# CISC 335 – Computer Networks – Winter 2021
# Programming Assignment 2/2

Due April 9th, 2021, 11:59 pm

- This is an individual project – You can consult with peers, but codes have to be written individually. Plagiarism in any form will be treated as per "Queen's University Academic Integrity guidelines"
- Allowed programming languages are Java and Python.
- Use the **same language** for all codes across the entire assignment.
- The source code and project report are to be uploaded to onQ.

## Deliverables

1. Report, detailing:

   a. Brief description of the code steps, operation, and considerations (if there are any considerations).

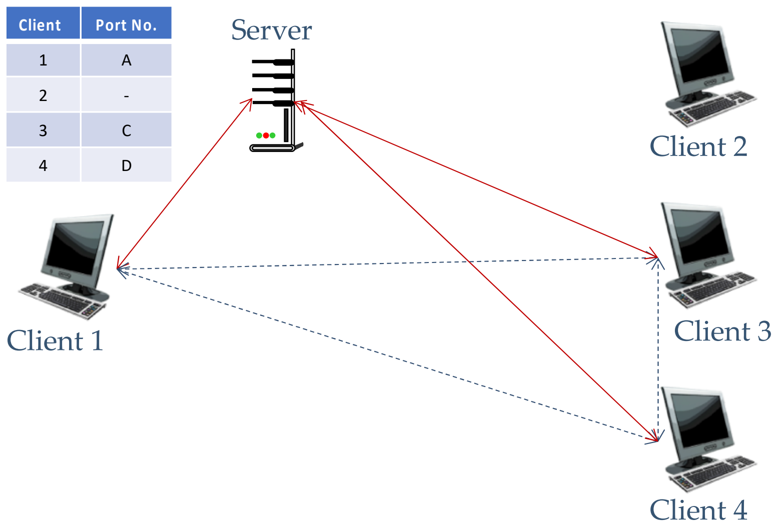
   b. Difficulties you faced and how you handled them (if any)

   c. Possible improvements: If you had more time, what would you add or do differently?

2. Source file (Use comments to document/describe your code)

## Description

In this assignment, you will learn to implement connection establishment and reliable data transfer (rdt) for applications running over UDP. As you recall from lectures, UDP is a best-effort transport layer protocol that does not establish connections before transferring packets nor tracks whether they have arrived properly to their destination. Yet, many applications still employ UDP for its benefits explained in class. If needed, these application designers must assume the responsibility of implementing the above two functions (i.e., connection establishment and rdt) in the application layer.



| Client | Port No. |
|---|---|
| 1 | A |
| 2 | - |
| 3 | C |
| 4 | D |

*Figure 1: Chatting Application Components*

You will be required to create a chatting application that establishes connections between clients and exchanges messages (between clients) reliably. The components of the system running this application are shown in Figure 1. As shown, it consists of 4 clients and 1 server.

When each of the clients is run, it opens a UDP socket and informs the server about the IP address and port number it can be reached at. The server maintains this information as shown in the table in Figure 1. When client X wants to send data to another client Y, it first consults with the sever to know the port number that client Y can be reached at, establishes an application-layer connection with it, then starts chatting. Once done talking, either party can initiate an application-layer termination of the chatting session. A client can always quit the application by notifying the server then closing the UDP socket. All above message exchanges must be acknowledged by the application layer to ensure rdt.

In this assignment and for the sake of simplicity, you will create the 4 clients and the server on the same machine. Since all elements will be on the same machine (same IP address), you will use the port number to distinguish between the processes. IP addresses do not need to be registered in the server nor used in the message exchanges.

## Requirements

The following is a list of the design requirements of the application. All steps (including starting server and clients, sending connection requests, accepting/rejecting connections, exchanged messages, connection terminations, etc.), must be made printable on the command line for normal application usage and code testing.

1. When started, the application server must create a UDP socket that listens on port 7070. The server must be able to handle multiple clients concurrently.
2. Upon starting, any client must establish a UDP socket and send its port number to the application server at “localhost:7070”. The list of clients that are active on the application and their reachable port numbers must be automatically displayed upon any change (any joining or leaving client) on the Server’s command line (same as the list shown in Figure 1).
3. For any client (e.g., Client 1) to establish a connection to another client (e.g., Client 2), it must follow the following procedure:
    a. User running Client 1 uses the command line to prompt the sending of a message to the server at “localhost:7070” asking about the UDP port number of Client 2.
    b. If Client 2 is active (the server has a UDP port number for it), the server returns the port number to Client 1. If not, the server returns a “Client 2 Inactive” message to Client 1.
    c. If Client 1 receives a port number for Client 2, it sends a “Connection Request” message, sets a 10-second timer, and waits for a response. The “Connection Request” message must have a numerical identifier for Client 1 (say no. 1). If no “Connection Accept” nor “Connection Reject” is received from Client 2 within 10 seconds, Client 1 resends the “Connection Request” message again and repeats the procedure until it gets one of these two messages.
    d. Once Client 2 receives the “Connection Request” message, it displays a message on the command line for the user to accept or reject the connection.
        i. If accepted. Client 2 responds with a “Connection Accept” message.
        ii. If rejected, Client 2 responds with a “Connection Reject” message.

    In either case, the message must have a numerical identifier for Client 2 (say no. 2).
4. When a connection is established between two clients, they can send short messages to one another through their command lines, which should be displayed on each other’s command lines. The messages sent by each client should include the client’s identifier. The messages must be sequenced incrementally and each of them must be acknowledged by the other party (using its sequence number) within 10 seconds, or else sent again. Duplicate messages (detected by duplicate sequence numbers) must be discarded and not displayed again.
5. Either clients can send a “Connection Termination” message to the other, which if acknowledged, must result in connection termination (again displayed in the command line).
6. To quit the application, a client must send a “Client Leave” message to the server. Once received the server erases the UDP port number from its table and sends an acknowledgement to the client. Only then, the client executes a socket close command.

**Bonus**: Clients must be able to handle multiple chats with multiple other clients. The client identifiers can be used to differentiate between messages coming from different clients.

If there is a detail not included in the requirements above, then it is totally up to you to decide on. All your design considerations must be detailed in the report. In the “possible improvements” section, you can detail all the things you might want to do if you had the time.

## Rubric

**Project** (No partial credit under each item, it is either fully met or receives no points)

| | | |
|---|---|---|
| 1. | Program can create server and clients | 1 point |

| | |
|---|---|
| 2. Successfully created clients send port numbers to server | 1 point |
| 3. Server automatically displays the list of active clients upon any change | 1 point |
| 4. Client receives “Client Inactive” message from server for inactive clients | 1 point |
| 5. Clients successfully establish connections to one another | 1 point |
| 6. Clients can reject connection requests from one another | 1 point |
| 7. Clients send messages with incremental sequence numbers | 1 point |
| 8. Received messages are acknowledged | 1 point |
| 9. Clients successfully terminate connections | 1 point |
| 10. Clients successfully leave the application | 1 point |
| 11. Bonus: Client can exchange messages with multiple clients | 2 points |

## Sample test cases

Make sure to test all items in the rubric.