COL334 Assignment-1

Shrey J. Patel, 2019CS10400

August 2021

1 Networking Tools

1.1 *ifconfig*

I use both my ISPs in wireless configuration. So, both the IP addresses are of the wlp3s0 type.

1. **ISP 1(Vodafone)**

   - **IPv4:** 192.168.1.102
   - **IPv6:** fe80::98a4:e1eb:6fa1:ae04

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ ifconfig wlp3s0
wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.102  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::98a4:e1eb:6fa1:ae04  prefixlen 64  scopeid 0x20<link>
        ether 58:a0:23:29:7f:5c  txqueuelen 1000  (Ethernet)
        RX packets 1504416  bytes 2128769414 (2.1 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 434330  bytes 57777365 (57.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2. **ISP 2(Reliance Jio)**

   - **IPv4:** 192.168.144.31
   - **IPv6:** fe80::6c32:c7a3:e9ca:5928

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ ifconfig wlp3s0
wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.102  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::98a4:e1eb:6fa1:ae04  prefixlen 64  scopeid 0x20<link>
        ether 58:a0:23:29:7f:5c  txqueuelen 1000  (Ethernet)
        RX packets 1504416  bytes 2128769414 (2.1 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 434330  bytes 57777365 (57.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

1.2 *nslookup*

1.2.1 ISP 1

1. **Local DNS server:** 127.0.0.53

   - **google:** 172.217.19.132
   - **facebook:** 31.13.79.35

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.19.132
Name:   www.google.com
Address: 2a00:1450:4006:806::2004

shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.facebook.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 31.13.79.35
Name:   star-mini.c10r.facebook.com
Address: 2a03:2880:f12f:83:face:b00c:0:25de
```

2. **Public DNS server(Google):** 8.8.8.8

   - **google:** 142.250.77.68
   - **facebook:** 157.240.16.35

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.google.com 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.250.77.68
Name:   www.google.com
Address: 2404:6800:4009:81d::2004

shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.facebook.com 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 157.240.16.35
Name:   star-mini.c10r.facebook.com
Address: 2a03:2880:f12f:83:face:b00c:0:25de
```

3. **Public DNS server(Cloudfare):** 1.1.1.1

   - **google:** 142.250.192.132
   - **facebook:** 157.240.7.35

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.google.com 1.1.1.1
Server:         1.1.1.1
Address:        1.1.1.1#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.250.192.132
Name:   www.google.com
Address: 2404:6800:4009:81b::2004

shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.facebook.com 1.1.1.1
Server:         1.1.1.1
Address:        1.1.1.1#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 157.240.7.35
Name:   star-mini.c10r.facebook.com
Address: 2a03:2880:f12f:83:face:b00c:0:25de
```

1.2.2 ISP 2

1. **Local DNS server:** 127.0.0.53

   - **google:** 142.250.193.4
   - **facebook:** 157.240.16.35

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.250.193.4
Name:   www.google.com
Address: 2404:6800:4002:821::2004

shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.facebook.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 157.240.16.35
Name:   star-mini.c10r.facebook.com
Address: 2a03:2880:f12f:83:face:b00c:0:25de
```

2. **Public DNS server(Google):** 8.8.8.8

   - **google:** 142.251.42.36
   - **facebook:** 31.13.79.35

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.google.com 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.251.42.36
Name:   www.google.com
Address: 2404:6800:4009:821::2004

shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking$ nslookup www.facebook.com 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 31.13.79.35
Name:   star-mini.c10r.facebook.com
Address: 2a03:2880:f12f:183:face:b00c:0:25de
```

3. **Public DNS server(Cloudfare):** 1.1.1.1
    - **google:** 142.250.76.196
    - **facebook:** 31.13.79.35

1.2.3 Observations

1. All the above query answers are non-authoritative, which means the query for a particular domain name is requested through a non-authoritative DNS server, like the local host or the public non-authoritative server. A list of authoritative server for a particular domain can be fetched from the command:

    host -t ns domainName

    However, regardless of the type of server used, the IP address is the same and it is sent by the authoritative server, although in case of a non-authoritative answer, the query reply is sent indirectly i.e. forwarded through the local/public DNS server.

2. Different host DNS servers give different IP addresses for the same domain name, which can be attributed to the fact that the same domain name my be assigned to multiple different servers, which is natural for domains that receive heavy traffic. So, different DNS servers may access a different server with the same domain name. The same reason can be applied to explain the fact that even different access networks(i.e. ISPs) also return different IPs for the same domain name. And the number of different servers is fairly representative of the amount of traffic on that site. For instance, Google seems to have the largest number of servers as expected, followed by Facebook. While a relatively less known domain such as IITD home site seems to have a single IP address(103.27.9.24).

3. Even when the ISP is changed, the IP address for the local DNS server is the same i.e. 127.0.0.53#53 (53 denotes the port used by the DNS server). But 127.0.0.53 points to the system's local cache which depends on the operating system. So, all the DNS queries which use the local DNS server(provided by the ISP) ago through this cache first.

1.3 *ping*

I have created a bash shell script named *ping.sh*, which uses ping command on console on different packet sizes and TTF values to perform **binary search** on the maximum value of packet size and the minimum TTL value required for the packet to reach the destination.

1.4 ISP 1

1) www.iitd.ac.in:
Maximum packet size = 1452 + 8 header bytes
Minimum TTL value = 16

2) www.google.com:
Maximum packet size = 68 + 8 header bytes
Minimum TTL value = 18

3) www.facebook.com:
Maximum packet size = 1452 + 8 header bytes
Minimum TTL value = 9

1.5 ISP 2

1) www.iitd.ac.in:
Maximum packet size = 1472 + 8 header bytes
Minimum TTL value = 21

2) www.google.com:
Maximum packet size = 68 + 8 header bytes
Minimum TTL value = 12

3) www.facebook.com:
Maximum packet size = 1452 + 8 header bytes
Minimum TTL value = 12

1.6 Observations

1. The maximum packet size may be dependant on the size allowance of the intermediate routers and switches, and since each of the packets follow different routes to reach different domains, and even same domains but on different access networks, the packet sizes are bound to be different.

2. Similarly, the minimum TTL value is dependant on the length of the route to the destination, and so different domains and different ISP networks have different TTL values.

1.7 *traceroute*

Traceroute implements the ping to send ICMP packets hop-by-hop i.e. for increasing TTL values. It exploits the feature of ping, by which if the packet doesn't reach the destination for the current hop value, then the last router on the path sends an ICMP error report(mostly) before discarding the packet. In this way, traceroute traces the path of routers and switches by their IPs until the destination. I have implemented a traceroute script using this method.

1.7.1 ISP 1: 192.168.1.102

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking/Part1$ traceroute www.iitd.ac.in -I
traceroute to www.iitd.ac.in (103.27.9.24), 64 hops max
  1   192.168.1.1  4.531ms  3.851ms  1.993ms
  2   192.168.0.1  2.194ms  2.101ms  1.961ms
  3   100.76.0.1  13.736ms  11.339ms  16.915ms
  4   203.187.200.184  16.307ms  15.269ms  17.230ms
  5   118.185.41.10  12.888ms  11.008ms  11.363ms
  6   182.19.106.103  21.609ms  25.558ms  22.673ms
  7   14.142.18.97  22.770ms  22.509ms  22.367ms
  8   *  *  *
  9   *  *  *
 10   14.140.210.22  36.134ms  36.297ms  36.505ms
 11   *  *  *
 12   *  *  *
 13   *  *  *
 14   103.27.9.24  37.901ms  43.561ms  45.224ms
```

1.7.2 ISP 2: 192.168.144.31

```
shrey@shrey-Inspiron-7570:~/Shrey/COL334/Basic-Networking/Part1$ traceroute www.iitd.ac.in -I
traceroute to www.iitd.ac.in (103.27.9.24), 64 hops max
  1   192.168.58.113  3.102ms  3.463ms  2.255ms
  2   *  *  *
  3   56.8.119.1  46.521ms  56.8.119.13  37.265ms  56.8.119.9  50.556ms
  4   192.168.38.2  39.778ms  39.617ms  192.168.38.0  39.034ms
  5   192.168.21.235  60.065ms  40.787ms  39.212ms
  6   172.26.101.4  39.791ms  30.227ms  40.013ms
  7   172.26.100.242  30.047ms  39.883ms  38.634ms
  8   192.168.38.23  40.794ms  192.168.38.25  39.830ms  192.168.38.23  54.377ms
  9   192.168.38.24  35.469ms  40.169ms  39.839ms
 10   172.26.40.5  43.681ms  49.519ms  74.146ms
 11   172.16.25.2  51.678ms  50.670ms  63.175ms
 12   172.16.1.218  66.083ms  58.891ms  58.568ms
 13   *  *  *
 14   *  *  *
 15   14.140.210.22  88.991ms  96.869ms  79.608ms
 16   *  *  *
 17   *  *  *
 18   *  *  *
 19   103.27.9.24  151.479ms  76.513ms  73.920ms
```

