Group no : ACN03

Members:

1. Rajeev Kumar :- 2020H1030152H

2. Priyanka Kedar :-2020H1030154H

3. Satish Phale : -2020H1030155H

4. Yash Jain :- 2020H1030158H

Part A:

Here we have successfully transferred the file from the Server to Client through two interfaces which is shown using network tools like wireshark , nload and netem and discovered that mpquic performs better for larger file as compared to smaller file through multiple interfaces with different delay.

Part B:

In this part we are implementing the ECF and SA-ECF algorithms and we are showing the comparison of different paths scheduling algorithms such as Low-RTT, Round Robin, ECF and SA-ECF. For the demonstration purpose using the file transfer scenario. We also tried a web traffic scenario using wprof tool based on dependency tree but due some system level challenges not able to complete.

Part C:

To reduce the service down time using Mpquic while live migration of virtual machines on cloud.

When cloud service providers want to migrate the virtual machine from one location to another by keeping the server down time as minimum they can use the Mpquic protocols .

**Project Part A**

Aim: Implementing multipath Quic to transfer file from Client to Server in the following

topology.

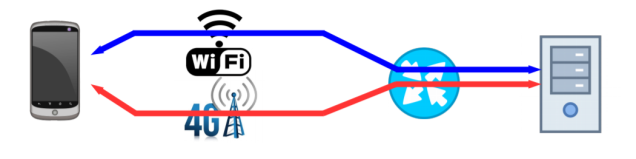


Fig 1: Topology implemented in PartA

Why we use Mpquic?

* Efficient approach to aggregate the bandwidth when there is heterogenity in the traffics among the multiple path.
* Fastest protocol and no kernel change is required.

Approach:

Step 1: Installed the Kvm hypervisor and created virtual machine in which we imported vmdk file in it which contains the following packages:

i. Mpquic setup

ii.Golang compiler

iii.mininet packages

iv. Netem packages

v. Wireshark packages

Step2: Installed GUI packages and started with “***startx***” command.

Step3: Cloned the package using command “***git clone*** [***https://github.com/qdeconinck/minitopo- experiences.git***](https://github.com/qdeconinck/minitopo-    experiences.git).”

Step4: Now running the topology created in sample file with defined delay, queue size and

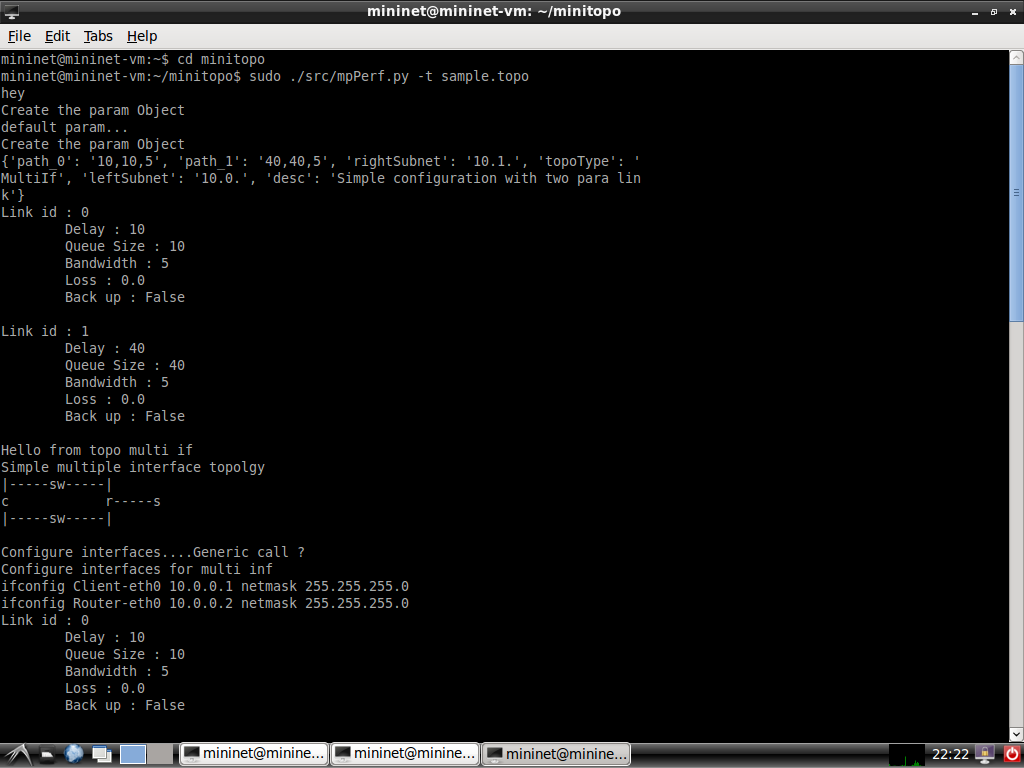
Bandwidth for two paths using command “***sudo ./src/mpPerf –t sample.topo***” shown in fig2.

