CS 3251 - Spring 2016

Programming Project (Part-1 and Part-2)

DUE DATE: Monday, April 20, 9pm

Design and Implementation of a Reliable Transport Protocol

1. Introduction

This is the main programming project for this course. You are asked to design and implement a Reliable Transport Protocol (referred to as RTP in the following). In Part-1 of the project, you can assume that the network will only delay and potentially duplicate your packets. In Part-2, the network can additionally corrupt, re-order or drop your packets.

You have significant flexibility in designing RTP. For instance, you can specify any RTP header fields and functionality you want. You can use any checksum, retransmission, or ACKing algorithm you want. For example, you can choose to design an error-correcting code (instead of just detecting packet errors).

However, you need to satisfy the following constraints:

- RTP should be using a (single) UDP socket to transfer packets over the network. In other words, you are not allowed to use TCP in this project or multiple UDP sockets.
- RTP must be connection-oriented. You can adopt the algorithms that TCP is using for connection establishment and termination -- or you can design yours.
- RTP must be bidirectional, meaning that data can flow in both directions of an RTP connection at the same time. (This is optional if you are doing the project individually or in a group of two students)
- RTP should be able to detect and recover from duplicate packets.
- RTP has to provide some form of receiver window-based flow control.

- RTP has to provide byte-stream communication semantics (similar to TCP).
- RTP must allow the multiplexing of multiple RTP connections at the same end-host (hint: you may need port numbers at the RTP layer).
- RTP should not make any assumptions about the underlying RTT or capacity of the network. In particular, we will be using a network emulator to limit the underlying network capacity to arbitrary levels and to create variable network delays.
- The maximum RTP packet size should not be larger than 1000 bytes.
- RTP must be able to perform data transfers of arbitrary size.
- RTP has to have a clean Application Programming Interface (API) through which it will be interacting with any higher-layer application. For instance, it is not ok if an application accesses any interval variables of RTP that are not exposed through this API.
- RTP should be efficient in its use of network resources. For instance, it is not acceptable if the sender only transfers one packet per round-trip time even though the receiver is advertising a larger window size. It is also not acceptable if the sender transmits more packets than allowed by the receiver window because that could create buffer overflows at the receiver.

In part-2 of the project, you need to also satisfy the following requirements, in addition to all requirements for part-1:

- RTP should be able to recover from packet losses.
- RTP should be able to recover from re-ordered packets.
- RTP should be able to recover from corrupted packets.
- RTP should perform some form of congestion control (it does not need to be as sophisticated as TCP's congestion control).

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 RTP end-points will be communicating over this private network, encapsulating each RTP packet in a single UDP packet.

We recommend that you work on this project with another student. We cannot allow groups of more than **two** students, unless if you have got the instructor's approval (but a three-student group will need to implement additional functionality). You are allowed to use 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. Applications running over RTP

We are asking you to write two different applications that run over RTP. These two applications will need to interact with RTP only through an API that you will design (e.g., the RTP interface should allow an application to create a connection, accept a connection, send or receive data, set the local receive window size, close a connection, etc).

The first application is the Relational DB application that you have already implemented in the first programming assignment (RDBA). The second will be a simple file transfer application.

2.1 Relational Database Application (RDBA)

In the first programming assignment, you implemented a simple client-server RDBA application over UDP and TCP. Here, you will reuse the same application but it will be running over your RTP protocol.

2.2 File Transfer Application (FTA)

FTA is a simple client-server file transfer application. The client will be connecting to a given FTA server and then the client will be able to either download or upload one or more files. The FTA server should be able to support multiple FTA clients (hint: think about multi-threading or other ways to support multiple simultaneous RTP connections).

The FTA commands should be as follows:

FTA Server

- `Command line: fta-server P W`

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

`P`: the UDP port number at which the FTA-server's socket is listening.
`W`: the maximum receive window size at the FTA-server (in bytes).

FTA Client

The FTA client should have an interactive command window that can (sequentially) execute the following commands :

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

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

`H`: the IP address or hostname of the FTA-server
`P`: the UDP port number of the FTA-server
`W`: the maximum receive window size at the FTA-client (in bytes)

- `Command: get F`

to download file `F` from the server (F should exist in the same directory as the FTA-server executable). The downloaded file at the client must be named “get_F”.

