# COMP3331/9331 Computer Networks and Applications

# Assignment for Term 3, 2022

Release Date: XXX

**Due: 11:59 am (noon) Friday, 11 November 2022**

Note:
- This is an individual assignment (group submission not allowed).
- Updates to the assignment, including any corrections and clarifications, will be announced on the course WebCMS.
- Late submissions will attract a penaly of 5% per day late beyond the deadline (12 noon to 12 noon will be counted as one day). No submissions will be accepted after 5 days beyond the deadline (no submissions after 12 noon 16 November).

## Designing and Implementing an IoT Data Collection and Sharing Network

## 1. Change Log & Author

Version 1.0 is released on 30/09/2022, Wei Song

## 2. Goal and learning objectives

The Internet of Things, or IoT, refers to the billions of physical devices around the world that are now connected to the Internet, all collecting and sharing data for improving people's quality of life. For example, the smartwatches which are very popular nowadays, can collect data from the wearer and share data with its central sever for monitoring the health of the wearer. In this assignment, you will have the opportunity to implement your own version of a data collection and sharing network based on the idea of IoT. Your application is based on the client-server architecture consisting of one server and multiple clients (i.e., the edge IoT devices) communicating concurrently as well as peer-to-peer networks. Your application will support a range of functions that are typically found from the existing edge networks including authentication (between edge devices and the central server), data generation at the side of edge devices, and data sharing between edge devices and the central server, and between one edge device and another edge device. You will be designing custom communication protocols for the edge network based on TCP and UDP.

### 2.1 Learning Objectives

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

1. Detailed understanding of how client-server and client-client (peer-to-peer) interactions work.
2. Expertise in socket programming.
3. Insights into designing and implementing a communication protocol.

## 3. Assignment Specification

The assignment is worth **20 marks**. The specification is structured in two parts. The first part, which is covered in Sections 3.2 – 3.3, involves the basic interactions between the edge device (i.e., the client) and the central server and includes functionality for edge devices to communicate with the central server via TCP. The second part, covered in Section 3.4, asks you to implement additional functionality whereby two edge devices can exchange video files with one another directly in a peer-to-peer fashion via UDP.

**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 data exchange between two edge devices over UDP). 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.

**Non-CSE Student**: The rationale for this option is that students enrolled in a program that does not include a computer science component have had very limited exposure to programming and in particular working on complex programming assignments. A Non-CSE Student is a student who is not enrolled in a CSE program (single or double degree). Examples would include students enrolled exclusively in a single degree program such as Mechatronics or Aerospace or Actuarial Studies or Law. Students enrolled in dual degree programs that include a CSE program as one of the degrees do not qualify. Any student who meets this criterion and wishes to avail of this option MUST email cs3331@cse.unsw.edu.au to seek approval before **5 pm**, 7 **October (Friday, Week 4)**. We will assume by default that all students are attempting the CSE version of the assignment unless they have sought explicit permission. **No exceptions**.

## 3.1. Assignment Specification

In this programming assignment, you will implement the client (Note that we will use client and edge device interchangeably) and server programs of an edge network, similar in many ways to the existing edge networks. Your application will support a range of operations including authenticating an edge device and let the edge device join the network, generating data at the edge devices, uploading the data from edge devices to the central server, interacting with the central sever for acquiring computation services, deleting the data from the central server, reading active edge devices' information, and exchanging data files from one edge device to another edge device (**CSE Students only**). You will design and implement a communication protocol for the edge network to achieve these functions. The server will listen on a port specified as a command line argument and will wait for a client to connect. The client program will initiate a TCP connection with the server. Upon connection establishment, the client will initiate the authentication process. The client program will interact with users through the command-line interface. Following successful authentication, the client will initiate one of the available commands. All commands require a simple request-response interaction between the client and server or two clients (**CSE Students only**). The client may execute a series of commands (one after the other) and eventually quit. *Both the client and server MUST print meaningful messages at the command prompt that capture the specific interactions taking place. You are free to choose the precise text that is displayed*. Examples of client-server interactions are given in Section 8.

