COMP3331/9331 Computer Networks and Applications

Assignment for Term 2, 2020

Version 1.1

Due: 11:59am (noon) Friday, 07 August 2020 (Week 10)

Updates to the assignment, including any corrections and clarifications, will be posted on the subject website. Please make sure that you check the subject website regularly for updates.

1. Change Log

Version 1.1 released on 18th July 2020.

2. Goal and learning objectives

COVIDSafe is a digital contact tracing app announced by the Australian Government to help combat the ongoing COVID-19 pandemic, which has infected more than 6 million people and killed more than 373,000. The pandemic has also caused major society and economy disruptions around the world, and effective contact tracing is one of conditions that we can go back to ``normal'' life before the development of a COVID-19 vaccine. The app is based on the BlueTrace protocol developed by the Singaporean Government. In this assignment, you will have the opportunity to implement a BlueTrace protocol simulator and investigate its behaviours. Your simulator is based on a client server model consisting of one server and multiple smartphone clients. The clients communicate with the server using TCP and among themselves using UDP. The server is mainly used to authenticate the clients and retrieve the contact logs in the event of a positive COVID-19 detection since the logs are stored locally in the smartphones for 21 days only to protect the privacy of the users. The clients broadcast and receive beacons (UDP packets) when they are in the communication range.

2.1 Learning Objectives

On completing this assignment, you will gain sufficient expertise in the following skills:

1. Detailed understanding of how client-server and client-client interactions work.
2. Expertise in socket programming.
3. Insights into implementing an application layer protocol.

3. Assignment Specification

The base specification of the assignment is worth **20 marks**. The specification is structured in two parts. The first part covers the basic interactions between the clients and server and includes functionality for clients to communicate with the server. The second part asks you implement additional functionality whereby two clients can exchange messages with each other directly in a peer-to-peer fashion. This first part is self-contained (Sections 3.2 – 3.4) and is worth **15 marks**. Implementing peer-to-peer messaging (beaconing) (Section 3.5) is worth **5 marks**. **CSE** students are expected to implement both functionalities. **Non-CSE** students are only required to implement the

first part (i.e. no peer-to-peer messaging). The marking guidelines are thus different for the two groups and are indicated in Section 7.

The assignment includes 2 major modules, the server program and the client program. The server program will be run first followed by multiple instances of the client program (Each instance supports one client). They will be run from the terminals on the same and/or different hosts.

3.1. BlueTrace[1] Overview

One of the core considerations of BlueTrace is to preserve user privacy. To this end, personal information is collected just once at point of registration and is only used to contact potentially infected patients. Contact tracing is done entirely locally on a client device using Bluetooth Low Energy (BLE) or UDP for this assignment, storing all encounters in a contact history log chronicling encounters for the past 21 days. Users in the contact log are identified using anonymous ``temporary IDs'' (TempIDs) issued by the health authority server. This means a user's identity cannot be ascertained by anyone except the health authority with which they are registered. Furthermore, since TempIDs change randomly on a regular basis (e.g., every 15 minutes), malicious third parties cannot track users by observing log entries over time.

The protocol is focused on two areas: locally logging registered users in the vicinity of a device (i.e., peer to peer communication or P2P), and the transmission of the log to the operating health authority (i.e., client to server communication or C2S). In the context of this assignment, the P2P component operates on top of the unreliable UDP communication, defining how two devices acknowledge each other's presence. The C2S component uses reliable TCP to communicate a timeline of visits to a centralised server owned by a health authority once a user has tested positive for COVID-19. The health authority can then, using the log, notify the users who came in contact with the infected patient.

3.2. Server

The reporting server is responsible for handling initial registration, provisioning unique user identifiers, and collecting contact logs created by the P2P part of the protocol. When the user first launches a BlueTrace app they will be asked to create a UserID (internationally format mobile phone number, e.g., +61-410-888-888) and password. This phone number is later used if the user has registered an encounter in an infected patient's contact log. Once registered, users are provisioned TempID uniquely identifying them to other devices. Each TempID has a lifetime of a defined period (e.g., 15 minutes) to prevent malicious parties from performing replay attacks or tracking users over time with static unique identifiers. Therefore, the server has the following responsibilities.

