CS 3251 - Spring 2017

2nd programming assignment

DUE DATE: Monday, March 27, 5pm

Reliable Data Transfer

1. Introduction

You are asked to design and implement a client-server application for bidirectional reliable data transfers. Instead of using TCP's reliable service, you will implement your own reliable service on top of the unreliable and connectionless service provided by UDP/IP.

Lets call this application RELDAT. The client will be called "reldat-client" and the server "reldat-server". Initially, the client will connect to the server. If the server is available, and not busy with another client, the client will first start pushing a text file to the server. The server will need to receive the incoming data, transform the text in some way (to simplify this part of the assignment, you can just transform the text from lower-case to upper-case), and then push the modified text to the client. The two transfers will need to be simultaneous, meaning that the server will be sending data to the client as it is still receiving data from the client. Also, the two transfers will need to be using the same RELDAT connection – instead of having two different uni-directional connections.

You have significant flexibility in designing the RELDAT reliability protocol. For instance, you can use a Selective-Repeat type of protocol or something that is more similar to TCP. However, there are some constraints that you need to meet:

- You should only use UDP sockets (you may need one or two sockets, depending on your design). TCP connections are not allowed.
- RELDAT should be connection-oriented, meaning that you need to include reliable connection establishment and connection termination mechanisms.
- RELDAT should be able to detect and recover from duplicate packets.
- RELDAT should be able to detect and recover from lost or corrupted packets.
- RELDAT should be able to detect and recover from out-of-order packets.
- RELDAT should be able to handle variable network delays.

- RELDAT should support window-based “pipelining” – the window size will be provided to the client and server as command-line arguments. Stop-and-Wait type of solutions are not allowed.
- RELDAT should be bidirectional, meaning that data can flow in both directions of a RELDAT connection at the same time.
- The maximum RELDAT packet size should have a payload of 1000 bytes. You should try to transfer max-sized packets when possible.
- RELDAT should be able to perform data transfers of arbitrary size.
- RELDAT should be efficient in its use of network resources. For instance, it is not acceptable if the sender transmits more packets than allowed by the receiver window because that could create buffer overflows at the receiver.
- Your design should be able to handle unexpected crashes at the client or server. For instance, if the client suddenly crashes in the middle of a transfer, the server should be able to detect that, timeout, and return to its listening mode. Similarly, if the server crashes, the client should detect that, timeout, and give an error message that it lost communication with the server.

On the other hand, you do NOT need to provide any of the following features:

- Flow control.
- Congestion control.
- Adaptive estimation of RTTs and of the retransmission timeout.
- Simultaneous client connections at the same server.

Because the Georgia Tech network has excellent performance, we will test your program on a special network (described in Section 4) on which we will introduce various network artifacts (limited capacity, delay, duplication, losses, bit errors, re-ordering) between the client and server. Your RELDAT end-points will be communicating over this private network, encapsulating each RELDAT packet in a single UDP packet.

You will work on this assignment in groups of two or three students. Groups of three students will need to do some extra work: the client and server will need to be written in two different languages (e.g., the client in Python and the server in Java).

You are allowed to use C/C++/C#/Python/Java. Please note that we will test your code on certain CoC machines that run a special network emulator -- it is not sufficient if your code only works on your laptop.

2. RELDAT interface

The RELDAT commands should be as follows:

RELDAT Server

- `Command line: reldat-server P W`

to start up the file server. The command-line arguments are:

`P`: the UDP port number at which the reldata-server's receiving socket is listening at.

`W`: the maximum receive window size at the reldat-server (in packets).

RELDAT Client

The RELDATA client should be able to execute the following commands:

- `Command line: reldata-client H:P W`

to create a reliable connection with the RELDAT server. The command-line arguments are:

`H`: the IP address or hostname of the reldat-server

`P`: the UDP port number of the reldat-server

`W`: the maximum receive window size at the reldat-client (in packets)

- `Command: transform F`

to transform file `F` by sending it to the server (F should exist in the same directory as the reldat-client executable). The downloaded file at the client must be named "F-received" and it should have the same suffix with F. For instance, if F is called "foo.txt", the received file should be named "foo-received.txt". The user should be able to call this command multiple times before closing the connection with the server.

- `Command: disconnect`

to terminate gracefully the connection with the reldat-server.

For both applications, the reliability of the data transfer will be the key criterion for assessing the correctness of your design. Performance will not be an evaluation criterion (unless if your design is clearly inefficient, causing a large transfer delay even for small files).

3. Design Documentation

The design report will need to describe at least the following:

- A high level description of how RELDAT works along with any special features that you may have designed and implemented.
- A detailed description of the RELDAT header structure and its header fields.
- Finite-state machine diagrams for the two RELDAT end-points.
- Algorithmic descriptions for any non-trivial RELDAT functions.

