Programming Assignment

COEN 233 Computer Networks - Fall Quarter 2014

Due date: Midnight December 4, 2014

The assignment should be built in 3 parts. You can demo each part as you are finishing them.

General Rules

- Projects are individual. Discussions are ok, but each student should have his/her own code.
- Each project requires a demo, during which the student should explain how the code works. Demos are part of the grade. You only get full credit if you demo.
- Besides the demo, each student should submit the code.
- If you do the project, deliver the code by the deadline, demo by the deadline, and it works properly, you get full credit for it.
- Keep the solutions simple!

Part 1: SFTP – Reliable Transfer over a Reliable Channel

This project consists of building an SFTP (Simple File Transfer Protocol). It consists of a client and a server that exchange one file using TCP. The client accepts 4 arguments: the name of the two files (<input> and <output>), and the IP address (or Internet name) and the port number of the server. The server starts first and waits for a connection request. The client requests a connection and then sends the name of the file <output> for output and the data in file <input> to the server, which saves the info received in the file <output>. The client reads the file and sends the data in chunks of 10 bytes. After sending the file, the client closes the connection and exits. The server receives the data and writes the file in chunks of 5 bytes.

The server needs to know when the transmission is over so that it can close the file. You need to come up with a mechanism for that! After executing, <input> and <output> should look the same. Your SFTP should be built on top of TCP. You must use C/C++ and Linux (any Unix will do it!). You will need to use the socket library. The man pages in the Unix/Linux systems have a lot of useful information. Start with <man socket>. There is a man page for each function in the socket library.

Your SFTP should be able to transfer binary files as well as text files.

Part 2: Reliable Transfer over an Unreliable Channel with Bit Errors

This project consists of building a Stop and Wait (S&W) reliable protocol. The S&W is going to be built on top of UDP, and it is supposed to provide a reliable transport service to the SFTP application (developed in part 1, which needs to change to call your new send and receive

functions). Messages are sent one at a time, and each message needs to be acknowledged when received, before a new message can be sent.

The S&W 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. Before sending each message, a checksum is calculated and added to the S&W 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 the S&W blocks on writes.

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!

The S&W message contains the header and the application data. No reordering is necessary, since the S&W is sending the exact message given by the application, one by one.

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.