Fig 2: Depicting implemented topology named as “sample.topo”

Step 5: Making connections between Client and Server using command “***xterm Client Server***” and transferring the files named as “this.tar(12.9 MB) ” from Server to Client shown in fig3.

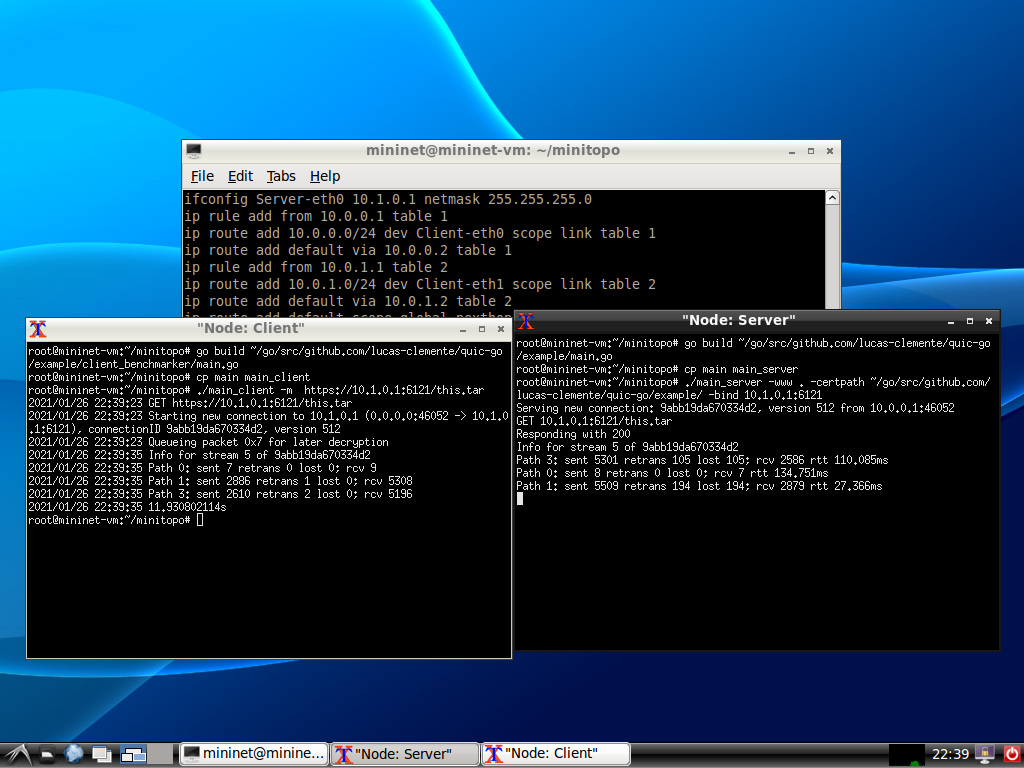


Fig 3:Depicting “this.tar” file transfer

Step 6: The file “this.tar” which was transferred is shown using wireshark in fig4.1 ,4.2 and 4.3.

