CS176B: Network Computing

FTPxpress

Project Deliverable B

Ben Patient
Danish Vaid
Jake Can

March 4, 2018

Abstract

As we are not finished with our project, there is no content for our abstract.

Team Members:

- Ben Patient
- Danish Vaid
- Jake Can

Team Size Justification: Having been in the same capstone team for CS 189A/B the three of us established a good workflow and our similar schedules let us coordinate and work well together. We hope to use these advantages to build a solid project and have many ideas for features we could implement and networking challenges for us to overcome.

Introduction

The vanilla FTP platforms we use today are good, yet rigid. We would like to implement a file transfer protocol that is versatile and customizable to how the user would like to handle his or her file transfers. Our implementation is light weight and manages more back and forth between the user and the program to ensure the user's preferences are heard. *Note: We also want to add info about vanilla FTP's implementation and our changes to that in the intro.*

Project Overview

For our project, we are implementing our own version of a FTP server, client, and protocol. In doing this we are also implementing custom headers to tailor the protocol to our priorities and for the features we want to implement. These headers would include information on file, packet ordering, payload size, etc. Looking more into these headers, we decided that it would be better to have a custom handshake that exchanged meta-data, file priority, and other such data explained below. Upon the basics of FTP, we also plan to add network security features (for user privacy and ensuring data integrity) and support for parallelized multi-file upload/download for performance. We hope for our server program to server multiple clients, this feature would have us identify specific users to differentiate who is sending what packet to ensure there are no errors.

Project Goal Summary

At minimum we want to deliver a basic FTP protocol with our custom headers and a minimal terminal UI for usage. Having finished the features discussed above, we would like to support basic commands such as list files, delete file, move file, and such. Finally, our physical deliverable will be our source code, and a report that describes the project, its completion and usage, and performance analysis.

Custom File Handshake

This process will set metadata for each file transfer. The benefit of including a handshake is to reduce the number of headers, therefore reducing overhead on each packet sent so there is more data per packet. Our metadata would include file name, file type, FUID (file unique identifier), client ID, and security keys. The client ID field will pair the file with the client. Knowing this ensures which file is being transferred by who, and can also be used to prevent collisions of file names. Lastly, the security key sent over will be used for asymmetric encryption, to make sure the connection is confidential.

Custom Headers

Note: Many of our custom headers became a part of the custom handshake to improve performance and efficiency.

Sequence Number

This is will be a 4 hex-decimal character describing the sequence number of this packet. This is used to rebuild the packets of the file in the correct order of data of the original file.

File Unique Identifier (FUID)

This is will be a 4 digit number assigned to each file transfer. The server will assign an ID to each file that is being uploaded or downloaded. This ID will pair with the file name and type so the header will only include the ID, and not the file name and type, thus reducing overhead.

Packet Type

This is will be a 1 character information that tells whether the packet contains command data, meta-data, or file data. We can use this character flag to know what logic to run on the packet payload.

Performance Analysis

We have not gotten to performance testing yet, but this section will contain an analysis on the performance of our FTP client. We plan to test the effects of sending encrypted packets vs unencrypted packets to see the effects of the increase in payload due to the encryption hashing. We want to test our parallelized upload/download methods by comparing the results of sending files over serialized and parallelized. We are going to load test our application by testing large payloads in a single transfer and also testing large numbers of concurrent connections and parallelized file transfers. We want to see how many files we can handle at once and the maximum size of files we can transfer. Once we have gathered all of this data we plan to compare the performance of our client to the performance of popular file transfer options like Cyberduck and Filezilla.

Timeline

Week 3: Design our headers and protocol schematics
Week 4: Successfully send over a file from 1 user to another
Week 5: Implement basic terminal UI and command support

Week 6: Implement parallelized upload/download
Week 7: Implement network security
Week 8: Run performance testing methods
Week 9: Refine code and finalize project/report.

We have fallen behind on our timeline slightly and are current in the week 6 phase of implementing parallel transfer. We spent the past week focused on other commitments, since those have ended we should have more time to catch up in the following weeks.

Conclusion

As we are not finished with our project, there is no content for our conclusion.

Comments

One user interface issue that we have ran into planning is listing the files of the client and server. It would be hard to separate the commands for a user wanting to list the user's files or the server's files. Going forward, it would be confusing for the user to keep track of the current working directories of the server and the client concurrently.

Another hurdle we have to handle is multiple clients connecting to the server, without disturbing the socket connections. This will take a lot of testing, as we do not have a lot of experience with multiple connections.