IIT HYDERABAD, DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CS3041 - Assignment 5

Antony Franklin

October 29, 2015

1 IMPLEMENTING A RELIABLE TRANSPORT PROTOCOL

In this programming assignment, you will be implementing a simple reliable data transfer protocol. There are two versions, the Alternating-Bit-Protocol version and the Go-Back-N version. This lab should be fun since your implementation will differ very little from what would be required in a real-world situation. Since you may not have experience to modify the OS, your code will execute at application layer. However, the programming interface that you would develop is very close to what is done in an actual UNIX environment.

2 THE PROGRAMS YOU WILL WRITE

The programs you will write are for the sending entity (A) and the receiving entity (B). Only unidirectional transfer of data (from A to B) is required. Of course, the B side will have to send packets to A to acknowledge (positively or negatively) receipt of data. Your routines are to be implemented in the form of the procedures described below.
The unit of data passed between the upper layers and your protocols is a message, which is declared as:

```
struct msg {
char data[20];
};
```

Your sending entity will thus receive data in some fixed size bytes (say 20-byte chunks) from layer 5; your receiving entity should deliver 20-byte chunks of correctly received data to layer 5 at the receiving side.
The unit of data passed between your routines and the network layer is the packet, which is declared as: //

```
struct pkt {
```

int seqnum;

int acknum;

int checksum;

char payload[20];

};

Your routines will fill in the payload field from the message data passed down from layer 5. The other packet fields will be used by your protocols to insure reliable delivery, as we've seen in class. The routines you will write are detailed below. As noted above, such procedures in real-life would be part of the operating system, and would be called by other procedures in the operating system.

- *A_output(message)*, where message is a structure of type msg, containing data to be sent to the B-side. This routine will be called whenever the upper layer at the sending side (A) has a message to send. It is the job of your protocol to insure that the data in such a message is delivered in-order, and correctly, to the receiving side upper layer.
- *A_input(packet)*, where packet is a structure of type pkt. This routine will be called whenever a packet sent from the B-side (i.e., as a result of a *tolayer3()* being done by a B-side procedure) arrives at the A-side. *packet* is the (possibly corrupted) packet sent from the B-side.
- *A_timerinterrupt()* This routine will be called when A's timer expires (thus generating a timer interrupt). You'll probably want to use this routine to control the re-transmission of packets.
- *A_init()* This routine will be called once, before any of your other A-side routines are called. It can be used to do any required initialization.
- *B_input(packet)*, where packet is a structure of type pkt. This routine will be called whenever a packet sent from the A-side (i.e., as a result of a tolayer3() being done by a A-side procedure) arrives at the B-side. packet is the (possibly corrupted) packet sent from the A-side.
- *B_init()* This routine will be called once, before any of your other B-side routines are called. It can be used to do any required initialization.

3 SOFTWARE INTERFACES

You also have to write additional following routines which can be called by the routines mentioned above. You can use a UDP socket between end systems to emulate a physical communication link.

- *tolayer3(packet)* at A-side, where packet is a structure of type pkt. Calling this routine will cause the packet to be sent into the network (over the UDP socket), destined for the other entity.

- *tolayer5(message),* where message is a structure of type msg. With unidirectional data transfer, you would be calling this at B-side. Calling this routine will cause data to be passed up to layer 5.

4 THE SIMULATED NETWORK ENVIRONMENT

A call to procedure *tolayer3()* sends packets into the medium (i.e., into the network layer (using UDP socket)). Your procedures *A_input()* and *B_input()* are called when a packet is to be delivered from the medium to your protocol layer at A-side and B-side, respectively.
The medium is capable of corrupting and losing packets. It will not reorder packets. This would be emulated at the receiver side as you have done in your previous assignment. You should be able to specify these values while running the simulation environment.

- Number of messages to simulate: Your emulator should stop as soon as this number of messages have been passed down from layer 5, regardless of whether or not all of the messages have been correctly delivered. Thus, you need not worry about undelivered or un-ACK'ed messages still in your sender when the emulator stops.

- Loss. You are asked to specify a packet loss probability. A value of 0.1 would mean that one in ten packets (on average) are lost.

- Corruption. You are asked to specify a packet loss probability. A value of 0.2 would mean that one in five packets (on average) are corrupted. Note that the contents of payload, sequence, ack, or checksum fields can be corrupted. Your checksum should thus include the data, sequence, and ack fields.