## 3.2 Authentication

When a client requests a connection to the server, e.g., for joining an edge network, the server should prompt the client to input the edge device name and password and authenticate the edge device. The valid edge device name 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. **Edge device names and passwords are case-sensitive, and you can assume that the edge device name is unique for the context of this assignment**. We may use a different file for testing so DO NOT hardcode this information in your program. You may assume that each edge device name and password will be on a separate line and that there will be one white space between the two. Device names and passwords will not contain any white space. If the credentials are correct, the edge device is considered to be successfully authenticated and joined the edge network and a welcome message is displayed.

On entering invalid credentials, the client is prompted to retry. After several consecutive failed attempts, this edge device is blocked for 10 seconds (the *number* is an integer command-line argument supplied to the server and the valid value of the *number* should be between 1 and 5) and cannot join the network during this 10-second duration (even from another IP address). If an invalid *number* value (e.g., a floating-point value, 0 or 6) is supplied to the server, the server prints out a message such as "Invalid number of allowed failed consecutive attempts: *number*. The valid value of argument number is an integer between 1 and 5".

**For non-CSE Students:** After an edge device has joined the network successfully, i.e, the server authenticates the edge device successfully, the server should record a timestamp of the edge device joining and the device name in the active edge device log file (*edge-device-log.txt*, you should make sure that write permissions are enabled for *edge-device-log.txt*). Active edge devices are numbered starting at 1:

```
Active edge device sequence number; timestamp; edge device name

1; 30 September 2022 10:31:13; supersmartwatch
```

**For CSE Students:** After an edge device has joined the network successfully, the edge device (i.e., the client) should next send the server the UDP port number on which it will listen for P2P connections. The server should record a timestamp of the edge device joining, the edge device name, its IP address, and its UDP port number, in the active edge device log file (*edge-device-log.txt*):

```
Active edge device sequence number; timestamp; edge device name; edge
device IP address; edge device UDP server port number

1; 30 September 2022 10:31:13; supersmartwatch; 129.64.31.13; 5432
```

For simplicity, an edge device will join the network once at any given time, e.g., multiple joins concurrently are not allowed, and we won't test this case.

## 3.3. Commands

After the edge device has successfully joined the edge network, the client needs to display a message informing all available commands and prompting it to select one command. In the context of this assignment, the following commands are available: EDG: Edge Data Generation which means the client side helps to generate data to simulate the data collection function in the real edge device, UED: Upload Edge Data, it allows the edge device to upload a particular edge data file to the central server, SCS: Server Computation Service, the edge device can practice this command to request the server to do some basic computations on a particular data file, DTE: Delete the data file (server side), AED: Active Edge Devices, request and display the active edge devices, OUT: exit this edge network, and UVF: Peer-to-peer Uploading Video Files (for **CSE Students only**). All available commands should be displayed in the first instance after the edge device has joined the network. Subsequent prompts for actions should include this same message. If an invalid command is selected, an appropriate error message should be displayed, and they should be prompted to select one of the available commands.

In the following, the implementation of each command is explained in detail. The expected usage of each command (i.e., syntax) is included. **Note that, all commands should be upper-case (EDG, UED, etc.)**. All arguments (if any) are separated by a single white space and will be one word long (except messages which can contain white spaces and timestamps that have a fixed format of `dd mm yyyy hh:mm:ss` such as `30 September 2022 10:31:13`). **You may assume that the communication data files only contain numbers (i.e., integers).**

*There are 6 commands for **Non-CSE Students** and 7 commands for **CSE Students** respectively, which users can execute.* The execution of each command is described below.

