COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 4320

Introduction to Computer Networks

Summer 2015

Project 1

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

Due: 11:55pm July 6, 2015

Objective

The purpose of this assignment is to implement a reliable File Transfer Protocol (FTP) service over the UDP transport service. This project will help you understand some of the important issues in implementing reliable data transfer protocols. You will write the reliable FTP client and server programs based on an Alternating Bit Protocol (ABP) that will communicate over 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, (3) recovery from lost packets, and (3) emulation of packet errors generation and detection.

Overview

In this project you will implement a reliable FTP client and server programs that ***must be written in C or C++*** and execute correctly 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 with a specified probability and lose packets). The overview of these software components is show in Figure 1 below.

The reliable FTP client and server program must have the following features, including an alternating bit protocol to ensure that the packets and received reliably.

The FTP client initiates the communication by sending an FTP request to the FTP server at a specific IP address, using 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. The FTP request message will be of the following form:

```
PUT TestFile
```

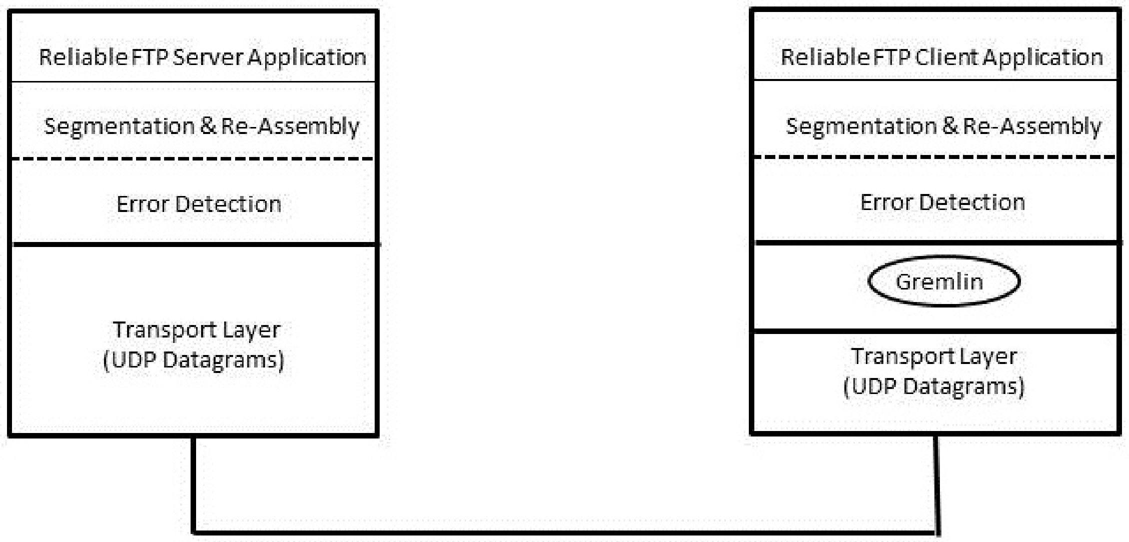
Reliable FTP Server Application
Segmentation & Re-Assembly
Error Detection
Transport Layer
(UDP Datagrams)
Reliable FTP Client Application
Segmentation & Re-Assembly
Error Detection
Gremlin
Transport Layer
(UDP Datagrams)

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 may be 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, which alternates between 0 and 1. 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 being 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. After it sends each packet, it will wait for a positive acknowledgement from the server before it sends the next packet. 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, ACK/NAK (acknowledgement) and data to indicate that it is sending and receiving the packets correctly, i.e. print each packet (say, only the first 48 bytes of data) that it sends and receives.

The reliable FTP server program will respond to reliable 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 sequentially. After it receives a packet and verifies that it is correct, it will send an acknowledgement. Each packet is processed by the error detection function that will detect possibility of error based on the checksum. 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
```

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

Alternating Bit Protocol

The Alternating Bit Protocol (ABP) allows the client to send error-free data to the server despite unreliable physical networks that can corrupt and lose packets. The client will send one packet at a time and wait for an acknowledgement before sending the next packet. Since packets can be corrupted or lost, the server will need to determine if the packet is corrupted, lost or duplicated. In order to perform these checks, the client will send additional information in the header of each packet. The header of each packet will contain the following information: (1) sequence number (0 or 1), (2) checksum, and (3) acknowledgement (ACK/NAK). You will have freedom in designing the packet header format.

Since packets can be corrupted, the client will first compute a checksum for the data in a packet and insert the checksum in the header of the packet. You can use any algorithm for computing checksums. When the packet is received by the server, it will compute the checksum using the same algorithm and compare the checksum that is in the header. If the checksums are identical, it can assume that there is no error in the data, in which case, the server will send back an acknowledgement and the sequence number of the packet that it is acknowledging. If the checksums are not the same, then the server will send back to the client a NAK and the sequence number. If the server detects an error in the packet, **it must print out in its output trace that the packet has errors with the packet sequence number**. The server will then drop the packet and not pass it to the function that reassembles the data stream.

The alternating bit protocol function that receives the packet must check if it contains the expected sequence number. **It prints the sequence number in the output trace** and indicate if there are lost packets. Since packets can be lost, after the client sends a packet, it will set a timer before blocking to receive the ACK or NAK from the server. ACK means “The packet was received OK, so send the next packet.” Assume that the ACK or NAK is never lost or damaged. When the ACK is received, the sender will increment its sequence number modulus the maximum sequence number. Since this is an alternating bit protocol, the sequence number is either 0 or 1. If the timer expires, the client will assume that the frame is lost and retransmit the frame.

You must set your time-out value to be no more than five times the round trip time and no more than 20 milliseconds. If the timeout value is set too long, the sender will block for a very long time and the throughput will degrade. The timer alarm will interrupt the client process that is waiting to receive the acknowledgement. When the timer times out, the client must retransmit the previous packet that it sent.

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 or pass the packet as it is to the receiving function.

If the gremlin decides to *lose a packet,* then the server's alternating bit protocol will not send an ACK back to the client. 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 of your alternating bit protocol will handle errors in packets by sending a NAK packet with the sequence number. The sender will then retransmit the packets that were NAKed.

Testing

Run the FTP client and FTP server programs with the alternating bit protocol for reliable 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 on different tux Linux computers. Capture the execution trace of the programs. In Linux, use the `script` command to capture the trace of the execution of the FTP client and FTP server programs. The trace must contain information when packets and ACK/NAK 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 server program and the output file received by the client program.

Submission

Submit your source codes and the script of the executions of the programs in Canvas on or before the due date. You will also demo your programs to verify that your programs execute correctly.