Contents

Chapter 1

Networking Tools (Windows)

a Tool: ipconfig

a.1 Goal:

To find IP address of laptop (Wifi) and observe its variation with change in Internet Service Provider (ISP) and Default Gateway.

a.2 Procedure:

- To get local IP and Default Gateway, execute `ipconfig` command on the machine.
- To get public IP and ISP, visit the site `www.whoismyisp.org` on the machine.

a.3 Observations/Remarks:

I followed the procedure (1.1.2) for **three** different settings (ISP/hotspots). I present my observations in the form of a table below:-

ISP	Public IPv4	Default Gateway	Local IPv4
BHARTI	122.161.143.242	192.168.1.1	192.168.1.9
Bharti Airtel	223.225.62.104	192.168.89.164	192.168.89.105
Bharti Airtel	106.223.112.148	192.168.43.148	192.168.43.105

Here, Public IPv4 and Local IPv4 indicate my laptop's interface's (Wifi) public and local IP addresses (version 4) respectively. Few **remarks**:

- On changing the ISP from BHARTI to Bharti Airtel, all three addresses change. This can be explained as on changing the ISP, I am also changing my local network environment (because I switch to my mobile's hotspot). As a result both the Default Gateway and local IP address of my laptop gets changed. Also since, the public IP address is provided by the ISP itself, change in ISP, generally implies a change in public IP address as well.
- On keeping the ISP same, but changing between mobile hostspots, again all three addresses change. Change in Default Gateway and local IP address can be explained by using the same argument as used in the above point (change in local network environment). Also, when connected to different mobile hotspots, the router to which the laptop is connected is different, and hence, the public IP address of the router (and laptop) will be different even when connected to the same ISP.

b Tool: `nslookup`

b.1 Goal:

To find Public IP addresses of `www.google.com` and `www.facebook.com` using `nslookup` command and observe its variation with change in DNS servers.

b.2 Procedure:

To get public IP, execute the command `nslookup <host> [DNS server]`

b.3 Observations/Remarks:

I followed the procedure (1.2.2) for both the hosts (google and facebook) and for **five** different DNS servers. I present my observations in the form of a table below:-

Site	DNS Server	Host IPv4	Host IPv6
Google	Broadcom.Home (192.168.1.1)	172.217.166.4	2404:6800:4002:804::2004
Google	Google (8.8.8.8)	142.250.193.228	2404:6800:4002:813::2004
Google	Cisco (208.67.222.222)	142.250.192.196	2404:6800:4002:824::2004
Google	Cloudflare (1.1.1.1)	142.250.194.68	2404:6800:4002:820::2004
Google	Quad9 (9.9.9.9)	142.250.207.68	2404:6800:4005:80b::2004
Facebook	Broadcom.Home (192.168.1.1)	157.240.16.35	2a03:2880:f12f:83:face:b00c:0:25de
Facebook	Google (8.8.8.8)	157.240.198.35	2a03:2880:f144:82:face:b00c:0:25de
Facebook	Cisco (208.67.222.222)	157.240.198.35	2a03:2880:f144:82:face:b00c:0:25de
Facebook	Cloudflare (1.1.1.1)	157.240.198.35	2a03:2880:f144:82:face:b00c:0:25de
Facebook	Quad9 (9.9.9.9)	157.240.235.35	2a03:2880:f10c:381:face:b00c:0:25de

Here **Google** and **Facebook** refer to hostnames `www.google.com` and `www.facebook.com` respectively. Also, **Host IPv4** and **Host IPv6** refer to host's public IP (version 4) address and public IP (version 6) address respectively. Few **remarks**:

- All the responses/answers from DNS servers are **Non-authoritative**.
- On changing the DNS server, the returned IPv4 and IPv6 addresses of both the hosts are changing in most situations. This can be explained by the fact that there can be multiple replicated servers with the same alias, like `www.google.com`. Since, the answer is non-authoritative, the IP address is coming from the DNS server's local cache (possibly after forwarding). Since, every DNS server has its own local cache, it is not unlikely that they will have different set of type A records stored for the same hostname (as there are multiple valid IPs). At the same time, two DNS servers can also have similar type A records in their cache. This explains why **three** out of **five** DNS servers gave the same response for `www.facebook.com`.
- Another important point is that even if two DNS servers are forwarding answers from the same authoritative name server, they may give different answers due to round-robin load-balancing algorithm.

c Tool: `ping`

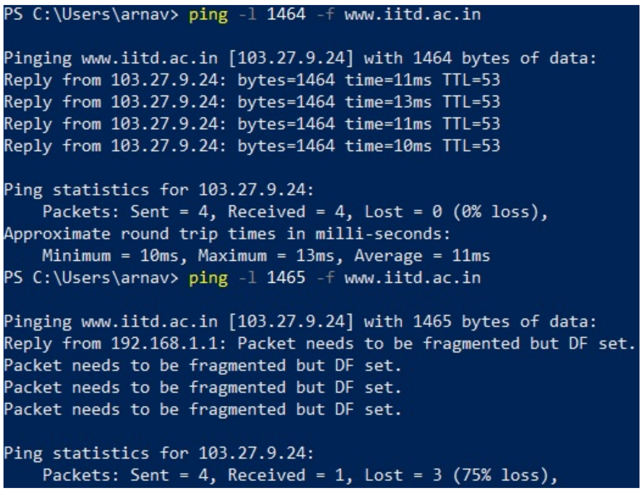
I experimented with `ping` command (Windows) and tried sending packets with different sizes (`-l`), different TTL values (`-i`), different count (`-n`) and *Do Not Fragment* flag (`-f`) to `www.iitd.ac.in`, `www.google.com` and `www.facebook.com`. From the experimentation, I determined the maximum packet size that I was able to send via `ping`. I present my observations below:

