COMP9331: Computer Networks and Applications

Assignment for Term 2, 2020

Name: Sidney Tandjiria
zID: z5272671

Language: Python 3.7 (tested on VLAB)
Files submitted: server.py, client.py, report.pdf

1. PROGRAM DESCRIPTION AND PURPOSE

The purpose of this project is to simulate the CovidSafe digital contact tracing app, based on the BlueTrace protocol. A client-server architecture is used, where multiple smartphone clients can communicate with a server via TCP, and with each other via UDP.

Two programs are used in this simulation:

- **server.py:** The server is responsible for authenticating users, generating tempIDs, and generating contact details of known contacts for a user that has tested positive to COVID-19.
- **client.py:** The client is responsible for allowing users to login, request a tempID, beacon other clients and to upload their contact log to the server.

2. PROGRAM DESIGN AND CONSIDERATIONS

2.1. Server

The server is designed to allow many clients to connect to it simultaneously. In the server program, Python's *selectors* module is used for I/O multiplexing. This module can check if a socket has I/O ready for reading or writing, which makes it suitable for handling multiple clients (Jennings, 2020).

The body of the server program is in a while loop, which listens for incoming messages from clients. There are two main type of messages addressed in the loop:

- A new client requesting a connection – in this case a new connection socket is created, and the socket is registered using *select.*
- A request from a client who is already connected – in this case the server parses the client's request and sends back an appropriate response and/or carries out required actions. To ensure messages received are complete, the server checks the header of messages to determine the length of the contents, and only parses it once all bytes are received. This is important in the case of the contact log, which may be longer than the other messages received by the server.

There are 3 functions that support the server program:

- **create_response**: This function determines the response to be sent back to a client and/or any actions that the server needs to carry out. The function will create the application-layer message, attach a header specifying the length of the response and encode it. Other actions fulfilled by this function are authenticating clients (by comparing inputs against credentials.txt), blocking clients (and starting the block duration timer) and writing to the tempIDs.txt file when a user downloads a tempID.
- **remove_block**: This function unblocks a blocked user and is called by *threading.Timer* in the *create_response* function after a user enters the wrong password three times and the block duration has elapsed. The *threading.Timer* function creates a new thread which waits for a specified amount of time before calling the target function *remove_block.*
- **check_contact_log**: When a client uploads their contact log, this function does the mapping from tempIDs to usernames and displays it on the server terminal. The data from the contact log and the data stored in tempIDs.text are transformed into pandas DataFrames, which can then be merged to get the mapping efficiently. This function also handles the cases where no valid mappings are found.

The server keeps track of the following state variables:

- A list of connected clients
- A dictionary which acts as a counter of password attempts for each user
- A list of blocked clients
- A list of tempIDs and start/end times associated with a user, stored in the tempIDs.txt file (this file is created when the server program is started).

The server program stays online indefinitely once it is executed.

2.2. Client

The client is responsible for initialising a TCP connection with the server. The client also creates two UDP sockets, one for receiving beacons, and another to send beacons.

At the beginning, the client prompts the user for a username, and if valid, a password. Once a user successfully authenticates, the client enters a loop to wait for user commands and/or receive beacons. Multithreading is used to allow a user to input commands and receive beacons concurrently. A thread for deleting old beacons from the contact log after 3 minutes is also used (using *threading.Timer*).

The body of the main while loop addresses the following user commands:

- **logout**: The client program sends a message to the server that the user is logging out, and subsequently closes the connection. At this point, the client terminal is closed.
- **Download_tempID**: The client sends a message requesting a tempID and waits for the server to provide one. This tempID (and its start and end time) is stored locally by the client (overwriting existing records), so that it can be accessed when the user wants to beacon another user.
- **Upload_contact_log**: The client checks that the contact log is not empty and reads in the contact log file as a formatted string. This string is then sent to the server.
- **Beacon <dest IP> <dest port>**: The client attempts to send its TempID to the specified IP and port using the UDP port. If the user doesn't have a tempID yet, the program will prevent them from beaconing and ask them to download one. In the event that a user has a tempID but it is expired, the program will warn the user, but will proceed to send the beacon (this was intentional, so that we can test the case where a received beacon is invalid).

