Programming Assignment 2 – CS 3251
Design and Implement your own Reliable Transport Protocol

1- Introduction

In this assignment you will design and implement your own **Reliable Transport Protocol** -- lets call it RTP for now (you can choose another name if you prefer).

You have significant flexibility in designing RTP. However, the design constraints that we give you are the following:

a. RTP has to be as reliable as TCP
b. RTP has to be connection-oriented
c. RTP has to provide window-based flow control
d. RTP has to provide byte-stream communication semantics (as TCP does).

Together with RTP, you will design and implement a simple File Transfer Application (FTA). This application will allow you to test RTP by transferring files between the RTP client and server. Additionally, the separation between FTA and RTP will help you to design a cleaner interface between the transport protocol and the application(s) that use it.

Because the Georgia Tech network has excellent performance, we will provide you with a Network Emulator (NetEmu) that will be used to drop, corrupt, reorder, duplicate and delay the packets between your client and server. Your RTP client and server will be communicating through NetEmu, encapsulating their messages in UDP/IP packets.

In the following, we provide more detail about RTP, FTA, NetEmu, and about the grading of this project.

Two key **due dates** for this project:

- Submission of a design report in which you describe RTP's operation and programming interface: **OCT 23**
- Submission of your implementation (including RTP and FTA): **NOV 21**

More information about the previous two submissions will be given later.

You can work on this project individually or with another student. We cannot allow groups of more than two students.

2- Reliable Transport Protocol

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

- a high-level description of how RTP works and of any special features you have designed
- 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)
- algorithmic descriptions for any non-trivial RTP functions (e.g., how it detects corrupted packets)

In the first page of the report (after the cover page), please provide clear answers to the following questions. We will use these answers in testing and grading your project:

- Is your protocol non-pipelined (such as Stop-and-Wait) or pipelined (such as Selective Repeat)?
- Can your protocol handle lost packets?
- Can your protocol handle corrupted packets?
- Can your protocol handle duplicate packets?
- Can your protocol handle out-of-order packets?
- Can your protocol support bi-directional data transfers?
- Does your protocol use a non-trivial checksum algorithm (i.e., anything more sophisticated than the IP checksum)?

- Does your protocol have any other special features for which you request extra credit? Please include a short summary of these features here (even if you describe them in more detail later in the report).

Note: you may modify the RTP protocol after you submit your design report. Your design report will be graded based on what you submit by October 23. If you modify the protocol later, you will need to submit a revised design report together with the final implementation on November 21. In that case, please make sure that you highlight the differences between your original design report and the modified design report that describes what you have actually implemented.

3- Network Emulator (NetEmu)

As previously mentioned, we will provide you with the source code for a Network Emulator (NetEmu) that will be causing random network artifacts, i.e., packet losses, corrupted packets, duplicate packets, packet reordering and packet delays. NetEmu will be parameterized with the following command-line arguments:

```
-l XX             Packet loss probability (e.g., "-l 10" means 10% loss probability)
-c XX       Packet corruption probability (note that this may also corrupt the RTP header)
-d XX       Packet duplication probability
-r XX       Packet reordering probability
-D YY       Average packet delay (in milliseconds) in NetEmu queue
```

Your RTP client and server will need to communicate through NetEmu -- they should never exchange packets directly. We will give you the IP address and port number for NetEmu.

Your RTP packets will need to be encapsulated in UDP packets. In other words, RTP will be using the best-effort connectionless services of UDP (instead of using TCP sockets or raw IP sockets).

Your RTP client and server will need to run at the same host (same IP address). This will allow NetEmu to easily forward packets between the client and server.

Additionally, you need to make sure that the RTP client's UDP socket binds to a port number that is an EVEN number, say X. Then, you need to make sure that the RTP server's UDP socket binds to the port number X+1 (an odd number). This simple "trick" will allow NetEmu to know if a received UDP packet was sent by the RTP client or the RTP server and forward it accordingly.

So, if your RTP client and RTP server run at a host with IP address A_H, while NetEmu runs at a host with IP address A_E and port number P_E, any packet sent from the RTP client will need to have:

Source IP address: A_H , Source-port: X, Destination IP address: A_E, Destination port: P_E

NetEmu will forward that message (if it is not dropped) to the RTP server with a new UDP packet that has:

Source IP address: A_E , Source-port: P_E, Destination IP address: A_H, Destination port: X+1

Similarly, any packet sent from the RTP server will need to have:

Source IP address: A_H , Source-port: X+1, Destination IP address: A_E, Destination port: P_E

NetEmu will forward that message (if it is not dropped) to the RTP client with a new UDP packet that has:

Source IP address: A_E , Source-port: P_E, Destination IP address: A_H, Destination port: X

4- File Transfer Application (FTA)

FTA is a very simple client-server file transfer application. The FTA commands should be as follows:

FTA SERVER

- **Command-line:** `fta-server X A P`

The command-line arguments are:

X: the port number at which the fta-server's UDP socket should bind to (odd number)

A: the IP address of NetEmu

P: the UDP port number of NetEmu

- **Command:** window W (only for projects that support pipelined and bi-directional transfers)

W: the maximum receiver's window-size at the FTA-Server (in segments).

- **Command:** `terminate`

Shut-down FTA-Server gracefully.

FTA CLIENT

- **Command-line:** `fta-client X A P`

The command-line arguments are:

X: the port number at which the fta-client's UDP socket should bind to (even number). Please remember that this port number should be equal to the server's port number minus 1.

A: the IP address of NetEmu

P: the UDP port number of NetEmu

- **Command:** connect-get F (only for projects that do **NOT** support bi-directional transfers)

This command does two things. First, the FTA-client connects to the FTA-server and, if the connection is successfully completed, the client downloads file F from the server (if F exists in the same directory that the fta-server executable is stored at).

If you only support uni-directional transfers (from the server to the client), the client should disconnect from the server at the end of the transfer.

- **Command:** connect (only for projects that support bi-directional transfers)

The FTA-client connects to the FTA-server (running at the same IP host).

- **Command:** get F (only for projects that support bi-directional transfers)

`The FTA-client downloads file F from the server (if F exists in the same directory as the fta-server executable).`

- **Command:** `post F (only for projects that support bi-directional transfers)`

`The FTA-client uploads file F to the server (if F exists in the same directory as the fta-client executable).`

- **Command:** `window W (only for projects that support pipelined transfers)`

`W: the maximum receiver's window-size at the FTA-Client (in segments).`

- **Command:** `disconnect (only for projects that support bi-directional transfers)`

`The FTA-client terminates gracefully from the FTA-server.`

5- Grading

The following table gives the maximum number of points for various components of the project. Note that if you design and implement successfully **all** of the following features, you can get up to 125 points (i.e., 25 points of extra credit). If you include any additional advanced features, you can get even more extra credit. Your final grade however cannot be higher than 140/100.

Feature	**Maximum number of points**
Design report (due on **OCTOBER 23**)	20
FTA application	10
Successful non-pipelined operation without any network artifacts (the RTP packets should still go through NetEmu)	30
Pipelined operation (you will need to provide a "window W" command in FTA)	10
Able to recover from lost and duplicate packets	10
Able to recover from corrupted packets	10
Able to recover from out-of-order packets	15
Able to do bi-directional data transfers	15
Advanced checksum (must be more sophisticated than IP checksum)	5
Additional features	Extra points (TBD)

6- FAQ

(in this section we will include answers to your questions)