User Authentication - When a client requests for a connection to the server, e.g., for obtaining a TempID or uploading contact logs after being tested as a COVID-19 positive, the server should prompt the user to input the username and password and authenticate the user. The valid username and password combinations will be stored in a file called *credentials.txt* which will be in the same directory as the server program. An example *credentials.txt* file is provided on the assignment page. Username and passwords are case-sensitive. We may use a different file for testing so DO NOT hardcode this information in your program. You may assume that each username and password will be on a separate line and that there will be one white space between the two. If the credentials are correct, the client is considered to be logged in and a welcome message is displayed. When all tasks)are done (e.g., TempID has been obtained or the contact log has been uploaded), a user should be able to logout from the server.

On entering invalid credentials, the user is prompted to retry. After 3 consecutive failed attempts, the

[1] Some aspects of the BlueTrace have been modified according to the context of this assignment.

user is blocked for a duration of *block_duration* seconds (*block_duration* is a command line argument supplied to the server) and cannot login during this duration (even from another IP address).

TempID Generation - TempIDs are generated as a 20-byte random number, and the server uses a file (*tempIDs.txt,* which will be in the same directory as the server program) to associate the relationship between TempIDs and the static UserIDs. An example *tempIDs.txt* file is provided on the assignment page.

Contact log checking - Once a user has been tested as a CVOID-19 positive, he/she will upload his/her contact log to the reporting server. The contact log is in the following format: TempID (20 bytes), start time (19 bytes) and expiry time (19 bytes). Then, the server will map the TempID to reveal the UserID and retrieve start time and expiry time (with help of the *tempIDs.txt* file). The health authority can then contact the UserID (phone number) to inform a user of potential contact with an infected patient. Therefore, you program will print out a list of phone numbers and encounter timestamps.

3.3. Client

The client has the following responsibilities -

Authentication - The client should provide a login prompt to enable the user to authenticate with the server.

Download TempID - After authentication, the client should be able to download a TempID from the server and display it.

Upload contact log - The client should be able to upload the contact logs to the server after authentication. For NON-CSE students, you may read the contact logs from a static file *(<your zID>_contactlog.txt).* For CSE students, you should generate the content of *<your zID>_contactlog.txt* file dynamically as discussed in Section 3.5 below.

3.4 Commands supported by the client

After a user is logged in, the client should support all the commands shown in the table below. For the following, assume that commands were run by user A.

Command	Description
Download_tempID	Download TempID from the server.
Upload_contact_log	Upload contact logs to the server.
logout	log out user A.

Any command that is not listed above should result in an error message being displayed to the user. The interaction with the user should be via the terminal (i.e. console).

We do not mandate the exact text that should be displayed by the client to the user for the various commands. However, you must make sure that the displayed text is easy to comprehend. Please make sure that you DO NOT print any debugging information on the client terminal.

Some examples illustrating client server interaction using the above commands are provided in Section 8.

3.5 Peer to Peer Communication Protocol (beaconing)

The P2P part of the protocol defines how two devices communicate and log their contact. We will simulate BLE encounters with UDP in this section. Each device is in one of two states, Central or Peripheral. The peripheral device sends a packet with the following information to the central device: TempID (20 bytes), start time (19 bytes), expiry time (19 bytes) and BlueTrace protocol version (1 byte). After receiving the packet/beacon, the central device will compare current timestamp with the start time and expiry time information in the beacon. If the timing information is valid (i.e., current timestamp is between start time and expiry time), the central device will store the beacon information in a local file (<*your zID*>*_contactlog.txt*) for 3 minutes[2]. Note that a client can behave in either Central or Peripheral states.

To implement this functionality your client should support the following command and remove the outdated (i.e., older than 3 minutes) contact log automatically (in addition to those listed in Section 3.4)

Command	**Description**
Beacon <dest IP> <dest port>	This command sends a beacon to another user (identified by destination IP address <dest IP> and UDP port number <dest port>). Note that in BlueTrace app, a beacon will be broadcasted via the BLE interface of a smartphone and all other BLE devices in the transmission range will receive the beacon. In this assignment, we use UDP unicast to mimic (and significantly simplify) the process. When you are testing your assignment, you may run the server and multiple clients on the same machine on separate terminals. In this case, use 127.0.0.1 (local host) as the dest IP address.