- Tracing. Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). A tracing value of 0 will turn this off. A tracing value greater than 2 will display all sorts of odd messages that are for my own emulator-debugging purposes. A tracing value of 2 may be helpful to you in debugging your code. You should keep in mind that real implementors do not have underlying networks that provide such nice information about what is going to happen to their packets.

5 THE ALTERNATING-BIT-PROTOCOL VERSION

- You are to write the procedures, *A_output(), A_input(), A_timerinterrupt(), A_init(), B_input(), and B_init()* which together will implement a stop-and-wait (i.e., the alternating bit protocol, which we discussed as rdt3.0 in the class) unidirectional transfer of data from the A-side to the B-side. Your protocol should use both ACK and NACK messages.

- For your sample output, your procedures might print out a message whenever an event occurs at your sender and receiver (a message/packet arrival, or a timer interrupt) as well as any action taken in response. You might want to hand in output for a run up to the point (approximately) when 10 messages have been ACK'ed correctly at the receiver, a loss probability of 0.1, and a corruption probability of 0.3, and a trace level of 2. You might want to annotate your printout with a colored pen showing how your protocol correctly recovered from packet loss and corruption.

6 THE THE GO-BACK-N VERSION

You are to write the procedures, A_output(), A_input(), A_timerinterrupt(), A_init(), B_input(), and B_init() which together will implement a Go-Back-N unidirectional transfer of data from the A-side to the B-side, with a window size of 8. Your protocol should use both ACK and NACK messages. Consult the alternating-bit-protocol version of this lab above for information about how to obtain the network emulator.
Some new considerations for your Go-Back-N code (which do not apply to the Alternating Bit protocol) are:

- A_output(message), where message is a structure of type msg, containing data to be sent to the B-side. Your A_output() routine will now sometimes be called when there are outstanding, unacknowledged messages in the medium - implying that you will have to buffer multiple messages in your sender. Also, you'll also need buffering in your sender because of the nature of Go-Back-N: sometimes your sender will be called but it won't be able to send the new message because the new message falls outside of the window.
- Rather than have you worry about buffering an arbitrary number of messages, it will be OK for you to have some finite, maximum number of buffers available at your sender (say for 50 messages) and have your sender simply abort (give up and exit) should all 50 buffers be in use at one point
- A_timerinterrupt() This routine will be called when A's timer expires (thus generating a timer interrupt). Remember that you've only got one timer, and may have many outstanding, unacknowledged packets in the medium, so you'll have to think a bit about how to use this single timer.
- Consult the Alternating-bit-protocol version of this lab above for a general description of what you might want to hand in. You might want to hand in output for a run that was long enough so that at least 20 messages were successfully transferred from sender to receiver (i.e., the sender receives ACK for these messages) transfers, a loss probability of 0.2, and a corruption probability of 0.2, and a trace level of 2, and a mean time between arrivals of 10. You might want to annotate parts of your printout with a colored pen showing how your protocol correctly recovered from packet loss and corruption.

7 HELPFUL HINTS

- Checksumming: You can use whatever approach for checksumming you want. Remember that the sequence number and ack field can also be corrupted. We would suggest a TCP-like checksum, which consists of the sum of the (integer) sequence and ack field values, added to a character-by-character sum of the payload field of the packet (i.e., treat each character as if it were an 8 bit integer and just add them together).
- START SIMPLE: Set the probabilities of loss and corruption to zero and test out your routines. Better yet, design and implement your procedures for the case of no loss and no corruption, and get them working first. Then handle the case of one of these probabilities being non-zero, and then finally both being non-zero.
- Debugging: We'd recommend that you set the tracing level to 2 and put LOTS of printf's in your code while your debugging your procedures.

8 SUBMISSION GUIDELINES

All the deliverable must be uploaded in the Course Web Page on Moodle as a single compressed file. The due date for submission of the assignment is Monday, Nov 10, 2015, 8:00 PM. There would be one day cut-off date for the submission. If you do late submission after the due date and before the cut-off date there would be 20% penalty in your evaluated score. After the cut-off date you would not be able to submit your assignment.

You can get your program execution evaluated by the TAs before Nov. 13, 2015 6:00 PM.

9 EVALUATION METHODOLOGY

- Correct implementation and emulation of Alternating-Bit-Protocol (40)
- Correct implementation and emulation of Go-back-B (40)
- Display of proper and informative output messages (10 marks)
- Code Documentation/Comments/README file (10 marks)