There are 4 functions that support the client program:

- **create_message**: This function checks that the user issued a valid command and creates an application-layer message containing the command, attaches a header specifying the length of the message and encodes it, ready to be sent to the server.
- **send_beacon**: This function takes in the user's current tempID and start/end time, creates the message and sends it via UDP to the specified IP and port.
- **recv_beacon**: This function listens out for incoming UDP beacons from other clients, checks if they are valid, and if so, adds them to the user's contact log. This function is run using a different thread, to ensure that a user can concurrently receive beacons and input commands. This function also sets up a threaded timer to remove a contact log entry after 3 minutes.
- **remove_from_contact_log:** This function is called in recv_beacon after 3 minutes of a beacon being received. It opens the contact log file and removes the first occurrence of the specified beacon to delete (the same beacon may occur multiple times).

The client keeps track of the following state variables:

- The user's most current tempID and its start/end time
- A list of all received and unexpired beacons, stored in the contact log.

The client program closes once a user logs out.

3. APPLICATION LAYER MESSAGE FORMAT

The application layer message format used is below. The message has a fixed length header that indicates how long the content is. The content itself is encoded in JSON, with two fields (*command* and *value*, or *command* and *status* if it is a response from the server). The message is encoded in UTF-8.

JSON was chosen as it is readable and readily encoded/decoded to/from python dictionaries.

<table>
<tr><td>HEADER (fixed length – 2 bytes)
Size of content length in bytes</td></tr>
<tr><td>CONTENT (variable length – specified in header)
Content encoded in JSON format. Example:
{
‘command’: ‘check_password’,
‘value’:
{
‘username’: ‘+61410666666’,
‘password’: ‘kara1234’
}
}</td></tr>
</table>

The *command* field tells the server how to interpret the attached *value* field. The table below summarises the possible structure of *value* and responses from the server.

command	**value**	**Possible responses returned from server**
check_username	username	OK: Username is valid NOT FOUND: Username does not exist BLOCKED: User is currently blocked
check_password	‘username’: username, ‘password’: password	OK: Password is valid INCORRECT: Password is incorrect BLOCKED: Password is incorrect the third time
logout	username	OK: User is successfully logged out
Download_tempID	username	‘tempID’: tempID, ‘startTime’: startTime, ‘endTime’: endTime
Upload_contact_log	‘username’: username, ‘contactlog’: contactlog	No response from server
Beacon <destIP> <destPort>	‘tempID’: tempID, ‘startTime’: startTime, ‘endTime’: endTime, ‘version’: 1	No response from receiving client

4. POSSIBLE IMPROVEMENTS

Possible improvements to the program are listed below:

- Functionality for a new user to register their details and store it in credentials.txt
- Client automatically downloads a new tempID when it expires
- Beacons are automatically sent to “nearby” users. The program can implement a way to simulate “nearness” – e.g. using randomly generated location data
- More secure authentication process (hide the password in terminal, encrypt it etc.)
- A more efficient/extensible message format (e.g. protocol buffers). The current format is not extensible and requires changing both the server and client programs if a new field is required.

5. REFERENCES

Jennings, N., 2020. *Socket Programming In Python (Guide) – Real Python*. [online] Realpython.com. Available at: <https://realpython.com/python-sockets/> [Accessed 27 July 2020].

Docs.python.org. 2020. *Selectors — High-Level I/O Multiplexing — Python 3.8.5 Documentation*. [online] Available at: <https://docs.python.org/3/library/selectors.html#selectors.BaseSelector.select> [Accessed 27 July 2020].

Docs.python.org. 2020. *Threading — Thread-Based Parallelism — Python 3.8.5 Documentation*. [online] Available at: <https://docs.python.org/3/library/threading.html> [Accessed 27 July 2020].