3.6 File Names & Execution

The main code for the server and client should be contained in the following files: `server.c`, or `Server.java` or `server.py`, and `client.c` or `Client.java` or `client.py`. You are free to create additional files such as header files or other class files and name them as you wish.

The server should accept the following two arguments:

- `server_port`: this is the port number which the server will use to communicate with the clients. Recall that a TCP socket is NOT uniquely identified by the server port number. So it is possible for multiple TCP connections to use the same server-side port number.
- `block_duration`: this is the duration in seconds for which a user should be blocked after three unsuccessful authentication attempts.

The server should be executed before any of the clients. It should be initiated as follows:

If you use Java:

[2] It is 21 days in COVIDSafe or BlueTrace.

```
java Server server_port block_duration
```

If you use C:

```
./server server_port block_duration
```

If you use Python:

```
python server.py server_port block_duration
```

The client should accept the following three arguments:

- `server_IP`: this is the IP address of the machine on which the server is running.
- `server_port`: this is the port number being used by the server. This argument should be the same as the first argument of the server.
- `client_udp_port`: this is the port number which the client will listen to/wait for the UDP traffic/beacons from the other clients.

Note that, you do not have to specify the TCP port to be used by the client. You should allow the OS to pick a random available port. Similarly, you should allow the OS to pick a random available UDP source port for the UDP client. Each client should be initiated in a separate terminal as follows:

If you use Java:

```
java Client server_IP server_port client_udp_port
```

If you use C:

```
./client server_IP server_port client_udp_port
```

If you use Python:

```
python client.py server_IP server_port client_udp_port
```

Note: When you are testing your assignment, you can run the server and multiple clients on the same machine on separate terminals. In this case, use 127.0.0.1 (local host) as the server IP address.

4. Additional Notes

- This is NOT group assignment. You are expected to work on this individually.
- **Tips on getting started**: The best way to tackle a complex implementation task is to do it in stages. A good place to start would be to implement the functionality to allow a single user to login with the server. Next, add the blocking functionality for 3 unsuccessful attempts. Then extend this to handle multiple clients. Once your server can support multiple clients, implement the functions for download TempID and upload contact logs. Note that, this may require changing the implementation of some of the functionality that you have already implemented. Once the communication with the server is working perfectly, you can move on to peer-to-peer communication. It is imperative that you rigorously test your code to ensure that all possible (and logical) interactions can be correctly executed. Test, test and test.
- **Application Layer Protocol:** Remember that you are implementing an application layer protocol for realising contact trace service to counter the COVID-19 pandemic. We are only considered with the end result, i.e. the functionality outlined above. You may wish to revisit some of the application layer protocols that we have studied (HTTP, SMTP, etc.) to see examples of message format, actions taken, etc.

- **Transport Layer Protocol:** You should use TCP for the communication between each client and server, and UDP for P2P communication. The TCP connection should be setup by the client during the login phase and should remain active until the user logs out, while there is no such requirement for UDP. The server port of the server is specified as a command line argument. Similarly, the server port number of UDP is specified as a command parameter of the client. The client ports for both TCP and UDP do not need to be specified. Your client program should let the OS pick up random available TCP or UDP ports.
- **Backup and Versioning:** We strongly recommend you to back-up your programs frequently. CSE backups all user accounts nightly. If you are developing code on your personal machine, it is strongly recommended that you undertake daily backups. We also recommend using a good versioning system such as github or bitbucket so that you can roll back and recover from any inadvertent changes. There are many services available for both which are easy to use. We will NOT entertain any requests for special consideration due to issues related to computer failure, lost files, etc.
- **Language and Platform**: You are free to use C, JAVA or Python to implement this assignment. Please choose a language that you are comfortable with. The programs will be tested on CSE Linux machines. So please make sure that your entire application runs correctly on these machines (i.e. your lab computers) or using VLAB. This is especially important if you plan to develop and test the programs on your personal computers (which may possibly use a different OS or version or IDE). Note that CSE machines support the following: **gcc version 8.2, Java 11, Python 2.7 and 3.7. If you are using Python, please clearly mention in your report which version of Python we should use to test your code.** You may only use the basic socket programming APIs providing in your programming language of choice. You may not use any special ready-to-use libraries or APIs that implement certain functions of the spec for you.
- There is no requirement that you must use the same text for the various messages displayed to the user on the terminal as illustrated in the examples in Section 9. However, please make sure that the text is clear and unambiguous.
- You are encouraged to use the forums on WebCMS to ask questions and to discuss different approaches to solve the problem. However, you should **not** post your solution or any code fragments on the forums.
- We will arrange for additional consultation hours in Weeks 7, 8 and 9 to assist you with assignment related questions if needed.