1.7.3 Observations:

1. According to the implementation of the traceroute, the total number of intermediate routers that the traceroute should encounter must be equal to the minimum TTL value, because this TTL value is the exact length of the route between the source and the destination.

   However, in the case of www.iitd.ac.in, I obtained the number of hops to be equal to **2 less than the minimum TTL value**, and on verifying with the ping command, I found that the

the last three hops in both ISPs all correspond to the routers whose IP address matches with the destination IP address. This might suggest that the Linux implementation of traceroute might be trying to find the first matching IP address for finding the destination instead of the actual destination based on the TTL values.

For instance, the TTL value for ISP 1 is 16, but the hop count of traceroute is 14, while the same for ISP 2 is 21 and its hop count is 19.

2. For the traceroute to get the information regarding the IP address of an intermediate router, the router must send back necessary information back to the source after discarding the packet. However some routers either don't respond to the pings or are not able able to send back the request before the timeout (which is set to 3 by default). The traceroute command sends the next ping and classifies the current router as private/hidden if it doesn't send any response before the timeout.

   So, one way to allow the identification of more routers on the path is to increase the timeout value of every ping. This can be done by using -w tag followed by the desired timeout value. However, this strategy might not be useful for routers which don't respond at all. This might be fixed by using the -I tag to prompt the traceroute to send ICMP packets instead of UDP packets as the routers sometimes don't respond to them because of unreliability.

3. The traceroutes for more traffic heavy domains like Google consist of variable number of hops because of existence of multiple different servers and therefore multiple routes and destinations. However, I found that the number of hidden/private routers are comparatively lesser in these cases, the reason of which is not quite clear to me.

