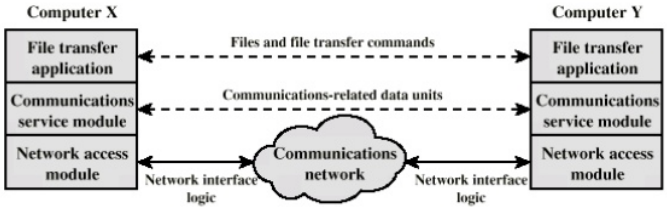

Computer Networks
Fall 2014-Project

Due Date: 1st Dec, 2014 **Due Time: 11:00 a.m.**

Motivation

More or less this same project statement has been around from last 13 years in FAST ISB Computer Networks course, therefore it is a legacy of our university now and it is a wonderful one. Our Instructor Dr. Muddassar Farooq made us do this project in one of our courses and despite of all of the misery of that time all of us agree that this was the project that made us learn most during our degree. We really appreciate the creativity and effort that Dr. Muddassar has put in the design of this project back in 2001. The beauty of the project is that it can't be copied because only the original creator of the code and design understand the complex implementation and no one else can justify most of the parts of the implementation. This project has been an inspiration for many keen learners and I hope most of you will make the best out of it this time also.

Introduction

In this project, you will design and implement UDP-based client-server for simple file transfer. You will manage the transfer of file data from client to server using only a UDP channel with unusually flaky delivery. The project allow you to better understand the common transport layer protocol mechanisms by applying your knowledge of fragmentation, error detection and/or correction, and retransmission to construct a Reliable Delivery Protocol (RDP) on top of UDP. Multiple **POSIX** threads in the server will allow logically concurrent operation between many clients. Because of the difficulty of managing concurrent accesses to data structures, you are strongly encouraged to use distinct UDP sockets for handling requests and to support each RDP stream in the server. However, since you really want to learn as much as possible, therefore **you are supposed to implement a two layer stack that will consist of RDP layer and an application layer.**

RDP Layer - Design and Implementation

The primary goal of this layer is to improve your understanding of the issues and approaches utilized to build a reliable byte stream abstraction across an unreliable network with artificial segment boundaries.

Your protocol should run on top of UDP, which is an unreliable best effort datagram communication service. UDP provides you with the necessary de-multiplexing service for process-to-process communication, so you do not have to replicate this in your transport layer. However UDP may not deliver some messages, may deliver out of order, and may deliver duplicates of a message. The

maximal segment size supported by UDP is 32768 bytes. Therefore you might have realized that use of UDP is asking for application writers to design their own algorithm on top of an unreliable stream. This layer should ensure normal operations of a reliable layer.

In this document only minimal design requirements are provided and you have the freedom to create your own design. So *be creative* but you should be able to justify your own design. Some of the alternatives that you should consider include:

- Packet size
- Header Format
- Sliding Window Protocol (GBN/Selective repeat/Hybrid)
- Flow Control
- ACK vs ACK/NACK
- Piggybacking or not
- RTT estimation
- Error control using CRC-16 ($x^{16} + x^{13} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^2 + 1$)
- Timer management i.e. timeouts using timer linked lists and doing retransmissions.

Please design appropriate calls so that functionality of this layer can be implemented. Since we are quite kind hearted instructors, hence you have been spared from taking care of delayed duplicates in your RDP layer; however, enthusiastic students are most welcomed to work in this area, hoping to get extra credit. **Your design document must clearly answer following parts:**

Part 1(a): Message Formats
Describe the message formats you will use for sending requests, indicating the non-existence of a file, indicating the length of a file, sending data, and sending acknowledgements. Explain and justify your framing decisions for each format; remember that UDP provides length-based framing for you, but if you put more than one piece of information into a message, you must solve the framing problem again.

Part1 (b): Window Data Structures
Define data structures to keep track of send and receive window information. On the send side, for example, this structure should include the last acknowledgement received (LAR) and an array of outstanding segments with timeout information. Provide versions of your data structures annotated with explanations of the purpose of each field.

Part1(c): Window Operations
Briefly outline the steps necessary for determining the minimum timeout value across outstanding packets in the send window (i.e., packets already sent at least once but not acknowledged). Also outline your strategy for processing acknowledgements. Do not turn in C code. Rather, provide descriptive paragraphs, supplemented if necessary by pseudo-code of the form used in algorithms textbooks.

Part1 (d): Error Correction

Discuss the conditions under which the server should use forward error correction. In particular, provide an Analysis of the minimum number of packets sent to transmit N bytes of data in packets no larger than L bytes for both an error detection scheme based on CRC-16 and an error correction scheme based on 2D parity. Assume that every packet has an independent probability p of suffering a single-bit error, and that only single-bit errors occur.

