CSCI651 Project #2: Developing a Reliable Data Transfer Protocol

1. Introduction

The IP protocol enables addressing, packet routing and forwarding in the network as a best-effort service, which does not guarantee the arrival of a packet at the destination. In this project you will implement an application layer reliable data protocol mimicking the basic TCP mechanism to overcome packet loss, delay, reordering, and/or corruption. The application will segment the input data, add a TCP-like header to the user data, and send the information to the server using UDP. While TCP is fully explained in RFC 1122 Section 4.2 (https://tools.ietf.org/html/rfc1122#page-82), not all details of TCP are required for this project. The application will be implemented using Java, C, or C++.

2. Reliable Data Transfer

TCP does not attempt to correct the bits of corrupted packets at the receiver, but retransmits those packets when the sender detects that something goes wrong with those packets.

The header format of your implementation should resemble the header format as defined in RFC 793 Section 3.1 (https://tools.ietf.org/html/rfc793). The source and destination port fields may be omitted as the UDP header will contain this information. The implementation should use sequence numbers, acknowledgements, timers, and a minimum Maximum Segment Size of 536 bytes as specified by TCP. Your implementation may negotiate an MSS of up to 1440 if desired according to the connection setup and teardown as specified by TCP.

3. Flow & Congestion Control

Your implementation will support Congestion Control as outlined in RFC 5681. (https://tools.ietf.org/html/rfc5681)

Specifically, the implementation will utilize slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. The sender should not send more than what the receiver can handle at a time (flow control) using the TCP Receive Window field.

4. Test and Output

A 1-2 MB binary file will be transferred between client and server using your reliable data protocol. To quickly verify the data has been sent reliably, the MD5 or SHA1 hash of the file will be used. That is, the server will calculate the MD5 or SHA1 hash of the data received and print the hash to standard out. The server should not attempt to write the received file to disk or otherwise store data; likewise the client should access the file with read-only expectations.

For output, display the sending rate when the sender sends packets, the sequence numbers, any actions for retransmission, timeout events, receipt of acknowledgments, and congestion control status. At the receiver side, display issuing acknowledgments, checksum pass or fail, and packet reordering. This output should track the current state of your implementation to help troubleshoot any issues in the event of protocol failure.

The client must calculate and output the hash of the file before sending; the server must calculate and output the hash of the data when receiving is complete.

5. System Requirements:

Application will be written using Java, C, or C++ programming language. The application should run on Ubuntu 14.04.3 LTS.

6. User Interface Requirements:

```
fcntcp -{c,s} [options] [server address] port
```

Command line options:

`-c, --client`: run as client

`-s, --server`: run as server

`-f <file>, --file`: specify file for client to send; may be relative or absolute path. *client applications*

`-t <#>, --timeout`: timeout in milliseconds for retransmit timer.

Default to 1000 mS (1 second) if not specified. *client applications*

`-q, --quiet`: do not output detailed diagnostics; only print minimal start/stop information and the MD5 or SHA1 hash of the data send/received. *client & server applications*

`server address`: address of server to connect to; may be IP address or host name. *client applications*.

`port`: primary connection port; server expects incoming packets at this port. *client & server applications*

7. Submission

When you complete, turn your program in as follows:

Be sure that everything is compiled and runs correctly. Submit early and submit often.

Put all the source files, header files, and `Makefile` (if there is any) into a single archive. myCourses will expect a single file for final submission. myCourses will only save the last submission.

```
Submit a single archive (zip/tar/tar.gz/rar) containing the following directory
structure and contents:
        /mhvcs            - top level directory (no files)
        /mhvcs/src        - *.java/*.c/*.cpp files for your program
        /mhvcs/doc        - any readme/documentation files
        /mhvcs/bld        - *.class/*.a/*.exe or other executable files derived
from source files.
    Substitute MY RIT username (mhvcs) with YOUR RIT username!
```

Be sure to include `README` to explain how to compile, run, and any other useful information.

8. Evaluation

Test Environment

This project will be executed using a virtual network environment constructed using Linux Containers (`lxc`) and Linux Traffic Control (`tc`) Queuing Disciplines (`qdisc`). This setup is detailed in the document *VirtualNetworkingLinux.pdf* available on myCourses and permits for controlled testing of ideal networks and networks with any combination of loss, corruption, delay, or re-ordering.

The test files will be random binary data a few MB in size. The execution of your implementation will be done using automated scripts which will capture all program output and expect to find the MD5 or SHA1 hash of the data printed by the server application.

Note: You may choose to create a similar virtual network to test, but you may also implement a 'debug' mode of your protocol which purposefully ignores valid packets to simulate loss. *While developing it will be easiest to test in a controlled manner such as dropping one packet every second.*

Point Distribution

10 points: Documentation

10 points: Code Organization/style/implementation

20 points: Use case #1 - ideal network

20 points: Use case #2 - network with loss

10 points: Use case #3 - network with corruption

10 points: Use case #4 - network with excessive delay

10 points: Use case #5 - network with re-ordering

10 points: Use case #6 - network with loss, corruption, delay, and re-ordering

9. Other Useful Utilities

`dd` – utility to generate test files. Example for 3MB binary file populated with random bytes:

```
dd if=/dev/urandom of=test3M.bin bs=1M count=3
```

`md5sum` – utility to calculate MD5 checksum of file. Typically installed by default.