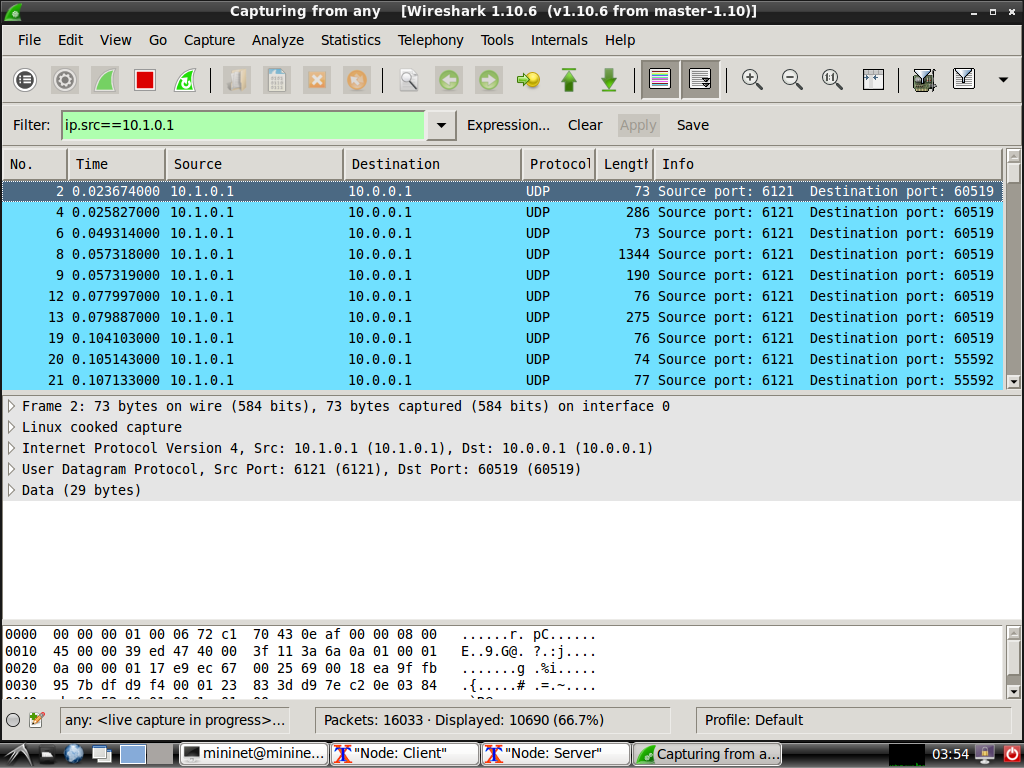


Fig 4.1: depicting data transfer on Client-eth0

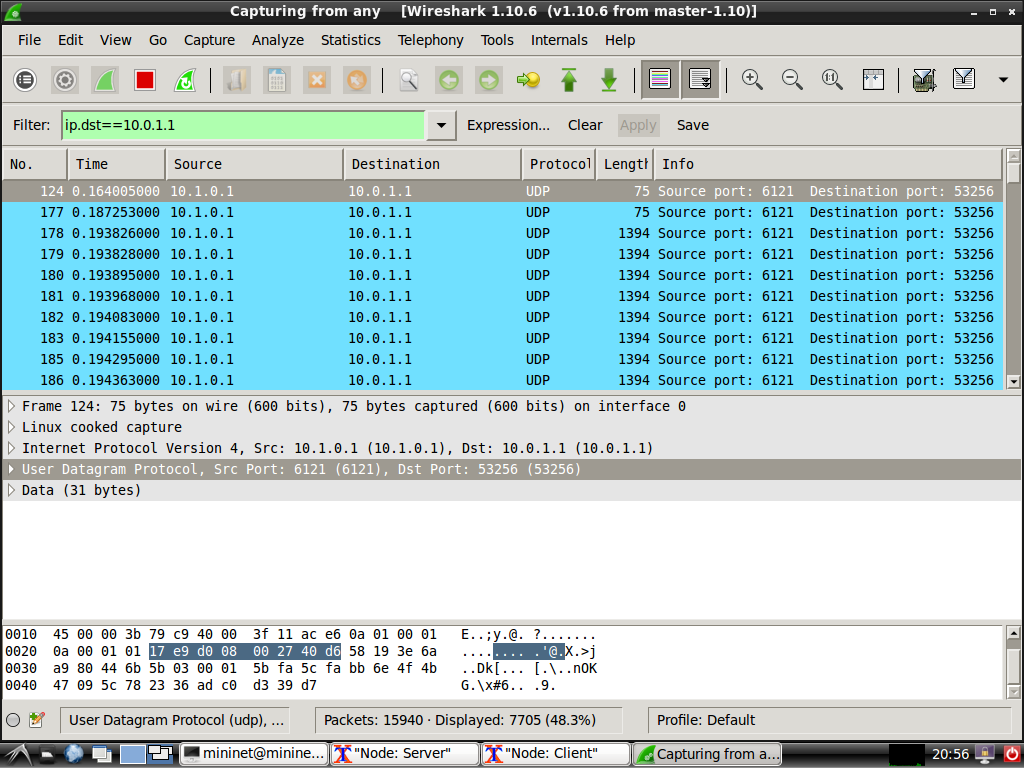


Fig 4.2: depicting data transfer on Client-eth1

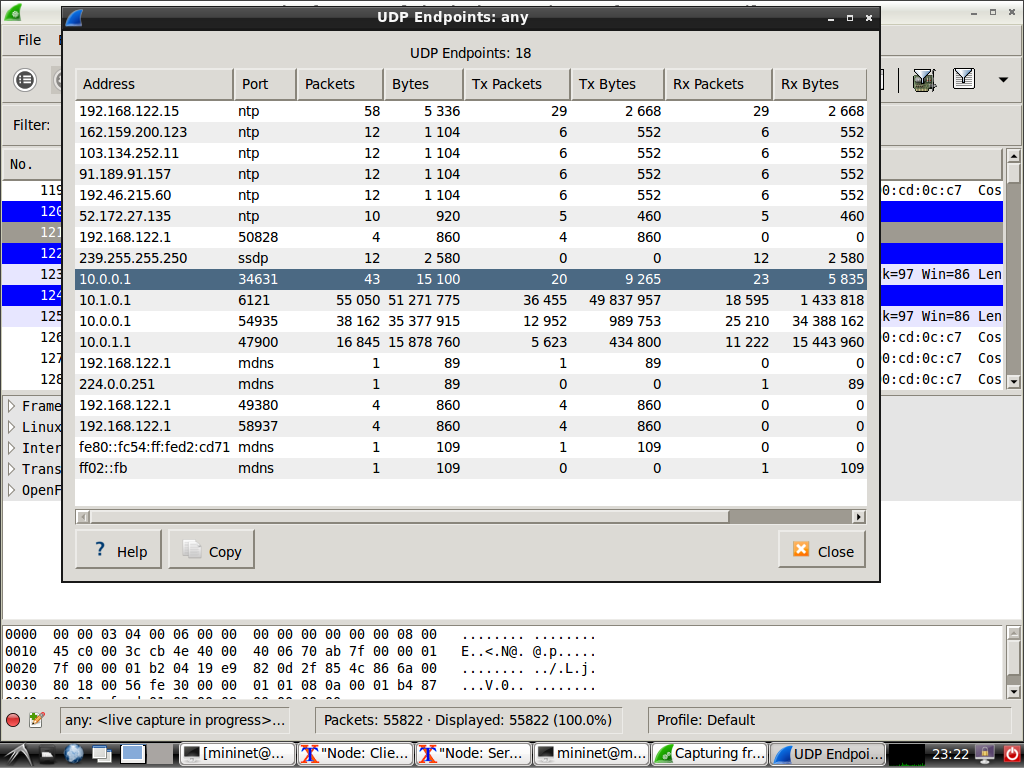


Fig4.3 : depicting endpoints in wireshrak

