Computer Networks I
Programming Assignment 2

Due: 28 October 2016

Overview

This project involves implementing a simple file transfer program using the sockets API. You will be required to use UDP transport, therefore you will need to implement your own reliability mechanism as well.

Project detail

Your submission will include two programs: a server, and a client. The server should listen on a UDP socket, then begin sending chunks of a specific file over UDP when the client initiates a download. The client should then save a copy of the file when it is done receiving the chunks.

File server program

The server should accept two command-line arguments: the name of the file to serve, and the port number to listen on. The server will send that particular file to any client which connects, unlike in FTP where the client decides which file to retrieve. Because UDP is connectionless and unreliable, you must implement mechanisms to (a) allow multiple clients to download simultaneously or in series, and (b) recover from lost or rearranged chunks. It is up to you how you implement these features. Some ideas are given in the Hints section below.

File retrieval program

The client should accept three command-line arguments: the IP address or hostname of the server, the port number that the server is listening on, and the filename which you would like to save the retrieved file under. Then, it must somehow retrieve the file from the server using the UDP socket, and save it under the given name. You must design some mechanism which will allow the server and client to communicate and determine when the end-of-file is reached.

Other specific requirements

- Use either C or C++ for this assignment.
- Your code must compile and run on the machines in the CS department labs.
- Code should be modular, and must follow the `.h`/`.c` modularization standards and naming conventions discussed in class.
- You must comment your code so that it is understandable. If your code is cryptic and undocumented, points will be deducted.
- Include a Readme which describes how to use your program, and a Makefile to compile your code. The Readme should also explain any interesting design or implementation decisions you have made.

- Do not use any external libraries in your implementation (only the standard C/C++ libraries and basic system libraries). If you are unsure if a particular library is allowed, do not hesitate to ask.
- Test your code throughly. It is also a good idea to run it under `valgrind` to ensure it is free of memory leaks.
- Make sure to handle all possible error conditions in your code. For example: file not found, no permission, port in use, host not found...
- You are to submit a tarball (.tar, .tgz) to Canvas including your code, the Readme, and the Makefile. Name the file using your own full name and the suffix `_Assignment2` (e.g. `JayMisra_Assignment2.tgz`).
- You'll be required to do this assignment in groups of not more than 2 individuals. If this is the case, the tarball (.tar, .tgz) submitted should include the full names of each group member separated by underscore and with the suffix `_Assignment2`
- Do not include the executable in your submission.

Hints

- To easily implement reliability and sessioning, you may want to make your file transfer protocol receiver-driven. In a sender-driven protocol, the sender bursts data to the receiver in sequence, then the receiver acknowledges the packets it receives. In a receiver-driven protocol, the receiver requests particular data chunks from the sender, then the sender replies with the chunks requested.

  The advantage of a receiver-driven protocol is that instead of maintaining state for several clients at the server, you can instead maintain all state within the client.
- The easiest way to implement reliability is with a stop-and-wait protocol. However, this will result in much slower downloads than TCP. Though it is not required, you may try to implement some sort of pipelining.
- You do not need to implement hash-checking or checksums in your program. Though UDP is an unreliable protocol, it still uses checksums on each packet. Therefore, mangled packets will not be delivered to your application. You only need to ensure that you get all of the chunks, and that they are in the right order.
- You can test your program using a utility like `md5sum`. Transfer a file to yourself, then check if the MD5 hash of the copy matches that of the original.
- You may want to add some kind of “debug mode” to your server program, which will simulate packet loss and reordering to facilitate testing your protocol's reliability. For example, debug mode may cause the server to randomly drop packets before sending them, or randomly choose to send chunks in the wrong order. This will allow you to check your reliability implementation without waiting for an error to occur on the network itself.
- You can also fake network unreliability without implementing a debug mode. The following page shows some ways to configure the Linux network stack to force delays and loss:

`http://stackoverflow.com/questions/614795/`. Unfortunately, you cannot use these programs on the lab machines, so you will need to have your own Linux installation to use them. Sufficient experience in Linux administration is recommended if you want to use this method.

Grading rubric

Design, delivery, and style		**30 points**
Modularization	Code is modularized according to the standards described in class and used in common practice.	5
Documentation	Code is thoroughly commented. Each method has a comment to describe its functionality and what inputs are expected. More complicated functions have comments throughout to explain the logic.	5
Readme	The Readme explains how to use the programs, and explains major decisions involved in the programs' designs and implementations. The Readme includes a description of the reliability protocol that is implemented.	10
Makefile	The Makefile compiles both executables without error.	10

Program functionality		**70 points**
File delivery	The program is capable of transferring a file over UDP. If this requirement is not met, none of the following requirements can be satisfied.	20
Session support	Multiple clients can connect to the server at the same time. Multiple clients can connect to the server in series without the server being restarted.	15
Reliability and integrity	The protocol is capable of delivering a file without corruption. Even when the network is dropping or rearranging packets, the file is delivered intact.	15
Correctness of interface	The programs implement the interface specified above. The command-line arguments are interpreted correctly and handled as they should be. The client is able to connect to the server regardless of whether a hostname or IP address is provided.	10
Error handling	The program gracefully terminates when an error occurs, whether it be due to malformed input or an unexpected event.	10