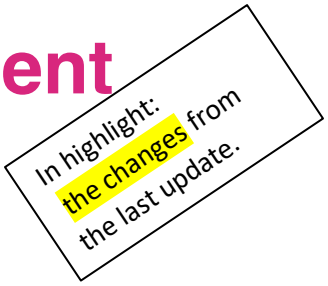
Networking Assignment



Reza Hassanpour

Ahmad Omar

Andrej Dobrkovic

Andrea Minuto

Assignment description and communication scenario

The networking assignment requires developing a client and a server application that will establish communication over THE INTERNET. This communication involves a scenario where the client requests a specific file called “test.txt” to be **downloaded** from the server. Your task is to satisfy communication requirements and comply with our custom protocol described in this document, to establish such a connection and exchange messages with it to receive all the packets of the file, both in the client and server. Our custom protocol is based on IP/UDP protocols.

After exchanging “hello” messages, we assume the *client* sends a request with the filename to the *download-server.* The server will locate the file, prepare it for transmission, and then starts sending it using the **UDP** protocol that implements *a sliding window* as in TCP. The *sliding window* implementation is needed because the **UDP** protocol is stateless and does not provide a guarantee that all the packets will be delivered messages could arrive in a different **order** than the server initially sent them. Additionally, messages sent by the server, or the client **may get lost**. Your server implementation has the responsibility to implement the GO-BACK-N sliding window. To grant that in case of an error, the server returns to the Nth packet and sends the missed data again starting from that point on.

Once all messages are received your client program will print the data of the message in an ordered way.

Note: we will also provide an implemented version of the server to compare and test. It will be distributed via docker image and ready to process requests and send data files.

The download procedure

The download procedure starts with the client sending a HELLO message which includes its name, and the name of the server. The server establishes a communication channel, assigns an ID to it, and replies with a HELLO_REPLY message. The HELLO_REPLY includes the connection ID. The client and the server will use this ID in all subsequent messages.

Next, the client sends a RequestMSG to download a file. The request message should include the names of the server and client, connection ID, and requested file name.

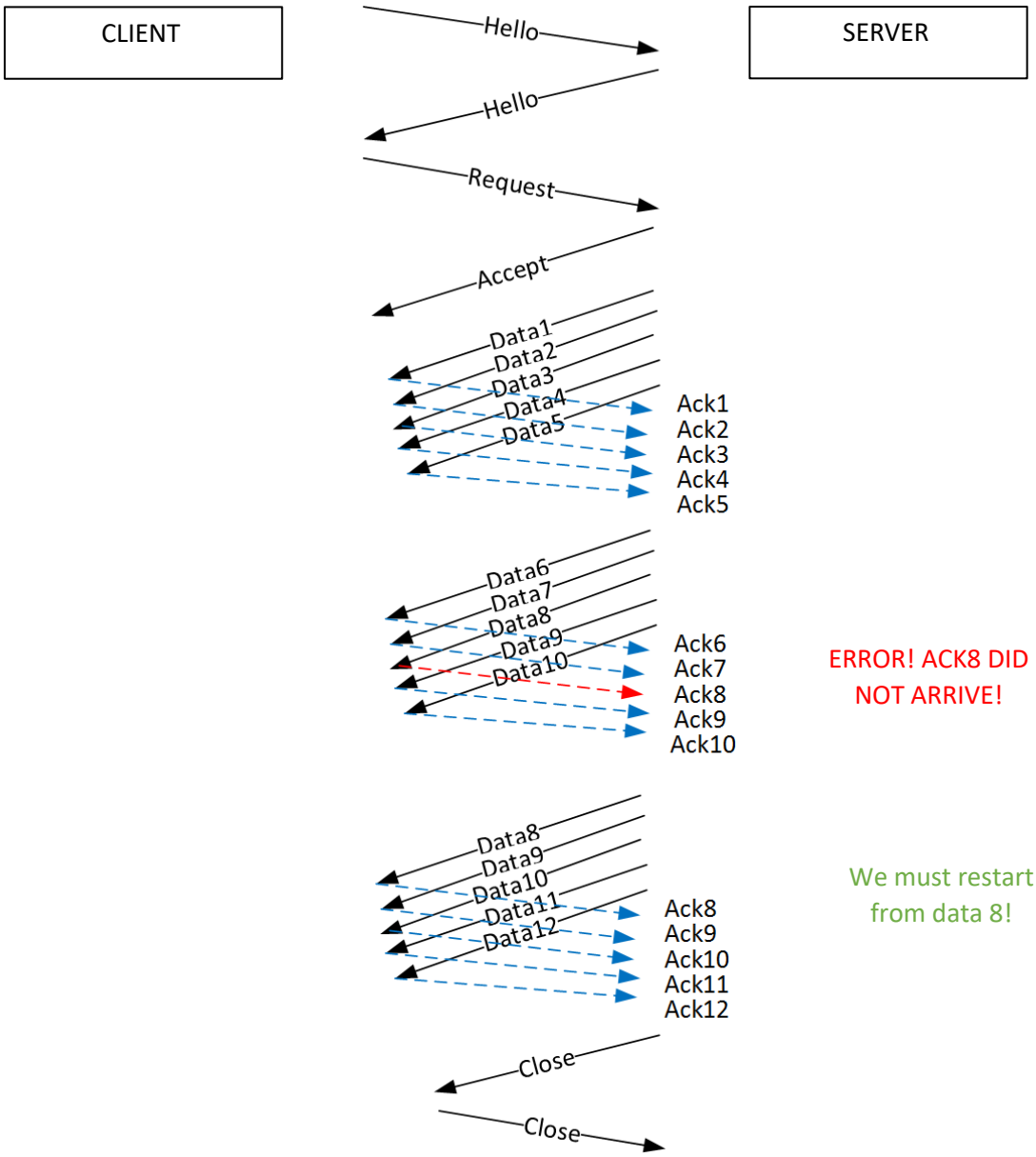
The server will confirm the request and starts sending the file data to the client. The data should be sent in DataMSG format (given in the codebase). Each data message should include as

