Department of Computer Science and Information Systems, BITS Pilani
Second Semester 2023-2024
Computer Networks (CS F303)
LAB TEST (OPEN BOOK)

Duration: 150 Mints (10:00 AM - 12:30 PM) **Date: 21-04-2024** **Maximum Marks: 30**

Instructions (Read carefully): The lab exam has two questions: socket programming and packet trace analysis using Wireshark.

You have been provided a file folder named **LABEXAM-CSF303**

The folder contains seven files: **labexam_qp.pdf**, **receiver_sr.c** (partial receiver code for network programming problem), **sender_sr.c** (partial sender code for socket programming question), **file.txt** (contains data to be transferred from sender to receiver).

The **network_capture.pcap** (packet capture file to be used for the packet trace analysis question), and **wireshark.docx** (to write solutions of the packet trace analysis question) are to be used to answer the packet trace analysis question.

The folder **Lab_Sheets** contains lab sheets used this semester.

Submission Instructions:
Create a folder on the desktop and name it with your ID_NO (e.g., P2022A7PS0007). Copy the following files into the folder and create a zip file with the same name.
i) receiver_sr.c **ii) sender_sr.c** **iii) out.txt** **iv) wireshark.docx**
Make sure (double-check) that these four files you will upload are the correct ones and contain the solution you wrote. (The **out.txt** file has to be used to save the data the sender transfers to the receiver for the socket programming question.)

==

Q.1 The socket programming question description starts from the next page. [21]

Q.2 The packet trace analysis question is provided in a separate file named **wireshark.docx.** Open this file using libre-office and answer the question in that file itself. [09]

==

Q.1 In this question, you will implement a modified version of the Selective Repeat (SR) sliding window protocol for the file transfer.

The protocol description is as follows:

1. The sender and receiver establish a connection over UDP sockets.
2. The sender sends the value window size (i.e., the number of packets that can be sent simultaneously without waiting for an acknowledgment (ACK)) to the receiver over the socket.
3. The sender reads data from a file named **file.txt** into a buffer.
4. The sender calculates the number of packets to transfer the file. The sender defines the maximum number of bytes in a packet in a variable **pktSize** (given in the code file). All packets the sender sends are of the maximum possible size except the last one, which may be smaller.
5. The sender sends data packets to the receiver individually as permitted by window size restrictions. After that, a retransmission timer for the oldest packet is started, which has yet to be acknowledged. The data packet format comprises two header fields: **pkt_no**, **pkt_length**, and the payload (i.e., a chunk of file data.). The value of **pkt_length** represents the number of bytes of the file copied into the packet. The **pkt_no** starts from 0 to n (n = total number of packets to be transmitted -1)
6. The receiver randomly discards packets based on a loss rate function to emulate the packet drop scenario. Such packets are not considered received at the receiver and require retransmission after the Retransmission Timeout **(RTO).**
7. The receiver sends an individual acknowledgment packet (**ACK**) for each successfully received data packet. The receiver does not send any **ACK** for discarded packets in Step 6. The **ACK** packet comprises **ack_no** (corresponding to **packet_no** received from the sender. (For example, for pkt_no 2, ACK 2 will be sent.)
8. The receiver copies the data received from the sender 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. After receiving an in-order ACK, the sender slides its window (i.e., changes the base) every time and sends new packets as allowed.
10. If a Retransmission Timer expires (**RTO**) for a packet, then the sender resends that packet and starts the Retransmission Timer.
11. The sender repeats Steps 5-10 to send the entire file to the receiver.
12. The sender and receiver close the connection after completing the file transfer.

Note: A time sequence diagram of the protocol is shown on the next page.