2 Packet Analysis

2.1 DNS filter

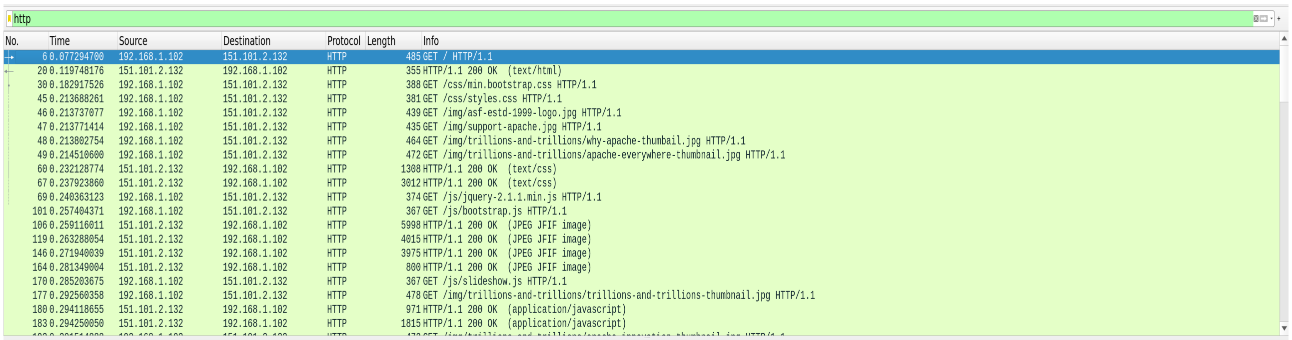
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.102	192.168.1.1	DNS	81	Standard query 0xb967 A apache.org OPT
2	0.040959344	192.168.1.1	192.168.1.102	DNS	97	Standard query response 0xb967 A apache.org A 151.101.2.132 OPT
23	0.182300501	192.168.1.102	192.168.1.1	DNS	91	Standard query 0xcaab A fonts.googleapis.com OPT
26	0.182690744	192.168.1.102	192.168.1.1	DNS	85	Standard query 0x3847 A cse.google.com OPT
28	0.182815811	192.168.1.102	192.168.1.1	DNS	88	Standard query 0xacbd A www.apachecon.com OPT
31	0.195637796	192.168.1.1	192.168.1.102	DNS	107	Standard query response 0xcaab A fonts.googleapis.com A 142.251.42.74 OPT
370	0.376175063	192.168.1.1	192.168.1.102	DNS	101	Standard query response 0x3847 A cse.google.com A 142.250.200.206 OPT
440	0.409493151	192.168.1.1	192.168.1.102	DNS	128	Standard query response 0xacbd A www.apachecon.com CNAME apache.org A 151.101.2.132 OPT
640	0.471802817	192.168.1.102	192.168.1.1	DNS	88	Standard query 0xc2c3 A fonts.gstatic.com OPT
651	0.484684674	192.168.1.1	192.168.1.102	DNS	140	Standard query response 0xc2c3 A fonts.gstatic.com CNAME gstaticadssl.l.google.com A 142.250.67.131 OPT
701	0.570140265	192.168.1.102	192.168.1.1	DNS	86	Standard query 0xb5ad A www.youtube.com OPT
703	0.584270718	192.168.1.1	192.168.1.102	DNS	376	Standard query response 0xb5ad A www.youtube.com CNAME youtube-ui.l.google.com A 142.250.199.142 A 216.58.203.14 A 172.217.167.174 A 142.250.76.206 A 216.58.203.46 A 142.250.67…
1252	1.189156303	192.168.1.102	192.168.1.1	DNS	98	Standard query 0x2331 A googleads.g.doubleclick.net OPT
1253	1.197313246	192.168.1.102	192.168.1.1	DNS	93	Standard query 0xa7e3 A static.doubleclick.net OPT
1254	1.200530436	192.168.1.1	192.168.1.102	DNS	114	Standard query response 0x2331 A googleads.g.doubleclick.net A 216.58.211.194 OPT
1256	1.209170865	192.168.1.1	192.168.1.102	DNS	158	Standard query response 0xa7e3 A static.doubleclick.net CNAME static-doubleclick-net.l.google.com A 142.250.200.198 OPT
1320	1.389610465	192.168.1.102	192.168.1.1	DNS	84	Standard query 0xf22a A yt3.ggpht.com OPT
1334	1.402429290	192.168.1.102	192.168.1.1	DNS	82	Standard query 0x0aeb A i.ytimg.com OPT
1335	1.403709908	192.168.1.1	192.168.1.102	DNS	145	Standard query response 0xf22a A yt3.ggpht.com CNAME photos-ugc.l.googleusercontent.com A 172.217.166.65 OPT
1337	1.413228684	192.168.1.1	192.168.1.102	DNS	338	Standard query response 0x0aeb A i.ytimg.com A 142.250.182.246 A 142.250.182.214 A 172.217.174.86 A 142.250.66.22 A 172.217.167.182 A 142.250.183.22 A 216.58.203.22 A 172.217.1…

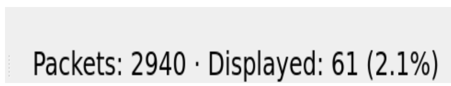
Frame 1: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface wlp3s0, id 0
Ethernet II, Src: IntelCor_29:7f:5c (58:a0:23:29:7f:5c), Dst: Tp-LinkT_fe:cd:5c (98:de:d0:fe:cd:5c)
Internet Protocol Version 4, Src: 192.168.1.102, Dst: 192.168.1.1
User Datagram Protocol, Src Port: 53961, Dst Port: 53
Domain Name System (query)

According to the figure,
192.168.1.102 = Local IP
192.168.1.1 = Local DNS server

Time of first request from host = 0.00 s
Time of first response from DNS server = 0.041 s
So, Difference = **0.041 seconds**

2.2 HTTP filter

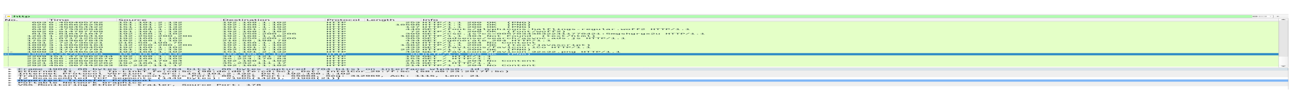




Observations:

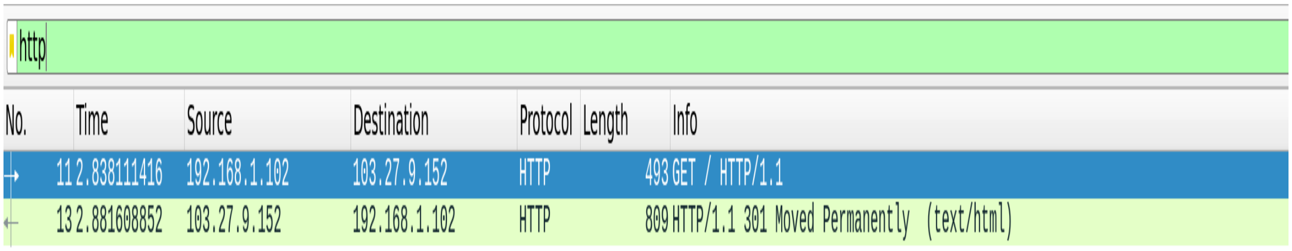
The total number of packets are 61. Most of these packets are transferred between the local host(192.168.1.102) and the IP address 151.101.2.132 (apache.org). All of them are either requests made by the local host to the web site to fetch some content, or are the responses of the website to these requests in the form of sending text or graphics files. The general order of sending the packets seems to be: First, the HTML source file of the website arrives, followed by the CSS and Javascript files which contain information for styles and links/references to other websites. The larger files, like the graphics files containing fonts, icons and images(png) seem to arrive later as expected. So, the simpler framework arrives earlier while the more complex elements arrive later. In fact, the last content delivered to the host by the website is an image file.

2.3 Total Download Time



From part 1 and the figure,
Time of first request from host = 0.00 s
Time when the last content file arrives = 3.217 s
So, total download time = **3.217 seconds**

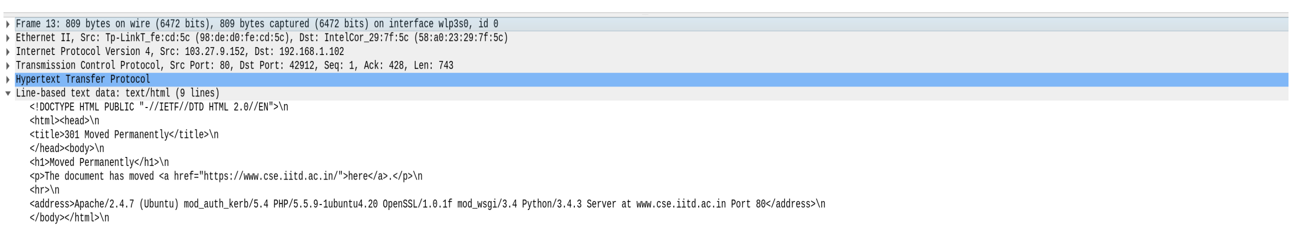
2.4 Comparison



Observations:

Compared to apache.org, cse.iitd.ac.in has very few HTTP requests as evident from the figure. At first glance, it might seem to be related to the amount of content which is loaded on both sites. But both the sites contain text, graphics, etc. while one of the sites has significantly lesser number of HTTP requests. So, that is clearly not the reason. Another possible reason might be the amount of traffic on the site, meaning that because apache.org is a site which is more visited than cse.iitd.ac.in, there is more traffic on it which means that the same amount of content must be fetched using multiple requests while the same content requires only a few requests on site with less traffic. However, this is also not the case as evident from the figure where the size of the content fetched is not even nearly close to the total content of the website.

One plausible reason can be derived from the next figure in which the info in one of the HTTP requests, says that the website is moved permanently to HTTPS from HTTP. HTTPS contains an SSL or a Secure Socket Layer due to which much of its internal content is not available for sniffing. That is the reason why the exact code or graphic content of the website is not available unlike the previous case.