Implementation Details:

You need to configure your client and server with the run time configurations.
Client configuration file should have at least following attributes:

- Server IP
- Server port
- Client Port
- File name
- Packet size
- ServerTimeout
- ACK number to drop
- ACK number to corrupt

Server IP:
This is the IP on which server will be active.
Server port:
The port on which server will be active.
Client Port:
The port on which the client will listen.
File name:
The name of the file in which the client will save the received data.
Packet size:
The size of each packet bytes.
Server Timeout:
The amount of time after which the client will assume that the server is dead.
ACK number to drop:
For every n^{th} number of packets the client will not send back the ACK.
ACK number to corrupt:
For every m^{th} number of packet the client will send back a corrupt ACK so that you apply error correcting and detecting algorithms.

Server File will have at least following attributes:

- Server port
- File name
- Packet size
- Packet number to drop
- Packet number to corrupt

Server port:
The port at which the server will be active.
File name:
The name of the file in which the server will save the received data.
Packet size:
The size of each packet in bytes.
Packet number to drop:
Server will not send every n^{th} number packet.
Packet number to corrupt:
Server will corrupt every m^{th} number packet so that you apply error correcting and detecting algorithms.

In the start you will read a simple text file in which you have saved these configurations. On both server and client side, then you will implement the RDP which will run according to the giving configurations. You have to show a well-structured message order to support the readability of your program.

Messages: To assist in debugging and in simplifying testing of your code, please prefix printed output with human-readable messages. For example,

ERROR: all error messages
REJECTED: bad request message received
NO FILE: requested file not found
ACCEPTED: valid request for existing file received
OBTAINED: client knows file length and is ready to receive data
SENT #: sent packet/ACK number #
RESENT #: resent packet/ACK number #
ACK #: server received ACK #
SEGMENT #: client-received segment #

And you are, of course, encouraged to design your own error message, if you feel they would help in debugging.

Application Layer

For the application layer, you should take the file as an input and get it transferred to the receiver's end. **Your application must be able to transfer *text* file and *pdf* file from sender to the receiver.**

Testing Layer

Some of intelligent students must have already realized that how the hell instructor is going to test the software. Ok in real life customers hardly write test code rather writing test code is the responsibility of software developer, customer only provides with test cases. Hence an important part of the project would be to write test code that would make the instructor confident that your code is working properly. The purpose of test software would be to make your code crash. Hence

group member that writes a test code that can make software of another group crash is an ideal candidate for highest grade in project, OK!!. Hence the competition is between groups, make another groups code crash and enjoy the life, but of course best group would be the one whose code can't be crashed in any adverse cases.

Please comply with the use of a few initialization and replacement functions. Please note that not using the functions will be treated as a deliberate attempt to disable the adversary code and will be graded as not addressing the assignment (zero credit).

```
ssize_t project_recvfrom (int s, void* buf, size_t len, int
flags, struct sockaddr* from, size_t* fromlen);
int project_poll (struct pollfd fds[], nfds_t nfds, int
timeout);
int project_pthread_create (pthread_t* thread, const
pthread_attr_t* attr, void* (*start_routine) (void*), void*
arg);
void project_init (int* argc, char*** argv);
```

Use the `project_recvfrom`, `project_poll`, and `project_pthread_create` routines just as you would the corresponding C library calls. For function prototypes, include the header file `project.h` file into your client and server executables. `project_recvfrom` must be used to receive all file data and acknowledgements; recvfrom may be used directly only for accepting client request messages. The semantics of `project_recvfrom` differ slightly from recvfrom in that `project_recvfrom` will never return messages longer than 256 bytes. Such messages are dropped silently, implying that your RDP will need to fragment and reassemble the file. Data packets (and ACK's) may be dropped because of the server/client configuration file or other reasons as well;

Please ensure that all members of different groups writing testing software should sit together and define similar interfaces so that one group's routines can be tested with another groups adversary code.

In both client and server, your function main must begin with a call to `project_init` to initialize the adversary code. Note that you must pass pointers to the `argc` and `argv` parameters to allow the routine to modify them.

Deliverables:

- ✓ Complete RDP Layer Design Document
- ✓ Please design your overall Makefile in such a way that once make is run, client and server executables are built. Please think carefully about command line options, a part from the one that you must implement, you must be smart enough to detect need of more and then clearly document that.
- ✓ You are supposed to write a README file that describes over all working of your project and how you have handled different adversaries in your design.

We have full faith in you that if you people 3 in a group work really hard,

1. Project is doable in the stipulated time
2. If you feel at any stage that you can't do this project in stipulated time, then please go to point 1 ☺

Note: This project, if done, could place you in line with students of TOP 10 US Universities because their students implement problems of same or less difficulty in their semester projects and I think that this reason alone is sufficient to put you on toes to complete this project.

Policy on Plagiarism

- **All the projects will be checked by MOSS script.**
- **Any one resulting in 40% or more similarity will be graded F in the course, irrespective of being a donor or receiver.**