**Lab 4:** Fast, Reliable File Transfer

**Names:** Chris Manna

Neha Vellakal Balasubramanian

Yubin Kwon

**EE 542: Internet and Cloud Computing**

**Introduction:**

In this lab we are trying to design and develop a fast and reliable custom internet protocol (IP). To achieve this, we followed the approach of setting up a basic UDP server client communication. After successful establishment of the basic communication, we upgraded the code to work in 2 main parts which include (i) Send and Receive and acknowledgement (ii) Multi-threading to send chunk of data packets. These 2 approaches are described in detail through the course of this report.

**Working:**

1. **Implementing Basic UDP Server-Client Model**

The initial server-client model using UDP shows the following:

|  |  |
| --- | --- |
|  |  |
| *Initial Server Side Terminal* | *Initial Client Side Terminal* |

As implemented in lab 1, the vyos is used. Here the IP addresses were used to route both the server and client to form the network topology shown below:

|  |
| --- |
|  |
| *VyOS topology with Server and Client* |

As shown in the network topology the server and client are connected via the VyOS server.

1. **Implementing with ACK**

This part of the implementation focusses on sending an acknowledgement packet from the receiver side after each data packet from the sender is Received. Thus, the sender and receiver alternatingly send packets to each other i.e the data and acknowledgement packets respectively.

This **improved the throughput by 4 times**

Thus, the new observed throughput simply by using ACK is \_\_\_\_\_\_\_\_\_\_\_\_\_\_.

|  |  |
| --- | --- |
|  |  |
| *Terminal Output without delay* | *Ping Output* |
|  |  |
| *Terminal Output with delay* | *Ping Output* |

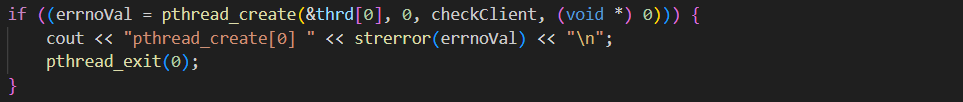
1. **Implementing Multi-threading**

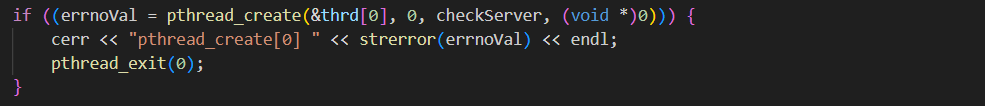
Multi-threading essentially allows concurrent execution of the code. It allows maximum utilization of resources, thereby increasing the throughput.

|  |  |
| --- | --- |
|  |  |
| *Multi-threading code snippets* | *Multi-threading code snippets* |
|  |  |
| *Terminal Output without delay* | *Ping Output* |
|  |  |
| *Terminal Output with delay* | *Ping Output* |

**Code explanation**

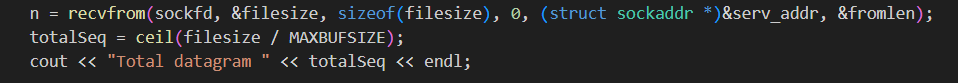
Creating Multithread (server.cpp, client.cpp)





: It uses a separate thread to perform this periodic checking and requesting of missing datagrams while the main thread (main) receives and processes incoming data packets from the server.

Receiving datagram count (client.cpp)



: It receives an initial datagram containing information about the total number of datagrams that will be sent. This information is used to determine when all expected datagrams have been received.

Implementing with ACKs (client.cpp)

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

: The client sends ACKs to the server 20 times in a loop to acknowledge the successful reception of data. These ACKs are sent back to the server to inform it that the corresponding data packets have been received successfully.

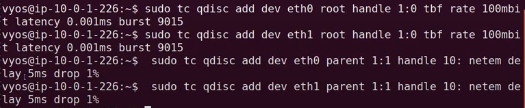
Observation of file transfer in default settings.

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

: observed with throughput of 97.88Mbsp in 83.70 seconds.

**Case 1**



Setting for Case 1

Delay: 10ms | Loss rate: 1%(bi-directional) at router | Link speed: 100Mbits/s for each

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

File Transfer Result for Case 1 – MTU 1500

: In 110.25 seconds, the file is transmitted with the throughput of 74.30 Mbps.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

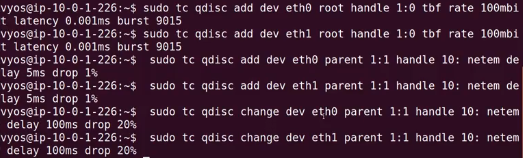
자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

File Transfer Result for Case 1 – MTU 9000

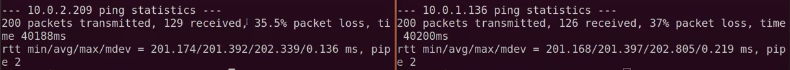
: In 83.71 seconds, the file is transmitted with the throughput of 97.87 Mbps.

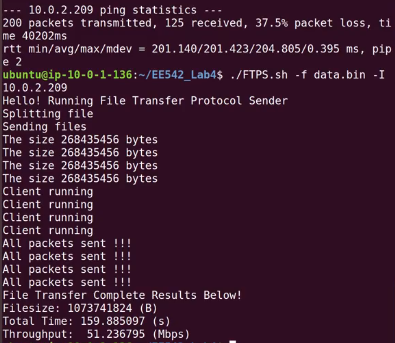
**Case 2**



Setting for Case 2

Delay: 200ms | Loss rate: 20%(bi-directional) at router | Link speed: 100Mbits/s for each



 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

File Transfer Result for Case 2 – MTU 1500

: In 159.89 seconds, the file is transmitted with the throughput of 51.24 Mbps.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

File Transfer Result for Case 2 – MTU 9000

: In 105.47 seconds, the file is transmitted with the throughput of 77.67 Mbps.

**Case 3**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Setting for Case 3

Delay: 200ms | Loss rate: - | Link speed: 80Mbits/s for router 100Mbits/s for client, server

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

File Transfer Result for Case 3 – MTU 1500

: In 132.21 seconds, the file is transmitted with the throughput of 61.96 Mbps.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

File Transfer Result for Case 3 – MTU 9000

: In 101.26 seconds, the file is transmitted with the throughput of 80.89 Mbps.

The results of multi-threaded code are:

1. Observed Throughput without delay is \_\_\_\_\_\_\_\_\_\_
2. Observed Throughput with delay is \_\_\_\_\_\_\_\_\_\_
3. Loss in Transmission is \_\_\_\_\_\_\_\_\_\_\_

**Bash Files**

The bash files created for the sender and the receiver have a few commands helpful in executing the program. They include port numbers and the names of the .bin files to be transmitted.

The important aspect to be noted is that the receiver script concatenates the split data packets into one.

While on the sender script we are using the start and end time of transmission to calculate the total time required for transmission. The total time (in seconds) and throughput (in MB/s) is printed at the end of execution.

|  |  |
| --- | --- |
|  |  |
| *Sender Bash Script code snippet* | *Receiver Bash Script code snippet* |

|  |  |
| --- | --- |
|  |  |
| *Terminal Output* | *Terminal Output* |

|  |
| --- |
|  |
| *GitHub repo* |