Department of Computer Science and Information Systems, BITS Pilani
First Semester 2023-2024
Advanced Computer Networks (CS G525)
Assignment#1: UNIX Socket Programming

Date of posting: 25/08/2023 **Submission Deadline: 31/08/2023** **Weightage: 10% (10 Marks)**

Problem Statement: Implement a reliable sliding window protocol for data transfer in C language using UNIX socket APIs.

The protocol description is as follows:

1. Client and server establish a connection over UDP sockets.
2. Client sends a value of **Window_Size** (**WS**) (i.e., amount of packets that can be sent simultaneously without an ACK) to the server (receiver) over the socket.
3. Client reads data from a file named **file.txt** into a buffer.
4. Client calculates the number of packets to be sent to transfer the file.
5. Client sends the **WS** amount of packets to the server one by one. After that, it runs a Timer for retransmission Timeout (**RTO**). Each packet comprises **packet_no**, **packet_size,** and the **data** (i.e., the chunk of file-data equals to **packet_size**).
6. Server randomly discards packets based on a loss rate function to emulate the packet drop scenario. Such packets do not consider as received at the server.
7. Server sends an individual ACK packet for each successfully received packet. The server does not send any ACK packets for discarded packets in ***Step-6***. The ACK packet comprises **ack_no** (corresponding to **packet_no** received from the client).
8. Server copies the data received from the client into a file named **out.txt**. Data must be copied to the file in order. (Finally, **file.txt** and **out.txt** must have similar contents (byte by byte.)
9. Client slides its window (i.e., change the base) every time after receiving an in-order **ACK** and sends new packets accordingly.
10. If an **RTO** expires for a packet, then the sender resends that packet and runs Timer for the next outstanding packet (i.e., a packet that has been sent but ACK is not received yet).
11. The client repeats ***Steps 5-10*** to send the entire file to the server.
12. After an entire file is transferred, the client and server close the connection.

Note: A time sequence diagram of this protocol is shown on the next page for further clarity.

Program Output(s):

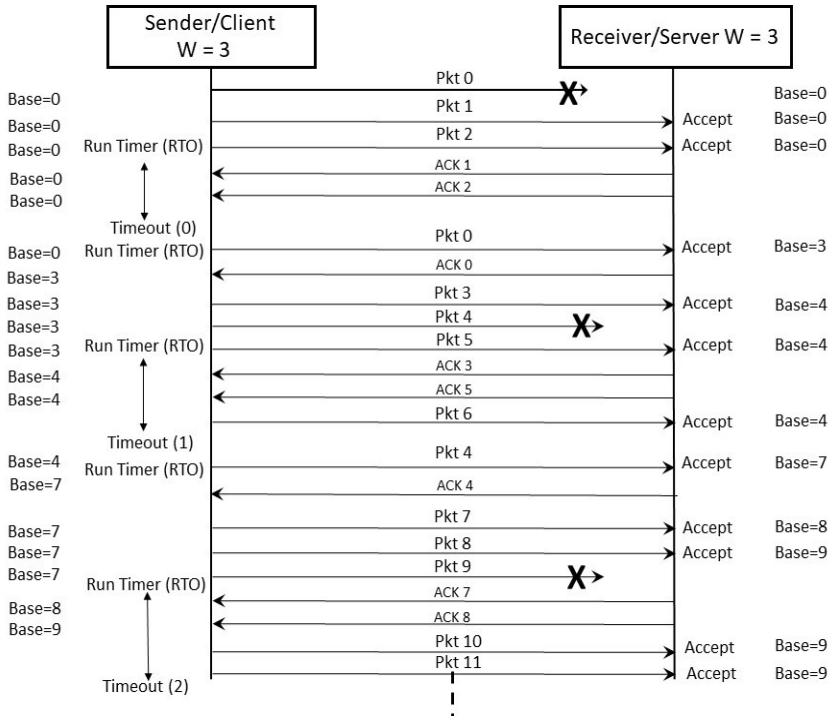
a) Client program must display the **packet_no** of each data packet transmitted, including retransmitted packets and **ack_no** of each ACK packet received. Also, display the base value of Window when the client sends or receives packets and display a Timeout message when **RTO** occurs. Sample output is shown below:

```
....
SEND PACKET 13: BASE 11
RECEIVE ACK 11: BASE 12
TIMEOUT 12

SEND PACKET 12: BASE 12
SEND PACKET 14: BASE 12
RECEIVE ACK 12: BASE 13
.....
```

b) Server program must display the **packet_no** of each packet received along with the information on whether it is **Dropped** or **Accepted** (based on probability calculation using a **rand()** function**).** Also, displays **the ack_no** of each ACK packet transmitted and the base value of the Window. Sample output is shown below:

…..
RECEIVE PACKET 12: DROP: BASE 11
RECEIVE PACKET 13: ACCEPT: BASE 11
RECEIVE PACKET 12: ACCEPT: BASE 12
SEND ACK 12
…..



Implementation Instructions/Hints:

- Write your program such that it can transfer data from any **file.txt**. After the program execution, the data received at the server should be put in a file with the filename **out.txt**.
- You are free to create packets of any size and test your program for all cases possible use cases to check the protocol functionalities.
- Write your program to work for any Window Size (**WS**). If you are not able to make it for any **WS** size, then specify the range for it works in the README.
- Since the client and server are both running on the same host machine, no packet drop would occur. Therefore, you must emulate the packet drop on the server side by randomly dropping packets. You can use the **drand48()** function (it returns a random value between 0 and 1) to emulate the packet drop.
- Packet drop emulation is applied only for the data packets. ACKs are not lost in any case. Also, packets are not corrupted in any case.
- To run a timer, you can use the **alarm()** function.

Submission Instructions:

Submit a client program (client.c), a server program (server.c), and a README file with your program compilation instructions on NALANDA on or before the submission deadline.