COMP6461 – Fall 2019
Data Communications & Computer Networks
Lab Assignment # 3
Due Date: Sunday, December 1 by 11:59PM

Introduction

In this assignment, you will re-implement the HTTP client and the HTTP remote file manager of Assignments #1 and #2 respectively using UDP protocol. In the previous assignments, you leverage TCP protocol for implementation to guarantee packet transmission over unreliable network links. Because you are going to use UDP protocol that does not guarantee the transfer, you need to insure reliability by implementing a specific instance of the Automatic-Repeat-Request (ARQ) protocol called: Selective Repeat ARQ / Selective Reject ARQ. Before starting on this Lab, we encourage you to review the programming samples and the associated course materials.

Outline

The following is a summary of the main tasks of the Assignment:

1. Study the simulation network infrastructure and message structure.
2. Replace TCP by UDP in your HTTP library in both the client and server.
3. Implement `Selective-Repeat` flow control to make a reliable transport.
4. (Optional) Support multiple clients at the server.

Objective

The primary objective of this assignment is to use the unreliable UDP instead of TCP transport protocol in the implementation of your both HTTP Client and HTTP file manager. To this end, you should manually ensure the reliability of the transport on top of UDP protocol by implementing Selective-Repeat ARQ technique.

Selective Repeat is part of the automatic repeat-request (ARQ). With selective repeat, the sender sends a number of frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-Back-N ARQ. The receiver may selectively reject a single frame, which may be retransmitted alone; this contrasts with other forms of ARQ, which must send every frame from that point again. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.[1]

It is important to emphasize that both the HTTP Client and the file manager should keep the same specifications as given in the previous assignments. The apps should be updated to be executed in the simulation environment as will be described in the following sections.

In addition to adjusting your previous apps (HTTP client and file manager server) to work with UDP and the provided simulation environment, you are required to implement the core functionalities of TCP protocol such as Mimicking the TCP three-way handshaking.

Important Note:

You must use only the bare-minimum socket APIs provided by the chosen programming language. Therefore, you must not leverage any library that could abstract the socket programming.

Important Considerations

In the following, we present a set of notes that you should take into account in the development of your HTTP client and HTTP file manager using UDP protocol.

Endianness

Endianness is the order of bytes in a multiple-bytes representation such as word, integer, float. When sending multi-bytes data type via a network, we must convert them from the host byte order to the network byte order (e.g., Big-Endian). And when receiving multi-bytes data types from the network, we must convert them from the network byte order to the host order. Most of the languages provide APIs for this conversion.

UDP methods

You need to read UDP Socket API for your selected language. Here are some quick notes.

1. There is no *accept* in UDP, it is connectionless.

2. The **recvfrom** method returns not only the data but also the **host:port** of the sender. You should use the **received_addr** to reply.

3. The **sendto** method requires two arguments: data to be sent, and the **host:port** of the receiver.

Addressing

An address of a network application includes two parts: the IP address of the host and the port number. To simplify the complexity of the assignment, we will use IPv4 addresses only. An IPv4 address is encoded using 4 bytes. For example, the address 172.16.254.1 is represented by an array of 4 bytes 172, 16, 254 and 1. Port number is represented by 2 bytes in Big-Endian order.

Development Environment

Local networks are more reliable than remote networks in the Internet. We could use UDP protocol in local networks with a negligible error if not zero. For this reason, you are requested to develop your apps in the simulation environment that we developed. The latter is a complete communication system that allows you to experiment the unreliability factor of the Internet such as packet delay and packet drop. Specifically, we provide a router app that simulates the previous factors. To make your app communicate with the router, you need to use its own custom packet structure. We urge you to consult the provided example to have a deeper understanding of the programming model in the simulation environments. The simulation environment is described the in the following.

Simulated network infrastructure

You are provided an unreliable UDP channel, in which packets can be dropped or delayed randomly. Instead of sending/receiving network packets directly between a client and a server, both of them have to send and receive packets via the router. The purpose of this infrastructure is to simulate and control the unreliable characteristic of UDP protocol. In the figure, both application A and B send and receive packets to/from the router.

