COMPUTER SCIENCES AND SOFTWARE ENGINEERING
AUBURN UNVERSITY

COMP 4320

Introduction to Computer Networks

2021 Summer Mini-Semester II

Project 1

Implementation of a Simple Web Service over the UDP Transport Service

Due in Canvas: 11:55pm Oct 28, 2021

Objective

The purpose of this assignment is to implement a simple Web service over the UDP transport service. You will write the Web client and server programs that will communicate over your own computer or the College of Engineering LAN. 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 a simple web client application and a simple web server application that ***must be written in C or C++*** and execute correctly either in your own computer(s) or in the COE tux Linux computers. Submission of these client-server programs written in any other languages will not receive any credit. You must also implement segmentation and re-assembly functions, an error detection function and a gremlin function (that can corrupt packets with a specified probability). The overview of these software components is shown in Figure 1 below.

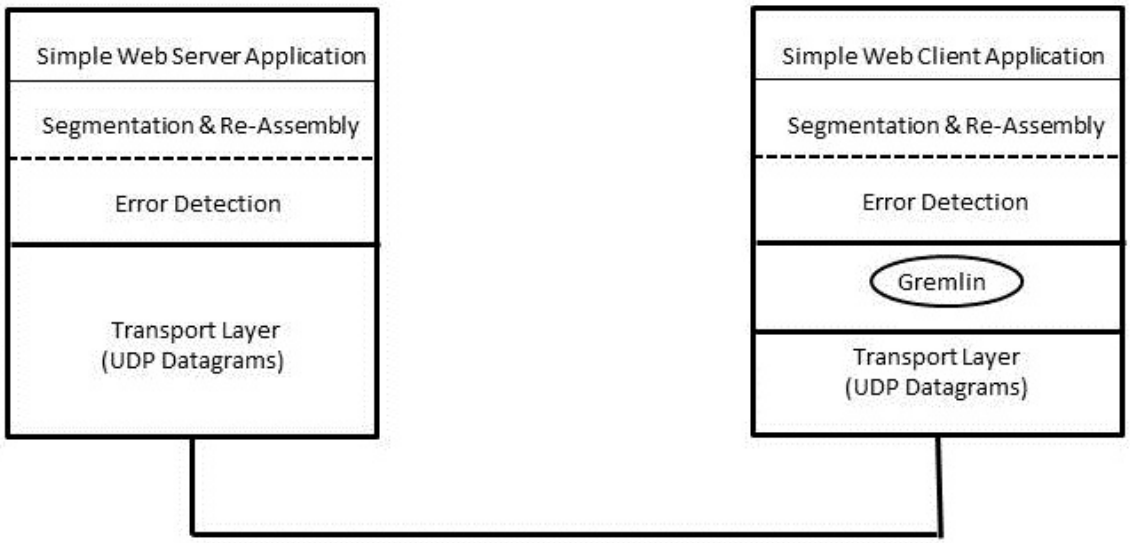

Figure 1. Overview of the Software Components

The Web client initiates the communication by sending an HTTP request to the Web server. This outgoing HTTP request is not processed by the segmentation and re-assembly, error detection or the Gremlin function. The HTTP request is sent through the transport UDP datagram socket to the Web server.

At the Web server host, the HTTP request is also not processed by those functions at the Web server host. The Web server will then process the request, e.g. GET command, by reading the file requested by the Web client. 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 *512-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. The packet is then passed to the error detection function which, at the server (sending process), will compute the checksum and place the checksum in the header. The packet is finally sent via the UDP socket to the Web client.

When the packet is received by the Web client host UDP socket, the packet will be processed by the Gremlin function which may randomly cause errors in some packet. This will emulate errors that may be generated by the network links and routers. The packet is then 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. The file is then displayed by the Web client application using a display software or browser.

Descriptions

This project introduces you to network programming in *your own computer* or the College of Engineering Unix computing environment (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.*

Use the UDP C/C++ client and server programs presented in class. You need to modify the programs to ensure that they run correctly in your environments. For example, you must change the server IP address to the address of the machine in which you will be running the server program. Also make sure that the correct ports are used.

Now, modify the programs to implement a *simple Web service* as follows:

1. Modify the client program so that it will send a HTTP request to a simple Web server to retrieve a data file. The HTTP request message will be of the form:

   ```
   GET TestFile.html HTTP/1.0
   ```

   The server will send HTTP response messages in *512-byte* packets until the end of the file (see below). The client will then receive each 512-byte packet in a loop and writes them into a file sequentially. When it reads a 1-byte message with a NULL character that indicates the end of the file, it will then close the file. Add print statements in the client program to indicate that it is *sending* and *receiving* the packets correctly, i.e. print the messages that it sends and receives.
2. Modify the server program so that it responds to clients' HTTP requests. The server constructs HTTP response messages by putting header lines before the object itself that is to be sent. The 4 header lines are supposed to be of the form:

   ```
   HTTP/1.0 200 Document Follows\r\n
   Content-Type: text/plain\r\n
   Content-Length: xxx\r\n
   \r\n
   Data
   ```

   (note: `\r` is a carriage return, `\n` is a line feed, `xxx` is the number of bytes in the HTML file being sent and `data` is the requested HTML file)

   The server then reads the requested HTML file (an ASCII file, must be at least *50 Kbytes*), put them in a buffer and sends the content of the buffer to the Web client who made the request. The HTTP response messages are sent in 512-byte packets until the end of the file. 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 print statements in the server program to indicate that it is *receiving* and *sending* the packets correctly, i.e. print the messages that it receives and sends.

Gremlin Function

Your program must allow the probability of damaged packets to be input as an argument when the program is executed. This packet damage probability is passed to your Gremlin function. You will implement a gremlin function to simulate two possible scenarios in the transmission line: transmission error that cause packet corruption and correct delivery. When the receiving process receive each packet, it first pass the packet to a gremlin function which will randomly determine whether to change (corrupt) some of the bits or pass the packet as it is to the receiving function. 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 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 of the client program 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 .5, the probability of changing two bytes is .3, and the probability of changing 3 bytes is .2. Every byte in the packet is equally likely to be damaged.
The packet is then passed from the gremlin function to the error detection function that will check for errors in the packet.

Error Detection Function

The sending process, e.g. the Web Server, 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 Web client, 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, you must not try to correct for errors in the packet using any network protocol or scheme; you are required only to indicate that a packet is in error.

Testing

Run the modified UDP C/C++ client and UDP C/C++ server programs, with the segmentation and re-assembly, error detection and gremlin functions, 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 members and I'll assign to you open port numbers for the tux computers so that you can 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 the UDP C/C++ client and UDP C/C++ server programs.

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, 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. Submit these in Canvas on or before the due date.