COMPUTER NETWORKS (CS F303)
SECOND SEMESTER 2020-21
Lab Exam

Mode of Exam: Online (Open Book) Max. Marks: 30
Date of Exam: 11-04-2021 Time: 10:00 AM to 2:30 PM

Read carefully the below mentioned INSTRUCTIONS and follow them:

a) Write your code in C language using gcc compiler. Code written in any other languages will not be evaluated.
b) Write your NAME and BITS ID in all code files and comment it.
c) Make a single ZIP file of prob1_client.c, prob1_server.c, prob2.c, prob3.c, and prob4.c and upload to the LAB EXAM link at NALANDA-AWS. Upload the correct solution files and respect the DEADLINE. Submissions will not be accepted through any other modes. Also, submit the error free code files that can be compiled and executed.
d) Submissions with plagiarism will be penalized with huge penality and it can be lead to the "NC" grade in the course.

The question paper comprises 4 different approaches for reliable file transfer between two nodes using Stop-and-Wait ARQ protocol at application level. These problems have been designed in such a way that you can reuse the code as you move from problem 1 to 4. In the first problem, you have to write the code for transferring a file using client-server paradigm whereas for second problem file transfer is to be implemented for peer-to-peer paradigm. The third and fourth problems are focused on peer-to-peer file transfer over a logical ring topology.

Problem 1: Client-server file transfer using UDP as the transport protocol and implementing Stop-and-Wait protocol at application level. [06 Marks]

Write client and server programs to upload a given file (**input.txt**) from client to the server using UDP as the transport layer protocol and Stop and Wait ARQ protocol at application layer to attain reliable communication.

Your program MUST include following features:

1. Each line of the **input.txt** file should be transmitted separately using Stop-and-Wait protocol by encapsulating it in the form of a packet structure. (*Note that the length of each line of the file may be different and hence the packet size would vary.*)
2. In addition to the payload, the packet structure should contain following information in the form of a header.
   a. The size (number of bytes) of the payload.
   b. The Seq. No. (Required for Stop and Wait protocol).
   c. Whether the packet is the last packet or not?
   d. The packet is DATA or ACK. In this way, you can utilize the same packet structure for both DATA send by the client and ACK send by the server.
3. The packet loss should be implemented at the server. The server will randomly drop a received packet and would not send ACK back to the client. The packet drops rate (PDR) should again be defined as **#define macro**, and its value can be kept as 10% for the submission purpose. Assume the ACKs are not lost. Hence no need to implement ACK losses.
4. The client should handle the retransmission of lost packets. You are free to choose any of the following techniques to implement retransmission:
   a) Explicit timers
   b) Non-blocking socket and receive function calls

c) Explicit NACKs (Negative ACKs) sent by the server(receiver) (*Note: Similar to ACK packets NACK packets are also not lost*)

5. The duration of retransmission time should be defined as a macro and can be set 2 seconds for the submission purpose. You should keep a copy of the transmitted packet to facilitate re-transmission, instead of re-constructing a new packet from the input file again.
6. At server side, you **should not** store the packets in a temporary buffer and write the entire buffer to the file at the end. Instead, directly write the received packets in the output file (**output.txt**) as they are received.
7. The client and server should produce the traces (using printf statement) of every sent (including retransmission) and receive events in the following format:

Client Trace	Server Trace
`SENT DATA:Seq.No---of size---Bytes`	`RCVD DATA:Seq.No---of size---Bytes`
`RCVD ACK:for PKT with Seq.No. ---`	`SENT ACK: for PKT with Seq.No.---`
`RESENT DATA: Seq.No---of size---Bytes`	`DROP DATA:Seq.No.--- of size---Bytes`

Deliverables: (i) prob1_client.c (ii) prob1_server.c

Problem 2: P2P file transfer using UDP as the transport protocol and implementing Stop-and-Wait protocol at application level. [08 Marks]

Now consider two identical peers who want to transfer files to each other instead of one being client and other being server. Instead of client and server we can refer them as sender and receiver. You have to write a single program. However, to differtiate between their running instances as different peers they should take PORT number on which they would be binding to receive the file from other peer node, as command line argument. The program will be tested by running it in two different terminals. So, initially both the peers ask the user to enter the name of the file to be transmitted while at the same time both the peers would be in position to receive the file from the other end. The program will tested by entering a file name to be transmitted in one of the terminals. Rest of the specifications (packet structure, Stop-and-Wait ARQ, packet drop mechanism) are same as given in Problem 1. Your program should produce the traces in the following format:

