CSE3461/5461 Lab2: Reliable Data Transfer 3.0 over UDP

Due Date for Electronic submission: Tue March 21, 2017 (23:59:59 PM)

1 Goal

The purpose is to use UDP Socket and C/C++ programming language to implement a reliable data transfer protocol.

2 Instructions

1. In this project, you will be implementing the data tranfer using UDP and reliable data transfer protocol (RDT 3.0) described in the textbook. You must write one program implementing the sender side, and another program implementing the receiver side. Only C/C++ are allowed to use in this project.
2. The receiver program will also act as a client application, requesting a file from the sender which acts as a server.
3. Task 1: UDP. The two programs will communicate using the User Datagram Protocol (UDP), which does not guarantee reliable data delivery.
   - The receiver program will take the hostname and port number of the sender, and the name of a file it wants to retrieve from the sender as a command line arguments. For example:

     to run the sender type: $shell > sender < portnumber >$
     to run the receiver type: $shell > receiver < sender\_hostname >< sender\_portnumber >< filename >$
   - The receiver will first send a message to the sender which includes the name of the file requested. If the file exists, the sender will send this file to the receiver via UDP. It will divide the entire file into multiple packets (the maximum packet payload size is 1K bytes), and then add some header information to each packet before sending them to the receiver. It is up to you what information you want to include in the header (e.g. Source, Destination port etc...), but you will at least need a sequence number field and the total sequence number. The client needs to use the sequence number to the restore the whole file. For example, the file has 10K bytes, it is divided into 10 packets. For each packet, you can add the two fields: the sequenence number of the current packet (0, 1, 2, ·, 9) and the maximum sequence number is 9. When the sequence number is equal to the maximum one, the file transfer completes.
   - You can create a large file being requested on your own, and this file should not be smaller than 5K bytes.
   - Test on your own. (1) Sender sends one file to Receiver (no loss and corruption probability); (2) Check file integrity: diff [send_file] [recv_file]. Both files should be identical.
4. Task 2: UDP over a lossy channel. Test your above code in case of packet loss.
   - Although using UDP does not ensure reliability of data transfer, the actual rate of packet loss or corruption in LAN may be too low to test your program. Therefore you should simulate packet loss in the following manner:
     - Packet loss: With probability P_l ignore arriving packets (pretend not receiving the arriving packets).
     - P_l ranges between 0 and an appropriate value (say, 0.40).
   - You can enable packet loss at the sender side, or the receiver side, or the both. That is, on the sender side, it will not send this packet at the probability of P_l. On the receiver side, it will drop this packet at the probability of P_l even when it receives this packet.
   - You should print messages to the screen when the server or the client is sending or receiving packets. Your message should include information such as whether it is a DATA packet and its sequence number, whether it is lost etc. Such messages will be helpful for you to debug the programs, and we can use them to examine the correctness of your programs. Finally, if the receiver should display how many packets

have been received out of the toal packet numbers. For example, " Receive 9 out of 10 packets. Fail" or "Receive 10 out of 10 packets. Succees".

- Test on your own. You should test with different probabilities (e.g, 0.1 and 0.4) and see how the receiption works. For example, to run the sender side: $> sender < portnumber > P_l$

  and to run the receiver side: $> receiver < sender\_hostname >< sender\_portnumber >< filename > P_l$

5. Task 3: RDT3.0 over a lossy channel. Now you develop your RDT3.0 protocol as specified in the textbook over the above lossy channel. You

   - You need to implement ACK, Seq number and timer. The sender will send the next packet upon receiving an ACK and will retransmit the packet upon timeout a corrupted ACK (not applicable here. We do not enable the errorneous channel). Note both data packets traveling from sender to receiver and acknowledgement packets traveling from receiver to sender may be lost.
   - Note that you need to add some header information to each packet before sending them to the receiver. You are free to define what kind of messages you will require, and the format of the messages. But the sequence number and the maximum sequence number is mandatory. The header can be the same as Task 1.
   - You need to display the packet information if it is retransmitted.
   - Finally, you need to display whether the file is successfully received and how many retransmission are used, as well as other important information.
   - Test on your own. For example, to run the sender side: $> sender < portnumber > P_l$ RDT
     and to run the receiver side: $> receiver < sender\_hostname >< sender\_portnumber >< filename > P_l$ RDT

6. Note that your programs will act as both a network application (file transfer program) as well as a reliable transport layer protocol built over the unreliable UDP transport layer.

3 Hints

The best way to approach this project is in incremental steps. Do not try to implement all of the functionality at once.

Fisrt, enable file transfer over UDP. Assume there is no packet loss or corruption. Just have the sender send a packet, and the receiver integrate them into one file.

Second, introduce pacekt loss and see how file transfer becomes unreliable.

Third, start to work on RDT3.0. To learn whether a packet is received, introduce ACK. The receiver responds with an ACK upon receiving a packet. Upon failure, you must implement some retransmission functionality. Now you have to add a timer at the sender side for each packet.

You should print messages to the screen when the server or the client is sending or receiving packets. Your message should include information such as whether it is a DATA packet or an ACK, the sequence number, whether it is lost etc. Such messages will be helpful for you to debug the programs, and we can use them to examine the correctness of your programs.

You should start this project early. We have three tasks and you should complete at least one per week.

The credit of your project is distributed among the required functions. If you only finish part of the requirements, we still give you partial credit. So please do the project incrementally.

4 Materials to turn in

1. Source codes (can be multiple files), as well as a makefile.

   Note: The grader will only type "make" to compile your code.
2. A report of your project (pdf or word format) should not exceed 3 pages (except from source codes). The report will contain:

- Your name and ID at the very beginning
- Include a brief manual in the report to explain how to compile and run your source code (if the grader can't compile and run your source code by reading your manual, the project is considered not to be finished). A makefile is highly recommended (by default).
- Give a high-level description of your server's design, especially highlighting what are unique in this project.
- Implementation description (header format, messages, timeouts, window-based protocol etc).
- Difficulties that you faced and how you solved them.
- Any other issues.

You do not have to write a lot of details about the code, but just adequately comment your source code.

To submit,

1. Put all your files into one directory.
2. Zip or tar all the files in this directory and get one submission file, which is named "lab2_name_SID.zip", where name and SID are your name and student ID.
3. Submit this zipped file to Carmen.

5 Bonus Points: In-Class Demo (Optional)

1-2 student(s) are strongly encouraged to demonstrate your RDT3.0 in class. Please sign it up with me in advance when you feel ready. You will be awarded with the full in-class participation points and even bonus points (for this class), depending on your presentation and project quality. Here is what you need to do.

- You need to bring your own laptop for in-class demo.
- You need to prepare 3 slides to present. 1 slide for design and implementation, 1 slide for experiences you gain, 1 slide for lesson learn from project and suggestion to project.
- Demo: you will demo the functions you have implemented step by step.
- Questions may be asked during demo or you may be requested to test your codes using different parameters.
- The whole demo and presentation will last for about 10 minutes.