MPTCP FINAL PROJECT

CPRE 489

DECEMBER 6, 2021

FINAL PROJECT DONE BY:
Kent Mark and Christian Williams

Table of Contents

Quick Intro:

This write-up details our implementation of the client and server programs that implement a form of MP-TCP type data transfer. The implementation described in this report was done for our final project for CPRE 489 and was meant to demonstrate our ability to make use of multiple connections to transport portions of data. As such, this document will provide the reader with an overview of the MP-TCP, a description of the two deliverables we were tasked with developing and subsequently implementing, and a brief explanation of the challenges we faced in completing this assignment and the steps we took to overcome them.

An Overview of MP-TCP:

MP-TCP is short for Muli Path TCP and can be described as a method that allows data transfer using various TCP connections. Multi Path connections are also capable of normal, single mode, TCP transfer when a given connection is incapable of utilizing MP. From this, one can correctly assume that a MP-TCP connection is only viable when both the source and the destination have the ability to use MP protocol. The way this kind of transfer normally works is by establishing a traditional two-way connection and then actively searching for additional connections that are currently available on the host. The hosts then make sure that extra paths are available and if so, additional TCP sessions are created on these paths. These connections are used to create multiple data paths and provide the infrastructure for sending allotted segments of data simultaneously between hosts. We wanted to conclude this segment by explaining that an extremely important part of MP is its ability to make connections seem like they are uninterrupted and making then behave like a single stream of data. This was a relatively quick and succinct overview of MP-TCP, a more robust explanation can be found online.

An Overview of the Project:

This project required us to implement a simplified version of the MPTCP that was focused using multiple paths, and mapping data to those paths. As per the provided pdf document the overall objective of this project was to establish 3 TCP data subflows between a client and a server. We were tasked with implementing a client program that will generate 992 bytes, numbered 0 through 991, with 16 repetitions of: the ASCII digits 0-9, the ASCII characters a to z, and the ASCII characters A-Z. Step three of the overall design required that the client then establish 4 TCP connections to a server program that will each be responsible for a different action, and also log the numbers used from 992 bytes to the sequence numbers on all 3 connections. Finally, we were also tasked with implementing a server process that will also have to accept connections, namely 4 connections, one being the control connection, and the other three being the TCP data subflows.

Deliverables

Client.c:

The client program is divided into a few main categories that are ach responsible for a specific task associated with the overall client design. These categories are a routine that creates a data buffer, routines that manage the connection, a method that uses child process to write over TCP sockets, and a main routine that would act as the control structure that manages the state like activity within the program.

The client is responsible for handling the creation of the data buffer as the first step in the process. This is done via a pointer that is allocated with a size of 992 characters. This is used to store the generated data buffer consisting of thee 16 repetitions looping through the sets of 0-9, a-z, and A-Z.

Server.c:

The server is responsible for accepting incoming data over three connections in addition to accepting a control connection. The control connection accepts DSS packets containing information needed to reassemble incoming data from multiple connections. The server program consists of different routines for initialization, creating and binding sockets, accepting incoming connections for data or control, a routine to implement a forked child process, and a routine that parses data for capturing and reassembling the incoming data packets.

Project Challenges and Difficulties:

We never really had a particular issue or challenge that we felt needed to be expanded upon in detail. Aside from minor bugs or syntax errors our implementation of the two programs were relatively problem free.