Please make sure that you provide a clear answer to (at least) the following questions in your report:

- How does RELDAT perform connection establishment and connection termination?
- How does RELDAT detect and deal with duplicate packets?
- How does RELDAT detect and deal with corrupted packets?
- How does RELDAT detect and deal with lost packets?
- How does RELDAT detect and deal with re-ordered packets?

- How does RELDAT support bi-directional data transfers?
- How does RELDAT provide byte-stream semantics?
- Are there any special values or parameters in your design (such as a minimum packet size)?

4. Testing on an Unreliable Network

Before you test your code in the adverse environment, it is always a good idea to first make sure that your design works fine under normal conditions, say on your laptop.

For this assignment, we have set up several physical machines to test your code. These machines are configured to delay packets, duplicate packets and to reduce the capacity of the network. This will allow you to test your implementation under adverse conditions. To access these machines, you need to be either on the Georgia Tech network (i.e., using GT machines or connected through the GT WiFi network) or using a Georgia Tech VPN client. Steps to install the VPN client are given by OIT (see http://anyc.vpn.gatech.edu). Make sure to start and login to the VPN client every time you plan to use the remote machines.

In order to access these special machines, you need to *ssh* as follows:

```
  ssh <gt_username>@networklabX.cc.gatech.edu
```

`where X is an integer between 1 and 8.`

Remember to use 130.207.107.*/127.0.0.1 as destination or source address (to listen on) in your code. For transferring your code to the remote machines, you may use *scp, sftp,* or the more user-friendly *filezilla*, which has a GUI.

We have used *netem* along with *tc* in order to setup adverse network conditions. If you feel more comfortable testing/debugging on your laptop, we show how to setup the network emulator at your laptop in Section-7 of this document.

5. Submission instructions

Please turn in well-documented source code, a README file, and a sample output file called sample.txt.

The README file must contain :

- Your name (or names for group projects), email addresses, date and assignment title
- Names and descriptions of all files submitted

- Detailed instructions for compiling and running your programs
- Design Documentation as described in section 3
- Any known bugs or limitations of your program

You must submit your program files online. Create a ZIP/tar archive of your entire submission. Use T-Square to submit your complete submission package as an attachment.

An example submission may look like as follows -

```
pa2.zip
  | -- pa2/
        | -- reldat-client.py
        | -- reldat-server.py
        | -- README.txt/.pdf
        | -- sample.txt
```

Note that you can choose the naming convention of the file containing RELDAT code but all the application code must be divided into 2 files and have names exactly as described above (see bold file and folder names). We will use automated script to test the code which may fail if you use a different naming convention.

Only one member of each group needs to submit the actual code and documentation. The other group members can submit a simple text file in which they mention their partner's name that has submitted the actual assignment.

6. Grading

The following table gives the maximum number of points for each component of this programming assignment.

Category	**Points**
Design report	10
Working transfer with window size=1 packet	20
Working transfer with larger window size	20
Correct handling of duplicate packets	10
Correct handling of lost-corrupted packets	20
Correct handling of re-ordered packets	20

7. How to configure the network emulator at your laptop (only if you choose to do so)

The goal of the following rules is to setup a network with the following network artifacts:
● Delayed packet delivery/transmission
● Lower transmission bandwidth
● Packet corruption
Please note that we may modify these parameters when testing your code. Also, we may add packet duplication and re-ordering.

The tc commands that we execute on the remote servers, are as follows

```
#parameters
IF = eth0
SUBNET = 130.207.107.12/30
BW = 10mbit
CORRUPT_PCT = 5%
DELAY_MEAN = 100ms
DELAY_STD = 30ms
# tc commands
sudo tc - s qdisc ls dev $IF
sudo tc -s filter ls dev $IF
sudo tc qdisc del dev $IF root
sudo tc qdisc add dev $IF root handle 1 : htb
sudo tc filter add dev $IF parent 1 : protocol ip prio 1 u32 flowid 1 : 1 match ip dst $SUBNET
sudo tc class add dev $IF parent 1 : classid 1 : 1 htb rate $BW
sudo tc qdisc add dev $IF parent 1 : 1 handle 1 0: netem delay $DELAY_MEAN $DELAY_STD
distribution normal corrupt $CORRUPT_PCT
```

Rules for Local Host

The previous rules only work when the client and server run on different physical hosts. In order to test the program on your own computer, you will need to be running a Linux distribution (ubuntu, fedora, centos, mint) and execute these rules with the following parameters

```
IF = lo
SUBNET = 192.168.56.2/32 # your local IP address
```

Note that the IP that you use here should be used in your code as well. For example, if you decide to use 127.0.0.1 in your code, you should use 127.0.0.1/32 here. Do not use the IP 0.0.0.0 or *localhost* for testing purposes. Feel free to post on piazza if you face any issues.

References

http://www.linuxfoundation.org/collaborate/workgroups/networking/netem
http://onlinestatbook.com/2/calculators/normal_dist.html