Ashraf Shaikh
Ben Simon
CPRE 489
Section: C

Final Project Report

Overview of Document

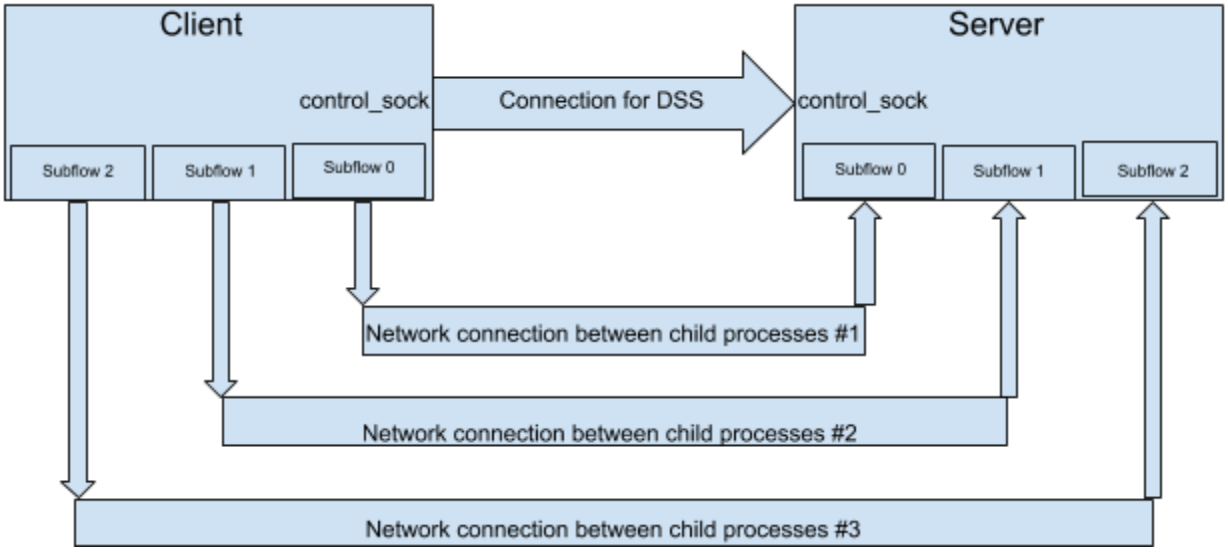
This document describes an implementation of MultiPath TCP (MPTCP). The document is divided into several sections. Design Requirements details the specific requirements of the protocol implementation. High Level Design describes the main components of the program. This section should give the reader a good understanding of key components but will not over burden them with the details. MPTCP Protocol explains this program's implementation of MPTCP. The Conclusion wraps up the document.

Design Requirements

The protocol described in this document is a simplified version of the MultiPath TCP (MPTCP). For demonstration, a program implemented the simplified protocol as defined in the MPTCP Protocol section. The program transmits 992 bytes from the client to the server. These will be the numbers 0-9, the lowercase letters a-z, and uppercase letters A-Z repeated 16 times. The server and the client will be located on the same machine. The client will only need to send the data once before quitting. It is assumed that the program will not need to worry about data lost or corrupted in transit. The only other requirement is that the program runs within the computer's memory without crashing.

High Level Design

Block Diagram



Client

Upon start up the client will establish connections and assemble the data to send. The client creates three socket pairs for interprocess communication. These sockets will be used for communication between the subflow and the client. The client will then use fork 3 times to create each subflow. In parallel, the subflows will start a connection with the corresponding server subflow. The server and subflow addresses will be given to the program on the command line. If any of these connections fail, the client will exit. The parent of the subflow will establish a connection with the server. This connection is for sending the DSS mappings.

After all connections are established the client will create the data for parsing. The data will be 992 bytes of the characters 0 to 9, lowercase a to z, and uppercase A to Z repeated 16 times. The client will take a round robin approach to send data over the subflows and sending the DSS mappings. Each subflow gets 4 bytes of data. Subflow 0 gets the first 4 bytes, subflow 1 get the next 4 bytes, subflow 2 the next 4 bytes until the server repeats the sequence with subflow 0. The client will send the DSS mappins for a round of the subflows after sending data to subflow 2. This will continue until all the data is sent. We assume that overall data length will be divisible

by 4 to prevent the need for padding. Once the client has finished sending data, it will close all connections and exit.

Server

The server starts by creating 3 sockets for IPC(Inter Process Communication) for communicating with child processes. It achieves this by using int socketpair(int domain, int type, int protocol, int sockets[2]) function that creates a socket pair which can be used by parent and child process for IPC. These unix sockets are made non blocking for achieving parallelism from the three connections. The server then creates a network TCP socket for receiving DSS from client. It then binds this socket to a fixed IP address so that its reachable by the client. The IP address shall be provided from the command line.

The server now proceeds to fork into three child processes. This is the most crucial part of MPTCP logic which ensures there are multiple TCP connections between server and client. The server's child processes then create their own network sockets, bind to the IP address of the server, though they'll use different TCP port numbers and wait for data from their corresponding client child processes. The read( ) used by child processes may be blocking. Once they receive data, they just directly send it to the parent server process.

The parent server process is the most important part of the logic. It loops through the connections of the three child processes to see if could get any data from them. It's also responsible for piecing together the output message by mapping from the DSS Sequence numbers, Subflow sequence numbers and Subflow number. The parent server process creates three mapping arrays for each of its three child processes where it stores the mappings for the data's final position in output string based on information provided from DSS packet received over the network socket. It also keeps track of how many bytes of data have been read from each of its child processes and how much mapping has been done. This is essential as the unix sockets were non blocking. When it sees that the DSS mapping has been done for packets from a particular child process, it attempts to read the data and add it to the final output string.

Once the client finishes transmitting data, the server will simply wait for client to connect again as the accept( ) is the server is a blocking call.

MPTCP Protocol

The program implements a simplified version of the MultiPath TCP (MPTCP) protocol. The protocol used multiple instances of TCP to increase its throughput and reliability of data transmission. Each instance of TCP is called a subflow. Each transmission over the subflow

sends a 4 byte chunk of the overall data. After the client connects to the server, each subflow will alternate sending the next portion of data.

To assemble the data, there will be a seperate control connection. This control connection will send Data Sequence Signal (DSS) to the server. This contains the mapping for each burst of data to its location in the assembled sequence. The DSS format is as follows:

Data Sequence Number (DSN) - 4 bytes
Subflow Sequence Number (SSN) - 4 bytes
Subflow Number - 1 byte

Data Sequence Number (DSN) is the starting byte index where a subflow burst should be stored in the output. Subflow Sequence Number (SSN) is which subflow sequence should be put at that DSN location. The subflow number indicate which subflow this mapping is for.

Each subflow is a single process in this protocol. Thus subflows can receive or transmit at the same time. Each subflow will communicate with a parent which splits the data (Client) for sending or reassembles it (Server).

Since this protocol will be implemented on a single machine, we assume that there will be no dropped packets or out of order bytes. This protocol will not respond with any acknowledgments. This allows for unidirectional communication between the server and the client.

If any connection cannot be established, the service will exit immediately. Upon startup, the client and server will immediately start a control connection and three subflows. The server will then wait for the client to begin sending data. The client will start sending data as soon as the connection is established, and it will stop by sending a FIN to the server once all the data is sent.

Conclusion

This paper summarizes an simple implementation of MPTCP. This implementation demonstrates the protocol by sending 992 bytes from a client to a server using three subflows.