CSC/ECE 573 (001) – Internet Protocols

Fall 2018

Project Proposal

Team

James Cross

Daniel Monazah

Nitesh Sainath

Deepak Patil

Introduction

Bandwidth is a very valuable resource, and networks must not waste it on redundant transfers and unnecessary overhead bits which are inherent with certain protocols. Especially in large enterprise networks, a lot of the bandwidth is wasted on control fields of “heavier" protocols, sacrificing throughput without significantly improving the user experience. For example, TCP introduces high overhead into the system because of all of the services it provides. However, these services are not always required, and a higher throughput may be more desirable. In addition, many versions of TCP discard out-of-order packets, requiring retransmissions of correctly-received packets.

Problem Statement

To develop a solution for efficient utilization of available bandwidth and reduce waste by implementing a lightweight solution in the application of remote directory exploration and file transfers.

Project Objectives

In this project, we will implement a low-overhead file transfer server using UDP which also supports out-of-order reception for segments. Clients will be able to interact with the server through a UNIX-like Command Line Interface (CLI) to explore the accessible directories and files on the server. This mechanism must be lightweight as well.

Implementation

Since this project deals primarily with file transfers, we use the client-server model to explore the directory structure and pull data.

- We implement a file transfer protocol built on UDP which is reliable and supports out-of-order reception within a window of blocks so that retransmissions can be kept to a minimum. This is done through a reliable transfer mechanism at the application layer which utilizes Selective Repeat ARQ. With a suitable window size, timer for each packet in the window, and a NAK scheme for missing or corrupted packets in the window buffer, we can optimize the performance for our application.

The resulting custom Trivial File Transfer Protocol (TFTP) code must be running on the client and the server.

- The file directory structure is represented using a JSON tree since it is lightweight, faster to transfer, and uses less bandwidth than a text-represented markup language file (html or xml).
- The client uses a CLI to interface with the server to explore the directories and files using predefined commands (`ls` for listing files and `cd` to change directory). JSON objects are transferred to the client once requested.
- Debugging and verification are done by printing out information to the console during runtime to see if any timeouts occur on the server side while transferring a block of the file or any serialized JSON strings. Similar information will be printed on the client side to ensure proper functionality. Other debug information is printed as needed.

Protocols used: UDP (RFC 768) and TFTP (RFC 1350)

Languages: Python3, C

Tools: PyCharm, Atom, Eclipse, NetBeans IDE

Project Evaluation

- Infrastructure setup required for the project including the establishment of a client-server architecture.
- The client must respond to predefined commands to correctly depict and navigate the directory structure as seen on the server side. The output of the terminal will be used to verify correct functionality.
- The downloaded file should not be corrupted or received out-of-order as seen by the user. Out-of-order reception is supported, which can be observed by intentionally dropping received datagrams and not acknowledging them. Packets that are successfully received and acknowledged will not be retransmitted. Only negatively acknowledged or timed-out packets will be retransmitted. This can be seen when the client prints out the packet it intentionally deemed corrupted, and the same packet is retransmitted from the server after receipt of a negative acknowledgment or time out.

Project Demonstration

- The server and client can be set up across different systems or through virtual machines. We will set up two different machines: one as the server and the other as the client. (8%)
- A folder will be made on the server side and a text file will be stored in it. Then, the server process will be run. (6%)
- Next, the client process will be run, and the appropriate commands will be entered. (6%)

- In order to check the different folders, hit `ls`. This should display the name of the folder(s) the client can navigate to using the `cd` command. The client can then initiate the custom TFTP file transfer using the `get` command, followed by the file name. (30%)
- The client must then notify the user that the transfer is pending by displaying "Downloading, please wait..." The client will then indicate when the file has been successfully received or if the transfer failed. (20%)
- When a packet received on the client side is intentionally dropped, it prints a message to the console to let the user know, and the server either receives a NAK or times out for that particular packet (the server will also print out messages so the user can see what is going on). The protocol will continue transmitting until the file is successfully transferred from the server to the client or until a designated timeout occurs (for example, stop trying if the file couldn't transfer within five minutes). (30%)