Project 2: UDP File Transfer

Authors: Tom Cardy & Mario Delprete

Class: CISC-450: Computer Networks

Date: 5/1/2017

Summary of Programs

Implement a Transport Layer protocol to transmit data from a Sender to a Receiver in the presence of errors and packet loss. The protocol to be implemented by you is the Stop-and-Wait protocol. 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.

Purpose of the Receiver

The purpose of the receiver is to maintain a loop that will listen to connections and packets being sent to it. Once a connection is established and the packets come in and the sequence number has been checked, the receiver begins to run a function that will intentionally cause the packet received to be lost, forcing the sender to retransmit the packet. Once a packet has been received and passes the lost packet test, the receiver will begin to write the file. Then the receiver constructs a packet with a 2-byte header that will contain an ACK. Before it is sent, it is ran through the simulate loss function wh­­­­ere it will try to lose the ACK. If the ACK is lost, the sender’s timer will trigger and resend the packet. This process will repeat until the last packet received will have a count of 0 in its­­ count field in the header. The step by step for this process is as follows:

* The Receiver loop begins
* Wait for packet to arrive from the Sender
* If the packet is received, check the field count
  + If the field count is set to 0, this is an EOT packet, so quit the loop and end program
* If the packet is received, but the field count is not 0, this is a regular data packet
  + Run Simulate Packet Loss
* If Simulate Packet Loss returns 0
  + The packet is lost and begin at first bullet again.
* If Simulate Packet Loss returns 1
  + The packet is correct and will proceed to check sequence number
* If Sequence number is not equal to expected
  + Don’t deliver it, and generate correct ack
* If Sequence number is equal to expected
  + Begin to file write and generate correct ack
* Before Ack is sent, Simulate Packet Loss
  + If Ack is loss, Let the sender timeout
  + If Ack is fine, send Ack

Purpose of the Sender

Once the sender has connected to the receiver, the purpose of the sender is to read an input file and send the receiver a series of packets with the text read from the input file. The receiver stores the text in the packets in a text file. As stated, the sender constructs these packets by reading one line at a time and placing it in a character array. The sender also places a header on the packet, which consists of 4-byte long header, which contains fields for the count of the number of bytes, and the sequence number of the packet for the receiver to verify if it is the correct packet. When the file has no more text to be copied, a last packet is created with a count of 0, signifying the receiver that the file transfer is completed. The client then breaks connection with the receiver and exits. The sender does this in the following order:

* Start an outer loop
* Parse a line from an input file into an array
* Make a packet
  + A header is placed on the file as well
* Transmit it
* Start a timer
  + Begin an inner loop to wait for an ACK
  + Check if an ACK has been received
    - If correct, quit inner loop
    - If wrong, resend and restart timer
    - If no Ack is received and timer is triggered, resend and restart
* Quit outer loop when file transfer is completed

Receiver Functions

Our Port Number: 5045

1. Void init\_variables()
   1. Initializes number of Packets and total Bytes variable to 0
2. Void open\_file()
   1. Opens the file that will store characters
3. Void close\_file()
   1. Close the file that was storing characters
4. Void write\_to\_file()
   1. Writes the character array to the text file
5. Int simulate\_packet\_loss(double rate)
   1. Generates random number between 0 and 1 and compares it to loss rate
      1. If random number is lower than rate, cause a loss. Continue elsewise.
6. Void calculate\_duration()
   1. Takes the difference between the time currently and the time when the receiver first received a packet
7. Void print\_report()
   1. Prints all the variables asked at the end of the program

