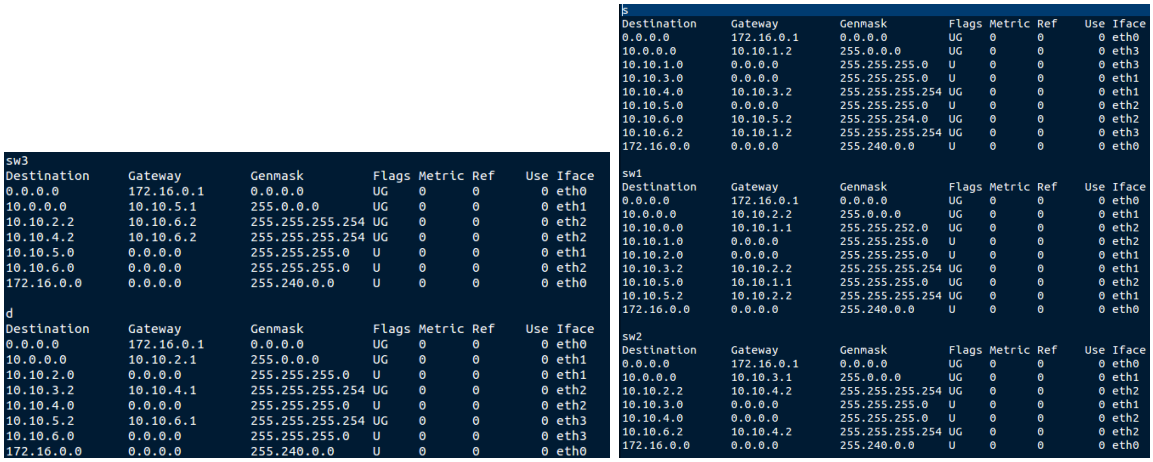
1. Initial configurations

I would like to explain how configurations to topology3 is done. If we have interface addresses to destionation as 10.10.2.2, 10.10.4.2, 10.10.6.2. And lets say we want to access to first interface through sw1, and second interface through sw2 and through sw3 to third interface. By writing "route -n" command to each node we can access routing table of node. In our case initially it gave these as output.



sw3

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.0.1	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	10.10.5.1	255.0.0.0	UG	0	0	0	eth1
10.10.2.2	10.10.6.2	255.255.255.254	UG	0	0	0	eth2
10.10.4.2	10.10.6.2	255.255.255.254	UG	0	0	0	eth2
10.10.5.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.10.6.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
172.16.0.0	0.0.0.0	255.240.0.0	U	0	0	0	eth0

d

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.0.1	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	10.10.2.1	255.0.0.0	UG	0	0	0	eth1
10.10.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.10.3.2	10.10.4.1	255.255.255.254	UG	0	0	0	eth2
10.10.4.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
10.10.5.2	10.10.6.1	255.255.255.254	UG	0	0	0	eth3
10.10.6.0	0.0.0.0	255.255.255.0	U	0	0	0	eth3
172.16.0.0	0.0.0.0	255.240.0.0	U	0	0	0	eth0

s

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.0.1	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	10.10.1.2	255.0.0.0	UG	0	0	0	eth3
10.10.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth3
10.10.3.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.10.4.0	10.10.3.2	255.255.255.254	UG	0	0	0	eth1
10.10.5.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
10.10.6.0	10.10.5.2	255.255.254.0	UG	0	0	0	eth2
10.10.6.2	10.10.1.2	255.255.255.254	UG	0	0	0	eth3
172.16.0.0	0.0.0.0	255.240.0.0	U	0	0	0	eth0

sw1

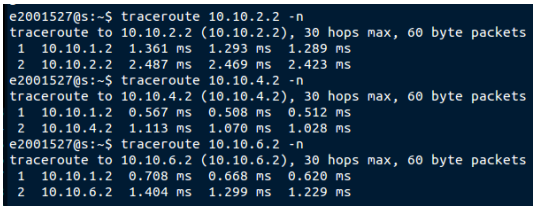
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.0.1	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	10.10.2.2	255.0.0.0	UG	0	0	0	eth1
10.10.0.0	10.10.1.1	255.255.252.0	UG	0	0	0	eth2
10.10.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
10.10.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.10.3.2	10.10.2.2	255.255.255.254	UG	0	0	0	eth1
10.10.5.0	10.10.1.1	255.255.255.0	UG	0	0	0	eth2
10.10.5.2	10.10.2.2	255.255.255.254	UG	0	0	0	eth1
172.16.0.0	0.0.0.0	255.240.0.0	U	0	0	0	eth0