Sender Trace	Receiver Trace
`SENT DATA:Seq.No---of size---Bytes`	`RCVD DATA:Seq.No---of size---Bytes`
`RCVD ACK:for PKT with Seq.No. ---`	`SENT ACK: for PKT with Seq.No.---`
`RESENT DATA: Seq.No---of size---Bytes`	`DROP DATA:Seq.No.--- of size---Bytes`

Deliverables: prob2.c

Problem 3: P2P file transfer between three peers connected in a logical ring fashion, using UDP as the transport protocol and implementing Stop-and-Wait protocol at application level. [10 Marks]

Now consider three identical peers who want to transfer files to each other in P2P manner. However, assume that the underlaying topology that connects them is logical ring topology. It means a peer node with ID (the ID starts from 0) i can only communicate directly with peer node having ID (i+1) mod 3. In other words all the packets (DATA, ACK and NACK) will move in clockwise direction only. You need to extend the packet structure by adding two fields source ID and destination ID to mention communication end points (similar to source and destination IP addresses in the IP header). For example, a packet from node 0 to node 2, would contain source as 0 and destination as 2. However, as the packet cannot go directly from 0 to 2 the node 0 should first send the packet to node 1 and using the destination field the node 1 can decide whether the packet is for itself or to be relayed. Note that the ACKs cannot take the same path taken by the DATA packet due to topology restriction. So, in this example the ACK/NACK from node 2 to 0 can be sent directly. The relay node can also drop the packet (as per the PDR value) in addition to the desination node.

You have to write a single program. However, to differtiate between their running instances as different peers they should take PORT number on which they would be binding to receive the file from their predecessor peer node, as command line argument. The program will be tested by running it in three different terminals. So, initially all three peers ask the user to enter the name of the file to be transmitted while at the same time all of them would be in position to receive the file from any other peer. The program will be tested by entering the name of the file to be transmitted and destination peer ID in one of the terminals. Rest of the specifications (packet structure, Stop-and-Wait ARQ, packet drop mechanism) are same as given in Problem 1.
Your program should produce the traces in the following format: (Note that if there is no relay node exists between the sender and the receiver then Relay Trace is not applicable.)

Sender Trace
`NODE_ID:--- SENT DATA:Seq.No---of size---Bytes`
`NODE_ID:---RCVD ACK:for PKT with Seq.No.---`
`NODE_ID:--- RESENT DATA:Seq.No---of size---Bytes`
Receiver Trace
`NODE_ID:--- RCVD DATA:Seq.No---of size---Bytes`
`NODE_ID:--- SENT ACK:for PKT with Seq.No.---`
`NODE_ID:--- DROP DATA:Seq.No.--- of size---Bytes`
Relay Trace
`NODE_ID:--- RELAY DATA:Seq.No.---of size---Bytes`
`NODE_ID:--- RELAY ACK:for PKT with Seq.No.---`
`NODE_ID:--- DROP DATA:Seq.No.--- of size---Bytes`

Deliverables: prob3.c

Problem 4: Fair P2P file transfer between three peers connected in a logical ring fashion, using UDP as the transport protocol and implementing Stop-and-Wait ARQ protocol at application level. [06 Marks]

In previous problem (**Problem 3**) of P2P file transfer system, if a node is transmitting a very large file then others will not get the chance to transmit until the complete file is not transmitted. According to the fairness rule each node should get equal opportunity for transmitting data. We can use a simple approach that puts an upper cap on the number of packets can be send by a node in one go.
Consider a token packet (TOKEN) that rotates in the logical ring topology and carries a user entered value that specifies the maximum number of data packets that can be sent by a node in one go. If a node has the data to send then it will capture the TOKEN and send one or more packets but not more than the value specified in the TOKEN. After that, the node forwards the token packet to the next node in order (clockwise), otherwise it simply forwards the TOKEN to the next node in order. In this manner, each node in the ring gets equal opportunity of sending data and there is no startvation.

Write a program that extends the problem 3 solution by incorporating the above mentioned fairness mechanism for P2P file transfer in reliable manner using Stop-and-Wait ARQ in a logical ring topology. The value field in TOKEN should be defined as macro and can be set to 4 packets for the submission purpose. Rest of the specifications are same as given in the Problems 1, 2 and 3 and it will be tested similar to Problem 3.
Your program should produce the traces similar to **Problem 3** by adding the traces of TOKEN sent and received events.

Deliverables: prob4.c

****