Department of Computer Science and Information Systems, BITS Pilani
Second Semester 2018-2019
Computer Networks (CS F303)
LAB TEST (OPEN BOOK)

Duration: 2 Hrs. **Date: 14-04-2019** **MM: 40**

Read the following instructions carefully and follow them strictly:

1) **Write your code in the virtual machine (NSLAB). Click on "Oracle VM Virtual Box Manager" to start the Virtual Machine. (In case your machine lock, use password bits@123 to unlock it.)**
2) **Create a folder on the Desktop of Virtual Machine (VM) and name it with your BITS ID (e.g., 2016A7PS0007) and copy all solution files into this folder. Keep file names as it is mentioned in each question (i.e., server1a.c, client1a.c, server1b.c, client1b.c, server1c.c, client1c.c and question2.tcl). Also, write your name and BITS ID in each solution file and comment it. Finally, create a zip file (named as your BITS ID) of this folder and save on the Desktop of the VM.**
3) **As the TCP is a byte-oriented (stream based) protocol, it may be possible that the two separate packet transmissions are getting clubbed together by the TCP into a single transmission. To avoid this, you can introduce a small delay of 100 milliseconds (either at the client or the server) so that each packet goes as individual transmission.**
4) **The modularity and the indentation of the code are highly desirable. You may lose some marks even if the code is working as per the given specifications but not modular.**

Question-1(a) Write client and server programs to upload a given file **("input.txt")** from client to the server using **stop-and-wait ARQ** protocol for the erroneous channel (data packets could be loss but not acknowledgment packets) by making a TCP connection between them. Assume that the channel does not corrupt the packets. **[10]**

Note: Server program file to be saved as server1a.c and Client program file to be saved as client1a.c

Implementation details

1. **Packetization**: Divide the file into equal sized chunks and transmit each chunk separately using stop and wait protocol by encapsulating it in the form of a packet structure. Keep the value of PACKET_SIZE as 100 Bytes. However, while evaluating your code, we will test for other values also, therefore, keep **PACKET_SIZE** as a variable. Total file size may not be in multiple of 100 Bytes.

2. **Packet Structure**: In addition to the payload the packet structure should contain the following information in the form of header fields.
   a. The **size** (number of bytes) of the payload
   b. The **seq_no** (in terms of byte) specifying the offset of the first byte of a packet wrt to the input file.
   c. One Byte flag **last_pkt** specifying whether the packet is the last packet or not?
   d. Use one Byte field **TYPE** to differentiate between DATA or ACK packet.

   (Note: The **seq_no** field in the ACK packet would correspond to the DATA packet with the same **seq_no** value received from the client.)

3. To simulate the erroneous channel (as described above), the server will randomly drop a received packet with probability PDR and would not send ACK back to the client. Keep the value of PDR as 10%. While evaluating we may test the working of your program for different PDR values.

4. Keep the retransmission timeout duration at the client as two seconds. Keep a copy of the transmitted packet to facilitate re-transmission, instead of reconstructing a new packet from the input file again.

5. Keep buffering the received packets (only payload) in an array of characters and write the complete array to the output file only on receiving the last packet of the file from the client using a separate function. Read the packet size amount of data from the **input.txt** file to construct a new packet at the client.
6. The client and server program should produce the output in the following format:

Client Output	Server Output
SENT PKT: Seq. No. = 0, Size = 100 Bytes	RCVD PKT: Seq. No. = 0, Size = 100 Bytes
RCVD ACK: Seq. No. = 0	SENT ACK: Seq. No. = 0
SENT PKT: Seq. No.= 100, Size = 100 Bytes	DROP PKT: Seq. No. = 100
RE-TRANSMIT PKT: Seq. No. = 100, Size = 100 Bytes	RCVD PKT: Seq. No. = 100, Size = 100 Bytes

Question-1 (b): Modify the Question-1 (a) solution to transfer a given file using the **modified stop-and-wait protocol** as described below. **[12]**

Note: Server program file to be saved as server1b.c and Client program file to be saved as client1b.c

Multi-channel stop-and-wait protocol without DATA or ACK losses.

1. The sender transmits packets through two different channels (TCP connections).

2. First two packets would be directly transmitted through channel 0 and 1 respectively. Thereafter any packet transmission by the client would happen only on the receipt of an ACK from the server.

3. The server acknowledges the receipt of a packet with a delay of 100 milliseconds after calling the recv() function via the same channel through which the corresponding packet has been received.

4. The sender transmits a new packet through the same channel on which it has received an ACK for its one of the previously transmitted packet. Note that, at a time there can be at most two outstanding unacknowledged packets at the sender.

5. Since neither the DATA nor the ACK can be lost, re-transmission of DATA packets is not part of the protocol requirement.

6. At the receiver, the packets transmitted through different channels may arrive out of order. In that case, the server has to perform temporary buffering to construct in order packet stream.

