CNT4007C Computer Network Fundamentals, Spring 2015

Programming Assignment 2

babak.ap@ufl.edu

Date assigned: Friday, March 13, 2015
Date due: Friday, March 27, 2015 (9:00 am EST)
NO LATE submission will be accepted for grading!
How to submit: Please submit via SAKAI following the guidelines.
Questions: Come to office hours or contact TA at babak.ap@ufl.edu

Introduction

In this programming assignment, we implement the RDT 3.0, Reliable Data Transfer over a Lossy Channel with Bit Errors, which is explained in the textbook. In order to complete this assignment, you will need to fully understand Section 3.4. In addition, you should be able to design FSMs (finite state machines) for both the sender and receiver sides, based on the description in the textbook for the RDT 3.0.

In this assignment, we will develop and/or modify the sender and receiver of the RDT 3.0 in addition to a Network program, where a network with a lossy channel and bit errors is emulated. Some examples of them are retransmission (sending the same packets), sending correct ACKs. Unless otherwise stated, we use the same assumption and environments as the programming assignment #1. For example, socket is used for TCP/IP communication and the program should

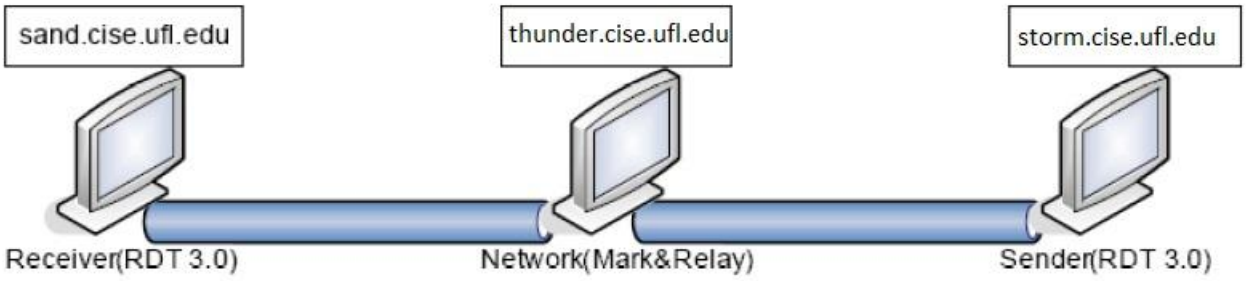

Fig. 1

Problem Outline

This assignment requires three programs running on different machines: a **Sender** program to send packets to the Network; a **Network** program to relay the packet from the sender to the receiver; and a **Receiver** program that receives a packet from the Network. Communications in the opposite direction should also be supported by these programs, i.e., when the Receiver sends back an ACK to the Network, the Network relays it to the Sender so that the Sender can receive the ACK from Receiver.

All three programs should support the RDT 3.0 functions described in the textbook.

Message Format

Before we get into the detail of each program, let's first take a look at the message format we're going to use.

Sequence No.	ID	Checksum	Packet Content…

The format is showing above. The first field (1 byte) is the sequence number of the packet, which is either 0 or 1. It's used for RDT 3.0. The second field (1 byte) is packet ID which represents the position of the packet in the whole message. It starts from 1. We will see this later. The third field (1 int = 4 bytes) is the checksum of the packet content. To simplify things, here we only add the ANSI value of each character in the content, instead of doing bit adding operation. And the last field (>= 1 byte) is the actual information in this packet.

Since we will need many packets to test the network and the program, what you should do is to read a message from a given file "message.txt" and break it into many words according to the blank space between them. For example, if the message is:

"You are my sunshine."

Then there should be 4 packets to contain it. The packets would look like:

0 1 x You

1 2 x are

0 3 x my

1 4 x sunshine.

Here x is a certain checksum number. We assume each message ends with a period '.' so if we find a period at the end of a packet, that means the whole message is complete.

The ACK format is simpler. It only has the sequence number field (1 byte), which takes the value 0 or 1, and the checksum value (1 byte). The checksum is initially set to 0 in ACK.

Network Program

In this assignment, the Network program is an emulator of a network. The main function of this program is to relay a packet and acknowledgement between a sender and a receiver. It's like a bridge between the two ends.

In order to emulate the lossy channel with bit errors, we let the Network choose an operation from the following three options: PASS, CORRUPT or DROP whenever it needs to relay a packet/ACK. In this implementation, the Network picks one of the three operations **RANDOMLY** with probability 0.5 for PASS, 0.25 for both CORRUPT and DROP. The reason we need to do this is that we want to test if the RDT 3.0 works correctly. Please see the Appendix for hints on generating random values.

If the chosen operation is PASS, the packet is forwarded to the receiver without any change. If CORRUPT, we add 1 to the checksum field. This is simpler than flip some random bits.

If the operation is DROP, we drop the packet or ACK which is to be relayed; instead, we send a DROP message to the **sender**. We can use ACK2 for DROP message, i.e., ACK with sequence number 2. The reason we need to do this is that we want to emulate a fake timeout. In a real RDT 3.0 system, if a packet or ACK is dropped, the sender will not receive any reply, so after a certain amount of time, it will timeout and resend the previous packet. However, we want to make thing a bit easier and avoid using timer function and interrupt handling. Thus, a fake timeout is used here, that is, when the sender receives a DROP message, it will consider it as a timeout.

