COMP3331: ASSIGNMENT REPORT z5112961

Program Design

The Bluetrace program is designed to simulate the COVIDSafe application. The server communicates with the clients (users) using TCP socket communication. The clients can log in to the server and there is appropriate credential checking. Once logged in they can do a variety of things including downloading a temporary ID and uploading their contact log by sending a message to the server. Several clients may be logged into the server at once. They can communicate with each other using UDP communication. Clients beacon each other which simulates the BLE encounters the COVIDSafe app records. The application uses multithreading to handle time delays and multiple users and file writing. Any time a file is written to a lock is used to stop two threads attempting to write to a file at once.

General overview of the Code

- Server is set up. Client is set up as TCP client for server.
- The server waits for a message from client in a while loop with a thread while the client waits for credential input from the user in a while loop.
- Once the user inputs credentials the client sends the information to the server for the server to validate
- If successful the server sets up another loop to wait for the next command and client is set up as a UDP client and server for beaconing purposes using function central(). If unsuccessful there are error messages to allow re-entering. After three attempts the user is blocked for the time inputted as block_duration, inputted as argument on server start up using function login_delay().
- User inputs commands (Download_tempID, Upload_contact_log and logout) into the client and the client does processing and sends info to the server which then processes the information using various functions and threads where needed and prints error messages if needed
- User can also beacon other clients which calls a function peripheral() to handle the call.
- User can logout with command logout and client sends info to the server for the server to take note of.

Application Layer Message Format

The table below shows the various outputs (including error messages) when certain commands are entered on both the server and client terminals. It also gives the format for the TCP and UDP messages sent. The user input is always in the client terminal. The server does not accept any user input. N.B. For the peer to peer, the server terminal is the central client (the one receiving beacons) and the client terminal is the peripheral client (the one sending beacons).

User Input	**Description**	**Server/Central Terminal**	**Client/Peripheral Terminal**	**Message Format**
Download_tempID	Generates tempID	Temp ID: <tempID>	TempID: <tempID>	“<tempID>”
Upload_contactlog	Sends contact log to server	Received contact log from: <user> <tempID>, <start>, <expiry>;	<tempID>, <start>, <expiry>;	“<tempID> <start> <expiry>”

		Contact log checking <uname>, <start>, <tempID>		
Upload_contactlog	If there is no file or the file is empty	Error: No contact log	Error: No contact log	”None”
Beacon <destIP> <destPort>	Beacon command is correct and port is valid and beacon is valid	Received beacon: <tempID>, <start>, <expiry>. Current time is: <currentTime> The beacon is: valid	-	(“<tempID> <start> <expiry> <version>”, <serverAddressPort>)
Beacon <destIP> <destPort>	Beacon command is correct and port is valid and beacon is invalid	Received beacon: <tempID>, <start>, <expiry>. Current time is: <currentTime> The beacon is: invalid	-	(“<tempID> <start> <expiry> <version>”, <serverAddressPort>)
Beacon <destIP> <destPort>	Beacon command is invalid	Error: Invalid command	-	-
logout		<user> logout	Program exits	-
Any other command		-	Error: invalid command	

Design Trade-offs and Considerations

The application has a couple of design trade-offs including the ability for a client to beacon itself. While this does not affect the applications functioning it is unnecessary for the client to be able to log contact with itself. A second design flaw is the system is only able to handle two users logging in at the exact instant because of amount allowed in the buffer. For the current use of the application this is not an issue as many clients can log in so long as it is not at the exact same time however for future expansion of the application it would be essential to rectify this problem

Possible Improvements and Extensions

The application currently works for all aspects of the spec however there are some improvements and features that could be added in the future. For example, currently the client can beacon itself. While this technically is not wrong in the scope of the spec, in practise it does not make a lot of sense for a user to record contact with themselves. Having a flag to check whether the beacon request is to its own port would rectify this issue. Additionally, a user can log into multiple terminals at once. To increase security for the user, it would be wise to implement a feature that logs the old terminal out if a user successfully logs in on another terminal. Finally developing a frontend graphical user interface would allow for a better user experience and would be essential for further development of the application.