Middle East Technical University

Department of Computer Engineering

CENG 435

Data Communications and Networking

Fall 2020–2021

Socket Programming Take Home Exam 2

Due date: 17.12.2020 - Thursday, 23:59

1 Introduction

In this assignment, you are going to be familiar with the *socket programming* concept. Since this is a programming homework, please read the whole document carefully before starting on the implementation.

A *network socket* is a software structure within a node of a computer network that acts as an endpoint for sending and receiving data across the network. The structure and properties of a socket are defined in an application programming interface (API) for the networking architecture. Sockets are created only during the lifetime of a process of an application running in the node [1].

Socket programming is a way of connecting two nodes on a network to communicate with each other. Server node listens on a particular port at an IP address, while the other node reaches out using a socket to form a connection [2].

In this activity, you are expected to develop an *UDP* and *TCP* socket application to transfer two files divided into chunks. Specifications about the socket application are explained in detail below. While implementing this homework, you can use any of the languages given in the course syllabus, which are *C*, *C++*, *Python3*, *Java* and *Bash*. However, using *Python3* is **strongly advised**.

First, while developing and testing your application, you will need two hosts, which are a "server" and a "client". You can use your own machines, virtual machines, etc. for these hosts, however, it is advised to use `inek` machines in our department labs. Since you cannot physically go to the labs, you can instead connect to two `inek` machines using `SSH`. In case you forgot the `ssh` commands, you can find them below. Please check the `ineks` status from here before connecting to an `inek` machine.

At your local machine

```
$ ssh e1234567@external.ceng.metu.edu.tr -p 8085
```

or

```
$ ssh e1234567@divan.ceng.metu.edu.tr -p 8085
```

then

```
$ ssh inekXX
```

In this homework, you must show what you have done in your code with *comments*. Add comments to explain your code and be sure that your implementation is apparent after reading them. Moreover, write a `README` file. In this `README` file;

1. Explain how to run and test your code (You should include a `Makefile` if necessary.).
2. Explain the main steps of how you approached this study, explain your work (in 5–6 sentences). You are free to include what you might deem necessary to understand your solution.
    - What have you done in this study?
    - Which part did you start with?
    - How did you plan what to do?
    - Which problems did you face?
    - What did you learn after the study?
    - How many days did it take?
3. Please explain shortly which RDT protocol you have used (your own protocol or a known protocol).

2 Implementation

You are going to implement *UDP* and *TCP* socket applications to transfer two files. You will transfer one file for each communication. However, UDP is an unreliable transfer protocol, therefore, you have to implement an RDT (Reliable Data Transfer) protocol on top of UDP to make it reliable. As you know, UDP, unlike TCP, is a connectionless protocol that does not inherently provide reliability against the underlying unreliable channel. In this homework, you are asked to solve two of the following problems of the unreliable channels with your implementation;

- Packet Corruption
- Packet Delaying / Reordering

You can implement your own RDT protocol or use some common protocols' solutions to these problems. There are some interactive animations that show some RDT protocols in these links: [3], [4], [5].

While working on inek machines, if two people were to use the same inek machine and choose the same port, it would be a problem. In order to avoid this problem, eight port numbers (Two for UDP, two for TCP, and the other four ports are in case you need to change the port number.) are assigned to your student ID and shared with you on our ODTUClass page as **"the2_studentIDs_ports.pdf"** (If you cannot find your student ID in the file, please contact with assistants of the course.). If you get an error about "your port is already in use", you may not have exited your program normally and this port might be seen as it is still using for a while (a couple of minutes), in this case please wait for a while than try again or try with the other ports assigned to you. Please be careful, while creating sockets, you should determine two ports for each communication, one port is for the "server" and the other one is for the "client". Generally, in socket programming, people sometimes do not care about the sender port, however, in this homework, you must determine the sender port and choose every port from the ports assigned to you in order to avoid congesting the same port by multiple people.