3 Traceroute using Ping

3.1 Specifications:

- **Input:** IP address of the destination
- **Output:**
  1. List of IP addresses of the intermediate hops (console)
  2. RTT vs Hop Number(TTL value) plot (output file)

3.2 Approach

The traceroute utility, as discussed above, traces the path that the packets of information travel from the host to the destination. The intermediate routers/switches/gateways are found by exploiting the ICMP or the Internet Control Message Protocol. An ICMP packet header has the information regarding its destination an the current Time to Live value. Whenever this packet reaches any router, the router reads the destination address and forwards it to another router on the path towards the destination and in the process, it decrements the TTL value of the packet. If the TTL value reaches zero, then the router discards the packet. However, following the ICMP protocol, some routers send an ICMP echo reply which contains the report of the discarded packet like the error type(mostly TTL exceeded in this case), along with the IP address of the router of the router itself.

3.3 Algorithm

- I have used the idea in the approach and transmit ICMP packets using the way the in-built *ping* utility sends packets, i.e. using a custom TTL value. We want to trace the entire path from the host to the destination, so we start with a TTL value of 1 and keep sending ICMP packets with increasing TTL values towards the destination, until it reaches the destination.
- In every iteration of sending the packet, we either receive an ICMP echo response, in which case we track the IP address of the router which sends it, thus tracing the path incrementally and noting the round trip time of every iteration, or some routers don't respond before discarding the packet, in which case, we set the round trip time of such routers to be zero.
- But how do we decide whether the packet has reached the destination? The strategy, that comparing the IP address of the intermediate router with the destination IP, seems to be plausible as our end goal is to reach the destination. But sometimes, certain destinations are preceded by gateways that mask the private IPs of certain routers in the internal network of the destination with the public IP, which here is same as the destination IP.
- Thus, finding a router with a particular IP may not guarantee that the packet has reached the destination. This can easily be confirmed by the inbuilt *ping command* which returns a TTL exceeded error message for such "seemingly" destination routers.

  For instance, while tracing the route for *www.iitd.ac.in*, the last three IPs have the same IP address i.e. 103.27.9.24, but only the third one is the actual destination. A traceroute implementation which simply compares the IP for termination would in that case, terminate at the first IP, and not reach at the real destination.

- So, again I have used the information stored in the received ICMP packet. Each ICMP packet header also stores the type code of the error which caused the router sending this packet to discard it. The error in most of the routers is of the type "TTL exceeded" (ICMP code 11). And in the 64 byte packet, this code occupies the 20th byte. So, for termination, we simply keep checking the 20th byte of the received packet. If the ICMP code is 11, it means that the TTL value is not sufficient to reach the destination, and if it is 0, then it means that the echo response has been sent by the destination. This forms the termination condition.

3.4 Implementation

- I have used C++ sockets for sending and receiving the ICMP packets across the network. The header and message contained in the ICMP packet are defined by the ICMP struct which is already defined by the GNU library.
- Each packet header stores the information regarding the IP address of its destination, the type of ICMP request/response and the checksum value (which is used for checking the manipulation/contamination of the packet data). It also stores the ID of the process through which it was created and sent, so as to maintain synchronisation between the sending and receiving of the packets i.e. they are received in the same order in which they are sent.
- At every iteration, we use this info stored in the packet for reporting and for termination, we refer to the error type code.
- The socket settings are set for the desired TTL value of the packet to be sent. Additionally, we also need to account for the routers which don't respond to the ICMP requests. So, to avoid infinite wait times, we also fix the timeout value of the socket for the packet to be received. If no packet is received from a router within this timeout after sending, we ignore the said router.
- The round trip time for all iterations have been calculated using the chrono library of C++. We simply find the difference in times between the complete sending/receiving processes to obtain the round trip time.

Execution: The code for the traceroute can be run using the *make host=domainName* command in the console.