5. Submission

Please ensure that you use the mandated file name. You may of course have additional header files and/or helper files. If you are using C, then you MUST submit a makefile/script along with your code (not necessary with Java or Python). This is because we need to know how to resolve the dependencies among all the files that you have provided. After running your makefile we should have the following executable files: `server` and `client`. In addition, you should submit a small report, `report.pdf` (no more than 3 pages) describing the program design, the application layer message format and a brief description of how your system works. Also discuss any design tradeoffs considered and made. Describe possible improvements and extensions to your program and indicate how you could realise them. If your program does not work under any particular circumstances, please report this here. Also indicate any segments of code that you have borrowed from the Web or other books.
You are required to submit your source code and `report.pdf`. You can submit your assignment using the give command in a terminal from any CSE machine (or using VLAB or connecting via SSH

to the CSE login servers). Make sure you are in the same directory as your code and report, and then do the following:

1. Type tar -cvf assign.tar filenames
e.g. `tar –cvf assign.tar *.java report.pdf`

2. When you are ready to submit, at the bash prompt type `3331`

3. Next, type: `give cs3331 assign assign.tar` (You should receive a message stating the result of your submission). Note that, COMP9331 students should also use this command.

Alternately, you can also submit the tar file via the WebCMS3 interface on the assignment page.

Important notes

- The system will only accept `assign.tar` submission name. All other names will be rejected.
- **Ensure that your program/s are tested in CSE Linux machine (or VLAB) before submission. In the past, there were cases where tutors were unable to compile and run students' programs while marking. To avoid any disruption, please ensure that you test your program in CSE Linux-based machine (or VLAB) before submitting the assignment. Note that, we will be unable to award any significant marks if the submitted code does not run during marking.**

- You may submit as many times before the deadline. A later submission will override the earlier submission, so make sure you submit the correct file. Do not leave until the last moment to submit, as there may be technical, or network errors and you will not have time to rectify it.

Late Submission Penalty: Late penalty will be applied as follows:
- 1 day after deadline: 10% reduction
- 2 days after deadline: 20% reduction
- 3 days after deadline: 30% reduction
- 4 days after deadline: 40% reduction
- 5 or more days late: NOT accepted

NOTE: The above penalty is applied to your final total. For example, if you submit your assignment 1 day late and your score on the assignment is 10, then your final mark will be 10 – 1 (10% penalty) = 9.

6. Plagiarism

You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Do not post this assignment on forums where you can pay programmers to write code for you. We will be monitoring such forums. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to **ZERO**. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the

discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or a URL when the code appears (as comments). Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

7. Marking Policy

You should test your program rigorously before submitting your code. Your code will be marked using the following criteria:

The following table outlines the marking rubric for both CSE and non-CSE students:

Functionality	**Marks (CSE)**	**Marks (non-CSE)**
Successful log in and log out for single client	0.5	0.5
Blocking user for specified interval after 3 unsuccessful attempts (even from different IP)	1	2
Successful log in for multiple clients (from multiple machines)	1	2
Correct Implementation of Download_tempID	2.5	3.5
Correct Implementation of Upload_contact_log	3	4
Correct Implementation of Contact log checking	3	4
Properly documented report	2	2
Code quality and comments	2	2
Peer to peer communications including removing beacons older than 3 minutes	5	N/A

NOTE: While marking, we will be testing for typical usage scenarios for the above functionality and some straightforward error conditions. A typical marking session will last for about 15 minutes during which we will initiate at most 5 clients. However, please do not hard code any specific limits in your programs. We won't be testing your code under very complex scenarios and extreme edge cases.