**EDG: Edge Data Generation**

```
EDG fileID dataAmount
```

The fileID and dataAmount should be included as arguments, the fileID is an integer which is used to uniquely identify the file which will be utilised to store the generated data, the name of the file should be edge device name-fileID and the file type should be txt (e.g., *supersmartwatch-1.txt*). The dataAmount argument is used to indicate the number the data samples to be generated. You can randomly generate the data samples, there is no strict requirements for the data generation, for example, if dataAmount is specified as 10, then you can generate any 10 integers (e.g., from 1 to 10, from 20 to 30, or any other 10 integers) and store them into the data file. In addition, when you store the data samples into the file, you should follow the rule of "one line one number", an example (*supersmartwatch-1.txt*) is provided. Note that, if the file already exists in the subsequent EDG command calls, you should directly overwrite the existing data samples with the new generated data samples. If the fileID or dataAmount argument are missing from the EDG command, you should prompt a proper error message, for example, "EDG command requires fileID and dataAmount as arguments.", and if the provided fileID and dataAmount parameters are not integers, you should prompt a proper error message, for example, "the fileID or dataAmount are not integers, you need to specify the parameter as integers".

After the edge device successfully generates the data samples and stores them into the file, you should prompt a proper message (e.g., "data generation done") to indicate this command has been successfully processed by the edge device.

**UED: Upload Edge Data**

```
UED fileID
```

The fileID of the particular file the edge device is going to upload is included as the argument. Upon receiving this command the edge device (i.e., the client) is expected to read the data samples from the corresponding file and transfer the data samples to the central server using TCP. The client needs to check if the fileID argument is provided or not, you should prompt a message for example "fileID is needed to upload the data" if the fileID is missed. The client also needs to check if the corresponding file exists or not, you should prompt a message e.g., "the file to be uploaded does not exist" if the file does not exist at the edge device side. After the central server successfully receives the file, the server should send a message to the edge device to inform that the server has successfully received the file, and the client also should prompt a proper message to indicate that the file uploading is done successfully. In addition, the central sever should maintain an uploading log file named as *upload-log.txt*. If everything is good, the server should append an uploading log message in the following format:

```
edgeDeviceName; timestamp; fileID; dataAmount
```

```
supersmartwatch; 30 September 2022 10:31:13; 1; 10
```

**SCS: Server Computation Service**

```
SCS fileID computationOperation
```

This command is designed to request the powerful central server to do various computations, because in reality the edge devices normally have very limited computation resources. The fileID and computationOperation are included as two arguments. The fileID is used to indicate the corresponding data file used for the computation purpose. If the fileID is not provided or the fileID is not an integer the client should prompt a proper error message e.g., “fileID is missing or fileID should be an integer”. The server also needs to check if the corresponding file exists or not, if the file does not exist the server should respond to the client with a message informing the client that the file does not exist, and the client should prompt a proper message indicating the file does not exist at the server side. For simplicity, we define a total of four computation operations for this assignment: SUM, AVERAGE, MAX, MIN. SUM – calculate the sum of the data samples in the corresponding file, AVERAGE – get the average of the data samples in the corresponding file, MAX – get the maximum value among the data samples, and MIN – get the minimum value among the data samples. If the provided computation operation argument is not one of these four, the client should display a proper error message. If everything is good, the server should send the computation result to the edge device (i.e., the client) and it should display the result properly at the terminal.

### DTE: Delete the data file

```
DTE fileID
```

The fileID of the particular file the edge device is going to delete at the central server side is included as the argument, upon receiving this command the edge device (i.e., the client) is expected to send a message to the central server to request the server to delete the corresponding file with that fileID provided in the argument. The server needs to first check if the file exists or not, if the file does not exist the server should make a response to inform the client the file to be deleted does not exist and the client should display a proper error message at the terminal, for example, “the file does not exist at the server side”. If everything is good, the server should remove the file from its folder completely. In addition, the central sever should maintain a log file named as *deletion-log.txt*. If everything is good, the server should append a delete operation log message in the following format:

```
edgeDeviceName; timestamp; fileID; dataAmount
```

```
supersmartwatch; 30 September 2022 10:33:13; 1; 10
```

After that, the central server should respond to the client with a message to inform the client the file has been successfully deleted and the client should display a successful message (e.g., “file with ID of fileID has been successfully removed from the central server”).

