Gregory Friedman

5511-4893

[friedmangreg37@ufl.edu](mailto:friedmangreg37@ufl.edu)

How to compile and run code:

To compile the code, go to the folder with the 3 .java files and the text file, then type:

javac \*.java

To run the program, you will need 3 terminal windows: one for each program

In the first window, type:

java Network [portNumber]

For example:

java Network 4893

Then, in the second window, type:

java Receiver [url] [portNumber]

For example:

java Receiver localhost 4893

Finally, in the third window, type:

java Sender [url] [portNumber] [filename]

For example,

java Sender localhost 4893 message.txt

I tested the code on the CISE machines, so this method should work under that environment.

Code Structure:

Sender.java

After setting up some variables and getting the necessary information from the command line, the sender program creates a socket to connect to the network and then waits for a response to say that the connection has been established. Then it reads the input file and splits it into an array of the words in the file, separated by spaces. It then enters a loop where it breaks each of these words into a packet: giving it a sequence number, an ID number and calculating its checksum. Then it puts the actual word into the packet, one byte at a time. Once it sees a period at the end of a word, it knows that the message should be done (since we’re assuming each message ends with a period), so it exits the loop. Or, if no period is ever found, it will just exit the loop once all of the words of the message have been made into packets, but this should never happen.

Once all the packets are made, the sender will begin to send them to the network. For each packet in the array of packets, it sends it to the network and then waits for a response (the ACK). When it receives the ACK, it checks the sequence number of the ACK. If it is 2, it knows the packet was “dropped,” so it does not move to the next packet in the array, but instead stays at the same position so that the next iteration of the loop will resend the same packet. If it receives an unexpected sequence number, it knows the packet was corrupted, so it will do the same thing and resend the packet on the next iteration of the loop. If it receives the expected sequence number, it will move to the next packet and also alternate the sequence number. If this is the last packet in the array of packets, it informs the user we are done and then breaks out of the loop, at which point it closes the socket and terminates the program.

Network

The network program creates two sockets: one for the sender and one for the receiver. It connects to the sender socket with the port number given by the user at the command line, and then connects to the receiver with one more than the port number given by the user. It connects to each of these sockets and sends a message to each of them to notify of this, then enters the main loop of the program.

In the main loop, the network waits for a packet from the sender. Once received (in the form of a string), it splits the packet into individual bytes. Then it gets a random value to decide if it should pass the packet (50%), corrupt the packet (25%), or drop the packet (25%). If it decides to drop the packet, it will send an ACK2 to the sender. If it corrupts the packet, it adds 1 to the packet’s checksum field then passes it along to the receiver. If it passes the packet, it passes the packet to the receiver without changing anything. Then it waits for the ACK from the receiver.

When it receives the ACK from the receiver, it passes it along to the sender. If the receiver told the network to terminate, it will close the sockets and exit the program.

Receiver

The receiver program connects to the network at a port number one greater than the one given by the user, so it doesn’t conflict with the connection between the sender and the network. Then it enters the main loop of the program.

In the main loop, the receiver waits for a packet from the network, in the form of a string. Then it splits the packet into individual bytes. It will then create a byte array for the ACK to be sent back, which has a sequence number to match that of the packet received and a checksum of 0. Next, the receiver gets the checksum from the packet received and also calculates what the checksum should be based on the message received. If they are not the same, it knows the packet was corrupted, so it inverts the sequence number of the ACK, to send the wrong one back to the sender so it will resend this packet. Otherwise, the receiver adds this packet to the total message string received so far and moves to the next state in the FSM. It then sends the ACK back to the network, and if it is at the end of the message (sees a period at the end of it), it prints the whole message, tells the network it is done, closes the socket, and ends the program.

Limitations:

No random on ACKs  
end connection the wrong way

Checksum and negatives

ID of 10 and 13