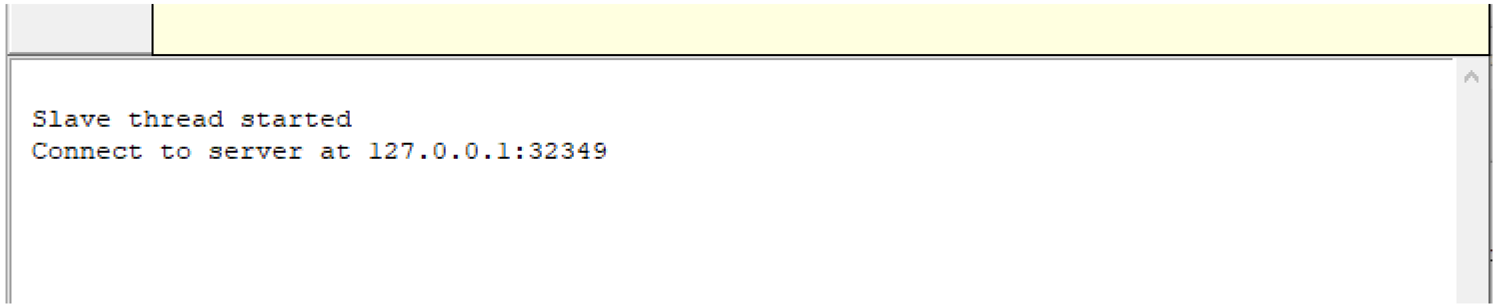
- Receive updated peer lists
  - Client would receive the updated peer list at any time (asynchronous)
  - Server sends the peer list using the **LIST** command
  - Displays the updated peer list using list_print() method
- Receive chat messages
  - Client would receive chat messages at any time (asynchronous)
  - Server sends the chat message using the **MSG** command
  - Displays the chat messages using chat_print() method with appropriate header and color scheme



```
                                                          [Mary] Amazing
                                                [Henry] For your information
                                                          [Peter] Hello Tony
[To: ALL] Hello all
```

Implementation Hint

- Some socket functions (e.g., recv()) will block the whole program if the socket event cannot finish/complete immediately
- When the process is blocked, the UI will not respond to any user's input
- Suggest using a python thread to handle all asynchronous recv() operations



```
Slave thread started
Connect to server at 127.0.0.1:32349
```