8. Sample Interaction

Note that the following list is not exhaustive but should be useful to get a sense of what is expected. We are assuming Java as the implementation language.

Case 1: Successful Login

Terminal 1

```
>java Server 4000 60
```

Terminal 2

```
>java Client 10.11.0.3 4000 8000(assume that server is executing on 10.11.0.3)
> Username: +61410888888
> Password: comp3331
> Welcome to the BlueTrace Simulator!
>
```

Case 2: Unsuccessful Login (assume server is running on Terminal 1 as in Case 1)

Terminal 2

```
>java Client 10.11.0.3 4000 8000    (assume that server is executing on 10.11.0.3)
> Username: +61410888888
> Password: comp9331
> Invalid Password. Please try again
> Password: comp8331
> Invalid Password. Please try again
> Password: comp7331
> Invalid Password. Your account has been blocked. Please try again later
```

The user should now be blocked for 60 seconds (since *block_time* is 60). The terminal should shut down at this point.

Terminal 2 (reopened before 60 seconds are over)

```
>java Client 10.11.0.3 4000 8000 (assume that server is executing on 10.11.0.3)
> Username: +61410888888
> Password: comp3331
> Your account is blocked due to multiple login failures. Please try again later
```

Terminal 2 (reopened after 60 seconds are over)

```
>java Client 10.11.0.3 4000 8000 (assume that server is executing on 10.11.0.3)
> Username: +61410888888
> Password: comp3331
> Welcome to the BlueTrace Simulator!
>
```

Example Interactions

Consider a scenario where users `+61410999999`, `+61410888888` and `+61410666666` are currently logged in. In the following we will illustrate the text displayed at the terminals for all users and the server as the users execute various commands.

1. `+61410999999` executes `Download_tempID` followed by a command that is not supported. `+61410888888` executes `Download_tempID`. `+61410999999` and `+61410888888` execute `logout`.

+61410999999's Terminal	**+61410888888's Terminal**	**server's Terminal**
> Download_tempID > TempID: 12345678901234567890		> user: +61410999999 > TempID: 12345678901234567890
> whatsthedate > Error. Invalid command		
	> Download_tempID > TempID: 12345678901234567891	> user: +61410888888 > TempID: 12345678901234567891
> logout		> +61410999999 logout

	>logout	> +61410888888 logout

2. +61410999999 executes Upload_contact_log. The server initiates contact tracing.

+61410999999's Terminal	+61410888888's Terminal	server's Terminal
> Upload_contact_log 1234567890123456789 1, 13/05/2020 17:54:06, 13/05/2020 18:09:05; 1234567890123456789 2, 14/05/2020 17:54:06, 14/05/2020 18:09:05;		> received contact log from +61410999999 1234567890123456789 1, 13/05/2020 17:54:06, 13/05/2020 18:09:05; 1234567890123456789 2, 14/05/2020 17:54:06, 14/05/2020 18:09:05; > Contact log checking +61410888888, 13/05/2020 17:54:06, 1234567890123456789 1; +61410777777, 14/05/2020 17:54:06, 1234567890123456789 2; (assume that TempID 1234567890123456789 2 is assigned to user +6141077777.)

3. +61410888888 sends a beacon to +61410999999 (which is inside its BLE transmission range)

+61410999999's Terminal	+61410888888's Terminal	+61410666666's Terminal (a potential adversary)
> received beacon: 1234567890123456789 3, 14/05/2020 17:54:06, 14/05/2020 18:09:05. Current time is: 14/05/2020 18:00:01. The beacon is valid. > received beacon: 1234567890123456789 3, 14/05/2020 17:54:06, 14/05/2020 18:09:05. Current time is: 14/05/2020 18:10:01. The beacon is invalid.	> Beacon 10.11.0.5 8001 (assume that the IP address and UDP port number of +61410999999 are 10.11.0.5 and 8001 respectively). 1234567890123456789 3, 14/05/2020 17:54:06, 14/05/2020 18:09:05.	> Beacon 10.11.0.5 8001 1234567890123456789 3, 14/05/2020 17:54:06, 14/05/2020 18:09:05,