

Latency

Timestamp (Task 1) (Screenshot attached)

```
Fatal: unable to access 'https://github.com/MdBaizil/scion.git/': Could not resolve host: github.com
ubuntu@ubuntu-xenial:~/go/src/github.com/scionproto/scion$ cd ..
ubuntu@ubuntu-xenial:~/go/src/github.com/scionproto$ cd ..
ubuntu@ubuntu-xenial:~/go/src/github.com$ sudo rm -rf tutorial/
ubuntu@ubuntu-xenial:~/go/src/github.com$ git clone git@github.com:manish978973/tutorial.git
Cloning into 'tutorial'...
remote: Enumerating objects: 55, done.
remote: Counting objects: 100% (55/55), done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 180 (delta 35), reused 0 (delta 0), pack-reused 125
Receiving objects: 100% (180/180), 44.57 KiB | 2.00 KiB/s, done.
Resolving deltas: 100% (114/114), done.
Checking connectivity... done.
ubuntu@ubuntu-xenial:~/go/src/github.com$ cd tutorial/new/
ubuntu@ubuntu-xenial:~/go/src/github.com/tutorial/new$ go run scion_client_timestamp.go -s 19-ffaa:1:152,[192.168.0.110]:30102 -c 19-ffaa:1:14c,[192.168.0.114]:30100
INFO[01-22|14:52:41] Started id=ebc1d683 goroutine=dispatcher_bck
INFO[01-22|14:52:41] Registered with dispatcher ia=19-ffaa:1:14c host=192.168.0.114 port=30100
^Csignal: interrupt
ubuntu@ubuntu-xenial:~/go/src/github.com/tutorial/new$ go run scion_client_timestamp.go -s 19-ffaa:1:152,[192.168.0.110]:30102 -c 19-ffaa:1:14c,[192.168.0.114]:30100
INFO[01-22|14:54:06] Started id=485e885d goroutine=dispatcher_bck
INFO[01-22|14:54:06] Registered with dispatcher ia=19-ffaa:1:14c host=192.168.0.114 port=30100

Client: 19-ffaa:1:14c,[192.168.0.114]:30100
Server: 19-ffaa:1:152,[192.168.0.110]:30102
LATENCY_TIMESTAMP_METHOD:
RTT is - -1548168846121710592.000ns
Latency is - -774084423060855296.000ns
ubuntu@ubuntu-xenial:~/go/src/github.com/tutorial/new$
```

The code has been written and implemented. I have referred the sensor and homework codes as a reference.

1. For the Ideal condition, the times in both the clocks (client and server side) should be same so that there is no hindrance in computing latency.
2. Latency = time_received - time_sent where time is computed from Bandwidth and the size of the packet.
3. Latency could be affected by bandwidth, size of the packet and time syncing.
4. These overhead are dynamic since bandwidth ,size,applications ,time varies.
5. The overheads depends on client and server applications which is not easy to overcome because of their complex code and timings .

Data Plane (task 2)

```
ubuntu@ubuntu-xenial:~/go/src/github.com/scionproto/scion/tutorial/new$ go run scion_cli
go -c 19-ffaa:1:148,[192.168.0.119]:30100 -s 19-ffaa:1:152,[192.168.0.110]:30102
INFO[01-22|15:40:44] Started id=76ba45ce goroutine=disp
INFO[01-22|15:40:44] Registered with dispatcher ia=19-ffaa:1:148 host=192.
=30100

Client: 19-ffaa:1:148,[192.168.0.119]:30100
Server: 19-ffaa:1:152,[192.168.0.110]:30102
LATENCY_DATAPLANE_METHOD:
    RTT - 235087198.000ns
    Latency - 117543599.000ns
ubuntu@ubuntu-xenial:~/go/src/github.com/scionproto/scion/tutorial/new$
```

The code has been written and implemented. I have referred the sensor and homework codes as a reference

1. In today's Non SCION Architecture the response from the server to the client would take some different path which would result in different timings and hence RTT would be the double of Latency. However in SCION Path(client to server) = Path(server to client)...Hence RTT would be the double of Latency.

2. The UDP connection (LISTENS TO SCION) in server and ensures the response back to the client only once some response or number is received.

Control Plane(task 3) (code not implemented)

I didn't implement this since I couldn't understand the logic behind its implementation.

1. B responds to A only when A requests for the echo which ensures B's echos are not sent before A's requests.

2. B can counterfeit the results hence extending the RTT. Hence it can be varied with changes done from B's side.

BandWidth

Assumptions. No screenshot since there are several errors in client code.

The server code has been implemented however the client code is incomplete. I am encountering several errors.

In the code (The main function determines the path used for establishing the scion communication and the udp connection is established. A map is created using the send and recieved packets. This map is used in calculating the average bottleneck bandwidth. The output is converted to Mbps after sorting the average time interval consumed for packet transmission ie send and recieved.)

At bottleneck link, the two packets should queue one after the other and not at some other link after that. If both packet reach at almost same time, they could queue together at bottleneck line

-

There wont be packet loss in between. If its lost, there will be big time difference. Moreover, the router treats all packets equally. Once first packet reaches, it will take sometime to find routing direction. Then second one is expected to have same characteristics so it will get processed fast