COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 4320

Introduction to Computer Networks

Fall 2020

Project 1

Implementation of a Simple File Transfer Service over the UDP Transport Service

Due in Canvas: 11:55pm October 29, 2020

Objective

The purpose of this assignment is to implement a simple File Transfer Protocol (FTP) service over the UDP transport service. This project will help you understand some of the important functions in implementing simple data transfer protocols in computer networks. You will write the simple FTP client and server programs that will communicate over either your own computer(s) or the College of Engineering LAN. You will also write a gremlin function that will simulate unreliable networks which will corrupt and lose packets. You will also learn other important functions in computer networks: (1) implementation of segmentation and re-assembly of long messages, (2) detecting errors in the received packets, and (3) emulation of packet errors generation and detection.

Overview

In this project you will implement simple FTP client and server programs that ***must be written in C or C++*** and execute correctly either in your own computer(s) or in the COE tux Linux computers. You will also implement segmentation and re-assembly function, an error detection function and a gremlin function (that can corrupt packets and lose packets with specified probabilities). The overview of these software components is shown in Figure 1 below.

The FTP client initiates the communication by sending an FTP request to the FTP server at a specific IP address and port number. If you are using the COE tux computers, use only the port numbers that are assigned to your group. You will only implement the PUT command of the FTP protocol, where the FTP client will send a FTP request to the FTP server to transfer a data file from the client to the server that will store the file in the file server system. The FTP request message will be of the following form:

```
PUT TestFile
```

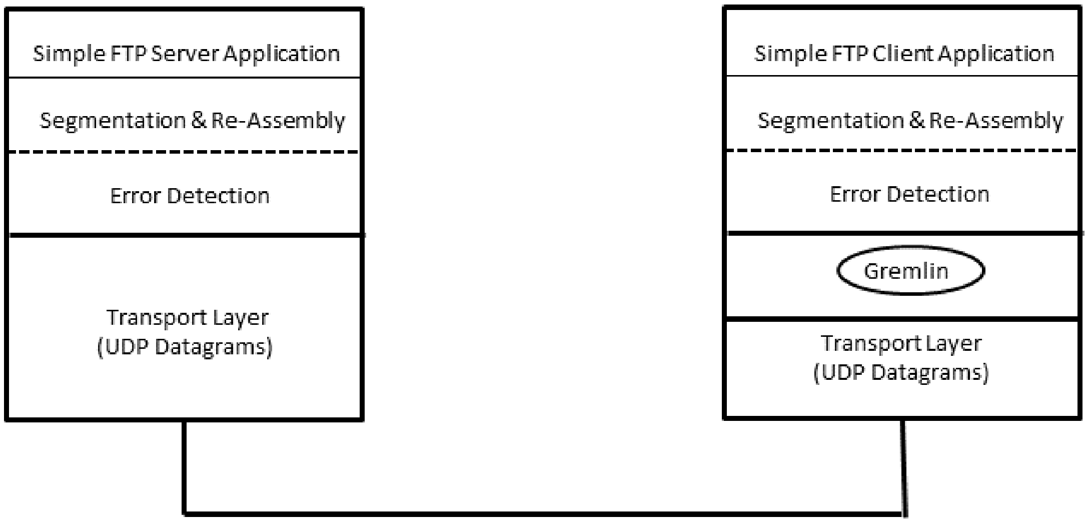
Simple FTP Server Application
Segmentation & Re-Assembly
Error Detection
Transport Layer
(UDP Datagrams)
Simple FTP Client Application
Segmentation & Re-Assembly
Error Detection
Gremlin
Transport Layer
(UDP Datagrams)

Figure 1. Overview of the Software Components

The file, `TestFile`, to be transferred is originally stored in the local secondary storage of the client host. The client first reads the file and put them in a buffer and sends the content of the buffer to the FTP server. Since the requested file is large, the server application will use the segmentation function to partition the file into smaller segments that will fit into a packet of size allowable by the network. Each segment is then placed into a 128-byte packet that is allowed by the network. The packet must contain a header that contains information for error detection and other protocol information. You may design your own header fields that are of reasonable sizes. Another field that must be in the header is a sequence number used in this simple FTP protocol for re-assembly of the segments back into the original message. The packet is then passed to the error detection function which, at the client process, will compute the checksum and place the checksum in the header. The packet is finally passed through the gremlin function and then sent via the UDP socket to the FTP server. The Gremlin function may randomly cause errors in some packet while it may randomly drop other packets. This will emulate errors that may be generated by the network links and routers.

The file is an ASCII file and must be at least 20 Kbytes in size. The file is sent in 128-byte packets (including the header) until the end of the file is reached. The last packet will be padded with NULL character if the remaining data of the file is less than 128 bytes. At the end of the file, it transmits 1 byte (NULL character) that indicates the end of the file. It will then close the file.