3.5 Output:

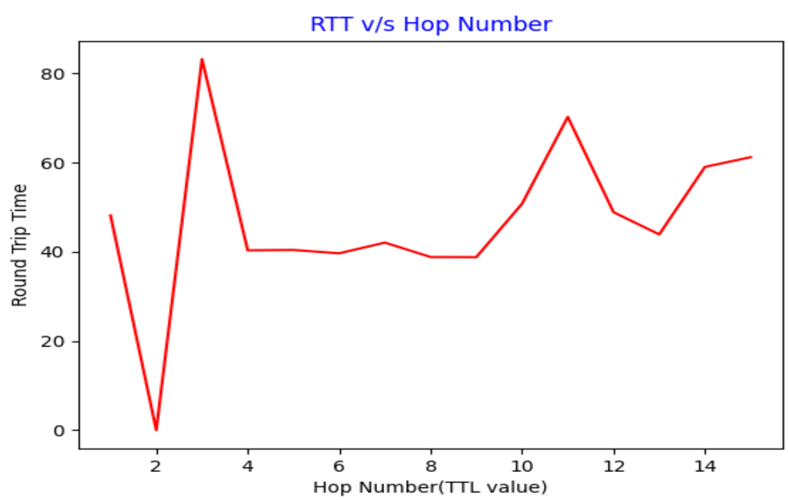
I have used the sites **www.google.com**, **www.facebook.com** and **www.iitd.ac.in** as testcases. The corresponding console outputs and their plots are shown below. In each row, the first field is the hop number, the second field is the corresponding IP address and the third field is the round trip time in microseconds(for better plotting). (Continued on next page)

3.5.1 Google(142.251.42.4)

Console output:

```
Hop,IP,RTT
1,192.168.240.42,4835
2,*,0
3,56.6.253.205,85988
4,192.168.38.6,29275
5,192.168.21.235,51410
6,172.26.101.6,28625
7,172.26.100.246,40566
8,192.168.38.25,39372
9,192.168.38.22,39671
10,172.26.40.7,52455
11,172.16.25.4,50902
12,172.16.1.220,45213
13,74.125.37.7,56545
14,209.85.248.61,44576
15,142.251.42.4,50063
```

Plot:

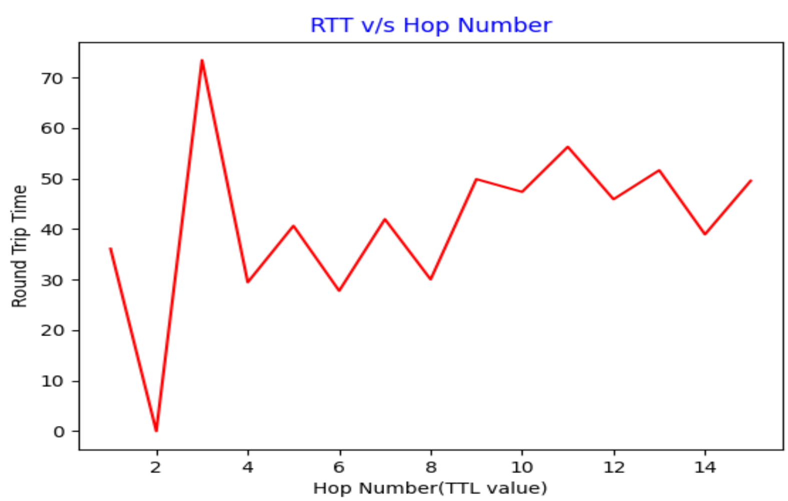


3.5.2 Facebook(31.13.79.35)

Console output:

```
Hop,IP,RTT
1,192.168.240.42,49579
2,*,0
3,10.72.234.133,79913
4,192.168.38.4,30348
5,192.168.21.237,40410
6,172.26.101.6,40284
7,172.26.100.246,40492
8,192.168.38.29,39162
9,192.168.38.26,50590
10,172.16.92.145,38276
11,172.16.25.0,69114
12,172.16.1.218,52163
13,157.240.53.21,61364
14,157.240.39.217,47931
15,31.13.79.35,50003
```

Plot:



3.5.3 IITD(103.27.9.24)

Console output:

```
Hop,IP,RTT
1,192.168.240.42,4124
2,*,0
3,10.72.234.145,92744
4,192.168.38.6,79176
5,192.168.38.5,85229
6,172.26.101.6,75815
7,172.26.100.247,79883
8,192.168.38.27,80680
9,192.168.38.28,78983
10,172.16.92.145,55222
11,172.16.25.2,64505
12,172.16.1.218,54194
13,172.26.40.64,62841
14,115.249.214.165,57230
15,115.255.253.18,60815
16,115.249.198.97,78896
17,*,0
18,*,0
19,*,0
20,*,0
21,*,0
22,*,0
23,103.27.9.24,137432
24,103.27.9.24,74062
25,103.27.9.24,55171
```

Plot: