COEN 146 - Winter 2019
Lab Project 3
TFv2 - Stop and Wait for an Unreliable Channel

Pre Lab – Week 4, upload a pseudo-code to Camino before the lab (10%)
Final Lab – Week 5, demo in the lab and upload by midnight on Friday (90%)

This project consists of building an S&W (Stop and Wait) reliable protocol. TFv2 is going to be built on top of UDP, and it is supposed to provide a reliable transport service to TFv1 (developed in week 3, which needs to change to call your new send and receive functions and use buffers of same size). Messages are sent one at a time, and each message needs to be acknowledged when received, before a new message can be sent. TFv2 implements basically the protocol rdt2.2 presented in the text book.

TFv2 consists of a client and a server. Communication is unidirectional, i.e., data flows from the client to the server. The server starts first and waits for messages. The client starts the communication. Messages have seq number 0 or 1 (start with zero). Before sending each message, a 1-byte checksum is calculated and added to the header. After sending each message, the client waits for a corresponding ACK. When it arrives, if it is not the corresponding ACK (or if the checksum does not match), the message is sent again. If it is the corresponding ACK, the client changes state and returns to the application, which can now send one more message. This means that TFv2 blocks on writes until an ACK is received.

The server, after receiving a message, checks its checksum. If the message is correct and has the right seq number, the server sends an ACK0 or ACK1 message (according to the seq number) to the client, changes state accordingly, and deliver data to the application.

The protocol should deal properly with duplicate data messages and duplicate ACK messages. Follow the FSM in the book!

TFv2 message contains the header and the application data. No reordering is necessary, since TFv2 is sending the exact message given by the application, one by one.

The checksum must be calculated for messages from the server and client. To calculate the checksum, calculate the XOR of all the bytes (header + data of just header if there is no data) when member cksum (see below) is zero. To verify your protocol, use the result of a random function to decide whether to send the right checksum or just zero. This will fake the error effect.

PROTOCOL

HEADER

seq_ack	int (32 bits)	// SEQ for data and ACK for Acknowledgement
len	int (32 bits)	// Length of the data in bytes (zero for ACKS)
cksum	int (32 bits)	// Checksum calculated (by byte)

PACKET

header	
data	char (10 bytes)

SENDER

- Member seq_ack is used as SEQ, and the data is in member data.
- Each packet may have 10 or less bytes of data, and the sender only sends the necessary bytes.
- After transmitting the file, a packet with no data (len = 0) is sent to notify the receiver that the file is complete.

RECEIVER

- Member seq_ack is used as ACK, and data is empty (len = 0).

TRICKY SITUATION

The server closes the file and terminates execution after the message with zero bytes arrives. If the ack sent for that last message does not make it to the client, the client will keep resending it forever. To avoid that, the client will start a counter after sending a message with zero bytes, and will only resend that last message 3 times. After 3 times, it will return to the main function.

INFO on 146 LABS

1. The TA will help you only if you have submitted a pseudo-code of your program.
2. For full credit, you will be expected to demo with another student, both the client and server.
3. You should open two terminal windows and execute a server in one a client on the other one. It is ok if both terminals are executing on the same machine. Just use the right IP address in the client side. The system will figure it out. To obtain the IP address of a machine type <ip addr show>.