Fast, Reliable File Transfer

EE542 Lab#4 Report

ANTISLEEP

Member: Yuzhe Zhang

Yuhao Lu

Yude Wei

In networks with high latency and packet loss, traditional TCP can’t provide satisfying throughput performance. In this lab, we tried to create a file transfer program which provides you better performance than any other program available.

74.4 Mbits/s

Throughput (Case-2)

74.4 Mbits/s

Maximum Throughput in Case-2

Under designed circumstance of 20% packet loss, 200ms RTT, and 100Mbit/s link speed, with MTU = 1500, the throughput of the network is impacted largely.

We re-designed the whole file transfer utility of Linux Socket for better speed performance. With dedicated optimization for the given network settings, the file transfer utility provides you high reliability and boosted throughput at the same time.

```
ubuntu@ip-10-0-1-172:~/lab4$ ./client ../test/data.bin 10.0.2.10 12345
Total file size = 1073741824
Total packet number = 735440
UDP socket created
Sent file info to server
File info received by server
Start 1st round send file
Finish 1st round send file
Start retransmission lost packet
File transmission end
Transfer file size = 8589.93 Mbits
Total file transmission time = 115.354sec
Throughput = 74.466 Mbits/s
```

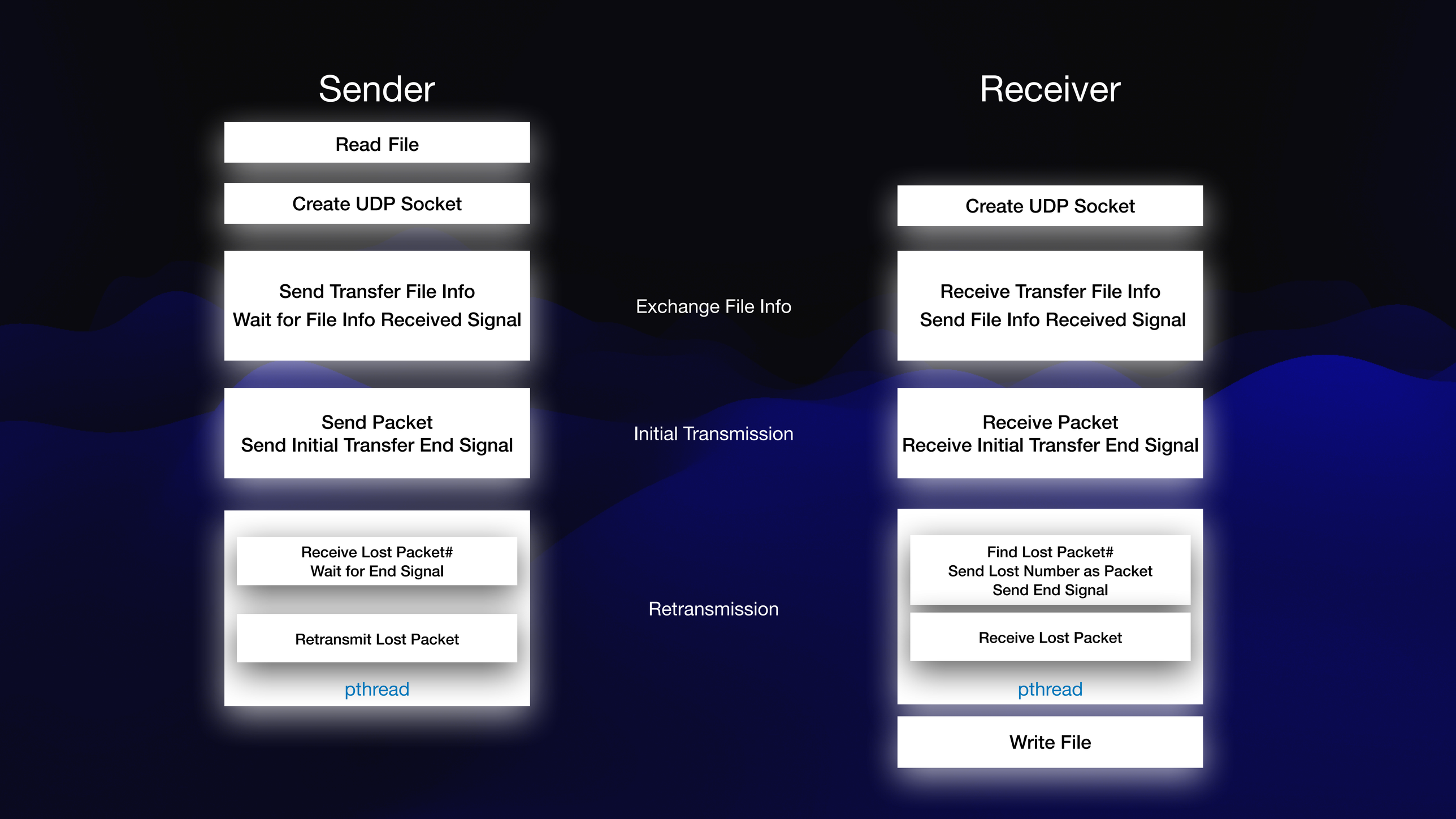
```
ubuntu@ip-10-0-2-10:~/lab4$ ./server ../test/data.bin 12345
UDP socket created on port: 12345
File info received by server
Send confirmation to client about file info
Start 1st round receive file
Finish 1st round receive file
Start request retransmission lost packet
Start receive retransmission packet
End retransmission lost packet
File transmission end
```

Throughput Performance

Our solution 74.4 Mbits/s

Common solution 50 Mbits/s

Traditional TCP solution 0.5 Mbits/s


Sender
Receiver
Read File
Create UDP Socket
Create UDP Socket
Send Transfer File Info
Wait for File Info Received Signal
Exchange File Info
Receive Transfer File Info
Send File Info Received Signal
Send Packet
Send Initial Transfer End Signal
Initial Transmission
Receive Packet
Receive Initial Transfer End Signal
Receive Lost Packet#
Wait for End Signal
Retransmit Lost Packet
pthread
Retransmission
Find Lost Packet#
Send Lost Number as Packet
Send End Signal
Receive Lost Packet
pthread
Write File

Customized Data Structure

Customized Data Structure

To achieve a higher throughput, we designed a customized packet structure from the ground up to meet the needs.

With a special considered control flow, our new packet structure can be used as either data delivery or retransmission info at the same time. It works as the key part of the retransmission.

We also created a file information structure dedicated for exchange file info

packet struct

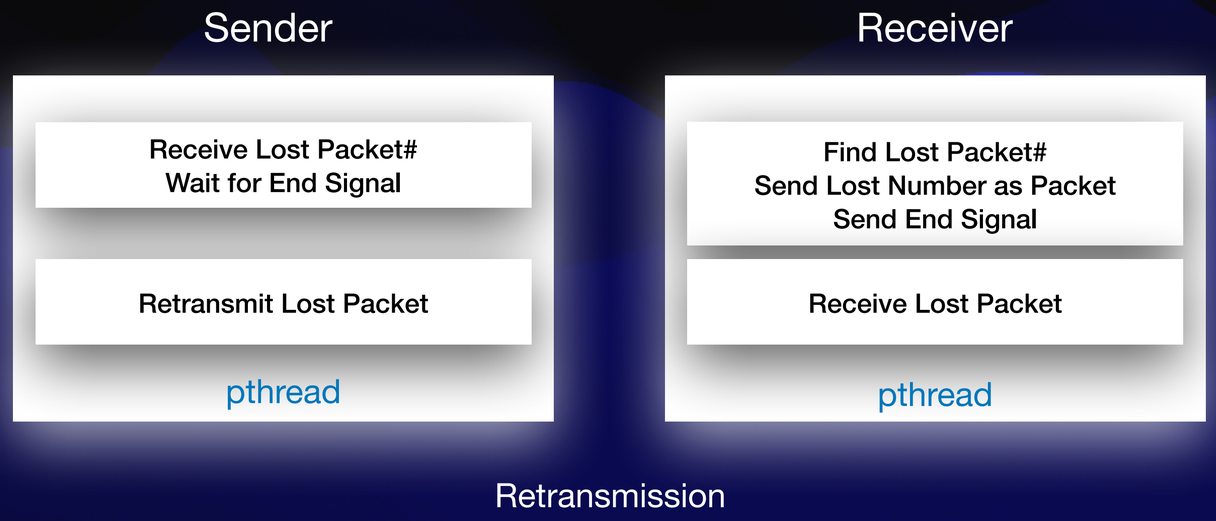
fileInfo struct

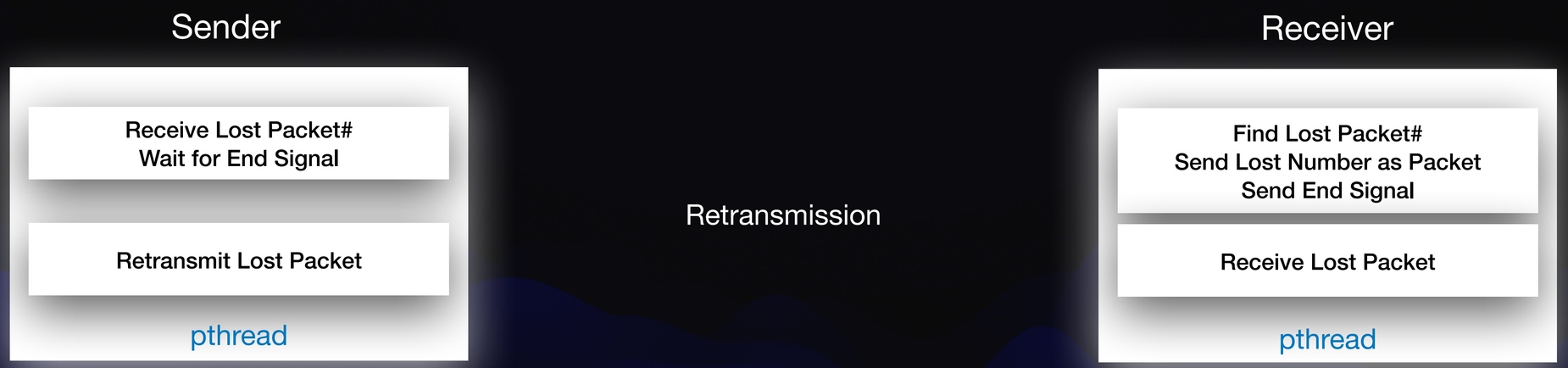
Dual Threads

Dual Threads

In a network with high packet loss, the retransmission process would be verbose. With Dual Threads, we can detect the lost packets and retransmit them at the same time.

Use two threads work in both sender and receiver to increase the bandwidth utilization significantly.





Sender:

Thread 1: Receive the lost packet number that need to retransmit.

Thread 2: Retransmit the lost packets

Receiver:

Thread 1: Actively scan the packet number buffer to find the lost packet and send the lost info to sender

Thread 2: Receive the retransmission packets

Buffer Accelerate

Buffer Accelerate

With transmission data loaded into the sender's buffer at the beginning, all IO process afterward can be extremely accelerated in read file and send packet phase.

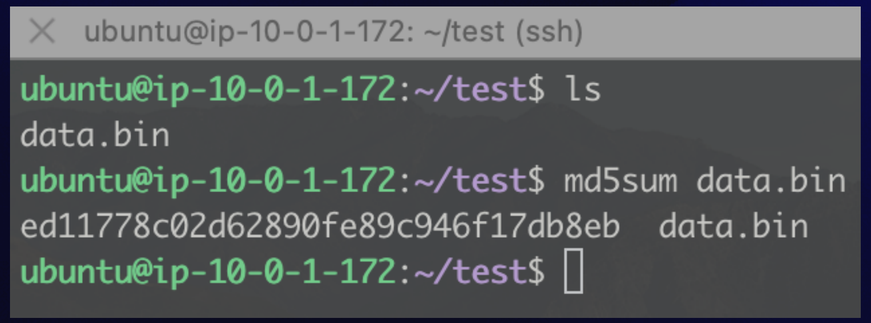
Meanwhile, after receive the file info, the receiver will create two buffer space. One for saving the packet and the other for saving the packet number.

