COM SCI 118 Spring 2018

Project 2: Simple Window-Based Reliable Data Transfer

1 Goal

The purpose of this project 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 a simple congestion window-based TCP protocol. You must write one program implementing the client side, and another program implementing the server side. Only C/C++ are allowed to use in this project as you did in the first project. You are not allowed to use any third-party libraries other than the standard libraries provided by C/C++.
2. The client and the server will communicate using the User Datagram Protocol (UDP) socket, which does not guarantee data delivery.
3. Here are commands for the server and the client.

   ```
   ./server [hostname] [port]
   ./client [filename]
   ```

   Two arguments are given to run the server program. The `hostname` is an IP address for the server and the `port` is the port number of the server. The client uses a single argument, which is the `filename` to request it to the server. Some parameters for the TCP congestion control such as maximum sequence number, time-out value, and slow start threshold value have to be encoded with default values.
4. The client should establish a connection to the server by using three-way handshake. During the handshake, the server has to send its initial sequence number to the client.
5. After establishing the connection, The client will first send a file request to the server. If the file exists, the server will divide the entire file into multiple packets, and then add some header information to each packet before sending them to the client. The maximum packet size is 1024 bytes including all the headers and data.
6. You have to design your own TCP header for this project. It is up to you what information you want to include in the header (e.g. sequence number, acknowledgment number etc.), but you will at least need a sequence number field. You are free to define what kind of messages you will require, and the format of the messages.
7. 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 Error Handling

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, we are going to emulate packet loss by using the `tc` command in linux.

- The file transmission should be completed successfully, unless the packet loss rate is 100 %.
- The timer on both the client and the server should work correctly to retransmission the lost packet.

- You can use the following commands to adjust parameters of the emulation:
- To check the current parameters for the private network (`eth1`):

  ```
  tc qdisc show dev eth1
  ```

- To change current parameters to loss rate of 20% and delay 100ms:

  ```
  tc qdisc change dev eth1 root netem loss 20% delay 100ms
  ```

- To delete the network emulation:

  ```
  tc qdisc del dev eth1 root
  ```

- If network emulation hasn't yet setup or you have deleted it, you can add it to, e.g., 10% loss without delay emulation:

  ```
  tc qdisc add dev eth1 root netem loss 10%
  ```

- You can also change network emulation to re-order packets. The command below makes 4 out of every 5 packets (1-4, 6-9, ...) to be delayed by 100 ms, while every 5th packet (5, 10, 15, ...) will be sent immediately:

  ```
  tc qdisc change dev eth1 root netem reorder 100% gap 5 delay 100ms
  ```

- or if you're just adding the rule

  ```
  tc qdisc add dev eth1 root netem reorder 100% gap 5 delay 100ms
  ```

- More examples can be found in [Network Emulation tutorial] (http://www.linuxfoundation.org/collaborate/workgroups/networking/netem)

4 Hints

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

- First, assume there is no packet loss. Just have the server send a packet, the client respond with an ACK, and so on.
- Second, introduce a large file transmission. This means you must divide the file into multiple packets and transmit the packets based on the current window size.
- Third, introduce packet loss. Now you have to add a timer on each sent and unAcked packet. If a timer times out, the corresponding (lost) packet should be retransmitted for the successful file transmission.

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.

5 Requirements

- You must use an UDP socket for both server and client.
- You should print messages to the screen when the server or the client is sending or receiving packets. There are four types of output messages and should follow the formats below. The output should follow the *exact* format as specified here; no additional debug output should be printed. If any SYN/FIN packet is retransmitted, you should also print "Retransmission" flag in the message.
  - Server: Sending packets

    "Sending packet" [Sequence number] [cwnd] ("Retransmission") ("SYN") ("FIN")

    Example:

    Sending packet 1999 5120 SYN

Sending packet 2000 5120
Sending packet 3024 5120
Sending packet 4048 5120
Sending packet 5072 5120 Retransmission
Sending packet 6096 5120 FIN

- Server: Receiving packets

  "Receiving packet" [ACK number]

  Example:

  Receiving packet 5096
  Receiving packet 6120

- Client: Sending packets

  "Sending packet" [Sequence number] ("Retransmission") ("SYN") ("FIN")

  Example:

  Sending packet 5095 SYN
  Sending packet 5096
  Sending packet 6120
  Sending packet 7144
  Sending packet 6120 Retransmission
  Sending packet 8168 FIN

- Client: Receiving packets

  "Receiving packet" [ACK number]

  Example:

  Receiving packet 2000
  Receiving packet 3024

Such messages will be helpful for you to debug the programs, and we can use them to examine the correctness of your programs.

