Sachin A Birajdhar

Fast and Reliable Communication Over Truly Unreliable Internet Connection

**Concept**

In this lab, our team implement the file transferring software using the **C Programming Language**. At first, we test the performance of **SCP** and **FTP**. Both software use **TCP** as a transport layer protocol. The result is that both software can’t even send a very small file by using the **TCP** protocol. We believe that the **TCP** mechanism will not work under high loss and delay alone, so we pick **UDP** as our transport protocol instead and use **TCP** only when we need to send very tiny packet. Even-though **UDP** is an unreliable protocol, we can use the application protocol on top of it to ask the server node to retransfer the file for us.

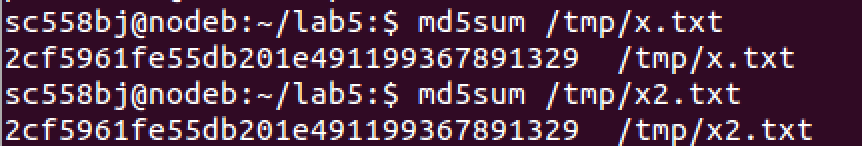
Now, let’s look at each step in our file transferring software. First, the server creates a **TCP** socket to wait for the client to connect and send the number of packet that it will send to the server. After the client finished sending the number of packet, the **TCP** session closed. Second, client will send a file packet with the Sequence number using **UDP** protocol, the packet consists of **4 bytes sequence number** and **60000 bytes data**. After the sequence number reach the last 100 sequences, the server node will start sending a **NACK** packet to the client node using **TCP** protocol so that it will retransmit the lost packet to the client. Our **NACK** packet is an array character of the value of **1** for received packet and **0** for lost packet packet, the array length is equal to the packet count. When client received this NACK, it will read each element of the array, when it finds the element that has value of **0**. It will read that file sequence index data and send it to the server node. After the **NACK** array elements’ value is equal to **1**. Then both client and server will be terminated and the server node will print an elapse time.

**Threading**

To be able to run both **TCP** and **UDP** socket at the same time, we are using the **pthread library** to implement threading in our software. Both client and server nodes start the **TCP Packet count thread** to send and receive the packet count. Then the client runs the **UDP packet send thread** and the server runs the **UDP packet receive thread.** Both server and client run the **NACK transmit and receive thread** respectively. Finally, client runs the **UDP packet resend thread**. When the server received the whole file, it then closes the threads and print the elapse time.

**Result**

We run our software on two conditions, the first is that we set the delay to 10ms and the loss ratio to 1 percent and the other condition is that we set the delay to be 200ms with loss ratio of 20 percent. Here is the result, for **10ms delay with 1 percent loss**, we get a throughput of **25Mbps** and for **200ms delay with 20 percent loss**, we have a trouble transferring the file on the DETERLab.



**Analysis**

When we compare **TCP** with **UDP**, we think of reliable vs unreliable packet transfer. Most of the time, for people who need easy and reliable protocol will use **TCP** as their main protocol. However, under certain environment, like in our lab environment, **TCP** seems to be a very bad option where the file can’t be transfer at all. **UDP** on the other hand, even though it is an unreliable protocol, **UDP** contains small header size and is suitable for our lan environment. Our application layer protocol adds one capability from **TCP** protocol, that is the **sequence** **number** that is used in the case that the packet has been lost or the packet arrived in different order.

At the end of the day, there is no perfect protocol that will work best for every environment. That is the reason why we should explore individual environment separately then look at existing protocols, both famous and infamous, then choose their capabilities that suit your environment. If you need any new capabilities, you can add your own ones.