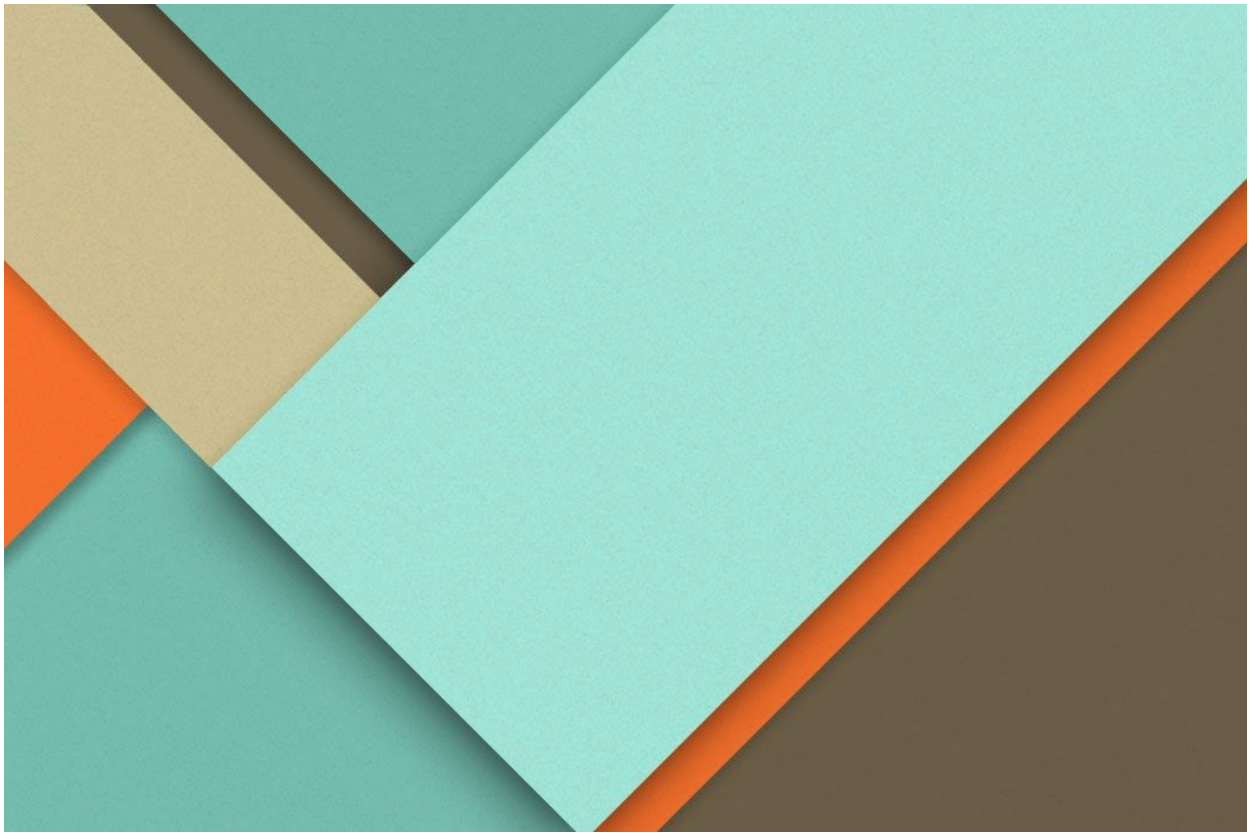

Implementing a Reliable Data Transport Protocol

16.01.2021

Mohamed Samy
(41)

Overview

Suppose you've a file and you want to send this file from one side to the other (server to client). You will need to split the file into chunks of data of fixed length and add the data of one chunk to a UDP datagram packet in the data field of the packet. You need to implement TCP with congestion control.

Goals

Since you've learned about the socket interface and how it is used by an application; by now, you're pretty much an expert in how to use the socket interface over a reliable transport layer, so now seems like a good time to implement your own socket layer and reliable transport layer! You'll get to learn how the socket interface is implemented by the kernel and how a reliable transport protocol like TCP runs on top of an unreliable delivery mechanism (which is the real world, since in real world networks nothing is reliable). This lab should be fun since your implementation will differ very little from what would be required in a real-world situation. The network communication in last assignment was provided through a reliable transfer protocol (TCP/IP). In this assignment, you are required to implement a reliable transfer service on top of the UDP/IP protocol. In other words, you need to implement a service that guarantees the arrival of datagrams in the correct order on top of the UDP/IP protocol, along with congestion control.

Milestones

I. The client sends a datagram to the server to get a file giving the its filename. This send needs to be backed up by a timeout in case the datagram is lost.

II. The server forks off a child process to handle the client

III. The server (child) creates a UDP socket to handle file transfer to the client.

IV. Server sends its first datagram, the server uses some random number generator random() function to decide with probability p if the datagram would be passed to the method send() or just ignore sending it

V. SWhenever a datagram arrives, an ACK is sent out by the client to the server.

VI. If you choose to discard the package and not to send it from the server the timer will expire at the server waiting for the ACK that it will never come from the client (since the packet wasn't sent to it) and the packet will be resent again from the server.

VII. Update the window, and make sure to order the datagrams at the client side.

VIII. repeat those steps till the whole file is sent and no other datagrams remains

IX. close the connection.

Description of the contribution introduced to code:

I. Server:

- Functions:

```
vector<packet> make_packets(string file_content);
// Function to split the requested file contents into packets
//each of size 500B
vector<string> read_from_file(string path);
//reading from file
void sending_packets(vector<packet>& packets, int
clientfd,struct addrinfo *p,struct sockaddr_storage
their_addr);
//sending the packets splitted to the client using udp sockets
bool packet_wiil_be_sent(double probability_of_loss);
//calculate the propabilty of sending the packets
```

- Data Structures:

```
vector<string> lines = read_from_file(server_in_path);
vector<packet> packets = make_packets(file_string_contnets);
```

II. Client:

- Functions:
  - `vector<string> read_from_file(string path);`
  - `void recieve_packets(int sockfd,struct addrinfo *servinfo);`
  - `void write_into_file();`
  - `void send_ack(uint32_t seqno, int sockfd,struct addrinfo *servinfo);`

- Data Structures:
  - `map<uint32_t,string> file_segments;`
  - `vector<string> lines = read_from_file(client_in_path);`