For this assignment you will design and implement your own reliable data transfer protocol. You will implement this protocol in application space on top of UDP. You will develop and document an API for your protocol. Then you will demonstrate your reliable API by building a simple file transfer application.

Part 1 of this assignment is to design and document your protocol and API. You will submit this documentation as the first milestone on November 4. The program implementation will be due on November 22.

You may work on this project individually or in teams of two. We can not allow teams of more than two. The same grading criteria will be applied for all projects, regardless of whether there is one or two people submitting the assignment. (When picking a partner for this project, make sure you have a conversation about remaining late days. We will use the minimum of the remaining late days for your team.)

Introduction

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

a. RxP must be as reliable as TCP

b. RxP must be connection-oriented

c. RxP must provide window-based flow control

d. RxP must provide byte-stream communication semantics (as TCP does).

2- Reliable Transport Protocol

The RxP design specification will need to specify at least the following:

- a high-level description of how RxP works and of any special features you have designed

- a detailed description of the RxP header structure and its header fields

- finite state-machine diagrams for the two RxP end-points

- a formal description of your API (the functions it exports to the application layer)

- algorithmic descriptions for any non-trivial algorithms in RxP (e.g., how you detect corrupted packets)

In the first page of the report (after the cover page), please provide clear answers to the following questions:

- Is your protocol non-pipelined (such as Stop-and-Wait) or pipelined (such as Selective Repeat)?

- How does your protocol handle lost packets?

- How does your protocol handle corrupted packets?

- How does your protocol handle duplicate packets?

- How does your protocol handle out-of-order packets?

- How does your protocol provide bi-directional data transfers?

- Does your protocol use any non-trivial checksum algorithm (i.e., anything more sophisticated than the IP checksum)?

Your protocol design must support all features to receive full credit (e.g. window-based flow control, bi-directional data transfer) even if you don't plan to completely implement them all.

Please note: you may modify the RxP protocol after you submit your design report. Your design report will be graded based on what you submit by the first deadline. You will need to submit a revised design report together with the final implementation. If you make changes to your design during the implementation phase, 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 re-ordering 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 corrupt your RxP header)

-d XX

Packet duplication probability

-r XX

Packet re-ordering probability

-D YY

Average packet delay (in milliseconds) in NetEmu queue

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

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

So, if your RxP client and RxP 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 RxP 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 RxP server with a new UDP packet that has:

Source IP address: A_E , Source-port: PE, Destination IP address: A_H, Destination port: X+1 Similarly, any packet sent from the RxP 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 RxP 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 (FxA)

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

FxA SERVER

● **Command-line**: FxA-server X A P

The command-line arguments are:

X: the port number at which the FxA-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 FxA-Server (in segments).

● **Command:** terminate Shut-down FxA-Server gracefully.

FxA CLIENT

● **Command-line**: FxA-client X A P

The command-line arguments are:

X: the port number at which the FxA-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 - The FxA-client connects to the FxA-server (running at the same IP host).

● **Command:** get F - The FxA-client downloads file F from the server (if F exists in the same directory with the FxA-server program).

● **Command:** post F - The FxA-client uploads file F to the server (if F exists in the same directory with the FxA-client program). This feature will be treated as extra credit for up to 20 project points.

● **Command:** window W (only for projects that support configurable flow window) W: the maximum receiver's window-size at the FxA-Client (in segments).

● **Command:** disconnect - The FxA-client terminates gracefully from the FxA-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 100 points.

This is a significant project and some of you may not be able to get it completely working. We strongly recommend that you use the grading rubric to plan out an incremental approach and save a snapshot of your working project at each point along the way so that you are sure to have something you can turn in for partial credit.

Design Report - 30 Homework Points

Working API with demonstration of connection establishment and close - 20 project points

Working file transfer using Stop-And-Wait ARQ (no injected errors) - 20 project points

Working file transfer with random bit errors - 10 project points

Working file transfer with random lost packets - 10 project points

Working file transfer with Sliding Window ARQ (with bit errors and lost packets) - 20 project points

Working file transfer with Sliding Window and re-ordered packets - 10 project points

Working file transfer with configurable flow window size - 10 project points

All of the above "file transfer" testing will be done using your "get" command. That is the required basic functionality. For up to 20 points extra credit you can implement "bi-directional" data and support the "put" command.