### AED: Active Edge Devices

```
AED
```

There should be no arguments for this command. The central server should check if there are any other active edge devices apart from the edge device that sends the AED command. If so, the server should send the edge device names, and timestamps since the edge devices joined, (and their IP addresses and Port Numbers, **CSE Students only**) from the active edge device log file to the client (the server should exclude the information of the client, who sends the AED command to the server). The client should display all the information of all received edge devices at the terminal. If there are no other active edge devices, a notification message of “no other active edge devices” should be sent to the client and displayed. The client should next prompt to select one of the available commands.

**OUT: Exit edge network**

```
OUT
```

There should be no arguments for this command. The client should close the TCP connection, and exit with a goodbye message displayed at the terminal. The server should update its state information about currently active edge devices and the active edge device log file. Namely, based on the message (with the edge device name  information) from the client, the server should delete this edge device, which entails deleting the line containing this edge device in the active edge device log file (all subsequent active edge devices in the file should be moved up by one line and their active edge device sequence numbers should be updated appropriately) and confirmation should be sent to the client and displayed at the terminal. Note that all the data files and messages uploaded by this edge device must NOT be deleted. For simplicity, we won't test the cases where an edge device forgets to exit or exit is unsuccessful.

### 3.4 Peer to Peer Communication (Video file upload, CSE Students only)

The P2P part of the assignment enables one edge device to upload video files to another edge device using UDP (A good example for this scenario would be the video camera uploading its video files to the smartphone). Each edge device is in one of two states, Presenter or Audience. The Presenter edge device sends video files to the Audience edge device. Here, the presenter edge device is the UDP client, while the Audience edge device is the UDP server. After receiving the video files, the Audience edge device saves the files and the device name of the Presenter. Note that an edge device can behave in either Presenter or Audience state.

To implement this functionality your client program should support the following command, and two example video files are provided to help implement and test this command.

**UVF: Upload Video File**

**UVF deviceName filename**

The Audience edge device and the name of the file should be included as arguments. You may assume that the file included in the argument will be available in the current working directory of the client with the correct access permissions set (read). You should not assume that the file will be in a particular format, i.e., just assume that it is a **binary file**. The Presenter edge device (e.g., *uav-camera*) should check if the Audience edge device (indicated by the device name argument, e.g., *wei-smartpone*) is active (e.g., by issuing command AED). If *wei-smartphone* is not active, the Presenter client should display an appropriate error message (e.g., wei-smartphone is offline) at the prompt to *uav-camera* client. If *wei-smartphone* is active, *uav-camera* should obtain the wei-smartphone address and UDP server port number (e.g., by issuing command AED) before transferring the contents of the file to *wei-smartphone* via UDP. Here, *uav-camera* is the UDP client and *wei-smartphone* is the UDP server. The file should be stored in the current working directory of *wei-smartphone* with the file name presenter deviceName_filename (DO NOT add an extension to the name. If the filename has an extension mp4, e.g., test.mp4 should be stored as uav-camera_test.mp4 in our example). File names are case sensitive and one word long. After the file transmission, the terminal of *uav-camera* should next prompt to select one of the available commands. The terminal of *wei-smartphone* should display an appropriate message, e.g., a file (test.mp4) has been received from *uav-camera*, before prompting to select one of the available commands.

**TESTING NOTES**: 1) When you are testing your assignment, you may run the server program and multiple clients' programs on the same machine on separate terminals. In this case, use 127.0.0.1 (local host) as the destination (e.g., wei-smartphone in our example above) IP address. 2) For simplicity, we will run different clients at different directories, and won't test the scenario that a file is received when a client is typing/issuing a command, i.e. the receiving client doesn't need to be notified of the file transfer until after they issue their next command.

### 3.5 File Names & Execution

The main code for the server and client (i.e., the edge device) 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 that the server will use to communicate with the edge devices (i.e., 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.

- `number_of_consecutive_failed_attempts`: this is the number of consecutive unsuccessful authentication attempts before an edge device should be blocked for **10 seconds**. It should be an integer between 1 and 5.