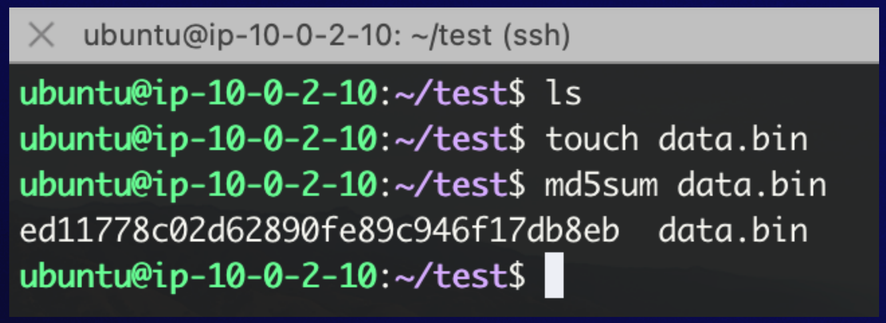
```
void readFile(const char* path, char* &buff_p) {
    // open file
    FILE *read_file;
    if((read_file = fopen(path, "rb")) == NULL){
        printf("Error opening specified file.\n");
        exit(1);
    }
    fseek(read_file, 0, SEEK_END);
    thisFileInfo.fileSize = (int)ftell(read_file);
    fseek(read_file, 0, SEEK_SET);
    thisFileInfo.packetNum = thisFileInfo.fileSize / PACKET_SIZE;
    if((thisFileInfo.fileSize % PACKET_SIZE) != 0) thisFileInfo.packetNum++;

    //map file into buffer
    buff = (char *)malloc(thisFileInfo.fileSize + 1);
    memset(buff, 0, thisFileInfo.fileSize + 1);
    fread(buff, 1, thisFileInfo.fileSize, read_file);
    fclose(read_file);

    cout<<"Total file size = "<< thisFileInfo.fileSize << endl;
    cout<<"Total packet number = "<< thisFileInfo.packetNum <<endl;
}
```

MD5 Check

MD5 Check

High packet loss can cause the file corrupted after transmission. After complicated transmission process, MD5 check provides us an easy way to ensure that file is intact and complete.





Case-1 and Case-3

Case1 Throughput: 93.4 Mbits/s

1% packet loss, 10ms RTT, and 100Mbit/s link speed, with MTU = 1500

```
ubuntu@ip-10-0-1-172:~/lab4$ ./client ../test/data.bin 10.0.2.10 12345
Total file size = 1073741824
Total packet number = 735440
UDP socket created
Sent file info to server
File info received by server
Start 1st round send file
Finish 1st round send file
Start retransmission lost packet
File transmission end
Transfer file size = 8589.93 Mbits
Total file transmission time = 91.9474sec
Throughput = 93.4223 Mbits/s
```

Case3 Throughput: 74.5 Mbits/s

200ms RTT, and 80Mbit/s link speed at router and server, 100Mbit/s link speed at client

```
ubuntu@ip-10-0-1-172:~/lab4$ ./client ../test/data.bin 10.0.2.10 12345
Total file size = 1073741824
Total packet number = 735440
UDP socket created
Sent file info to server
File info received by server
Start 1st round send file
Finish 1st round send file
Start retransmission lost packet
File transmission end
Transfer file size = 8589.93 Mbits
Total file transmission time = 115.168sec
Throughput = 74.5862 Mbits/s
```

Change MTU to 9001 will not affect the throughput of the program

Because the program fixed the customized data structure with packet size 1460 Bytes

Critical Thinking

Critical Thinking

For UDP, the two cases will be similar. In case 2, The bandwidth is 100Mbps and the loss is 20%, so there will be 80Mbit data transferred successfully in 1 second on average, which is roughly the same as case 3 with data rate of 80Mbps.

For TCP, the vital drawback in case 2 is that the congestion window will become very small after several packet losses and the speed will drastically go down. But in case 3, the data rate is a little lower but there is no loss (or just a little loss due to the drop of the network buffering), so the TCP congestion window still remain large and the speed is still high. And that makes the difference.

Thank you

ANTISLEEP