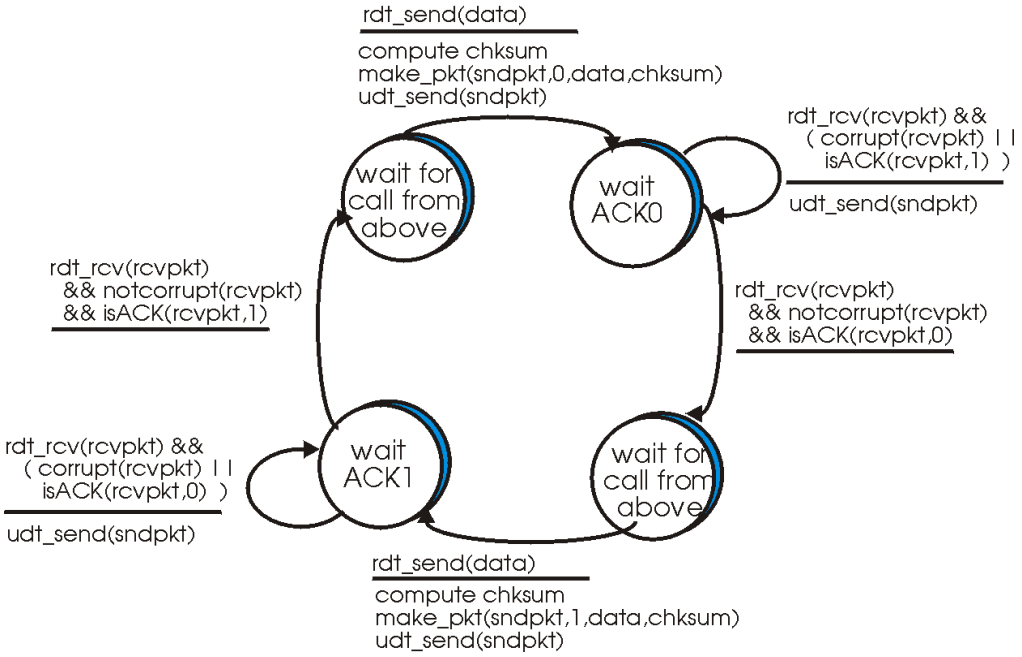

Fig 1: Sender

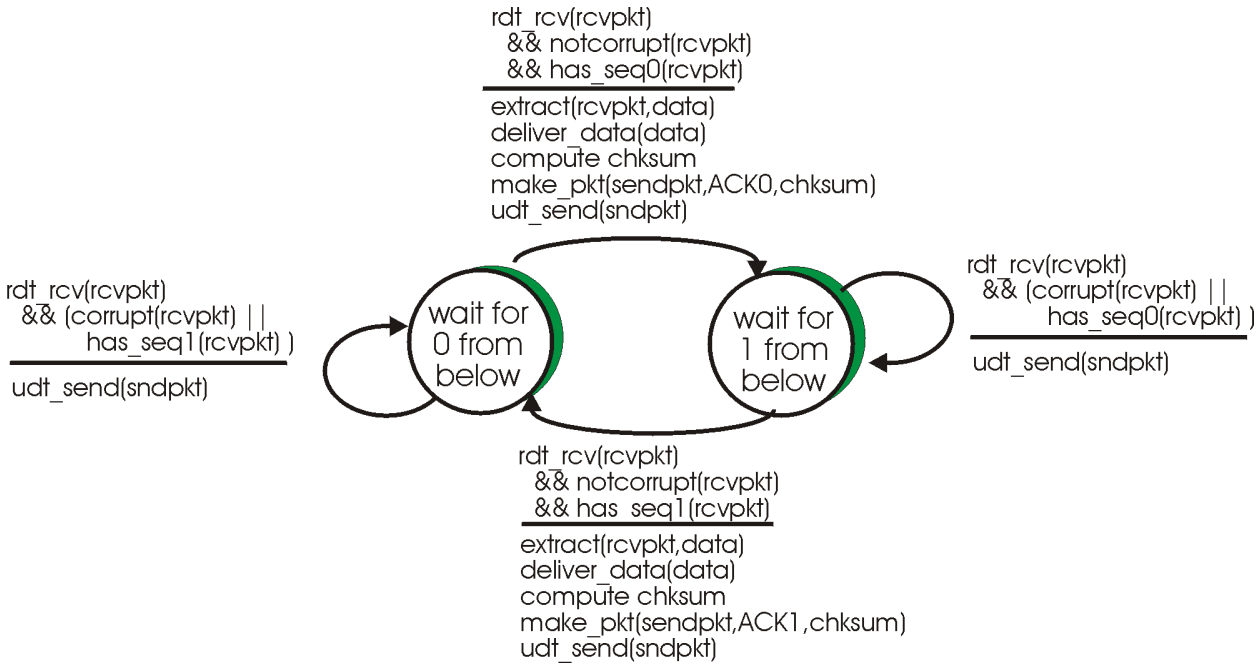

Fig. 2: Receiver

Part 3: Reliable Transfer over an Unreliable Channel that can also loose packets

This project consists of extending the protocol developed in part 2 to enable it to deal with loss of messages. The E_S&G 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. Before sending each message, a checksum is calculated and added to the E_S&G header. After sending each message, the client starts a timer (use alarm or sleep). When the timer goes off, the client tries to read a corresponding ACK message. If the corresponding ACK is not there, the message is sent again and the timer is started again. If the corresponding ACK is there, the client returns to the application which can now send one more message. This means that the E_S&G blocks on writes. The server, after receiving a message, checks its checksum. If the message is correct, the server sends an ACK0 or ACK1 message (according to the seq number) to the client and delivers the message to the application. If the message is not correct, the server repeats the last ACK message.

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, and to decide whether to send the right checksum or just zero. This will fake the error and loss effect.

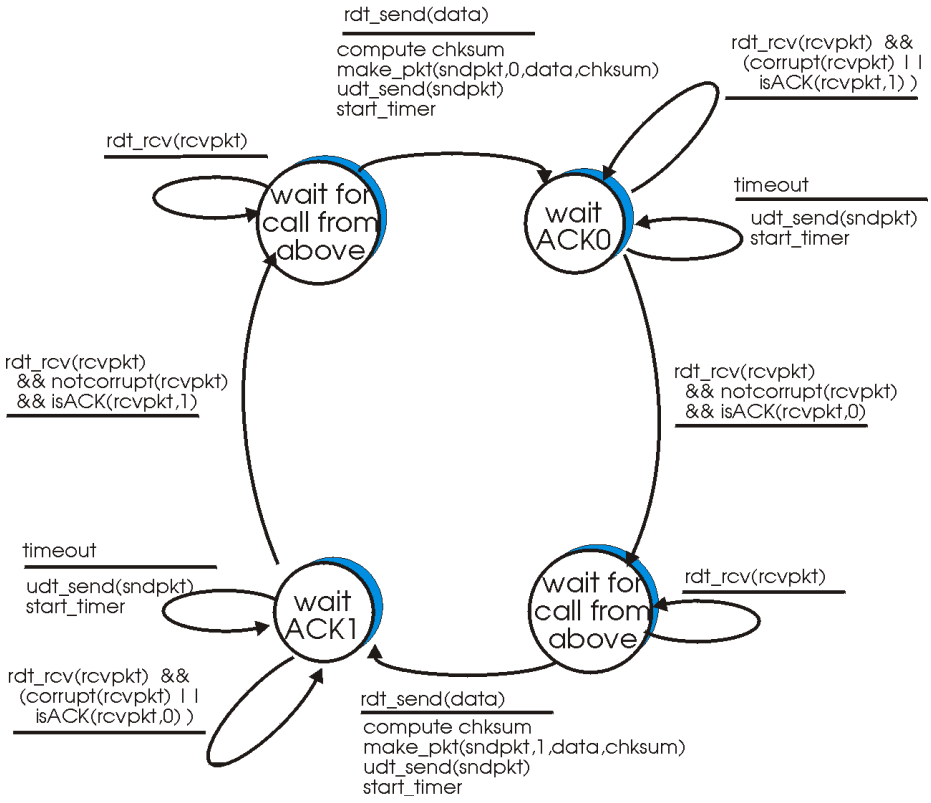
rdt_send(data)
compute chksum
make_pkt(sndpkt,0,data,chksum)
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) | |
isACK(rcvpkt,1))
rdt_rcv(rcvpkt)
wait for
call from
above
wait
ACK0
timeout
udt_send(sndpkt)
start_timer
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,1)
rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& isACK(rcvpkt,0)
timeout
udt_send(sndpkt)
start_timer
wait
ACK1
wait for
call from
above
rdt_rcv(rcvpkt)
rdt_rcv(rcvpkt) &&
(corrupt(rcvpkt) | |
isACK(rcvpkt,0))
rdt_send(data)
compute chksum
make_pkt(sndpkt,1,data,chksum)
udt_send(sndpkt)
start_timer

Fig. 3: Sender