When the Network gets a packet or ACK, it will print on the screen the packet type, ID (no ID for ACK) and the operation it picks. For example:

Received: Packet0, 5, DROP
Received: ACK1, PASS

Receiver Program

The receiver program utilizes the RDT 3.0 receiver side protocol by returning a proper ACK after receiving a packet. Please check out the homework 2 for detail. The important thing is that your receiver needs to cooperate with the RDT 3.0 sender and sends back ACK0 or ACK1 to handle PASS or CORRUPT packet.

When a packet arrives, the receiver will perform the checksum and examine the serial number and ID to see if this packet is wanted and not corrupted.

Every time the receiver got a packet, the screen should display: its current state, total number of received packets so far (including corrupted), print whole packet, and the proper ACK to be transmitted. For example:

Waiting 0, 10, 1 4 x sunshine., ACK1
Waiting 1, 3, 1 5 x gators, ACK1

If the packet containing the end of the whole message is received, the receiver should also display the message with blanks between each word. Like:

Message: You are my sunshine.

Sender Program

The sender program first reads the message file and converts it into many packets. Then it sends a packet to the Network and awaits a relayed ACK from the RDT 3.0 receiver. It should be able to cooperate with your RDT 3.0 receiver, meaning that it needs to be able to send a new packet or re-send the same packet according to different network situations such as PASS, CORRUPT or DROP. This is really determined by the ACK it received or by timeout.

Moreover, when it receives a DROP message from the Network, it realizes that this is the fake timeout and then resends the same packet.

Every time an ACK or DROP is received, the sender should display on screen: its current state, total number of packets sent so far (including failure), packet received, proper action. For example (Here Packet0 means the serial number 0, not the ID):

Waiting ACK0, 8, DROP, resend Packet0
Waiting ACK0, 15, ACK0, no more packets to send
Waiting ACK1, 120, ACK1, send Packet0

Running the System

The three programs are running at three different CISE machines. The port number can be the same as Project #1. The Network is started first and waits for 2 incoming connections. Network program is implemented with two sockets; one for the sender and the other for the receiver, as shown in Fig. 1. Then the sender and the receiver are started and both connected to the Network using URL and port number. So that means the Network program might need two threads to handle the two connections simultaneously.

Here is an example on how to run the system:

[C] ./network [portNumber]
[Java] java network [portNumber]

[C] ./receiver [URL] [portNumber]
[Java] java receiver [URL] [portNumber]

[C] ./sender [URL] [portNumber] [MessageFileName]
[Java] java sender [URL] [portNumber] [MessageFileName]

Important Note

• The URL, port number and file name should be command line input, do NOT hard code it.
• Print the required messages on all 3 machines according to the specified format.
• Test your program on CISE machines; make sure it's able to connect.

System Termination

After receiving the final correct ACK from the receiver, the sender will initiate the termination process. It will send a single byte -1 to the Network and exit. Upon receiving -1, the Network will also send -1 to receiver and exit. The receiver exits the last when it got -1 from Network.

Note on Run-away Processes for the Graceful Termination:

Your program should terminate gracefully. While testing your programs, run-away processes might exist. However, these run-away processes should be killed. Please check frequently if there are any remaining processes after termination. CISE department has a policy regarding this issue and your access to the department machines might be restricted if you do not clean these processes properly.

Some useful Linux/Unix commands:

To check your running processes: ps -u <your-username>
To kill a process: kill -9 pid
To kill all Java processes: killall java
To check processes on remote hosts: ssh <host-name> ps -u <your-username>
To clean Java: ssh <host-name> killall java

Report

Submitted report file should be named report.pdf or .txt and include the following:

• Your personal information: Full name, UF ID, and Email
• How to compile and run your code under which environment.
• Description of your code structure.
• Show some of the execution results.
  - Discuss the results you got with your program and include screenshots.
• Explain about the bugs, missing items and limitations if there is any.
  - Honesty is valued here.

Submission Guidelines:

1. The source code files should be named "sender.c/java"," receiver.c/java" and "network.c/java".
2. Only submit your C/Java source code files and report, do not include .obj/.class or executable.
3. Include "makefile" if you have one. (Not required)

4. Include the report in .pdf or .txt format. Name it "report.pdf/txt".
5. Zip all your files into a packet: Firstname_Lastname_ID.zip
6. Upload the zip packet as attachment to SAKAI before deadline.

Grading Criteria:

Correct Implementation / Outputs	50%
Graceful Termination / Exception handling	20%
Report / Code Style	30%
Total:	100%

Appendix: Generating Random Number According to Some Probability

To get a random event with certain probability, we can generate a uniformly distributed random number between (0, 1) and check if the number is within the probability value. For example, if the number we get is less than 0.5, we choose PASS. If it's between 0.5 and 0.75, we choose CORRUPT and if it's larger than 0.75, we pick DROP.

The following are some simple examples to show how to generate random values.

```
import java.util.*;
public class RandomJava {
	Random r;
	RandomJava() { r = new Random(); }
	public double getRandomValue() { return r.nextDouble(); }

	public static void main(String[] args)
	{
		RandomJava rj = new RandomJava();  //test random variables
		for (int i=0;i<10;i++ ) {
			System.out.println("random: "+rj.getRandomValue());
		}
	}
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
	int seed = (int) time(NULL);
	srand(seed); //Since this, you can call rand() for random values
	int i=0;
	for (i=0;i<10;i++)
	{
		int r = rand()%100; //rand will generate random integer
		printf("random: %d \n", r);
	}
	return 0;
}
```