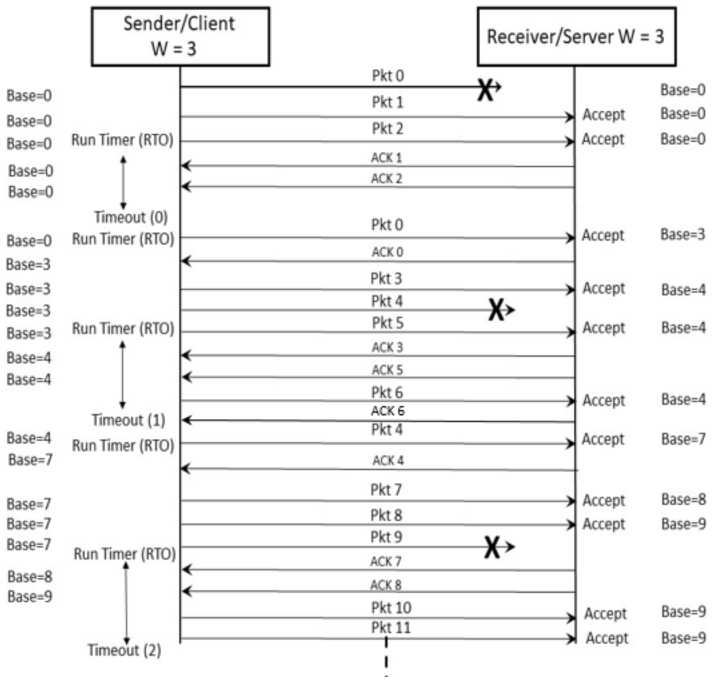
Program Output(s):

a) The sender program must display the **packet_no** of each data packet transmitted/retransmitted and **ack_no** of each **ACK** received. Also, it should display a message whenever **RTO** occurs (timer expires) and a **window BASE** value. A sample output snapshot is shown below:

```
…..
SEND PACKET 13: BASE 11
RECEIVE ACK 11: BASE 12
SEND PACKET 14: BASE 12
TIMEOUT 12                              (Note: timeout for packet no. 12)
SEND PACKET 12: BASE 12
RECEIVE ACK 12: BASE 13
```

b) The receiver program must display the **packet_no** of each arrived packet and the information on whether it is **DROPPED** or **ACCEPTED.** Also, it displays the **ack_no** of each ACK packet transmitted and the current **window BASE** value. A sample output is shown below:

```
…..
ARRIVE PACKET 12: DROPPED: BASE 12
ARRIVE PACKET 13: ACCEPTED:  BASE 12
ARRIVE PACKET 12: ACCEPTED: BASE 14
SEND ACK 13
SEND ACK 12
…..
```



Implementation Instructions: You are provided with the partial source code files (sender_sr.c and receiver_sr.c) for both the UDP client (sender) and UDP server (receiver). You MUST use only these files to write your program and NOT delete any code from these files in your final submission. You are ALLOWED to add as much code as you need to complete the program in these files.

You have to implement the protocol in two parts, i.e., without packet loss scenario and with packet loss scenario (it is also indicated in the partial code files!) so that both can be tested independently. The command line argument value passed through the main() function selects either one.

1. The data you are supposed to send from the sender to the receiver is provided in the filename **"file.txt"** file. After the program execution, the receiver must write the received data in a file with the filename "**out.txt**." (Note: You have to write code to create the file.)
2. The code to read the data from the file at the sender into a buffer has been provided as part of the partial code.
3. No packet drop will occur since the sender and receiver are running on the same host machine. Therefore, you must emulate the packet drop at the receiver side by randomly dropping a couple

of packets based on the **Packet Drop Rate (PDR)**. You can use the **drand48()** function (it returns a random value between 0 and 1) to emulate the packet drop.

4. Packet drop emulation is applied only for the file-data packets. ACKs are not lost in any case. Also, packets are not corrupted in any case.
5. The code to handle the retransmission timeout is provided. You would need this to handle the timeout situation for retransmitting packets. You do not have to emulate premature timeout. To run a timer, you can use the **alarm()** function. However, you can implement the RTO using any other method**.**
6. Assume both the programs (sender and receiver) would run on the same machine.

Evaluation Scheme:

a) Your code MUST compile (both client and server) to be evaluated. Non-compiled codes will not be evaluated.

b) Without packet loss case will be evaluated for **10** Marks.

c) With packet loss case will be evaluated for **11** Marks.

*****