Implementation details

1. Use two TCP connections between client and server to simulate two different channels. For your reference, a sample server code (**Multiple_Connections_Server.c**) is provided that can handle multiple TCP connections simultaneously. You can test the server code by running the following client program at multiple terminals while the server is running in one terminal.
   $ telnet 127.0.0.1 8888
   Note that the client should also be able to receive ACK from multiple connections simultaneously.
2. **Packetization** same as described in problem 1
3. In addition to the **Packet Structure** from problem 1, the packet structure should contain another header field **channel_ID** specifying the ID (either 0 or 1) of the channel through which the packet has been transmitted or received.
4. Implement a mechanism to handle out of order received packets at the server. Unlike problem 1, this time you **should not** store the packets in a temporary buffer and write the complete buffer to the file at the end. Instead, you should directly write the received packets in the output file as and when they are received.
5. The client and server program should produce the output in the following format.

Client Output	Server Output
SENT PKT: Seq. No. = 0, Size = 100 Bytes, CH=0	RCVD PKT: Seq. No. = 0, Size = 100 Bytes, CH=0
SENT PKT: Seq. No.= 100, Size = 100 Bytes, CH=1	SENT ACK: Seq. No. = 0, CH=0
RCVD ACK: Seq. No. = 0, CH=0	RCVD PKT: Seq. No. = 100, Size = 100 Bytes, CH=1
SENT PKT: Seq. No.= 200, Size = 100 Bytes, CH=0	SENT ACK: Seq. No. = 100, CH=1

Question-1 (c) Modify the client and server programs for **Question-1 (b)** to transfer the file from client to server using multi-channel stop-and-wait protocol considering "**erroneous channel"** as described in **Question-1 (a).** Use the implementation of **Question-1 (a)** to simulate the erroneous channel. You need to handle both the re-transmission of lost packets at the client and also the reception of out of order packets at the server. A packet would be re-transmitted through the same channel that was used to transmit it the first time. The output format for client and server program should be in similar format as given for Question-1(a) and 1(b). Make appropriate assumptions wherever necessary. **[5]**

Note: Server program file to be saved as server1c.c and Client program file to be saved as client1c.c

Question-2 Write a TCL script (**question2.tcl**) to simulate the network topology shown in Fig.1 in ns-2 simulator. All links are duplex with 2ms delay and DropTail queue. The bandwidth of individual links is mentioned in Fig.1. Use the Session routing policy for all the nodes. Create the UDP and TCP flows with required parameters as per the information mentioned in the Fig.1. Use different colors to designate the three different traffic flows for animation. Enable Nam and trace in the script. Set simulation run time as 20 seconds. **[13]**

For the TCP flow between nodes n1 and n6:

1. Write a procedure in the **question2.tcl** named as **plotWindow** to generate a graph (**congestion.xg**) using Xgraph utility to plot the TCP congestion window size with-respect-to time. Plot the size of congestion window at every 0.1 second interval of time.
2. Write a script/commands to obtain dropped packets count for the TCP flow. Add this script/commands at the end of the **question2.tcl** file and comment it.

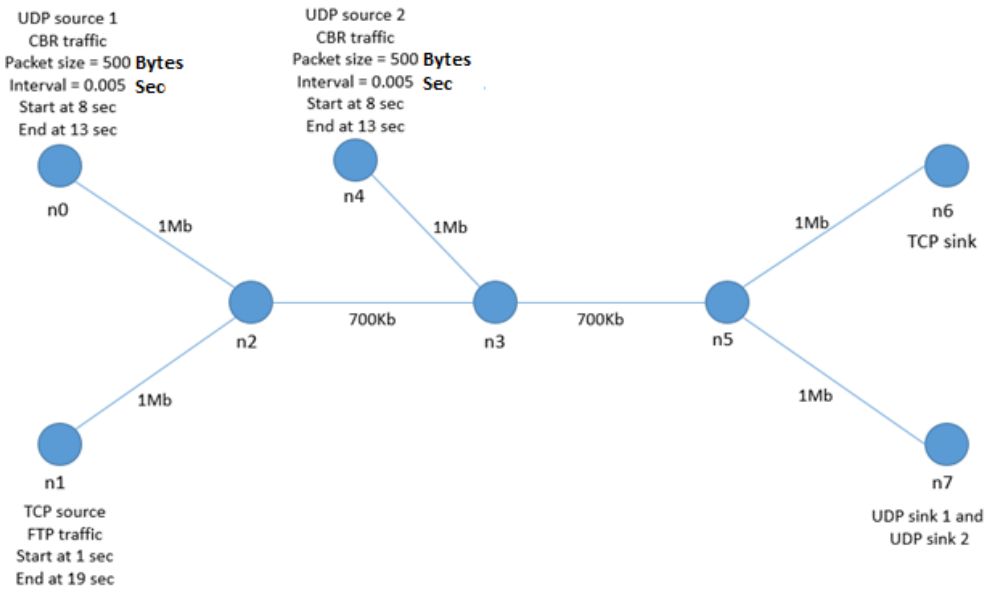

Fig.1

*****