COEN 146: Computer Networks
Lab assignment 6: Stop and Wait for an Unreliable Channel, with Loss

Objective

1. To build a Stop and Wait reliable protocol on top of UDP to provide a reliable transport service while considering loss

TFv3 – Stop and Wait for an Unreliable Channel, with Loss

In Lab 5, you have developed the TFv2 which implements a reliable file transfer using Stop and Wait protocol rdt2.2, which examines whether or not a packet and/or an ACK is in error. In this lab, you will extend the protocol to examine also the loss of a packet and/or an ACK. This version of file transfer is then TFv3.

TFv3 implements basically the protocol rdt3.0 presented in the textbook. The FMS of the sender is shown below. 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 sequence or ack number 0 or 1 (start with zero). Before sending each message, a checksum is calculated and added to the header.

After sending each message, the client starts a timer, using `select()`. If the return of `select()` is zero, there is no data, and the client needs to retransmit, restart the timer, and call select again. If select returns non-zero, there is data, so the client calls `recvfrom()` to receive the ACK and then processes it. If the ACK is not corrupted and the ACK number is right, the client can now send the next message.

After receiving each message, the server checks its checksum. If the message is correct and the seq number is right, the server sends an ACK message with the right seq number, and the data is ready to be written to the file. Otherwise, the server repeats the last ACK message and waits to receive a message again. So, your server code will be the same as the server code in Lab5. You only need to simulate a drop of ACK.

To verify your protocol, use the result of a random function to decide to send or skip a message, to decide to send or skip an ACK message (only change to the server), and to decide whether to send the right checksum or just zero. This will fake the error and loss effect.

SELECT

This is an example on how to use `select( )`:

```
// local variables needed
struct  timeval tv; // timer
int rv;     // select returned value

// set up reads file descriptor at the beginning of the function to be checked for being ready to
read
fd_set  readfds;
fcntl (sock, F_SETFL, O_NONBLOCK);
…
// start before calling select
FD_ZERO (&readfds); //initializes readfds to have zero bits
FD_SET (sock, &readfds); //sets readfds bit

// set the timer
tv.tv_sec = 1;
tv.tv_usec = 0;

// call select () which returns the number of ready descriptors that are contained in the bit masks.
// if the time limit expires, select returns zero and sets errno
rv = select (sock + 1, &readfds, NULL, NULL, &tv);
```

```
if (rv == 0)
{
   // timeout, no data
}
else if (rv == 1)
{
   // there is data to be received
}
```

Important note

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 a non-responding server. 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.

Requirements to complete the lab

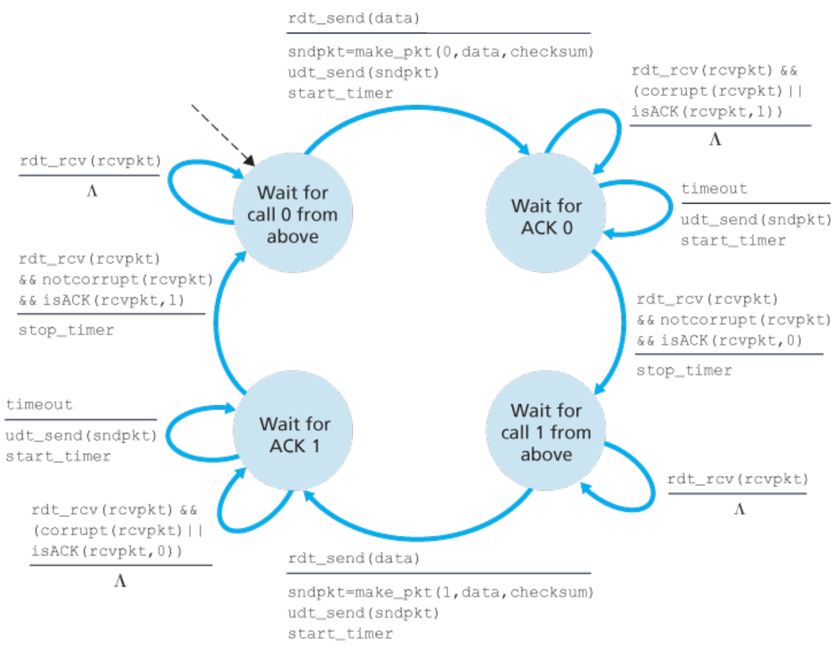
Show the TA correct execution of the programs you wrote and upload source code to Camino.

Be sure to retain copies (machine and/or printed) of your source code. You will want these for study purposes and to resolve any grading questions (should they arise)

Please start each program with a descriptive block that includes minimally the following information:

```
/*
* Name: <your name>
* Date:
* Title: Lab6 - ….
* Description: This program … <you should
* complete an appropriate description here.>
*/
```

Rdt3.0 sender - FSM


rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
Λ
rdt_rcv(rcvpkt)
Λ
Wait for call 0 from above
Wait for ACK 0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,1)
stop_timer
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)
stop_timer
timeout
udt_send(sndpkt)
start_timer
Wait for ACK 1
Wait for call 1 from above
rdt_rcv(rcvpkt)
Λ
rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,0))
Λ
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)
start_timer