sw2

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.16.0.1	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	10.10.3.1	255.0.0.0	UG	0	0	0	eth1
10.10.2.2	10.10.4.2	255.255.255.254	UG	0	0	0	eth2
10.10.3.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.10.4.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
10.10.6.2	10.10.4.2	255.255.255.254	UG	0	0	0	eth2
172.16.0.0	0.0.0.0	255.240.0.0	U	0	0	0	eth0

Figure 1: Initial routing tables

Then with help of "traceroute" command we can see the path which packet will follow while it is send to destination. It is executed as traceroute followed by ip of destination.In our case it gave this as result. We can see that all packets are going through 10.10.1.2 which belongs to sw1



```
e2001527@s:~$ traceroute 10.10.2.2 -n
traceroute to 10.10.2.2 (10.10.2.2), 30 hops max, 60 byte packets
 1  10.10.1.2  1.361 ms  1.293 ms  1.289 ms
 2  10.10.2.2  2.487 ms  2.469 ms  2.423 ms
e2001527@s:~$ traceroute 10.10.4.2 -n
traceroute to 10.10.4.2 (10.10.4.2), 30 hops max, 60 byte packets
 1  10.10.1.2  0.567 ms  0.508 ms  0.512 ms
 2  10.10.4.2  1.113 ms  1.070 ms  1.028 ms
e2001527@s:~$ traceroute 10.10.6.2 -n
traceroute to 10.10.6.2 (10.10.6.2), 30 hops max, 60 byte packets
 1  10.10.1.2  0.708 ms  0.668 ms  0.620 ms
 2  10.10.6.2  1.404 ms  1.299 ms  1.229 ms
```

Figure 2: Initial traceroute output

Now we have to modify our routing table inorder to send packets as we want.For 10.10.2.2 we don't have to do anything since it goes through sw1(10.10.1.2) as we wanted.Inorder to send packets to 10.10.4.2 through sw2, we should modify routing table with help of "route add" command,by executing "sudo route add -net 10.10.4.2 netmask 255.255.255.255 gw 10.10.3.2 dev eth1" command in source and "sudo route add -net 10.10.3.1 netmask 255.255.255.255 gw 10.10.4.1 dev eth2" in

destination. In first command we are saying if my destination ip is 10.10.4.2 exactly use gateway 10.10.3.2. Here we are saying exactly by assigning 255.255.255.255 to netmask. Same thing can be said for second command. Here 10.10.3.2 and 10.10.4.1 belongs to sw2 and 10.10.3.1 belongs to source. We can check it with help of "ifconfig" command in sw2 and source, also it is provided in Geni Portal while we are reserving resources. Using same logic we should execute "sudo route add -net 10.10.6.2 netmask 255.255.255.255 gw 10.10.5.2 dev eth2" in source and "sudo route add -net 10.10.5.1 netmask 255.255.255.255 gw 10.10.6.1 dev eth3" in destination. Here I would like to mention the reason why all ips used gateway 10.10.1.2 initially. The reason is, in routing table of source, 2nd line we see that if destination is 10.0.0.0 use gateway 10.10.1.2 but also netmask is set to 255.0.0.0 which means that 2,3,4 components of ip can be any number between 0-255. But why it did not use it after modification of routing table the reason for this is, it looks for longest prefix match.