Step7: The file “transfer.tar” which is transferred from server to client is also shown using nload command in fig 5.1 and fig 5.2.

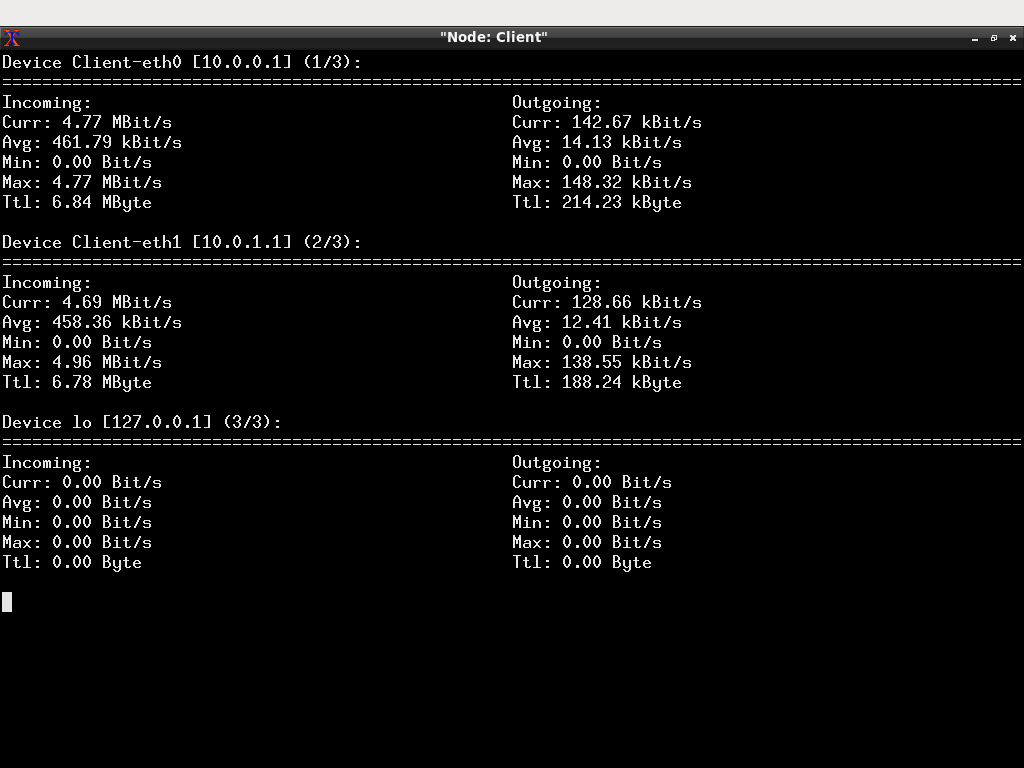


Fig5.1:depicting clients’ incoming and outgoing packets in multiple paths(Client-eth0 & Client-eth1)

