CS 571: Computer Networks **Spring 2021**

Programming Assignment 2

Assigned: April 4, 2021 *Due: April 23, 2021*

1 Introduction

In this programming assignment, you will implement a reliable data transfer protocol similar to TCP, but much simplified. You can modify the UDP echo client (UDPEchoClient-Timeout.c) and the UDP echo server (UDPEchoServer.c) from the "TCP/IP Sockets in C" book. *You should not use TCP at all.* Use the UDP echo client as the sender and the UDP echo server as the receiver. If you run the UDP code on a pair of machines that are not far away (or even on the same machine), most likely you will not see any loss. We will have parameters on the receiver to control the loss.

2 Sender

You can modify the UDP client code to send packets. The sender will send data in a buffer using UDP. Its interface should be:

```
myclient server_IP server_port
```

where `server_IP` is the IP address of the server, and `server_port` is the port number the server is listening to. If you run the client and the server on the same machine, `server_IP` should be 127.0.0.1.

You can assume the sliding window size is fixed at 5, meaning that at any time, the maximum number of oustanding packets (sent but not acked) is 5. To avoid the burden of reading data from a file, you can declare a variable called `buffer` of type character array and initialize it with a constant. Here is an example:

```
char buffer[240]="The University of Kentucky is a public, research-extensive,
     land grant university dedicated to improving people's lives through
     excellence in teaching, research, health care, cultural enrichment,
     and economic development.";
```

The message sent to the receiver is in the following format: type (int), sequence number (int), length (int) and data (the size of the data is specified by `length`, which is less than or equal to 1024). The field `type` should be 1, which indicates it is a data message. You can define the following structure for the messages you will send.

```
struct data_pkt_t {
    int type;
    int seq_no;
    int length;
    char data[1024];
};
```

However, the size of messages you will send should be sizeof(int)*3+`length`. To simplify the assignment, we assume that the length of data is always 10. Every time you need to send 10 bytes

from the buffer. So the total size is actually fixed at 22 bytes. The total number of packets that need to be received successfully at the receiver is 24. Their sequence numbers are $0, 1, 2, \cdots, 23$.

You can declare two variables to indicate the boundaries of the sliding window.

1) One is `base`, which is the smallest sequence number within the window, and has an initial value of 0. All packets up to `base-1` have been acknowledged by the receiver.

2) The other is `last_sent`, which is the packet with the largest sequence number in the window you can send or have sent. The number of packets that the sender can send is determined by the size of the sliding window, which is fixed at 5. The packets in the window are from `base` to `base+4`. Usually after sending all packets in the window, `last_sent` will be equal to `base+4`. Because we have only 24 packets to be sent out, the highest sequence number of packets can be sent in any window will be `last_sent = min(base+4, 23)`.

At the beginning, the sender will send all packets from the initial window to the receiver. After that, `base=0, last_sent = 4`. The sender sets a timer of 4 seconds. Then it has to wait for ACKs from the receiver or a timeout.

To detect duplicate ACKs, the sender side needs to declare another variable called `ndups`, recording the number of duplicate ACKs received. It is initialized to 0.

Case 1: It receives an ACK from the receiver.

Step 1: It should clear the timer. [1]

Step 2: It processes the ACK.

- If the ack number (`ack_no` in the ACK message) is 23, the sender is done and can exit.
- If the ack number is less than `base-1`, just ignore it.
- If the ack number is equal to `base-1`, it is a duplicate ACK. Add 1 to `ndups`. If `(ndups == 3)`, retransmit the packet with sequence number `base` only.
- If the ack number is equal to or greater than `base`, it is a new ACK. In this case, set `ndups` to 0, update `base` to be equal to the ack number + 1, and send *new* packets allowed by the `window_size`, i.e., packets with sequence number from `last_sent+1` until the sequence number `min(base+4, 23)`; update the `last_sent` variable to `min(base+4, 23)`.

Step 3: The sender should set the timer after processing ACK. [2]

Case 2: If it is a timeout, it will retransmit all packets from `base` to `last_sent`. After that, it will set a timer.

In both cases, it will go back to wait for the ACK from the receiver or a timeout.

The sender must print out the sequence numbers of all the packets transmitted and retransmitted in the format of "`SEND PACKET seq_no`" and the sequence numbers of all the ACKs received in the format of "`-------- RECEIVE ACK ack_no`".

[1] This is an over-simplified version of the reliable transmission protocol. In the TCP implememtation, only if it is a new ACK and there are outstanding packets unacked, or the sender just retransmited a packet, the timer will be reset.

[2] Again, this is an over-simpliefied version of the reliable transmission protocol. In the TCP implementation, it will set the timer only if $base <= last\_sent$.

3 Receiver

The modified UDP Echo server is the receiver and its interface should be:

```
myserver port_no a_list_of_numbers
```

where `port_no` is the port number that the server will listen to, and `a_list_of_numbers` is a list of sequence numbers that we use to force drop of packets.

The receiver declares a receiving buffer to hold the content sent from the sender. After it finishes, it should hold the same content as `buffer` in the sender. It uses an array called `packet_rcvd[24]` to record the packets received. It is initialized with all 0.

After receiving a packet from the sender, it will determine whether to drop the packet or not by checking with the `a_list_of_numbers` parameter provided as arguments when running the program. If the sequence number is equal to the first number in `a_list_of_numbers`, then this packet is dropped (no data will be copied to the receiving buffer) and no ACK will be sent to the sender. At the same time, remove this number from the list (only logically removed).

If the packet is not dropped, it will set the element in the array `packet_rcvd` indexed by the sequence number to 1. Copy the data in the packet to the correct location of the receiving buffer. Send an ACK with `ack_no` equal to the maximum sequence number of consecutive packets received starting from 0. [3] If the `ack_no` is equal to 23. It should also print out the whole message in the receiving buffer.

The message sent to the sender is in the following format: type (int), acknowledgement number (int). The field `type` should be 2 to indicate that it is an ACK for data messages.

You can define the following structure for the ACKs to be sent.

```
struct ack_pkt_t {
    int type;
    int ack_no;
};
```

The receiver must print out the sequence numbers of all the packets received in the format of "`RECEIVE PACKET seq_no`" and the acknowledgement numbers of all the ACKs sent (including duplicate ACKs that repeat previous `ack_no`) in the format of "`-------- SEND ACK ack_no`" or the messages dropped "`---- DROP seq_no`".

4 Hints

In the UDP echo client and echo server code, the client and server send string between them. If you want to send a structure, you do not need to transform it into a string to send and receive it. Rather you can send a structure directly. Here is an example to show how to send and receive a data packet structure.

At the sender side, you can send a structure

```
struct data_pkt_t send_dp;
int len = 22;
/* fill in the content properly */
if (sendto(sock, &send_dp, len, 0, (struct sockaddr *)
            &echoServAddr, sizeof(echoServAddr)) != len)
  ......
```

[3] The receiver uses an ACK only, cumulative ACK, and current packet number scheme.

At the receiver side, you can receive a structure

```
struct data_pkt_t receive_dp;
int rlen = 22;
int recv_len;
if ((recv_len = recvfrom(sock, &receive_dp, rlen, 0,
         (struct sockaddr *) &echoClntAddr, &cliAddrLen)) < 0)
  ......
```

5 Submission

You need to provide a Makefile. Your programs will be tested on a Linux environment similar to OpenStack VM. Robustness of the server/client is one of important aspects when your code is tested. Comments are required. Also write a README file to give a general description about your programs and instructions to run them, and state any limitations of the implementation. You should tar or zip all the files together and submit one tar/zip file.