- `Command: get-post F G`

to download file `F` from the server and at the same time upload file `G` to the server through the same RTP connection (`F` and `G` should exist in the same directory as the FTA-server and FTA-client executables, respectively). The downloaded file at the client must be named “get_F”, while the uploaded file at the server must be named “post_G”.

- `Command: disconnect`

to terminate gracefully the connection with the FTA-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).

For testing purposes, please make sure that you have stored a binary file (a JPEG image, more than 100KB) at the same directory as the FTA server, and name that file “3251.jpg”.

3. Design Documentation

You need to submit only one design report for both Parts of the project.

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

- A high level description of how RTP works along with any special features that you may have designed and implemented.
- A detailed description of the RTP header structure and its header fields.
- Finite-state machine diagrams for the two RTP end-points.
- A formal description of the protocol’s programming interface (the functions it exports to the application layer, including return values and any error conditions).
- Algorithmic descriptions for any non-trivial RTP functions.

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

- How does RTP perform connection establishment and connection termination?
- How does RTP perform window-based flow control?

- How does RTP perform congestion control?
- How does RTP detect and deal with duplicate packets?
- How does RTP detect and deal with corrupted packets?
- How does RTP detect and deal with lost packets?
- How does RTP detect and deal with re-ordered packets?
- How does RTP (de)-multiplex data to different RTP connections at the same host?
- How does RTP support bi-directional data transfers? (if applicable)
- How does RTP 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 have released the netem/tc commands that we use to setup the adverse network here. This is not required to complete the assignment however.

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/**

| -- **ftaclient.py**

| -- **ftaserver.py**

| -- **dbengineRTP.py**

| -- **dbclientRTP.py**

| -- rtp.py

| -- README.txt/.pdf

| -- sample.txt

| -- 3251.jpg

Note that you can choose the naming convention of the file containing RTP code but all the application code must be divided into 4 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.

NOTE: you can choose to include two different versions of your code, one for Part-1 and another for Part-2. This may be useful if your code works fine for Part-1 but only partially for Part-2.

6. Grading

We will test Part-1 and Part-2 of your project separately. Part-1 accounts for 15% of the final course grade, and Part-2 accounts for another 15% of the final course grade.

GRADING OF PART-1

The following table gives the maximum number of points for each component of Part-1 of the project.

Category (1- or 2-student groups)	**Points**
Design report (those parts of the design that cover part-1)	20
Working file transfer (GET) with window size=1 packet	20
Working file transfer (GET) with larger window size	20
Working bidirectional data transfer (GET-POST) - EXTRA CREDIT OPPORTUNITY	20
Correct handling of duplicate packets	10
Handling multiple connections at the same server	20
Correctly working RDBA	10

If you are in a group of **three** students, the GET-POST command is a mandatory feature (no extra credit for supporting this feature). You can get extra credit (up to 20 points) if you design and implement an additional and advanced feature(s) that the instructor has previously approved. Your project grade will be determined based on the following table:

Category (3-student groups)	**Points**
Design report (those parts of the design that cover part-1)	20
Working file transfer (GET) with window size=1 packet	10
Working file transfer (GET) with larger window size	15
Working bidirectional data transfer (GET-POST)	15
Correct handling of duplicate packets	10
Handling multiple connections at the same server	20
Correctly working RDBA	10

GRADING OF PART-2

The following table gives the maximum number of points for various components of Part-2 of the project.

Category (1- or 2-student groups)	**Points**
Design report (those parts of the design that cover Part-2)	20
Working file transfer (GET) with window size=1 packet	20
Working file transfer (GET) with larger window size	20
Working bidirectional data transfer (GET-POST) - EXTRA CREDIT OPPORTUNITY	20
Demonstration of some form of congestion control	10
Handling multiple connections at the same server	20
Correctly working RDBA	10

If you are in a group of **three** students, the GET-POST command is a mandatory feature (no extra credit for supporting this feature). You can get extra credit (up to 20 points) if you design and implement an additional and advanced feature(s) that the instructor has previously approved. Your project grade will be determined based on the following table:

Category (3-student groups)	**Points**
Design report	20
Working file transfer (GET) with window size=1 packet	10
Working file transfer (GET) with larger window size	15
Working bidirectional data transfer (GET-POST)	15
Demonstration of some form of congestion control	10
Handling multiple connections at the same server	20

Correctly working RDBA	10

7. FAQ

(In this section we will be including answers to your questions and clarifications)

- You are free to use any serialization/de-serialization library in this project