# Lab 3 – Custom TCP/UDP File Transfer Utility over Python

Authors: Angela Zhou ([azhou@hmc.edu](mailto:azhou@hmc.edu)), Bruce Yan ([byan@hmc.edu](mailto:byan@hmc.edu))

## Overview

Using the native unix application, scp, to transfer files may be reliable but still limited to the limitations of ssh. We have decided to use Python for the implementation of socket transfer. Our server sends and receive data using both TCP and UDP protocols. TCP is primarily used for transferring of metadata and UDP is primarily used for transferring of data chunks (pieces of files).

## Approach

We have decided to transfer the files using two separate streams, transferring the metadata using TCP and transferring the actual file pieces using UDP protocol. Since we know that metadata about the file, such as file size and checksum, need to be preserved; we’ve decided to send that through the TCP protocol. We decided to use Python’s file read function to read a certain blocksize and transfer the amount that’s read.

A summary of the metadata information is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| TCP - metadata | | UDP - data | |
| Client | **Server** | **Client** | **Server** |
| File name | Resend data information | Send data chunks | Receive data chunks |
| Total # of blocks |  | Block Index (marks which block # is being transferred) |  |
| Checksum of file |  |  |
| Checksum of each block |  |  |
| Total Size of file |  |  |
| Size of each block |  |  |

## Functions

generateMetaData – This function will take in a file and generate the following metadata information:

* Size

## Modules/Libraries

In our project, we made use of the following public python libraries

* socket – this is the low-level networking interface for socket servers and clients
* os – contains portable tools to help the user use operating system dependent functionality, such as getting the size of the file
* hashlib – we used md5 checksum function in this security interface to ensure that files sent and received matched the same hashed md5 checksum
* sys – we used sys to grab the arguments (such as file name) that were being passed into the system

## Metadata Transfer (TCP)

The idea behind our TCP protocol implementation is to ensure the successful transferring and receiving of important information, such as blocksizes and checksums. To store all the metadata, we’ve decided to use an array of arrays to send the data over. We chose arrays because they are relatively cheap in cost and quite versatile in terms of functionality.

## Data Transfer (UDP)

The idea behind our UDP protocol implementation is to ensure the sending of packets as fast as we can, and send as many packets as possible. If packets are dropped along the way due to either delay, link loss, or an inconsistent connection, our TCP implementation will come in hand if we drop packets along the way due to delay and link loss, we can use our TCP implementation to confirm and re-transmit the missing pieces. Instead of sending and checking the checksum at the same time, we propose to send all data using UDP first, then verify using TCP at the end

# Appendix

## Presentation

Our presentation is hosted on Google Slides (requires HMC login): <https://docs.google.com/a/g.hmc.edu/presentation/d/13wsIZ2vtJckYLAp41PUPLskLWEJXtkDF7xMN3odpDWM/edit?usp=sharing>

## Delete me below!!!

May not need to check checksum of every chunk because per UDP, <http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/>, either it gets to the destination wholly or not at all. It will never be a partial send. The thing that messes UDP up is that it will scramble the order you send the chunks in, so the key is to remember that the order that the chunks are sent are important, we need a way to keep track of that.

http://stackoverflow.com/questions/13993514/sending-receiving-file-udp-in-python