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

If you use Java:

```
java Server server_port number_of_consecutive_failed_attempts
```

If you use C:

```
./server server_port number_of_consecutive_failed_attempts
```

If you use Python:

```
python server.py server_port number_of_consecutive_failed_attempts
```

Note that all references to `python` in this specification may be replaced by `python3` if you use Python 3 rather than Python 2.

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 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 randomly available UDP source port for the UDP client. Each client should be initiated in a separate terminal as follows:

**For non-CSE Students:**

If you use Java:

```
java Client server_IP server_port
```

If you use C:

```
./client server_IP server_port
```

If you use Python:

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

**For CSE Students:**

If you use Java:

```
java Client server_IP server_port client_udp_server_port
```

If you use C:

```
./client server_IP server_port client_udp_server_port
```

If you use Python:

```
python client.py server_IP server_port client_udp_server_port
```

**Note**: 1) The additional argument of client_udp_server_port for **CSE Students** for the P2P UDP communication is described in Section 3.4. In UDP P2P communication, one client program (i.e., Audience) acts as UDP server and the other client program (i.e., Presenter) acts as UDP client. 2)

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.

### 3.6 Program Design Considerations

#### Client Program Design

The client program should be fairly straightforward. The client needs to interact with people through the command-line interface and print meaningful messages. Section 8 provides some examples. **You do not have to use the exact same text as shown in the samples.** Upon initiation, the client should establish a TCP connection with the server and execute the authentication process to join the edge network. Following authentication, the client should be prompted to enter one of the available commands. Almost all commands require simple request/response interactions between the client with the server.

**For CSE Students**, the client program also involves P2P communication using UDP. Similar to the above, the user should be prompted to enter the available P2P communication command: UVF. This function should be implemented using a new thread since the client program may need to run other commands when the file is uploading. The thread will end when the upload finishes. Similarly, the client UDP server should be implemented with another thread. However, this thread should be run until the client goes offline since it is a UDP server thread. You should be particularly careful about how multiple threads will interact with the various data structures. Code snippets for multi-threading in all supported languages are available on the course webpage.

#### Server Program Design

When the server starts up, you can assume that there are no active edge devices. The server should wait for an edge device to connect, perform authentication, and service each command issued by the client sequentially. Note that, you will need to define several data structures for managing the current state of the edge network (e.g., active edge devices and data files) and the server must be able to interact with multiple edge devices simultaneously. A robust way to achieve this is to use multithreading. In this approach, you will need the main thread to listen for new connections. This can be done using the socket accept function within a while loop. This main thread is your main program. For each connected edge device/client, you will need to create a new thread. When interacting with one particular client, the server should receive a request for a particular operation, take necessary action and respond accordingly to the client and wait for the next request. You may assume that each interaction with a client is atomic. Consider that client A initiates an interaction (i.e., a command) with the server. While the server is processing this interaction, it cannot be interrupted by a command from another client B. Client B's command will be acted upon after the command from client A is processed. Once a client exits, the corresponding thread should also be terminated. You should be particularly careful about how multiple threads will interact with the various data structures. Code snippets for multi-threading in all supported languages are available on the course webpage.

## 4. Additional Notes

- This is **NOT a** 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 edge device to log in with the server. Next, add the blocking functionality for several unsuccessful attempts. Then extend this to handle multiple clients. Once your server can support multiple clients,

implement the functions for interacting with the server. Note that, this may require changing the implementation of some of the functionalities that you have already implemented. Once the communication with the server is working perfectly, you can move on to peer-to-peer communication (**CSE Students only**). It is imperative that you rigorously test your code to ensure that all possible (and logical) interactions can be correctly executed.

- **Application Layer Protocol:** Remember that you are implementing an application layer protocol for the edge network. We are only considering the end result, i.e., the functionalities outlined above. You may wish to revisit some of the application layer protocols that we have studied (HTTP, SMTP, etc.) to see examples of the message formats, actions taken, etc.