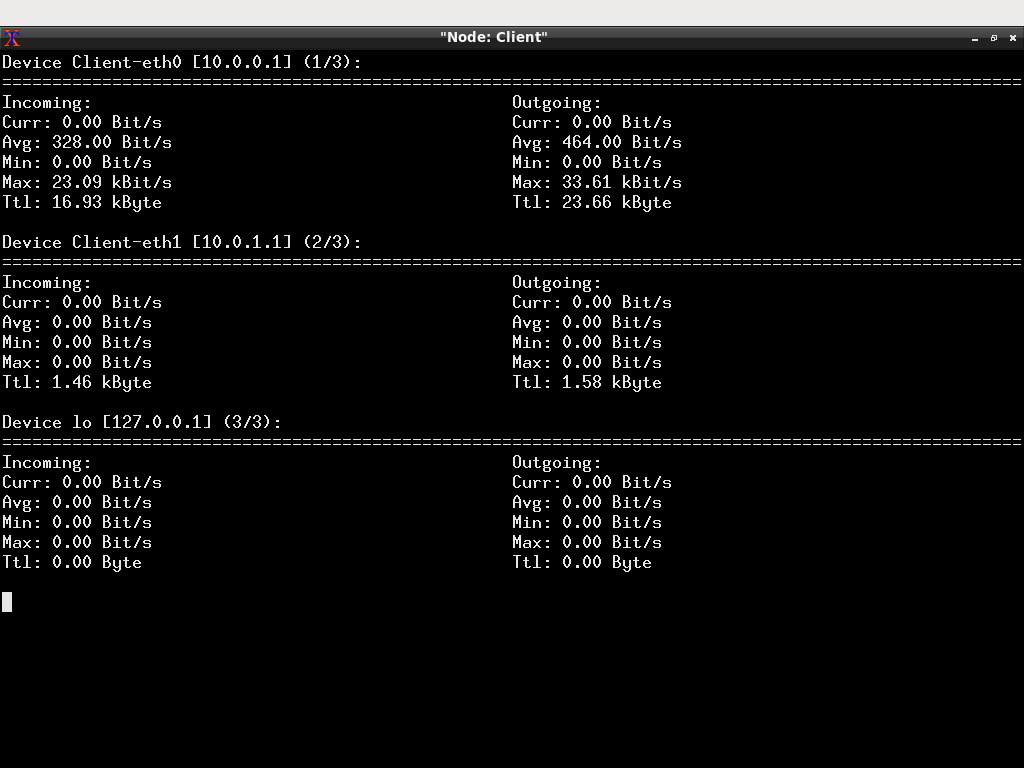
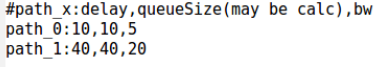


Fig5.2:depicting clients’ incoming and outgoing packets in single path(Client-eth0).

Step 8: Created two files named as “file\_4mb.tar” (3.2 MB) and“transfer.tar” (22.2 MB) and compared delay by modifying bandwidth parameter in both single and multiple path as shown belowwith following parameter of created topology



Netem: sudo tc qdsic add dev Client-eth0 root tbf rate 1000mbit burst 32kbit latency 100ms

sudo tc qdsic add dev Client-eth0 root tbf rate 1000mbit burst 32kbit latency 100ms

|  |  |  |  |
| --- | --- | --- | --- |
| **file\_4mb.tar** | **With different delay** | **Single path performance** | **Multipath performance** |
|  | Client:  C:\Users\Purushottam\Desktop\25.PNG  Server:  C:\Users\Purushottam\Desktop\27serv.PNG | Client:  C:\Users\Purushottam\Desktop\24.PNG  Server:  C:\Users\Purushottam\Desktop\26ser.PNG |
| **With same delay on both interface** | **Single path performance** | **Multipath performance** |
| Rate:  1000mbit  Latency:  100ms  (Client-eth0 & Client-eth1) | Client:  C:\Users\Purushottam\Desktop\vii.PNG  Server:  C:\Users\Purushottam\Desktop\vi.PNG | Client:  C:\Users\Purushottam\Desktop\viii.PNG  Server:  C:\Users\Purushottam\Desktop\v.PNG |
| **transfer.tar** | **With different delay** | **Single path performance** | **Multipath performance** |
|  |  | Client:  C:\Users\Purushottam\Desktop\20.PNG  Server:  C:\Users\Purushottam\Desktop\22.PNG | Client:  C:\Users\Purushottam\Desktop\21.PNG  Server:  C:\Users\Purushottam\Desktop\23.PNG |
| **With same delay on both interface** | **Single path performance** | **Multipath performance** |
| Rate:  1000mbit  Latency:  100ms  (Client-eth0 &Client-eth1) | Client:  C:\Users\Purushottam\Desktop\10.PNG  Server:  C:\Users\Purushottam\Desktop\9.PNG | Client:  C:\Users\Purushottam\Desktop\9.PNG  Server:  C:\Users\Purushottam\Desktop\10.PNG |

