Department of Computer Science and Information Systems, BITS Pilani (Pilani Campus)
First Semester 2018-2019
Advanced Computer Networks (CS G525)
ASSIGNMENT

Date of posting: 16-8-2018
Submission Deadline: 23-8-2018 (mid night)
Maximum Marks: 7 **Mode: Individual**

Objective: This assignment problem will help you to tune up your programming skills. Particularly, you are going to learn TCP/UDP socket programming concepts using UNIX libraries.

Submission Instructions:

a) **Create a zip file (named as: assignment.zip) which comprises, source codes for both client program (client_src.c) and server program (server_src.c) and text file (in.txt) used for transferring from client to server.**
b) **Upload the zip file on NALANDA on or before 23 Aug 2018 (mid night)**

Problem Statement: The goal of this assignment is to implement a modified version of a Selective Repeat (SR) sliding window protocol (as described below) for the file transfer. You can use either C or C++ programming language for implementation. Client/server will communicate over the network and exchange the data.

Protocol Description:

1. Client and server establish a connection over UDP sockets.
2. Client sends value of **window_size** (**W**) (i.e., amount of packets can be send simultaneously without an ACK) to server (receiver) over socket.
3. Client reads data from a file named as **in.txt** into a buffer. (create a file with sufficient data to send several packets)
4. Client calculate the total number of packets (**N**) to be send to transfer the file.
5. Client sends **W** amount of packets to the server one by one. After that, it runs a Timer (**RTO**). Each packet comprises of **packet_no**, **packet_size** and the **data** (i.e., chunk of file-data equals to **packet_size**).
6. Server randomly discards packet(s) based on a loss rate function to emulate the packet drop scenario. Such packets do not considered as received at server.
7. Server sends an individual ACK packet for each successfully received packet. Server does not send any ACK packet for discarded packets in **(Step-6)**. The ACK packet comprises of **ack_no** (corresponding to **packet_no** received from client).
8. Server copies the data received from client into a file named as **out.txt**. Data must be copied to the file in order. (At the end **in.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 a Timer expires for a packet then sender resends that packet and run Timer for next outstanding packet (i.e., a packet which has been sent but **ACK** is not received yet).
11. Client repeats the **Steps 5-10** to send the entire file to the server.
12. After entire file is transferred, client and server close the connection.

Note: A time sequence diagram of this protocol is shown in Fig. 1

Sample Output(s):

a) Client program must displays the **packet_no** of each data packet transmitted including retransmitted packets and **ack_no** of each **ACK** packet received. Also, it should display a message to indicate for which packet no. **Timeout (RTO)** occurred. A 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 displays **packet_no** of each packet received along with the information whether it is **Dropped** or **Accepted.** Also, displays **ack_no** of each **ACK** packet transmitted and window BASE value. A 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
…..
```

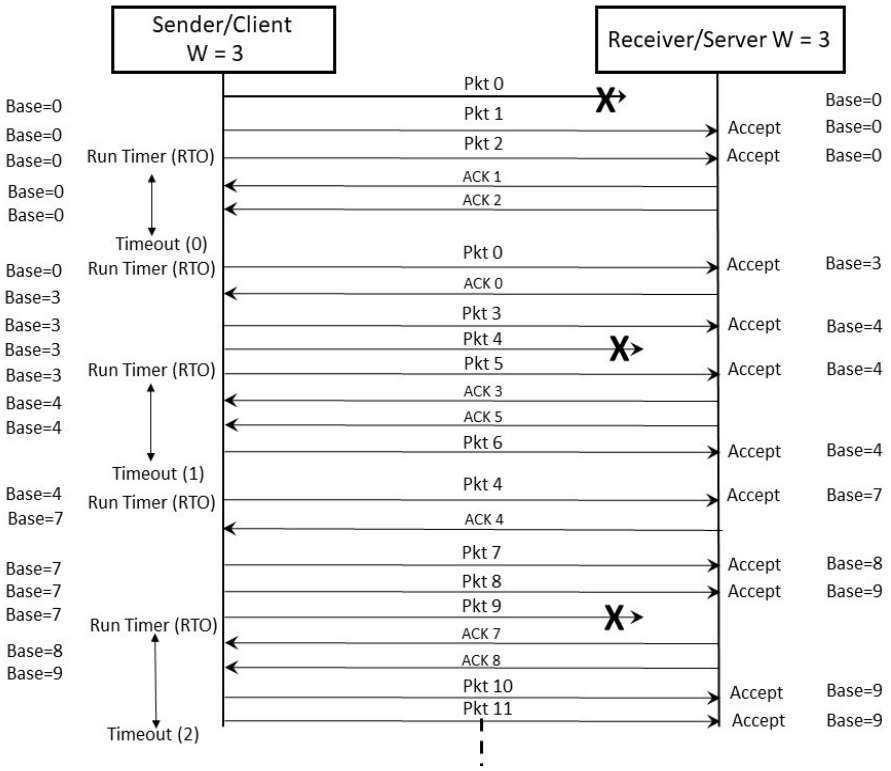

Fig. 1

Important points to be considered in your implementation:

1. It would be better if you decompose the protocol implementation into two scenarios i.e., without packet loss scenario and with packet loss scenario and control the execution of a particular scenario by command line arguments.
2. As the client and server both are running on a same host machine (or within same subnet) no packet drop would occur. Therefore, you have to emulate the packet drop at the server side by randomly dropping a couple of packets based of the **Packet Drop Rate (PDR)**. You can use **drand48()** function (it returns a random value between 0 and 1) to emulate the packet drop.
3. Packet drop emulation is applied only for the file-data packets. ACKs are not lost in any case. Also, you can assume packets are not corrupted in any case.
4. The code to handle the re-transmission timeout is provided. You would be needing this to handle the timeout situation for the re-transmission of packets. To run a timer you can use **alarm()** function.

****