UNIVERSITY OF DELAWARE
DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

CISC 450 / CPEG 419: Computer Networks I

Fall Semester, 2018 Professor: Adarsh Sethi

Programming Project 2
Due Date: 11 pm on Wednesday November 28, 2018

This project is a group assignment; you should work on it with your group partner.
Collaboration with or help from anyone except your group partner is not permitted.
Only one submission per group is required.

Important note: Before you start this project, read the accompanying document *Project 2 Hints,* which contains important and useful information about this project.

In this project, you will implement the Stop-and-Wait protocol to transmit data from a Sender to a Receiver in the presence of errors and packet loss. The protocol will be unidirectional in which data is sent in one direction only with acknowledgments being sent in the reverse direction. Only positive ACKs are used. The transmission of packets and ACKs will be done over UDP (that represents an unreliable network layer channel) using fixed UDP ports.

To implement this protocol, you will write two separate programs called *Sender* and *Receiver* which represent the actions to be executed by the sending node and the receiving node respectively. Both the Sender and the Receiver must run on the host *cis450.cis.udel.edu*, as you will have to come in to my office to give me a demo of your programs. For convenience, both the hostname and the server port number may be hardcoded into both the Sender and Receiver, but this should be done in such a way that they should be easy to change.

Both the Sender and Receiver will require the user to enter some configuration parameters (see later section for details). These parameters should be provided as command-line arguments.

General Actions and Packet Format

The Sender constructs packets to send to the Receiver by reading one line at a time from an input file named *input.txt*. The Receiver receives packets and puts their data into distinct lines in an output file named *output.txt*. You must make sure that duplicate or out-of-sequence packets are discarded and do not get their data stored in the output file. When the file transfer is complete, both the Sender and the Receiver terminate execution.

Each line in the input file contains a sequence of printable characters (no control characters, etc.), with no more than 80 characters on a line. The contents of each line should be transmitted as a separate packet. The "newline" character read from the file at the end of each line is also transmitted in the packet and is included within the limit of 80 characters per line. Note that the "newline" character is different from the C "null" character; the "null" character should *not* be transmitted in any of the packets.

The format of a data packet is shown in Figure 1 below. Each data packet contains a 4-byte long header followed by a number of data characters. The header contains 2 fields, each of length 16 bits (2 bytes) as shown in the figure. You must convert the values in these fields into the network byte order when they are transmitted, and convert them back to host byte order when they are received.

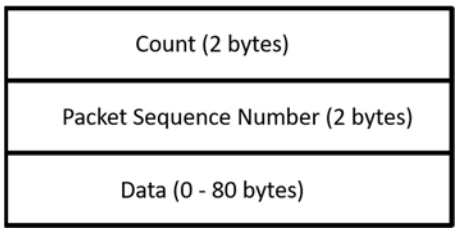

Figure 1: Format of a data packet

The first field of the header is a count of the number of data characters in the packet. This value must be in the range 0 through 80. If the count is 0, then there are no data characters in the packet.

The second field of the header is called the packet sequence number. Each packet transmitted by the Sender is assigned a sequence number that alternates between 0 and 1, in accordance with the Stop-and-Wait protocol.

After sending each data packet, the Sender waits for an ACK to be returned from the Receiver. The format of an ACK packet is shown in Figure 2 below, where the ACK sequence number is either 0 or 1.

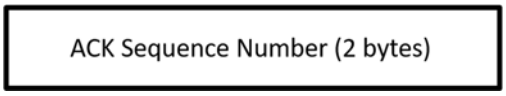

Figure 2: Format of an ACK packet

When the Sender is finished transmitting all the lines in the data file, *and has received ACKs for all of them*, it will send a special last packet signifying "End of Transmission". This packet will have a Count of 0 and no data characters. It will have a Sequence Number that is the next sequence number that would have been used if this were a valid data packet. It is important that this packet be transmitted only *after* the Sender has received ACKs for all transmitted data packets. The Receiver will not transmit an ACK for the EOT packet, and the Sender will not expect any ACK to be returned for it. The Sender program can terminate once this packet has been transmitted. When the Receiver receives the EOT packet, it closes the output file and also terminates.

The Project 2 Hints document that accompanies this project describes in detail how to implement the Stop-and-Wait Protocol for the purposes of this project. Although your packets and ACKs are transmitted over UDP which provides unreliable data transfer, it is not common to observe packet loss when transmitting to the same host (which is the case with us). For this reason, we will explicitly simulate packet and ACK loss by implementing the two functions described below. Both of these functions are implemented in the Receiver.

Functions *SimulateLoss* and *SimulateACKLoss*:

In the Receiver program, the function *SimulateLoss* simulates loss in the reception of data packets from the Sender. Errors and loss are not distinguished from each other, as they both result in the packet being dropped. This function uses the configuration parameter *Packet Loss Rate* which is read when the Receiver starts (see below).

Below are the actions of this function:

- Generate a uniformly distributed random number between 0 and 1.
- If the random number is less than *Packet Loss Rate*, then the packet is dropped. Return the value 0.
- Otherwise, the packet is not dropped. Return the value 1.

The function *SimulateACKLoss* simulates loss for ACKs using the configuration parameter *ACK Loss Rate* as described below. This function is also implemented in the Receiver and it causes the Receiver to not transmit an ACK when a loss is indicated.

- Generate a uniformly distributed random number between 0 and 1.
- If the random number is less than *ACK Loss Rate*, then the ACK is dropped. Return the value 0.
- Otherwise, the ACK is not dropped. Return the value 1.

