CPRE-489 DATA COMMUNICATION AND NETWORKING

FINAL PROJECT REPORT

MULTIPATH TCP IMPLEMENTATION

PREPARED BY :
SHUBHAM MAKWANA
LOHITH REDDY KALLURU

DATE:12/07/2021

OVERVIEW OF MPTCP:

This document consists of Introduction, Design Requirement , Levels of design , Protocol and Assumptions
Design Requirement: Gives the details regarding the MPTCP and requirements needed for simplified implementation.
Levels of Design: Provides the details regarding prototype method, processes, execution flows and so on.
Protocol: Provides the details regarding error validations ,inputs given,outputs displayed and so on.

INTRODUCTION:

TCP was a widely used and implemented protocol that is well-known due to its reliability. Now, apart from the advantages of the TCP, designers of TCP had an idea that TCP is not an appropriate option as network links. Now network layer protocols should be separated from the transport layer protocols, and Network reroute was required, but there was the need for the dynamic routing protocols, which could have made this transmission of data less tedious. MPTCP is the answer to that problem. in MPTCP, the TCP connection will be generated according to the congestion of the Network. While designing the MPTCP few things are needed to be considered: Application Compatibility and Network Compatibility. Under Application compatibility, there was a requirement of such a system that can easily get merged with the current system and need not change much. Network Compatibility is multipath TCP

should operate on any Internet path where TCP was possible. In layman terms, it should be compatible with the adjacent systems.

The main reliability is secured connection, reliable data transfer, and easy connection release. Over here, MPTCP has a three-way handshake. The client is responsible for sending the first Syn and then Syn, Ack-Syn, Ack. There would be a tuple with all the data of the connection. After this successful connection, the data is ready to transfer, and over here, the role of the sequence number is to detect the loss of the data and the losses. If the segments are lost, there are many other ways to retransmit those TCP segments through TCP connection. After this whole thing, the TCP connection should be released, and that can be done in a few ways: by sending the RST packet or the FIN packet. The FIN method connection would be closed when both sides (server and client) simultaneously acknowledge that. Now additional sub-flows can be added and connected to the socket and used for the connection. Every sub-flow will have the same working method mentioned above, starting with a three-way handshake and ending with FIN or RST packets. Data that is being transferred in the sub-flow if by any chance is going out of the capacity of the sub-flow (e.g., sub-flow has a capacity of 32-bit characters and data of 64 bits), then this recovery of data will be covered by other sub-flows. Note that multipath TCP will not send data on every TCP because that would consume a lot of resources, and which will result in waste of resources so because the MPTCP uses its consecutive sequence number where every segment sent will have two sub-flows sequence numbers (SSN) and data sequence number (DSN). Multipath TCP can then send some data sequence numbers on one path and the remainder on the other path; old middleboxes will ignore the DSN option, and the Multipath TCP receiver will use it to reorder the byte stream before it is given to the receiving application. In short, MPTCP is the solution to better performance and efficient use of resources which is not achieved in the case of TCP.In the below figure, consider the top and bottom are running on TCP connection individually. Now over here, multipath TCP uses both links and moves most of the traffic to the less congested link.