- **Transport Layer Protocol:** ***You should use TCP for the communication between each client and server, (and UDP for P2P communication between two clients, CSE Students only).*** The TCP connection should be set up by the client during the authentication phase and should remain active until the edge device exits, 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 **CSE Students only)**. The client ports for both TCP and UDP do not need to be specified. Your client program should let the OS pick any available TCP or UDP ports.

- **Backup and Versioning:** We strongly recommend you 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. Do not, however, post your code to a public repository. There are many services available 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., CSE 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 10.2, Java 11, Python 2.7 or 3.9. 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 provided 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 8. However, please make sure that the text is unambiguous.

- You are encouraged to use the forums on ED 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 - 9 to assist you with assignment-related questions if needed.*

## 5. Submission

Please ensure that you use the mandated file names. 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.

Alternatively, 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 as possible 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.

## 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 the 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 join in and exit out for single client | 0.5 | 0.5 |
| Blocking the user for 10 seconds after the specified number of unsuccessful attempts (even from different IP) | 1.5 | 1.5 |
| Successful connection for multiple clients (from multiple terminals) | 4 | 4 |
| Correct Implementation of EDG: Edge data generation | 1 | 2 |
| Correct Implementation of UED: Upload edge data | 1 | 2 |
| Correct Implementation of SCS: Server computation service | 1 | 2 |
| Correct Implementation of DTE: Delete the data file | 2 | 3 |
| Correct Implementation of AED: Active edge device | 2 | 3 |
| Properly documented report | 1 | 1 |
| Code quality and comments | 1 | 1 |
| Peer to peer communications including Correct Implementation of UVF: Upload Video File | 5 | N/A |
| TOTAL | 20 | 20 |

**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 Authentication (underline denotes input)

**Terminal 1**

```
>java Server 4000 3
```

**Terminal 2**

**For Non-CSE Students:**

```
>java Client 10.11.0.3 4000
```
(assume that server is executing on 10.11.0.3)

```
> Username: supersmartwatch

> Password: comp9331

> Welcome!

> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT):
```

**For CSE Students:**

```
>java Client 10.11.0.3 4000 8000
```
(assume that server is executing on 10.11.0.3)

```
> Username: supersmartwatch

> Password: comp9331

> Welcome!
```
(the client should upload the UDP port number 8000, i.e., the second argument, to the server in the background after a client is authenticated).

```
> Enter one of the following commands (EDG, UED, SCS, DTE, AED, UVF, OUT):
```

Case 2: Unsuccessful Authentication (assume server is running on Terminal 1 as in Case 1, underline denotes user input)

*The unsuccessful login examples below are for **Non-CSE Students.** For **CSE Students**, the client program should have an additional argument client_udp_server_port (see the example above with UDP port number 8000).*

**Terminal 2**

```
>java Client 10.11.0.3 4000
```
(assume that server is executing on 10.11.0.3)

```
> Username: supersmartwatch
> Password: comp3331
> 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 edge device should now be blocked for 10 seconds since the specified number of unsuccessful login attempts is 3. The terminal should shut down at this point.

**Terminal 2 (reopened before 10 seconds are over)**

`>java Client 10.11.0.3 4000 8000` (assume that server is executing on 10.11.0.3)

```
> Username: supersmartwatch
> Password: comp9331
> Your account is blocked due to multiple authentication failures. Please try again
  later
```

**Terminal 2 (reopened after 10 seconds are over)**

`>java Client 10.11.0.3 4000 8000` (assume that server is executing on 10.11.0.3)

```
> Username: supersmartwatch
> Password: comp9331
> Welcome!

> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT):
```

Example Interactions (underline denotes user input)

*Example Interactions 1 and 2 below are for **Non-CSE Students.** For **CSE Students**, the client command prompt has one more command UVF, and AED command returns extra active users' information including IP addresses and UDP port numbers. Please see Example Interaction 3 (P2P communication via UDP).*

Consider a scenario where *the central server*, *supersmartwatch*, *superwristband* are currently active. 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. *supersmartwatch* executes EDG command followed by a command that is not supported. And *superwristband* executes AED followed by OUT.

| supersmartwatch's Terminal | superwristband's Terminal | central server's Terminal |
|---|---|---|
| `> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT):` **`EDG 1 100`**<br><br>`> The edge device is generating 100 data samples…`<br><br>`> Data generation` | `> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT):` | `>` |

| | | |
|---|---|---|
| done, 100 data samples have been generated and stored in the file supersmartwatch-1.txt<br><br>> Enter one of the following commands | | |
| (EDG, UED, SCS, DTE, AED, OUT): **whatsyourname**<br>> Error. Invalid command!<br>> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): | | |
| | > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): **AED**<br><br>> supersmartwatch, active since 1 October 2022 13:31:13.<br><br>> Enter one of the following commands (EDG, UED, SCS, DTE, AED OUT): | > Edge device superwristband issued AED command<br><br>> Return messages: supersmartwatch, active since 1 October 2022 13:31:13. |
| > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): **OUT**<br><br>> Bye, supersmartwatch! | | |

| | | |
|---|---|---|
| | | > supersmartwatch exited the edge network |
| | > Enter one of the following commands (BCM, ATU, SRB, SRM, RDM, OUT): **OUT**<br><br>> Bye, superwristband! | > superwristband exited the edge network |

2. *supersmartwatch* and *superwristband* executes a series of valid commands and interact with the central server

| supersmartwatch’s Terminal | superwristband’s Terminal | central server’s Terminal |
|---|---|---|
| > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): **UED 1**<br><br>> Data file with ID of 1 has been uploaded to server<br><br>> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): **SCS 1 SUM** | > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): | > Edge device supersmartwatch issued UED command<br>> A data file is received from edge device supersmartwatch<br><br>> Return message:<br>The file with ID of 1 has been received, upload-log.txt file has been updated |

| | | |
|---|---|---|
| Computation (SUM) result on the file (ID:1) returned from the server is: 100 (100 is an example, you need to replace it with the real value)<br><br>> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): **DTE 1** | > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT): | > Edge device supersmartwatch requested a computation operation on the file with ID of 1<br><br>> Return message<br><br>SUM computation has been made on edge device supersmartwatch data file (ID:1), the result is 100 (100 is an example, you need to replace it with the real value)<br><br>> Edge device supersmartwatch issued DTE command, the file ID is 1<br><br>> Return message<br><br>The file with ID of 1 from edge device supersmartwatch has been deleted, deletion log file has been updated |
| | | |

3. P2P communication via UDP **CSE-students only**. Before *uav-camera* (assume the edge device *uav-camera* already joined the network in this example) uploads a video file lecture1.mp4 to *wei-smartphone, uav-camera* firstly issues the AED command to find out the IP address and UDP server port number of *wei-smartphone*.

| **uav-camera Terminal** | **wei-smartphone Terminal** | **server's Terminal** |
|---|---|---|
| `> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT, UVF):` **`AED`** `> wei-smartphone; 129.129.2.1; 8001; active since 1 October 2022 13:31:13.` | `> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT, UVF):` | `> The edge device uav-camera issued AED command` `> Return other active edge device list:` `wei-smartphone; 129.129.2.1; 8001; active since 1 October 2022 13:31:13. (assume that the IP address and UDP server port number of Obi-wan are 129.129.2.1 and 8001 respectively.)` `(note that the server is not aware of the P2P UDP communication between uav-camera and wei-smartphone)` |

| | | |
|---|---|---|
| > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT, UVF):<br>**UVF wei-smartphone lecture1.mp4**<br><br>> lecture1.mp4 has been uploaded | | |
| > Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT, UVF): | | |
| | > Received lecture1.mp4 from vav-camera<br><br>> Enter one of the following commands (EDG, UED, SCS, DTE, AED, OUT, UVF):<br>(For simplicity, we won't test the scenario that a file is received, when a client is typing/issuing a command.) | |

End of Assignment Specification