Sender and Receiver Configuration Parameters

When the Receiver starts, it is given the following configuration parameters as command-line arguments:

- *Packet Loss Rate:* A real number between 0 and 1.
- *ACK Loss Rate:* A real number between 0 and 1.

When the Sender starts, it is given the following configuration parameter as a command-line argument:

- *Timeout:* The user enters an integer value *n* in the range 1-10, and the timeout value is then stored as 10^n microseconds. Note that the resultant timeout value should be stored with both seconds and microseconds components.

It is not acceptable to hard code values for the above configuration parameters in your code.

Output of your program

At specific places in both your Sender and Receiver programs, you must print out specific messages. The symbol "*n*" below refers to the sequence number of the transmitted or received packet **(note that the sequence number must always be either 0 or 1)**, and the symbol "*c*" below refers to the count (number of data bytes) in the transmitted or received packet.

The messages to be printed by the Sender are:

When a new data packet numbered *n* is sent by the Sender:

Packet n transmitted with c data bytes

When a data packet numbered *n* is retransmitted by the Sender:

Packet n retransmitted with c data bytes

When a timeout expires:

Timeout expired for packet numbered n

When an ACK is received with number *n*:

ACK n received

When the "End of Transmission" packet is sent:

End of Transmission Packet with sequence number n transmitted with c data bytes

The messages to be printed by the Receiver are:

When a data packet numbered *n* is received correctly by the Receiver for the first time:

Packet n received with c data bytes

When a data packet numbered *n* is received correctly by the Receiver, but is a duplicate packet:

Duplicate packet n received with c data bytes

When a data packet numbered *n* is received by the Receiver, but the *SimulateLoss* function causes it to be dropped:

Packet n lost

When an ACK is generated with number *n*, but is dropped by the function *SimulateACKLoss*:

ACK n lost

When an ACK is generated with number *n* and is transmitted without loss:

ACK n transmitted

When the "End of Transmission" packet is received:

End of Transmission Packet with sequence number n received with c data bytes

At the end, before terminating execution, the following statistics should be printed. Do not include the last special "End of Transmission" packet in the count of data packets in these statistics.

For Sender:

Number of data packets transmitted (initial transmission only)

Total number of data bytes transmitted (this should be the sum of the count fields of all transmitted packets when transmitted for the first time only)

Total number of retransmissions

Total number of data packets transmitted (initial transmissions plus retransmissions)

Number of ACKs received

Count of how many times timeout expired

For Receiver:

Number of data packets received successfully (without loss, without duplicates)

Total number of data bytes received which are delivered to user (this should be the sum of the count fields of all packets received successfully without loss without duplicates)

Total number of duplicate data packets received (without loss)

Number of data packets received but dropped due to loss

Total number of data packets received (including those that were successful, those lost, and duplicates)

Number of ACKs transmitted without loss

Number of ACKs generated but dropped due to loss
Total number of ACKs generated (with and without loss)

Testing

The files *test1.txt* and *test2.txt* in the directory */usa/sethi/networks/proj2* on *cis450.cis.udel.edu* are sample input files that may be used by you to test your programs. It is strongly suggested that you first use *test1.txt* for all your testing, and only if you have thoroughly debugged your programs, then proceed with using *test2.txt* for further testing. Copy the input file that you wish to use to the name *input.txt*, so the Sender can read it from there.

It is also suggested that you test your programs in phases using the following configuration parameter values:

- Packet and ACK loss rates 0, Timeout value $n = 5$.
- Packet loss rate 0.2, ACK loss rate 0, Timeout value $n = 5$.
- Packet loss rate 0, ACK loss rate 0.2, Timeout value $n = 5$.
- Packet and ACK loss rates 0, Timeout value $n = 3$.

Once you have tested and debugged with the above parameter values, then you should try other combinations of the various parameters. In particular, feel free to experiment with larger or smaller timeout values if the above values don't appear to work well for you. Make sure your programs work well under all conditions, because we will test them out under a variety of different conditions.

Submission

Create two scripts, one each for the Sender and the Receiver, and show the transfer of the file *test2.txt* with the first set of configuration parameters listed above (i.e., no packet loss, no ACK loss). The scripts should contain a long listing of the directory that contains your files (using *ls -l*), should show them being compiled, then another long listing (using *ls -l*) of the directory after compilation is complete, a listing of the input file (for the Sender), show the execution of the programs including the program outputs, and finally, a listing of the output data file (for the Receiver). Finally, do a *diff* on the input and output files.

Submit a zipped copy of your project directory. This directory should include all your original source files, the executables, the input and output data files, and the scripts generated by you for your test run.

Demo: You will be asked to schedule a time after your submission to come in and give a demo of your programs. During the demo, we will run your programs under a variety of different loss and timeout conditions. Both group members must be present for the demo.

Grading

Your programs will be graded on correctness, proper output, readability, and documentation. The grade distribution is as follows:

Correctness: 60 %
Proper Output and Testing: 20 %

Readability and Documentation: 20 %

Part of the Correctness grade will depend on how your program performs during the demo. As with Project 1, points for documentation will not be awarded lightly; we will be looking for meaningful variable and function names, good use of comments, good modular structure with appropriate use of functions, good programming style, and proper indentation.

Deadline for submission: Wednesday November 28, 11 pm.
No late assignments will be accepted, because we will have to schedule times for demos.