many bytes as `SEGMENT_SIZE` except for the last message. The data messages must include a sequence number, length (number of data bytes), and an attribute to indicate if this message is the last or not. The client should confirm each message with an AckMSG.

The server must send as many messages as `WINDOW_SIZE` before waiting for acknowledgements. If a message is lost, the client will not confirm it. If a message is not confirmed (data message or the Ack is lost), the server will repeat transmitting the messages from the one which was not confirmed (Slide Window protocol with go-back-n).

At the end of the transmission, the server sends CloseMSG to close the communication channel. The message should be confirmed by the client.

The following schematic diagram shows the order and the type of messages used during the communication.



Any ack or data could be lost during the transition.

Error handling

Both the client and the server should validate the input messages before reacting to them. The error checking should include the following (at least)

1- Sender's name
2- Receiver's name
3- Connection ID

4- Expected message type at each state
5- Expected sequence numbers (in case of data messages)

The client or the server should respond appropriately according to the type of error occurring (ignoring the message, closing the connection, etc.). It is a good policy to check for exceptions thrown by called methods and to handle those appropriately.

Logging:

Print or log out the errors so the user can check what happened. It is a good policy to print or log important messages to give the user an idea of what is happening in the application.

The custom protocol

TCP is a connection-oriented protocol, UDP does not grant the same properties. In our custom protocol, you must implement these properties (granted from TCP) by using a UDP socket in C#.

The client will communicate with the server in port 5004.

The server will communicate with the client in port 5010.

The type of all the messages is provided in the codebase of the assignment. Each of them is referring to a missing property in UDP that you need to handle (implement).

You will have to implement and handle a Sliding Window protocol in UDP.
We provide a set of parameters to implement the Sliding Window.

```
BUFFER_SIZE = 1000,
WINDOW_SIZE = 5,
SEGMENT_SIZE = 10
```

BUFFER_SIZE is the number of bytes you can have at any time in your memory.
WINDOW_SIZE is the total number of data fragments you can wait for confirmation.

SEGMENT_SIZE is the size of each data fragment.

Type of messages:
To communicate with the download-server program, you need to comply with the custom protocol requirements that the server supports. Your client application must send and receive different types of messages to be able to download the file and keep track of the state.
For example, a HelloMSG message must be sent to the server to exchange messages with the server. After creating a socket, you should be able to send and receive UPD messages.
The *sendTo()* method of a socket expects the message to be a byte-array. Therefore, the *HelloMSG* needs to be encoded and serialized properly before sending.
If a client receives a message it. The *ReceiveFrom()* method stores a byte-array in the data buffer. The received *HelloMSG* need to be deserialized and decoded accordingly before further processing.