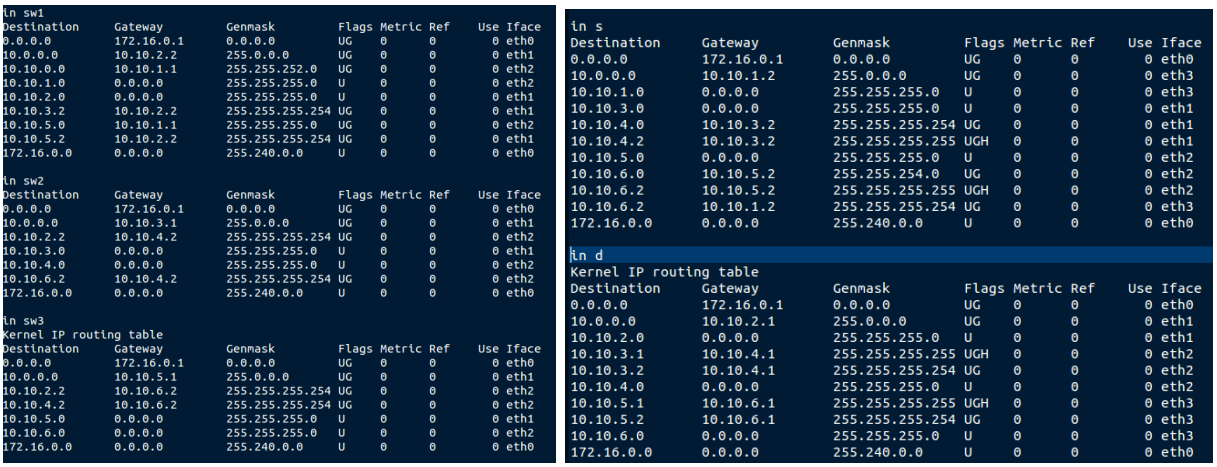


Figure 3: Modified routing tables

Our new routing tables look like above. And now if we execute traceroute function we get output as below.

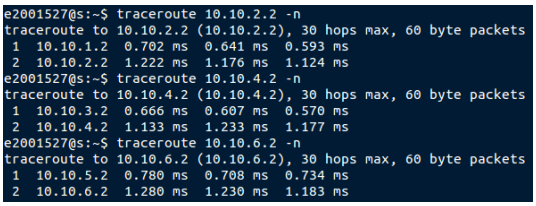


Figure 4: Final traceroute outputs

We can see that if we want to send packet to 10.10.2.2 we use 10.10.1.2(sw1), for 10.10.4.2 we use 10.10.3.2(sw2) and for 10.10.6.2 we use 10.10.5.2(sw3).

Now let me briefly mention how we configured delay, corruption, duplicate, reordering properties of links. We configured delay of all links to 1ms with the "tc" command, by executing "tc qdisc add dev eth1 root netem delay 1ms" command we can do it for eth1 of a node where we

executed this command. Also in source side for packet duplication with 1% we should execute this command "tc qdisc change dev eth1 root netem duplicate 1%".
For packet corruption with 0.1 we executed "tc qdisc change dev eth1 root netem corrupt 0.1%" command.
For reordering of a packet in second form with 25% of packets(with a correlation of 50%) sending immediately,others sending with 1ms delay we execute "qdisc change dev eth1 root netem delay 1ms reorder 25% 50%" with this, 25% of packets (with a correlation of 50%) will get sent immediately, others will be delayed by 1ms.
Inorder to set packet loss option we used "tc qdisc change dev eth1 root netem loss 1%" command. With this 1 percent of (i.e 1 out of 100) packets will be randomly dropped.

4. Implementation

We used c as our programming language. In Readme file we provided how our code works and how to run it. Also in our source code we provided clear explanations about each function. Workshare was like this we wrote code together but Batyr Charyyev give special attention to thread part of code, and Kadir Berkay take main part in packet design and checksum part of code. Also Batyr Charyyev explored about how routing configurations are done and Kadir Berkay explored how delay, duplication, corruption and reordering properties are set in links. The rest of job was done together.

3. Problems that we encountered

In server side we created thread for each receiving interface, all of them initially writes to buffer then, after receiving all the packets we write that buffer to output.txt. Here synchronization of threads were difficult for us. Our code worked for first topology without any problem but it didn't work for topology2 and 3.We printed every packet after we received it and result was correctly inorder but we could not manage to write it to output.txt properly. We had to study this topic. And the problem was due to zombie threads for which we did not wait for finish.
Also configuration of routing tables were very difficult for us.
We noticed that increasing timeout time increases our FTT because of packet loss. For example for topology1 we got FTT nearly 60seconds so we decreased our timeout time so FTT decreased considerably.

4. Experiment

In graph1, for each topology we took 4 results of FTT and took mean value of it. For topology1 our results were: 12.118s, 12.269s, 13.020s, 12.192s.
For topology2 : 8.428s, 7.208s, 8.807s, 8.777s.
For topology3 : 8.009s, 7.857s, 7.003s, 7.306s.

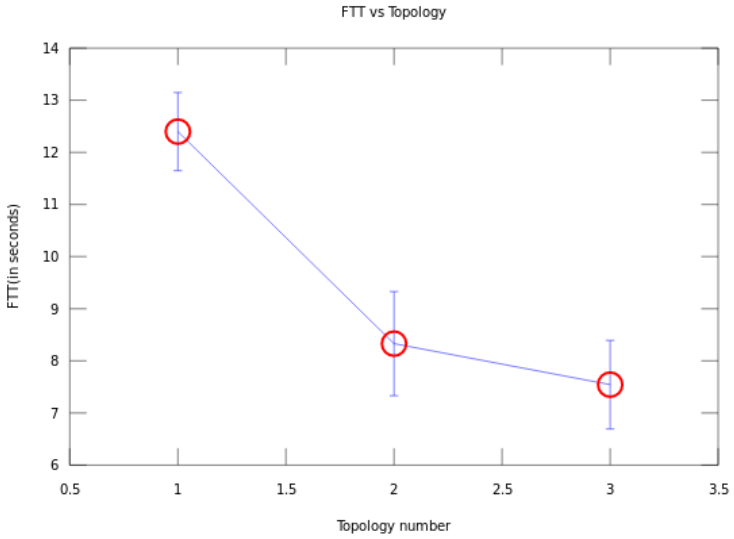


Figure 5: Chart1

In chart1 we can see decrease in FTT as expected. Because for topology1 we use only one thread for sending packets, but in topology2 we created two threads so workload of one thread is halved. Also FTT is decreased nearly half of topology1. But between topology2 and topology3 there is not so much difference because we used stop-and-wait kind of approach in our implementation as we mentioned before. Which means that all threads must wait for each other to receive ack and send packets together.

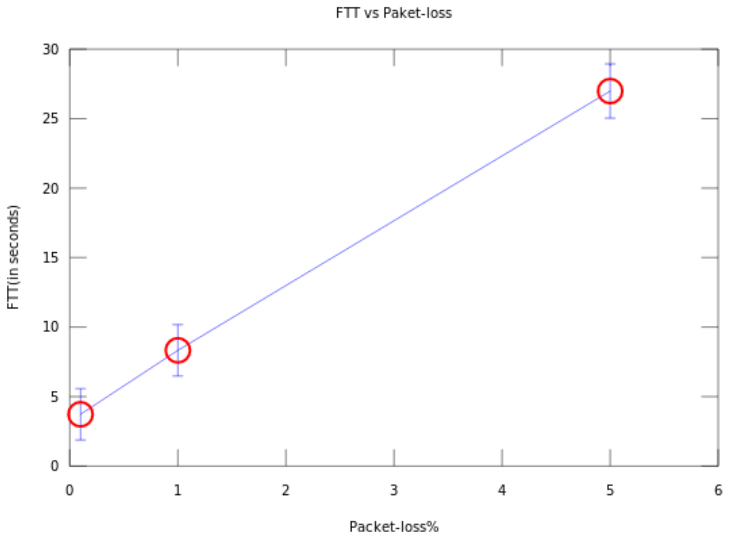


Figure 6: Chart2

In graph2, for topology3 we set packet loss to three values (0.1%, 1% , 5%) and calculated

FTT results for each of them. Again we took 4 values and calculated mean of them. We used the results of graph1 for 1% packet loss. When packet loss occurs sender waits for timeout so the total file transfer time(FTT) rises. From the our graph it can be seen clearly.

For 0.1% : 3.937s, 3.608s, 3.659s, 3.697s
For 5% : 25.394s, 26.163s, 27.623s, 28.761s

5. Conclusion

As a programmer everything can not be good according to real life. After implementing our program when we add delays, losses, duplicates we faced with too many errors. So after modifying our program everything started to work better. After adding loss FTT rises and after setting time-out value of sender we decreased the FTT. In topology 2 and 3 FTT are decreased as expected but it did not decrease too much for the topology 3 because we implemented our programs in StopWait manner. FTT ratio for Topology 3 and topology on is 40%. FTT for topology 3 is 7.5 and for topology 1 is 12.4. With the help of parallelism we speed up our program.