**Project Part B**

**A Stream-Aware Multipath QUIC Scheduler for Heterogeneous**

**Paths**[2]

**Earliest Completion First (ECF):**

* Old path schedulers like Round Robin do not take path characteristics into consideration which impact the performance.
* ECF gives priority to Lowest RTT path(fastest path). Once it finds the fastest path congested ECF evaluates whether it is actually beneficial to send over slower paths.
* The decision is based on the how much data left to send, along with the path RTT and capacity estimation.

**Stream Aware - Earliest Completion First (SA - ECF):**

* It extends the ECF scheduler for connections where multiple streams are transmitted concurrently over the same connection.
* Unlike ECF, SA-ECF schedules data on a per-stream basis, using the estimated completion times of each individual stream instead of the completion time of the connection as a whole.

**Implementation:**

* To implement this experiment we have added ECF and SA - ECF code inside the scheduler.go file of original lucas clemente code in which Round Robin and Low-RTT code are already present. For SA-ECF we have added stream\_schedular.go file inside the original code.
* Minitopo is used for creation of topology and adding network conditions.
* Wireshark is used to measure the results.

**Evaluation for single File Transfer**

To evaluate the performance of different path/stream scheduling algorithms, a set of experiments were conducted where the time required to download a file using HTTP over increasing path asymmetry was measured.

Below are the four different algorithms used in this implementation:

1. Round Robin
2. Low RTT
3. ECF
4. SA - ECF

**Results:**

**File Size: 3 MB**

**File transfer on symmetric path**

Evaluated the above mentioned four algorithms on the symmetric paths shown in Table1. Used the same Delay and Bandwidth 20ms and 50 Mbps respectively the result is shown is table 2 where it clearly shows that round robin gives better performance than other algorithms. Since paths are the same round robin do not take path characteristics into consideration and give better performance in this case.

Table 1:

Link parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Path 0(WLAN) | Path 1(MBB) |
| Delay(ms) | 20 | 20 |
| Bandwidth(Mbps) | 50 | 50 |

Table 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | Round Robin | Low RTT | ECF | SA-ECF |
| **Time(sec)** | 6.63 | 6.75 | 7.06 | 7.18 |

**File transfer on asymmetric path**

In this case different Delay and Bandwidth applied given in the table 3. The result is shown in table 4 where it clearly shows that Low RTT gives better performance than other algorithms. Since paths are the asymmetric Low RTT finds the fastest path and gives better performance in this case.

Table 3:

Link parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter** | Path 0(WLAN) | Path 1(MBB) |
| **Delay(ms)** | 20 | 100,50 |
| **Bandwidth**  **(Mbps)** | 50 | 10,50 |

Table 4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | Round Robin | Low RTT | ECF | SA-ECF |
| 100ms,10Mbps | 11.78 sec | 11.14 sec | 12.82 sec | 12.72 sec |
| 50ms,50Mbps | 9.66 sec | 7.47 sec | 12.74 sec | 12.65 sec |

**Path utilization in different algorithms:**

In this module, the four algorithms are evaluated based on the path utilization. Here one path i.e. Path 0 parameter kept constant and by changing other path parameters.

The path parameters are given in table 5 and results are given in table 6. First parameter is the same as path 0 parameter and it clearly shows both the paths are equally utilized in such a case. In the second and third parameter as we increase the delay on path 1, the utilization of path 0 is more in ECF and SA-ECF compared to other algorithms.

Table 5:

Link parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Path 0(WLAN) | Path 1(MBB) |
| Delay(ms) | 20 | 20,100,200 |
| Bandwidth(Mbps) | 50 | 50,10,0 |

1. Percentage of packets sent on Path 0(WLAN) for symmetric scenarios in terms of

bandwidth.

Table 6:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sr. No. | Path 1(MBB) | Round Robin(%) | Low RTT(%) | ECF(%) | SA-ECF(%) |
| 1 | 50 Mbps,  20 ms | 50 | 51 | 50 | 50 |
| 2 | 50 Mbps,  100 ms | 77 | 81 | 98.80 | 99.24 |
| 3 | 50 Mbps,  200 ms | 84 | 89 | 99.01 | 98.99 |

1. Percentage of packets sent on Path 0(WLAN) for asymmetric scenarios in terms of

Bandwidth.

In the first parameter in table 7. We can see that the delay of path 0 and path 1 is the same bandwidth does not impact more here as we are getting the same utilizations on both paths. Here we can clearly see that on path 0 SA-ECF achieves higher utilization but after increasing the delay ECF achieves better performance than SA-ECF.