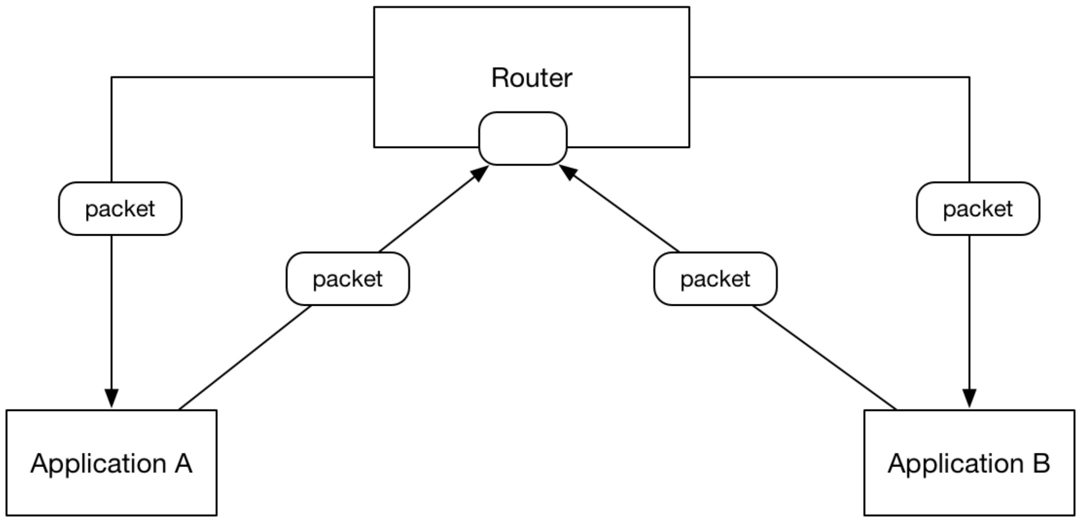


Figure 1: The Communication Model

Router

We provide you the executable of the router on Windows, MacOS, and Linux. This is the manpage of the router application.

```
Router is a logical router that dispatches UDP packets between applications.
It receives UDP packets, then dispatches them to the associated destination
```

```
of the packets. During the routing, the value of a peer address will be
modified from 'destination' to 'source'.



Usage:
    router --port int --drop-rate float --max-delay duration --seed int

    --port int-number
        port number that the router is listening for the incoming packets.
        default value is 3000.

    --drop-rate float-number
        drop rate is the probability of packets will be dropped during on the way.
        use 0 to disable the drop feature.

    --max-delay duration (eg. 5ms, 4s, or 1m)
        max delay the maximum duration that any packet can be delayed.
        any packet will be subject to a delay duration between 0 and this value.
        the duration is in format 5s, 10ms. Uses 0 to deliver packets immediately.

    --seed int
        seed is used to initialize the random generator.
        if the same seed is provided, the random behaviors are expected to repeat.

Example:
    router --port=3000 --drop-rate=0.2 --max-delay=10ms --seed=1
```

Message structure

To interact properly with the router, all the packets (e.g. UDP message) should follow the following structure; otherwise, the router may reject or fail to dispatch it to a proper destination.

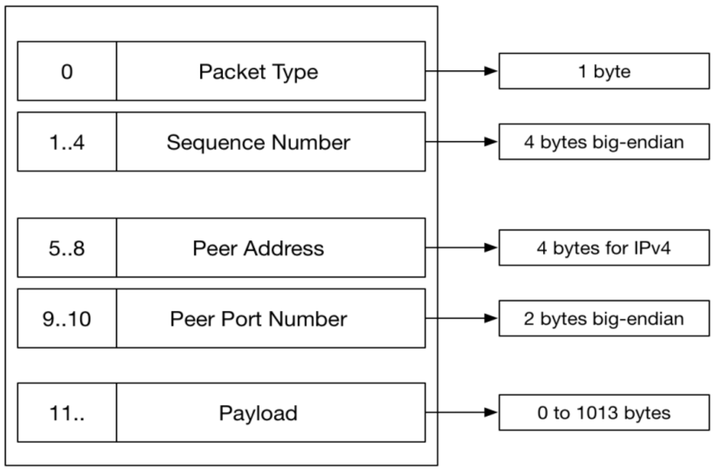


Figure 2: Packet Structure

Packet Type: 1 byte

This value indicates the type of the packet, which can be Data or ACK or SYN or SYN-ACK or NAK. You are free to define and manage this value.

Sequence number: 4 bytes - unsigned and big-endian

The sequence number has two meanings which depend on the value of packet type. If the packet type is SYN, this value is the initial sequence number during the handshaking. Otherwise, the sequence number is the accumulated sequence number of the first byte of the payload (like TCP). You can use the sequence number as the packet number (e.g., 1, 2, 3) as it can be simpler.

Peer Address and Peer Port

The peer address of a packet also has two meanings. When you send a packet, the peer address is the address of the destination that you want to send. Thus, you have to set the peer address and port of the packet by the values of the receiver. On the other hands, when you receive a packet, the peer address is the address of the original sender. The router executes this translation. When dispatching the packet, the router modifies the peer address from Destination to Source.