Add `cout` or `printf` statements in the client program to print the sequence numbers and data to indicate that client is sending the packets and also receiving packets from the server, i.e. print each packet sequence number and data (say, only the first 48 bytes of data) that it sends and receives.

The FTP server program will respond to the client FTP requests. On receiving the PUT request, the server will receive data of the file in 128-byte packets, i.e. the server will receive each 128-byte packet in a loop and writes them into a local file system sequentially. Each packet is processed by the error detection function that will detect possibility of error based on the checksum. If there are errors in the packet, just print the error message and the packet sequence number. You do not need to handle errors or lost packets in this project. The packet is then processed by the segmentation and re-assembly function that re-assembles all the segments of the file from the packets received into the original file. When it receives a 1-byte message with a NULL character, then it knows that the last packet has been received and it closes the file. The server then constructs FTP response messages by putting the status on the header lines. The header line will be of the following form:

```
PUT successfully completed
```

This FTP server response packet is not processed by the segmentation and re-assembly, error detection or the Gremlin function.

Add `cout` or `printf` statements in the server program to print the sequence numbers and data to indicate that the server is receiving packets and sending the packets, i.e. print each packet sequence number and data (say, only the first 48 bytes of data) that it receives and sends.

Gremlin Function

Your program must allow the probabilities of damaged *and lost* packets to be **input as arguments when the program is executed**. These parameters for packet damage *and lost* probabilities are passed to your Gremlin function. You will implement a gremlin function to simulate *three* possible scenarios in the transmission line: (1) transmission error that cause packet corruption, (2) *packet loss,* and (3) correct delivery. Before the client process sends each packet through the UDP socket, it first pass the packet to a gremlin function which will randomly determine, depending on the damage probability, whether to change (corrupt) some of the bits or pass the packet as it is to the server receiving function. It will also decide whether some packets will be dropped based on the loss probability. The gremlin function uses a random-number generator to determine whether to damage a packet, drop (lose) a packet or pass the packet as it is to the receiving function. For example, a packet's loss probability of 0.2 would mean that two out of ten packets will be dropped.

If it decides to damage a packet, it will decide on how many and which byte to change. The probability that a given packet will be damaged, P(d), is **entered as an argument at runtime**. If the probability of damaging a packet is .3, then three out of every ten packets will be damaged. If the packet is to be damaged, the probability of changing one byte is .7, the probability of changing two bytes is .2, and the probability of changing 3 bytes is .1. Every byte in the packet is equally likely to be damaged. The packet is then passed from the gremlin function to be sent through the UDP socket.

Error Detection Function

The sending process, e.g. the FTP client, will compute the checksum for the packet that is to be sent. The checksum is calculated by simply summing all the bytes in the packet. The checksum is then inserted into the checksum header field of the packet.

The receiving process, e.g. the FTP server, will then use the same algorithm for computing the checksum that the sending process used. It will calculate the checksum by summing all the bytes in the received packet. It then compares the computed checksum with the checksum received in the packet. If the two checksums match, then it assumes that there is no error, otherwise there is at least an error in the packet.

When the receiving process detects an error in a packet, it will print out a message indicating the packet's sequence number and that there is an error in the packet.

In this project, the receiver need **not** handle errors in packets; instead, it just prints out a message indicating that the packet has an error and prints the packet sequence number. The sender does **not** need to handle the packet error either.

Testing

Run the FTP client and FTP server programs for a simple data transfer. Other software, such as the segmentation and re-assembly, error detection and gremlin functions must also function correctly. The FTP client and server programs must execute correctly either on *your own computer* or the College of Engineering tux Linux workstations. In both cases, *all communication between the client and the server must be through the socket API with real IP addresses, i.e.* ***you cannot use the 127 loopback IP addresses, e.g. 127.0.0.1****. Your computer must be connected to the Internet and you must be able to find the real IP address assigned to your computer. Use that real IP address for creating your sockets and for all communication between the client and the server.*

If you use Auburn University's tux Linux workstations you should use different tux Linux computers for the client and the server processes. The tux computers that are available for you use through remote access are tux050-tux065, tux237-tux252. Altogether, there are 32 tux machines. If you are using the tux computers, send me email with all your group member names and I'll assign to you open port numbers for the tux computers so that you can use and avoid interfering with each other's message transmission.

In both environments that you use for your execution environment, capture the execution trace of the programs. In Linux, use the `script` command to capture the trace of the execution of your FTP client and FTP server programs. The execution traces must contain information when packets are sent or received, when packets are corrupted, and when packets are lost. Sequence numbers and other relevant information on the packets must be printed.

Print the content of the input file read by the client program and the output file received by the server program.

Submission

Submit your source codes, the script of the execution traces of the programs, the file that was sent by the server and the file that was received by the client. ***Make sure that all the group member names are clearly marked in the source codes and other files***. Submit these in Canvas on or before the due date.