Table 7:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sr. No. | Path 1(MBB) | Round Robin(%) | Low RTT(%) | ECF(%) | SA-ECF(%) |
| 1. | 10 Mbps,  20 ms | 50 | 49 | 50 | 52 |
| 2. | 10 Mbps,  100 ms | 78.1 | 82 | 99.01 | 99.08 |
| 3. | 10 Mbps,  200 ms | 86 | 88 | 99.21 | 98.99 |

Note : As the values are changing every experiment here is considered the average value of 3 experiments.

**Conclusion:**

In this experiment we demonstrated the simple file transfer using the various path algorithms and measured them on different parameters. In the original research paper, the author evaluated EA-ECF algorithm on web-based traffic and showed that it performs better than Low-RTT and Round Robin. Another algorithm pstream[1] shows that it performs better for web based traffic than SA-ECF in heterogeneous scenarios.

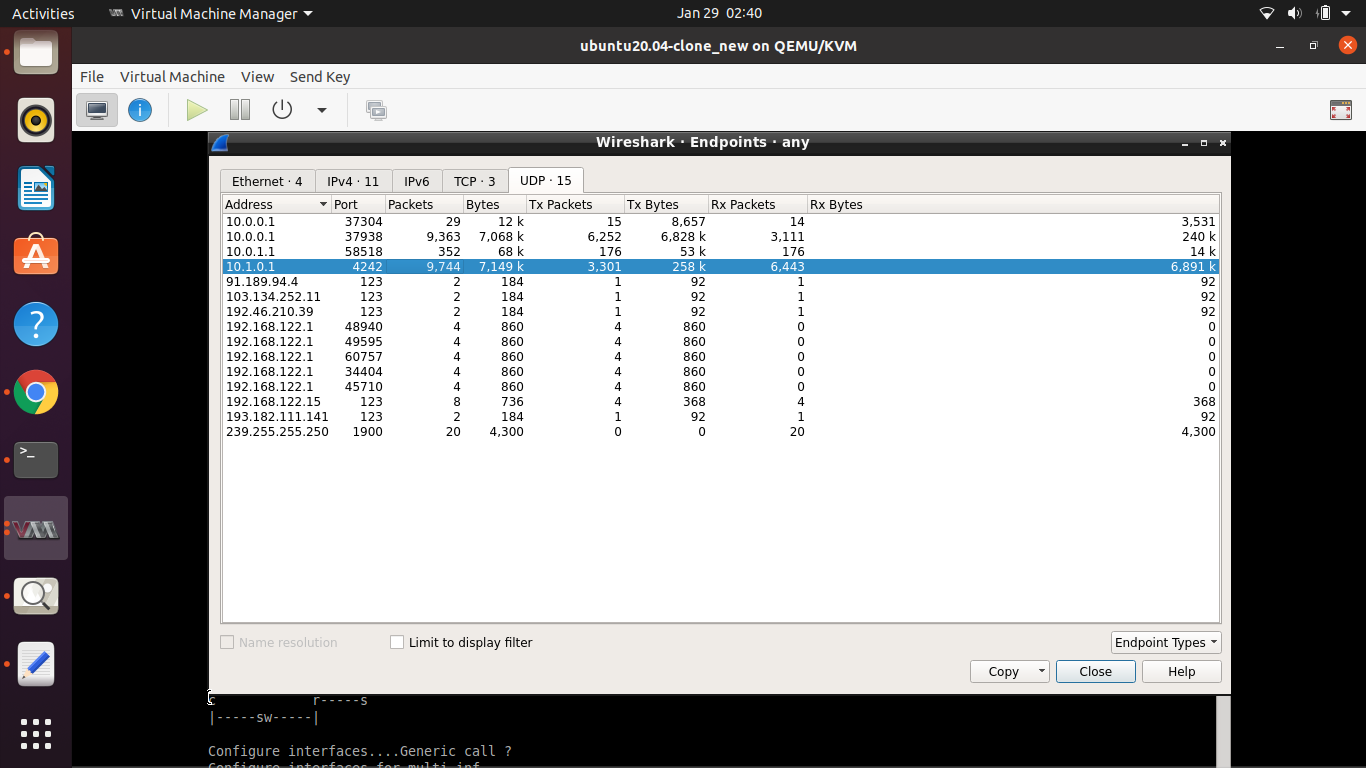
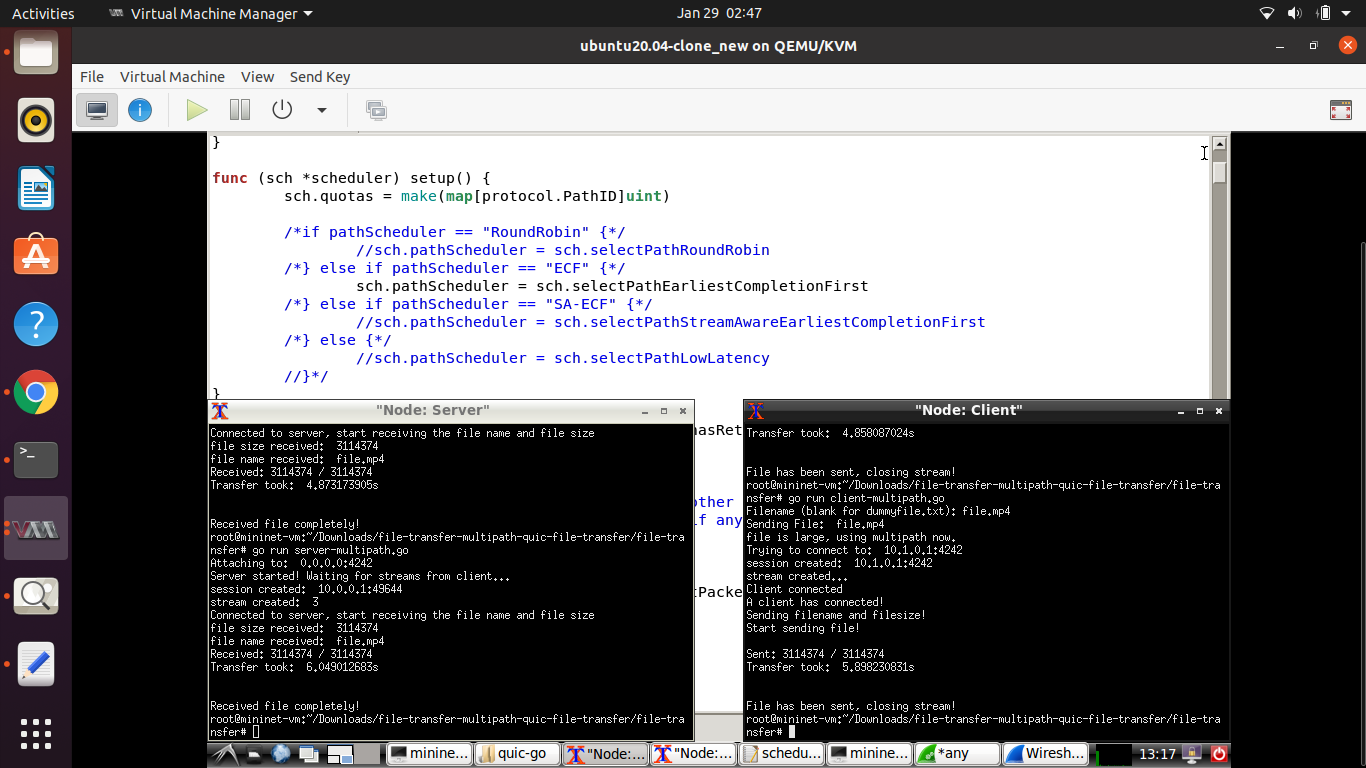
**References:**