Payload

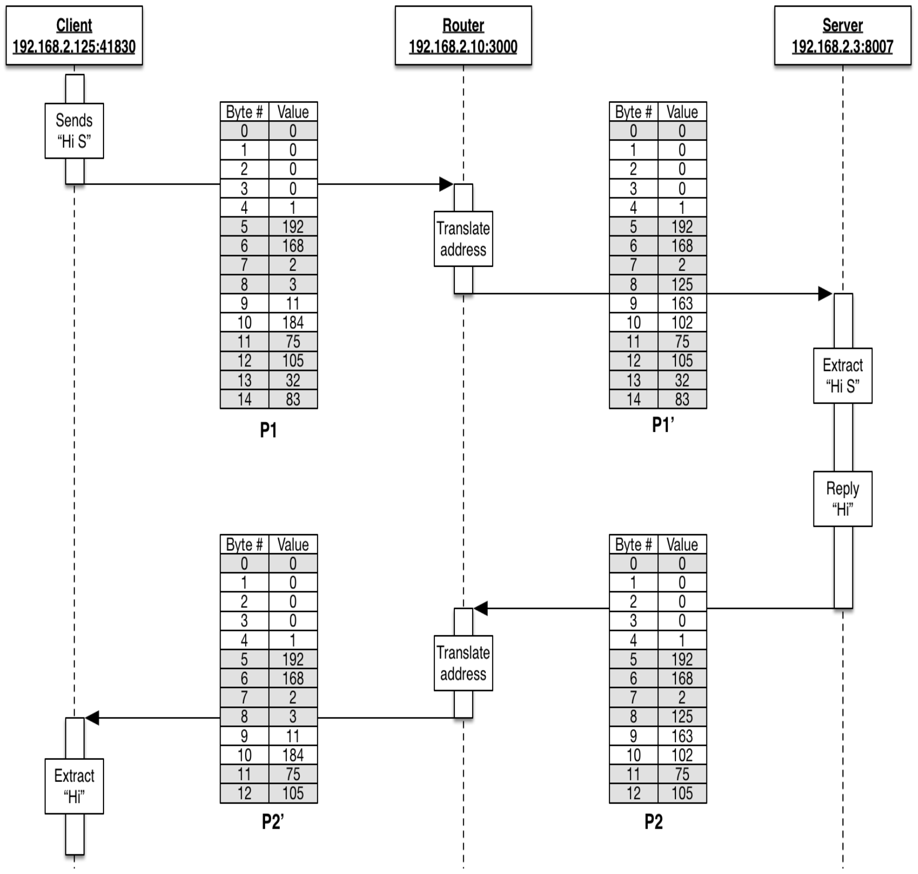
You are allowed to send a maximum of 1013 bytes' payload for each packet. In other words, each packet must be in range 11 (when payload is 0) to 1024 bytes.

Flow Example

In this section, we describe step by step a simple interaction between a client and a server.

Preconditions

1. Router is running at port 3000 at the host 192.168.2.10
2. Server is running at port 8007 at the host 192.168.2.3
3. Client is running at port 41830 (uses an ephemeral port) at the host 192.168.2.125



P1

Byte #	Value
0	0
1	0
2	0
3	0
4	1
5	192
6	168
7	2
8	3
9	11
10	184
11	75
12	105
13	32
14	83

P1'

Byte #	Value
0	0
1	0
2	0
3	0
4	1
5	192
6	168
7	2
8	125
9	163
10	102
11	75
12	105
13	32
14	83

P2'

Byte #	Value
0	0
1	0
2	0
3	0
4	1
5	192
6	168
7	2
8	3
9	11
10	184
11	75
12	105

P2

Byte #	Value
0	0
1	0
2	0
3	0
4	1
5	192
6	168
7	2
8	125
9	163
10	102
11	75
12	105

Figure 3: Flow Example

Step 1: Client wants to send "Hi S" to the server

In order to send a payload to the server, the client has to construct a packet P1 which is 15 bytes. These bytes are broken down as follows.

- **Byte 0:** - Packet Type - in this example we use 0, which means this is a data packet.
- **Byte 1-4:** - Sequence Number - assume this packet has the sequence number 1. Therefore, it is encoded by 4 bytes 0, 0, 0, and 1 in BigEndian order.
- **Byte 5-8** - Peer Address - the server (eg. destination) is running at host "192.168.2.3", thus the client has to put this value to the packet as [192, 168, 2, 3].
- **Byte 9-10:** - Peer Port Number - the server is listening at port 8007, thus the client has to put this value in BigEndian order to the packet as [11, 184].
- **Byte 11-14:** - Payload - the client wants to send "Hi S" which is encoded as [105, 32, 83].