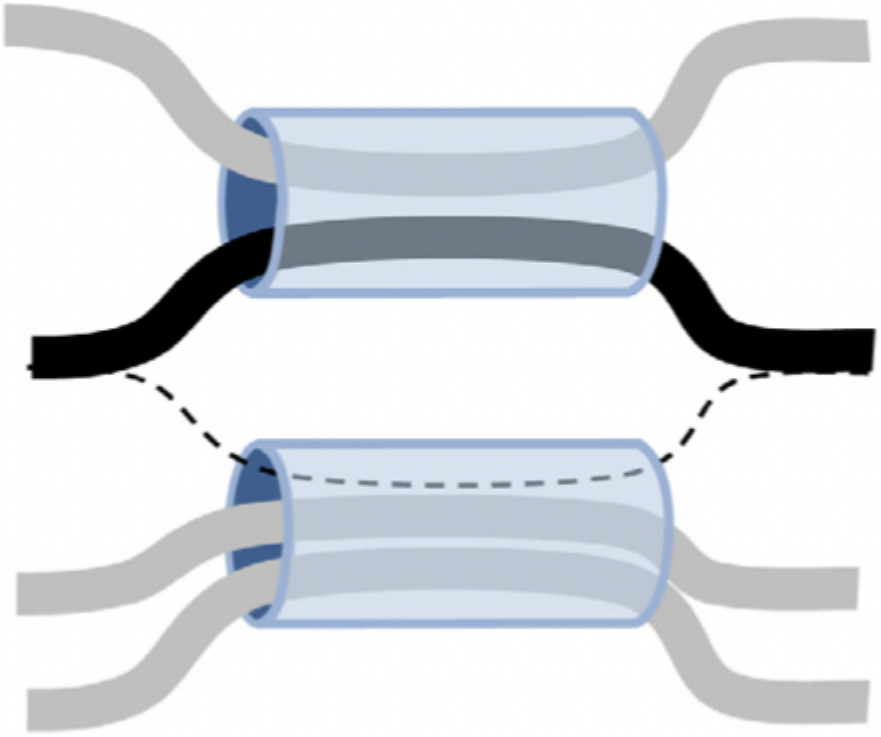
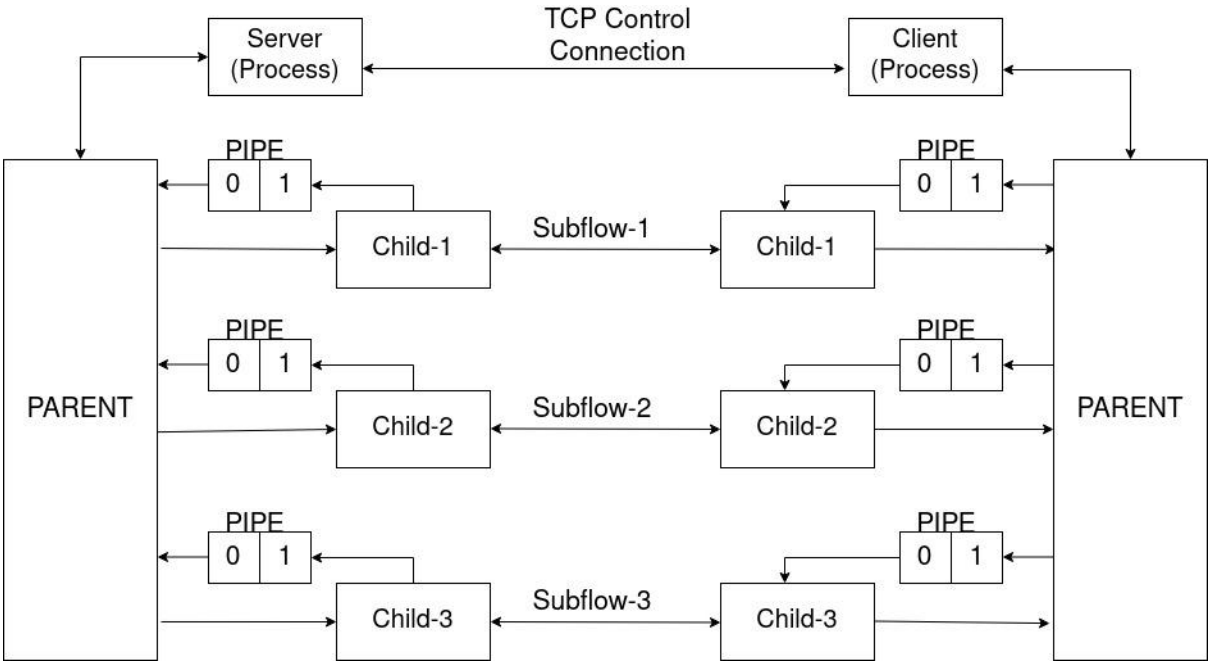


Figure-1 MPTCP with two TCP

DESIGN REQUIREMENT :

1. Create 4 sockets, one for control connection and other three are subflow connections between server and client.
2. Parent processes in both client and server should be forked to create 3 child processes which interact with main socket file descriptors using subflow pipes .
3. Client =>send DSS message before each packet is sent through subflow
4. Client=>Entire data of 992 bytes should be split into packets of 4 bytes each which cover alphabets,numbers(repeated) sent sequentially using different subflows. Store the mappings into a file(dsn,ssn,subflow,data)
5. Server=>Read the DSS message and using port number, read data from that particular subflow. Map the data received using dsn and print the output.



LEVELS OF DESIGN DETAILS :

The flow of the functions are given below:

1. Initially we create a socket connection from server to the client which is a control connection and once it is created successfully , we establish each of the 3 subflow connections which have sockets setup in the client (This can also be done by having all the sockets setup on either side of client and server).
2. Here on the server we are using an **assign method** for creating the structures of serveraddress_len with the IP address 127.0.0.1 for each of the ports(6344,6345,6446,6447).
3. We create 3 pipes and 3 forked child processes on both client and server. On the client,initially a DSS message is sent which consists of data size, port number,sub-flows sequence number(SSN) and data sequence number (DSN). The parent process splits the entire data of 992 bytes and prepares it into packets of size(4 bytes) which is sent through each of the child processes using pipe . On the server, we will read it on the forked child process which writes into the corresponding pipe. The DSS message is read on the parent process of server and based on port number we read the respective pipe in a non blocking way and assign the DSN to the data. Also, the parent process writes back **ack** to the control connection. This ack is read by the parent process in the client and writes the next 4 bytes data into the next pipe . This pipe is read by the corresponding child process of the client and cycle repeats. The main process in the client also creates a log.txt indicating the mapping of DSN, SSN and data

The methods used here are:

Server => **child_write(int fd, int sflow[2])**
Reads the data from the **fd** of the subflow socket and writes into a **sflowPipes[i][1]**

Client => **child_read(nt fd, int sflow[2])**
Reads the data from the **sflowPipes[i][1]** and writes into a **fd** of the subflow sockets

4. **clearSubflows(int process[3])**
   Closes all the subflow processes using SIGKILL once the read is done . It is implemented both in client and server.

PROTOCOLS:

1. It takes input of a custom server address and 4 port numbers as arguments to the server and client . It also throws errors to the user when these inputs are not provided.
2. The error validations provided here are for socket creation , connection errors, pipe creation errors, fork errors, write errors, dsn errors and read errors.
3. The data considered in the client here is hardcoded.
4. When it is run, the server generates an output on the terminal indicating the data received, which subflow is being created and which pipe is being accessed. The client also displays the mapping.
5. It is a simple implementation and does not consider the options

ASSUMPTIONS:

- The checksum is not considered here.
- We consider all data received on the server to be not corrupted.