[1]PStream: Priority-Based Stream Scheduling for Heterogeneous Paths in Multipath-QUIC

Xiang Shi ∗ , Lin Wang †‡ , Fa Zhang ∗ , Biyu Zhou § and Zhiyong Liu ∗[2020]

[2]Alexander Rabitsch, Per Hurtig, and Anna Brunström. A stream-aware multipath QUIC scheduler for heterogeneous paths[2018]

**Running ECF algorithm Wireshark result**



**Project Part C:**

Scenario based : Live Migration technique

Aim : To reduce the service down time using Mpquic while live migration of virtual machine on cloud.

Definition: Live Migration is a technique used to transfer the virtual machine from one server to other server at same or different locations.

When we need Live migration ?

* When the Server is down or crashed .
* When the server is under maintenance .
* When we are changing the geographical location of the servers.

When cloud service providers want to migrate the virtual machine from one location to another by keeping the server down time as minimum they can use the Mpquic protocols .

While transferring the file we can modulate the traffic on different paths.

1. using netem command we can decrease the latency.

“ sudo tc qdisc add dev h1-eth0 root netem delay 5ms”

2. using high bandwidth in topology.

We will use such a Mpquic scheduling algorithm which will dynamically select the interface based on the bandwidth as we know that dynamic network-bandwidth adaptation allows migration to proceed with minimal impact on running services, while reducing total downtime.

here we have to transfer very large data sets and various log files so that the customers can resume from the same checkpoints from where they have left since we have seen in Step 1 that the mpquic performs better when we are transferring the large files.