- The maximum packet size is 1024 bytes including a header.
- The window size must be in **the unit of bytes**. (Not in the unit of packet count) For example, if the window size is 5120 bytes, then five packets can be transmitted simultaneously.
- The sequence number is given in the unit of bytes as well. The maximum sequence number should correspond to 30 KB (30720 bytes). You have to reset back the sequence number when it reaches the maximum value.
- Packet retransmission should be triggered when he timer times out.
- Here are the default values for some variables.
  - Maximum packet length (including all your headers): 1024 bytes
  - Maximum sequence number: 30720 bytes
  - Window size: 5120 byte
  - Retransmission time out value: 500 ms
- The client should save file to received.data in the current working directory.
- After server finishes transmission, it should terminate the connection using FIN/FIN-ACK procedure. The client should implement TIME-WAIT mechanism, e.g., using 2 × RTO as a waiting time.

6 Congestion Control: Extra Credit

If you implement TCP congestion control features then you can get up to 15% extra credit of this project. Here are requirements for the congestion control.

- Create separate execution files for both server and client to show congestion control.
- Set initial slow start threshold as 15360 bytes.
- Set initial (congestion) window size as 1024 bytes.
- You should print slow start threshold when the server sends data packets.
  (e.g. *Sending packet 5095 5120* ***15360***)
- Following congestion control features will be checked for grading
  - Slow start
  - Handle packet loss (cwnd and ssthresh)
  - Fast retransmission and fast recovery
  - Congestion avoidance

It your responsibility to clearly demonstrate how your congestion control worked (in terms of cwnd growth, retransmission etc.).

7 Due Date and Demo

1. You are required to demo your program on 6/13 and 6/15 (tentative schedule).
2. Submit an electronic copy of the project on CCLE on **11:55 p.m.** 6/8.
3. Late submission is not allowed!

7.1 Demo

Demo should be done on an up-to-date Ubuntu or Linux system. You then use make to compile your programs, run your programs to deliver a test file from the server side to the client side. It is required by your program to print out the operations, which explain the delivery process on the screen. We may ask you to use different values for tc command to test your programs. We will ask you to compare the received test files with the original one using the tool diff.

Demo Procedure:

1. Sign up for Demo: TA will distribute the demo signup sheet in the discussion section of the 8th week.
2. In Demo: You need to prepare 3 slides to present. One slide for design and implementation, one slide for experiences you gain, one slide for lesson learn from project and suggestion to project.
   - TA will ask you to demo the function step by step.
   - TA will also ask you questions during demo, you need to answer the question clearly. All question will be related to your project implementation.
3. Demo time is 15 minutes.

8 Project Submission

1. Put all your files into a directory called `project2_UID1_UID2`, where UIDs are your UCLA ID numbers. Compress the directory and submit the file `project2_UID1_UID2.tar` to CCLE course website.

2. The `project2_UID1_UID2.tar` should contain the following files

    - Source codes (can be multiple files)
    - A PDF format report file (.pdf) no more than 3 pages. The report will contain:
        - Student names and Student IDs at the very beginning (**2 students per-group**).
        - Implementation description (header format, messages, timeouts, window-based protocol etc).
        - Difficulties that you faced and how you solved them.
    - Makefile
    - README file that includes your group information. The README will contain:
        - Group number, students name, and UID.
        - Brief description about how you divide the workload.

3. The TAs will only type `make` to compile your code, make sure your Makefile works on macOS/Linux system.

4. Each group just needs to submit one set of files.