Sender Functions

1. Void sigalrm\_handler(int sig)
   1. Sets timeExpired to 1
   2. Adds 1 to the timeout counter
2. Void initialize\_variables()
   1. Initializes all the variables to 0
3. Int open\_file(char filename[])
   1. Checks and Opens a file
4. Int close\_file()
   1. Closes the file created in open\_file
5. Int wait\_for\_call\_from\_above(char buffer[])
   1. Returns an Int that is the number of bytes read
6. Int read\_ack(char ack\_packet[])
   1. Checks to see if the ack returned is the correct ack
      1. Compares it to the expected sequence number
      2. Adds to the number of ACKs recieved
7. Void recv\_ack()
   1. Receives an ack
   2. Passes ack to read\_ack
8. Int send\_packet(char packet[])
   1. Sends the Packet
   2. Adds to the counter of the number of bytes sent
9. Int resend\_packet(char packet[])
   1. Resends the last packet
   2. Adds to the counter to the number of resent packets
10. Int make\_packet(char buffer[])
    1. Makes a packet
    2. Places a header on the packet for the receiver to read
11. Void send\_eot()
    1. Sends a specific packet with 0 databytes
    2. Places a special header with 0 as its Countfield
12. Void set\_timer()
    1. Begins the timer
    2. Sets the timeExpired to 0
13. Void stop\_timer()
    1. Stops the timer
    2. Sets the timeExpired to 0
14. Void determine\_timeout\_values()
    1. Calculates the amount of time to pass for a timeout
15. Void wait\_for\_ack()
    1. Loop that waits for an ack or timeout
16. Void next\_iteration()
    1. Resets variables to 0 except tracking variables
    2. Adds 1 to number of packets created
17. Void print\_report()
    1. Prints out all the requested variables
       1. Number of Packets
       2. Total bytes Sent
       3. Number of Resent Packets
       4. Number of Acks
       5. Number of Timeouts
       6. Time to complete the file transfer
18. Void calculate\_duration()
    1. Calculate the duration it took to send the packets

Walkthrough of the Program (Sender Side)

The sender is prompted to enter an appropriate timeout exponent. This number can range between one and ten. It then begins to calculate timeout values. The file then proceeds to open the text file. Right after that, the sender takes the current time. Then the sender starts a do while loop that will read the file, make a packet, send a packet, set a timer, wait for an ack. This do while repeats until the text file has no more data bytes to be sent. If this loop breaks, begin to send the eot packet, signaling the end of transmission. At this point, the sender will take the current time and calculate the duration based on the original time taken. The Sender then finishes by printing out all the variables tracked and closes the file and the program. There are a couple of issues that can occur. One issue is that the packet is lost. If the packet was lost, the Sender can resend the packet when the timeout occurs. If the Ack packet gets lost on the receiver side, the Sender will resend the packet. The Receiver will recognize that this packet is a duplicate and send an Ack back to the sender so the sender can continue.

Walkthrough of the Program (Receiver Side)

The receiver first seeds a random number. This is required or else the simulate loss function will constantly generate the same random number. The user is then prompted to initialize the Packet and Ack loss rate. After that, the receiver proceeds to open a socket, initialize the receiver, and binds the socket to a port. After, the receiver opens an empty text file. The receiver then begins a loop, starting with waiting for an incoming message on that port. A message comes in and the receiver begins to parse the header details. The receiver first looks at the count field. If the count field is equal to 0, it will start to end the file transfer. If not, the next comparison the receiver will do is to see if the sequence number on the header matches the expected sequence number. If the sequence number matches, the receiver will continue by simulating a packet loss. If the packet loss function returns a 1, the packet was not loss and the file writing can begin, else the packet was “lost” and will not send an ACK, causing the sender to timeout and send it again. If the packet was received and written to the file, the receiver must make an ACK packet to send back. Before this is done, the simulate loss function will trigger an ACK loss, causing the sender to resend upon timeout. The packet that is resent is already written in the file, but the sequence number is not the number that is being expected, so the receiver will recognize that this packet is a duplicate, and will proceed by discarding that packet and sending an ACK back for it. This entire process will continue until the packet with a 0 in its header count field is seen. Once the last packet is in, the file will close and the receiver will terminate.