Once the packet p1 is constructed, client calls **`sendto(p1, router_addr)`** to send the packet to the router. The client is provided with the router address in advanced.

Step 2: Router receives packet p1 and routes to the server

After the client invokes **`sendto(p1, router_addr)`**, the router can receive the packet **p1** via **`recvfrom`** function. The **`recvfrom`** function returns not only the data but also the `host:port` of the sender. In particular, the router receives packet **p1** and the sender's address is **"192.168.2.125:41830"**. The router translates the peer address for packet **p1**. It creates a new packet **p1'** which is almost similar to the packet **p1** except that the values of the peer address and port. The new values of peer **address:port** are the **address:port** of the client. The router then calls **`sendto(p1', packet_dest)`** with the **`packet_dest`** is the address of the server which is extracted from the **peer_address** of the packet **p1**.

Step 3: Server receives packet p1' from the router

Once the router executed **`sendto(p1', packet_dest)`**, the server may receive packet **p1'** with the carrier address (eg. the router address). It exacts the message's payload, and got "Hi S".

Step 4: Server wants to reply "Hi" to the client

The server in turn wants to reply a message "Hi" to the client. It has to construct a packet called **p2**. The packet **p2** is shorter than **p1** and **p1'** as its payload is only 2 bytes while **p1** and **p1'** are 4 bytes. The peer address of the packet **p2** is the address of the client (extracted from the peer address of packet **p1'**). To reply to the packet, the server calls **`sendto(p2, router_addr)`**. The router address is the carrier address in Step 3.

Step 5: Router receives packet p2 and routes to the client

Step 5 is similar to Step 2. When the router receives the packet **p2** from the server, it modifies the peer address from **"192.168.2.125:41830"** to **" 192.168.2.125:8007"** and forwards the modified packet to the client.

Step 6: Client receives packet p2' from the router

Step 6 is similar to Step 3.

Optional Tasks (Bonus Marks)

If you have successfully completed the material above, congratulations; as you now understand the implementation of reliable protocols over unreliable networks. For the rest of this lab exercise, we have included the following optional tasks. These optional tasks will help you gain a deeper understanding of the material, and if you can do so, we encourage you to complete them as well. Bonus marks will be given for that.

Multi-Requests Support

In order to support multiple clients at the server, you are required to mimic the `accept` behavior of the TCP. In particular, for each new handshaking request, you create a new UDP Socket for the requested client exclusively. Another approach to support multiple clients is to use the client address to as its identifier.

Submission and Grading

Important Note: You can this assignment individually or in a group of **at most 2 members** (i.e. you and another student). No extra marks or any special considerations will be given for working individually.

Deliverable

1) Create one zip file, containing the necessary source-code files (.java, .c, etc.)

You must name your file using the following convention:

If the work is done by 1 student: Your file should be called A#_studentID, where # is the number of the assignment. studentID is your student ID number.

If the work is done by 2 students: The zip file should be called A#_studentID1_studentID2, where # is the number of the assignment. studentID1 and studentID2 are the student ID numbers of each of the group members.

2) Assignments must be submitted in the right folder of the assignments. Upload your zip file at the URL:

https://fis.encs.concordia.ca/eas/ as **Programming Assignment 3**. Assignments uploaded to an incorrect folder will not be marked and result in a **zero mark**. **No resubmissions will be allowed.**

Demo

A demo is needed for this assignment and your lab instructors will communicate the available demo times to you, where you **must** register a time-slot for the demo, and you must prepare your assignment and be ready to demo at the start of your time-slot. If the assignment is done by 2 members, then **both members must** be present for the demo. During your presentation, you are expected to demo the functionality of the application, explain some parts of your implementation, and answer any questions that the lab instructor may ask in relation to the assignment and your work. Different marks may be assigned to the two members of the team if needed. **Demos are mandatory. Failure to demo your assignment will entail a mark of zero for the assignment. <u>Failure to show up for a scheduled demo will entail you to a zero marks as no replacement of a demo time is/will be allowed.</u>**

Grading Policy (10 Marks)

1- With reliable environment: 6 Marks
   - Mimicking the TCP three-way handshaking technique when you in or start the communication with the server: 1 Mark
   - GET: 2.5 Marks (e.g. read the content of a file and listing files)
   - POST: 2.5 Marks (e.g. create a new file)

2- With drop rate only: 1.5 Marks

3- With delayed only: 1.5 Marks

4- With both drop and delay: 1 Mark

Optional Tasks (2 Marks)

1. Support multiple clients at the server: 2 Marks

References

[1] Selective Repeat. https://en.wikipedia.org/wiki/Selective_Repeat_ARQ.