Prepare two files ("server" and "client") and one README file. You can have more than two files as long as you explain them in the README file, however, you must have "client" and "server" files (such as client.py and server.py) because we will run them while evaluating your homework. The direction of file transfers is from "client" to the "server". The "client" program takes

- the "server"s IP address,
- the "server"s UDP listen port,
- the "server"s TCP listen port,
- the "client"s sender port for UDP communication,
- the "client"s sender port for TCP communication

as inputs through command line arguments and sends the packets from its sender port to the "server" which has the given IP address and port for the UDP and TCP communication. The "server" program takes the port which listen for incoming packets as for UDP and TCP inputs through command line arguments. For example, if your "server"s IP address is 10.10.1.1, its listen ports are 10000 for UDP, 11000 for TCP, "client"s sender ports are 12000 for UDP, 13000 for TCP and you use "Python3", we will run your code as follows;

For "server",

```
$ python3 server.py 10000 11000
```

Then, for "client",

```
$ python3 client.py 10.10.1.1 10000 11000 12000 13000
```

You can test your TCP and UDP communication with two inek machines by running your code. If you use the channels between the ineks (for our intents and purposes), you may not observe packet corruption, delaying, or reordering as you expect to test your RDT protocol. Therefore, we want you to test your RDT implementation in a simulation environment against these problems for UDP communication, in other words, you need an unreliable channel simulation. Hence, a "simulator" executable shared with you in ODTUClass. Details about how to use explained below;

- This "simulator" creates an unreliable channel with packet corruption and packet delaying/reordering between your "client" and "server" only for UDP communications. For TCP communication it only forwards the packets from "client" to "server".
- Your "client" should send your packets to the "simulator" instead of sending them directly to the "server". Your packets will be sent to the "server" through the "simulator".
- If you send a reply packet from "server" to "client", simulator also supports that. When you reply the incoming packets normally to where it comes from in the "server", it automatically sends the packet to "client" through the "simulator".
- The test environment works like this, "server" $\Longleftrightarrow$ "simulator" $\Longleftrightarrow$ "client"
- After you send your packets from "client" to "server" through "simulator", if any packet does not transfer, "simulator" program close itself after approximately 10 seconds.
- How to run:

```
$ ./simulator <client_address> <server_address> <the_port_that_UDP_packets_send_to_
from_client> <the_port_that_TCP_packets_send_to_from_client>
<udp_client_sender_port> <udp_server_listener_port> <tcp_server_listener_port>
<free_port_assigned_to_you> <free_port_assigned_to_you> <packet_corruption_
percentage(0-100)> <delaying_reordering_percentage(0-100)> <delay_time(second)>
```

- Because you will send your packets to simulator instead of "server", <the_port_that_UDP_packets_send_to_from_client> <the_port_that_TCP_packets_send_to_from_client> ports must be different than the ports "server" listens.
- First, run the "simulator" and then run "server" and "client".
- Example: In a normal TCP and UDP communication, let's say
  - "client(IP address: 10.0.0.1)" sends its packets from its 12001. port to "server(IP address: 10.0.0.2)"s 12003. port for the UDP communication and "server" listens 12003. port.

- For the TCP communication, “client” sends its packet from its 12002. port to “server”s 12004. port and “server” listens 12004. port.
- However, while using “simulator”, you should change the ports that “client” sent to. Then, let’s say “simulator” listens in 12005. port for UDP, 12006. port for TCP and you have chosen two port numbers from the ports that was assigned to you (12007 and 12008).
- Moreover, you have implemented and want to test with 10% packet corruption possibility, 5% delaying/reordering possibility and if delay happens, let delay time be 3 seconds.
- The run command will be

```
$ ./simulator 10.0.0.1 10.0.0.2 12005 12006 12001 12003 12004 12007 12008 10 5 3
```

- “simulator” will print the information about the packets, such as if a packet delays or if a packet corrupts.
- You can stop the simulator with double “Ctrl + C”.

The files you are asked to transfer are on our ODTUClass page (“transfer_file_TCP.txt” and “transfer_file_UDP .txt”) and their size is exactly 1 MB. One file must be transferred with TCP (“transfer_file_TCP.txt”) while other file must be transferred with UDP (“transfer_file_UDP.txt”). Only the “client” program is allowed the read these files from the disk. The “client” program must divide both of the files into chunks of up to 1000 bytes before sending. This process is called fragmentation and the 1000 bytes has chosen considering the MSS (Maximum Segment Size) [6].