Chatserver.py program

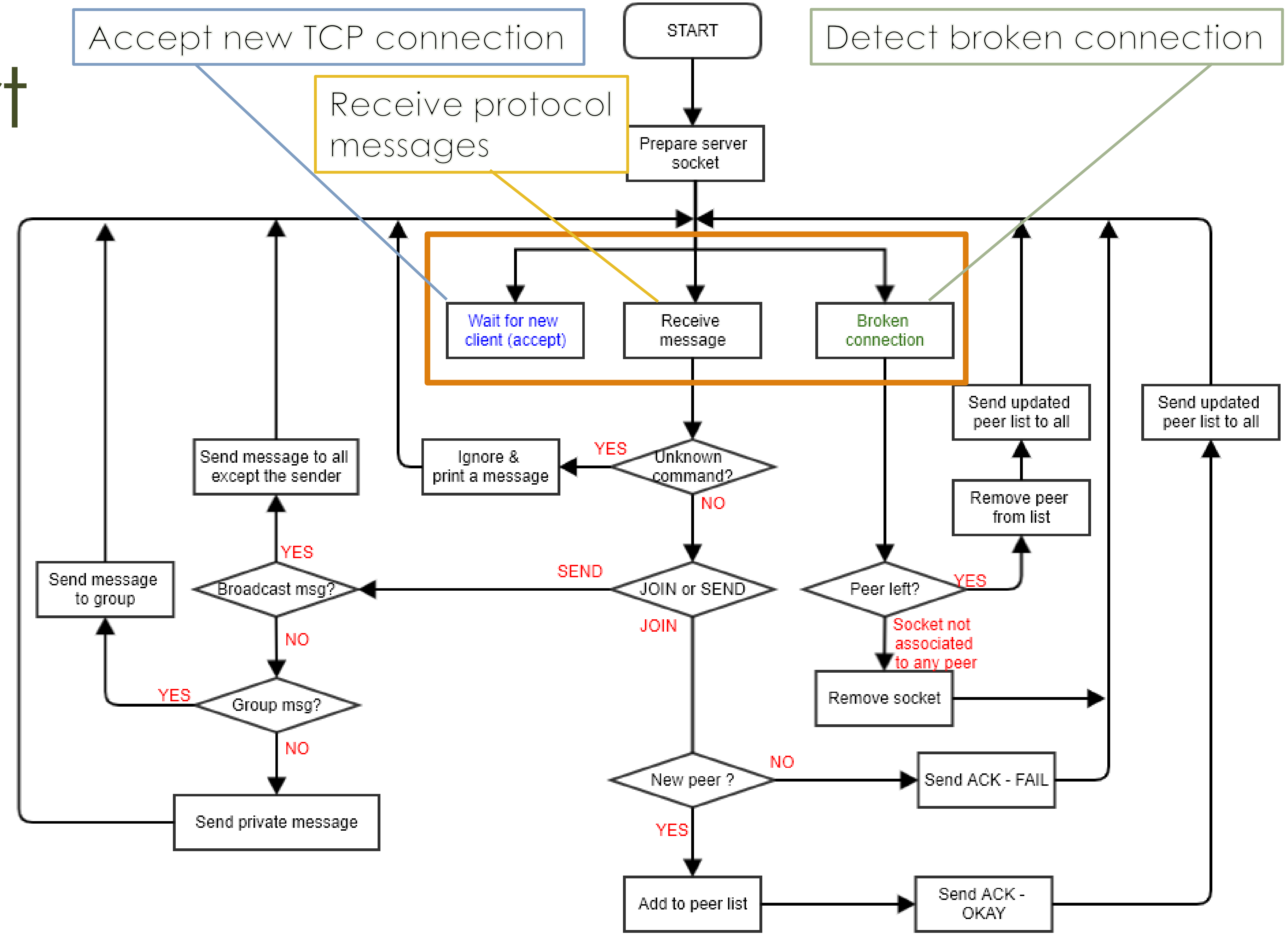
- To run the program,

  python3 Chatserver.py [listen port]

- listen port – optional argument
  - If not provided, the server should use the default port number XXXXX
  - A port number under your assigned port number range

Flowchart

Suggest using a single-threaded process with the select I/O operation



Communication Protocol

- All commands are in JSON string format
  - JSON is a standard text-based format for representing structured data based on JavaScript object syntax.
  - JSON is a human and machine-readable format
- Protocol data is converted to JSON string before sending, carried by TCP, and turned back to protocol data format at the other end
- Many programming environments (including Python) come with functions to convert information between JSON string and structured data.

Communication Protocol

- JSON data is written as key-value pairs with a colon between the key and the value
- Each key-value pair is separated by a comma
- A JSON string is a key-value data format that is typically rendered in curly braces
- JSON values can be of one of 6 **simple data types**:
  - strings – must be written with **double quotes**
  - **numbers**
  - objects (dictionaries) – circumscribed by curly braces { }
  - lists – circumscribed by square brackets [ ]
  - booleans
  - null or empty

```
{
  "USERID": "Tony@hku.hk",
  "NICKNAME": "Tony",
  "SERVER": "127.0.0.1",
  "SERVER_PORT": 32349
}
```

Python functions

json.loads() – convert a JSON string to a Python object

json.dumps() – convert a Python object to JSON string

```
import json

# a JSON string:
x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])    # that prints an integer 30
print(y["name"])   # that prints the string John

# a python list
obj = ['apple','orange','mango']

# encode obj:
jstr = json.dumps(obj)

print(jstr)        # that prints ["apple", "orange", "mango"]

# parse jstr:
nobj = json.loads(jstr)

print(nobj[1])     # that prints orange
```

JOIN Command

$$\begin{tikzcd}\boxed{\text{Client}} \arrow[r] & \boxed{\text{Server}}\end{tikzcd}$$

- The JOIN command consists of three fields:
  - CMD - should have the value JOIN
  - UN - user's nickname
  - UID - user's userid
- e.g.,
  - `{"CMD": "JOIN", "UN": "Tony", "UID": "Tony@hku.hk"}`