The signature of both methods:

public int SendTo (byte[] buffer, int size, System.Net.Sockets.SocketFlags socketFlags, System.Net.EndPoint remoteEP);

public int ReceiveFrom (byte[] buffer, int size, System.Net.Sockets.SocketFlags socketFlags, ref System.Net.EndPoint remoteEP);

```
public class ConSettings
  {
     public Messages Type { get; set; }
     public string From { get; set; }
     public string To { get; set; }
     public int ConID { get; set; }
     public int Sequence { get; set; }

  }

public class AckMSG //in the diagram Ack followed by order number.
  {
     public Messages Type { get; set; }
     public string From { get; set; }
     public string To { get; set; }
     public int ConID { get; set; }
     public int Sequence { get; set; }

  }

public class CloseMSG
  {
     public Messages Type { get; set; }
     public string From { get; set; }
     public string To { get; set; }
     public int ConID { get; set; }

  }


public class HelloMSG
  {
     public Messages Type { get; set; }
     public string From { get; set; }
     public string To { get; set; }
     public int ConID { get; set; }

  }
```

```
public class Enums
    {
        public enum ErrorType
        {
            NOERROR = 0,
            BADREQUEST = 1,
            CONNECTION_ERROR = 2
        }
        public enum Params : int
        {
            BUFFER_SIZE = 1000,
            WINDOW_SIZE = 5,
            SEGMENT_SIZE = 10
        }
        public enum Messages : int
        {
            HELLO = 1,
            HELLO_REPLY = 2,
            REQUEST = 3,
            REPLY = 4,
            DATA = 5,
            ACK = 6,
            CLOSE_REQUEST = 7,
            CLOSE_CONFIRM = 8
        }
    }


public class RequestMSG
    {
        public Messages Type { get; set; }
        public string From { get; set; }
        public string To { get; set; }
        public string FileName { get; set; }
        public int ConID { get; set; }
        public ErrorType Status { get; set; }
    }


public class DataMSG
    {
        public Messages Type { get; set; }
        public string From { get; set; }
        public string To { get; set; }
        public int ConID { get; set; }
        public int Size { get; set; }
        public bool More { get; set; }
        public int Sequence { get; set; }
```

```
    public byte[] Data { get; set; }
}
```

Grading requirements:
Any missing, imprecise, partial, or incomplete requirement will be graded as a failure.
Other requirements details can be extracted from the description.

The client and server need to implement the sliding window with the GO-BACK-N functionality. Packets can arrive in any order and may be lost. Have a look here for more information: https://www.baeldung.com/cs/networking-go-back-n-protocol

- The program should be written in .Net 6.*x*.
- Only allowed libraries are those that are NOT involved in the networking and handling of messages (ex: no “UdpClient”, nor “TcpListener”, or any other that avoid port binding and automated message handling). There is NO NEED FOR THREADS, the application will be synchronous (NOT CONCURRENT). No need for an external config unless already in the template.
- Other built-in libraries such as System and System.IO are allowed if it does not conflict with the learning goals.
- NO EXTERNAL LIBRARIES THAT NEED INSTALLATION OR EXTRA FILES.
- You need to use our project **template** to implement your application. Do not change any existing filename in the provided template AND FILL IT APPROPRIATELY.
- When running the application there must be **no errors/warnings**. Apply proper exception handling where needed (failure to do so will result in a FAIL).
- All the communication between the programs must be carried out with the proper socket kind with appropriate settings, wrong settings will be regarded as a syntax error in the code (ex: wrong kind of socket TCP, wrong port communication settings, etc.).
- Your program will be tested using **different sequences of possible errors in the protocols** (anything that is a valid format but contains different values) therefore, make sure that the data you are reading is from the same location as one of the programs running.
- The path/location should be handled independently from the running operating system (Mac/Windows/Linux).
- The solution should be the **original** work of the submitting group (an automatic check for plagiarism will be carried out).
- The submitting group can be composed of up to 2 students with the same teacher. Retakers can pick any class/companion.

- The solution should include the **names** of the participants (and students' numbers) as a comment in the template files and/or where indicated.

The delivery of the assignment

The delivery is expected before **06/11/2022 11:00 pm**.

Link for the submission: https://forms.office.com/r/cCatvVApRE

Remember to rename the zipped solution in the ".dot" file. Keep student numbers in the file name as indicated in the submission, and do not forget to clean up your solution BEFORE zipping it.

The testing ground will be provided via Docker images. To load the image, you can do the following:

- Install docker on your machine
- Run this command in your terminal (command line):
  docker load < file_download_server_image.tar (or name of the file image).
- *Check if the image is loaded* and check if the name matched the name in the next command:
  docker image ls
  if it does not match, adapt the *file name in the next command*!
- To run the container: ***docker run -i -t -p 5004:5004/udp file_download_server-image:latest***

Remarks:

The provided docker server is for testing purposes. Not all exceptions and possible errors are handled explicitly.

In the code, there are comments/todos to help your development.

If you find errors or unclear parts let us know. This includes description, code and/or delivery.

Ask per time. Asking later will not be useful. Feel free to give feedback if you think there is any imperfection or doubt (please do so in the appropriate manner and channel).