After the fragmentation step, packets of the “transfer_file_TCP.txt” file must be transferred with **TCP** and “transfer_file_UDP.txt” must be transferred with **UDP**. You can transfer TCP and UDP packets one after the other or concurrently. In other words, your program can transfer UDP packets firstly and after it was finished, it can transfer TCP packets secondly. You can also transfer the packets concurrently with using threads or processes, it is your own choice, however, “transfer_file_UDP.txt” file must be sent with UDP and the other “transfer_file_TCP.txt” file must be sent with TCP. As you can understand, after all of the packets for both of the files are transferred from the “client” to the “server”, fragmented packets must be reassembled to create the transferred files (with the same names “transfer_file_UDP.txt” and “transfer_file_TCP.txt”) in the “server”s location.

While transferring your packets, if you want to set a timeout to check the packets are transmitted properly, it is advised to set “1 second” timeout. It is only an advice, you can choose the timeout freely, however, inform us by explaining in the README file about how many seconds you have chosen for the timeout in your code.

Due to the underlying unreliable channel, a proper checksum control mechanism is required to check if a packet has transferred correctly. We will evaluate your code using different files to send so do not make assumptions about the files you are transferring.

Your “server” program must not listen to a port forever. You should construct a mechanism that the “server” can realize when the whole files are transferred. Then, “server” stops listening, reassembles, and saves the files to the same location with the “server” file.

2.1 Expected Outputs

Firstly, calculate the average packet transmission times for UDP and TCP transmissions separately. In order to do that, in the “client”, you should start the time before sending a packet to “server” and stop the time after the packet is received from “server”. For UDP and TCP packets, you should calculate this time difference for each packet separately, then take the average of them and at the end print (in the “server”) as in the format below right before the “server” exits.

Note: If you resend a packet, consider only the last packet and ignore the previously sent packets while calculating average transmission times.

For TCP communication:

```
TCP Packets Average Transmission Time:  ...  ms
```

For UDP communication,

```
UDP Packets Average Transmission Time: ... ms
```

After calculating the average transmission time, you should also calculate the total transmission time for the UDP and TCP transmissions. In order to do that, in the "client", you should start the time before sending the first packet to "server" and stop the time after the last packet is received from "server". In other words, calculate the time between the start and end of TCP communication and do the same for UDP communication. Then print (in the "server") as in the format below right before the "server" exits.

For TCP communication:

```
TCP Communication Total Transmission Time:  ...  ms
```

For UDP communication,

```
UDP Communication Total Transmission Time: ... ms
```

Also count and print the packets that you have sent again, after the UDP transmission ends. In your RDT protocol you may send your packets if your data was corrupted or if there is a delay. Therefore, in these cases, count how many packets you have sent again and at the end print (in the "client") as in the format below right before the "client" exits.

```
UDP Transmission Re-transferred Packets:  ...
```

Notice that, your implementations must be reusable for any topology or any network configuration based on bandwidth, delay, and packet size. Show these details in your codes with your comments.

3 Other Specifications

- **This is an individual assignment. Using any piece of code, discussion, explanation, etc. that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homework, or the Internet. The violators will be punished according to the department regulations.**
- **Late Submission: Late submission is allowed as stated in the course syllabus.**
- Follow the announcements on our ODTUClass page for any updates and clarifications. Please use the discussion forum on ODTUClass first for your questions instead of e-mailing if the question does not contain code or a solution. Your question might have already been answered or the answer you get might help your peers.
- Grading:
  - Proper "README": 10 Points
  - Fragmenting Files into Chunks: 5 Points
  - TCP Communication: 30 Points
    - Creating TCP communication: 5 Points
    - Successful File Transfer: 15 Points
    - Calculating and Printing "TCP Packets Average Transmission Time": 5 Points
    - Calculating and Printing "TCP Communication Total Transmission Time": 5 Points
  - UDP Communication over RDT: 55 Points

* Creating UDP communication: 5 Points
* Successful File Transfer: 35 Points
* Calculating and Printing "UDP Packets Average Transmission Time": 5 Points
* Calculating and Printing "UDP Communication Total Transmission Time": 5 Points
* Calculating and Printing "UDP Transmission Re-transferred Packets": 5 Points

4 Submission

- Include the source code of the server and the client program, along with any other additional source files you may have written and the README file, prepared according to the specification above.
- Compress these files in **.zip** format called **e<your_student_id(7 digit)>.zip** (Ex: **e1234567.zip**) and submit to ODTUClass.