ACK Command

$$\begin{tikzcd} \text{Server} \arrow[r] & \text{Client} \end{tikzcd}$$

- The server responds with an ACK command for each JOIN command
- It consists of two fields:
  - CMD - should have the value ACK
  - TYPE - either OKAY or FAIL
- For example, the server accepts the JOIN request:
  - `{"CMD": "ACK", "TYPE": "OKAY"}`
- Another example, the server rejects the JOIN request:
  - `{"CMD": "ACK", "TYPE": "FAIL"}`

LIST Command

$$\begin{tikzcd} \text{Server} \arrow[r] & \text{Client} \end{tikzcd}$$

- The LIST command consists of two fields:
  - CMD - should have the value LIST
  - DATA - is a list data type that keeps the peer list
    - Each peer is structured as a dictionary type with the UN and UID fields
- For example, this is the LIST command with two peers in the list

  -
    ```
    {"CMD": "LIST", "DATA": [{"UN": "Mary", "UID":
    "mary@hku.hk"}, {"UN": "Henry", "UID": "henry@hku.hk"}]}
    ```

SEND Command

$$\begin{tikzcd} \text{Client} \arrow[r] & \text{Server} \end{tikzcd}$$

- The SEND command consists of four fields:
  - CMD - should have the value SEND
  - MSG - contains the chat message
  - TO - is a list data type that contains the userids of the recipients.
  - FROM - contains the userid of the sender
- e.g., Private message

  -
    ```
    {"CMD": "SEND", "MSG": "Where are you?", "TO":
    ["Tony@hku.hk"], "FROM": "mary@hku.hk"}
    ```
- e.g., Group message

  -
    ```
    {"CMD": "SEND", "MSG": "For your information", "TO":
    ["peter@hku.hk", "mary@hku.hk"], "FROM": "Tony@hku.hk"}
    ```

SEND Command

- To send a broadcast message, set an empty list to the TO key
  - `{"CMD": "SEND", "MSG": "Dear all, you can reach me via 91176842", "TO": [], "FROM": "henry@hku.hk"}`

MSG Command

$$\begin{tikzcd} \boxed{\text{Server}} \arrow[r] & \boxed{\text{Client}} \end{tikzcd}$$

- The MSG command consists of four fields:
  - CMD - should be the value MSG
  - TYPE - indicates the type of message; either ALL, GROUP, or PRIVATE
  - MSG - contains the chat message
  - FROM - contains the userid of the sender
- e.g., private message

  -
    ```
    {"CMD": "MSG", "TYPE":"PRIVATE", "MSG": "Where are you?",
    "FROM": "mary@hku.hk"}
    ```
- e.g., group message

  -
    ```
    {"CMD": "MSG", "TYPE":"GROUP", "MSG": "For your information",
    "FROM": "Tony@hku.hk"}
    ```

MSG Command

- e.g., broadcast message
  - {"CMD": "MSG", "TYPE":"ALL", "MSG": " Dear all, you can reach me via 91176842", "FROM": "henry@hku.hk"}

Computer Platform

- You should develop and test your Chat programs in any platform installed with Python 3.6 or above
  - Mac OS, Linux, Windows
- Script files – quickly start 4 chat clients and the chatserver
  - start.ps1 for Windows
  - start-linux.sh for Linux
  - start-OSX.sh, & start-OSX-tab.sh for Mac OSX

Submission

- Deadline: **April 1, 2022 (Friday) at 17:00**.
- Name the client to ChatApp.py
- Name the server to Chatserver.py

Be careful if your work is based on Workshop 2 program, they have the same name!!!

- At the head of the submitted programs, state the
  - Student name and No.:
  - Development platform:
  - Python version:

Grading Rubric

Documentation	• **Include the required program and student's info at the beginning of the program (-0.5 points)**
ChatApp program (8 points)	• [JOIN] button (2 points) • [SEND] button (3 points) • [LEAVE] button (0.5 points) • Display peer list (1 point) • Display received chat messages (1.5 points)
Chatserver program (6 points)	• Connection and peer management (3 points) • Handle chat messages (3 points)