Host	Do Not Fragment	Max. Packet Size	MTU*
www.iitd.ac.in	Yes	1464 bytes	1492 bytes
www.iitd.ac.in	No	1472 bytes	1500 bytes
www.google.com	Yes	1464 bytes	1492 bytes
www.google.com	No	1464 bytes	1492 bytes
www.facebook.com	Yes	1464 bytes	1492 bytes
www.facebook.com	No	1464 bytes	1492 bytes

***MTU:-** denotes the **Maximum Transmission Unit** (in bytes) for the corresponding network path. It is determined by adding `20` (IP header) `+ 8` (ICMP header) bytes to the max packet size

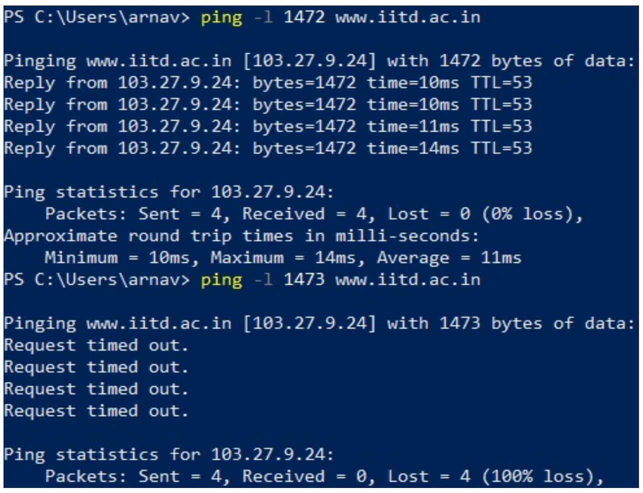
c.1 Output/Remarks:

- With `-f` flag set, the largest ping packet that I was able to transmit (without loss) was the same for all the three domains (`= 1464 bytes`)
- With `-f` flag off (fragmentation), the largest ping packet that I was able to transmit (without loss) was the same for **Google** and **Facebook** (`= 1464 bytes`), but was `1472 bytes` for **IITD** server.



```
PS C:\Users\arnav> ping -l 1464 -f www.iitd.ac.in

Pinging www.iitd.ac.in [103.27.9.24] with 1464 bytes of data:
Reply from 103.27.9.24: bytes=1464 time=11ms TTL=53
Reply from 103.27.9.24: bytes=1464 time=13ms TTL=53
Reply from 103.27.9.24: bytes=1464 time=11ms TTL=53
Reply from 103.27.9.24: bytes=1464 time=10ms TTL=53

Ping statistics for 103.27.9.24:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum = 13ms, Average = 11ms
PS C:\Users\arnav> ping -l 1465 -f www.iitd.ac.in

Pinging www.iitd.ac.in [103.27.9.24] with 1465 bytes of data:
Reply from 192.168.1.1: Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.

Ping statistics for 103.27.9.24:
    Packets: Sent = 4, Received = 1, Lost = 3 (75% loss),
```

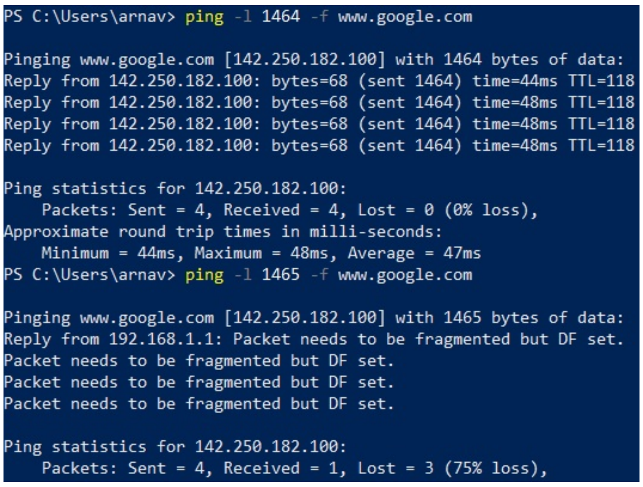
(a)



```
PS C:\Users\arnav> ping -l 1472 www.iitd.ac.in

Pinging www.iitd.ac.in [103.27.9.24] with 1472 bytes of data:
Reply from 103.27.9.24: bytes=1472 time=10ms TTL=53
Reply from 103.27.9.24: bytes=1472 time=10ms TTL=53
Reply from 103.27.9.24: bytes=1472 time=11ms TTL=53
Reply from 103.27.9.24: bytes=1472 time=14ms TTL=53

Ping statistics for 103.27.9.24:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum = 14ms, Average = 11ms
PS C:\Users\arnav> ping -l 1473 www.iitd.ac.in

Pinging www.iitd.ac.in [103.27.9.24] with 1473 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 103.27.9.24:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

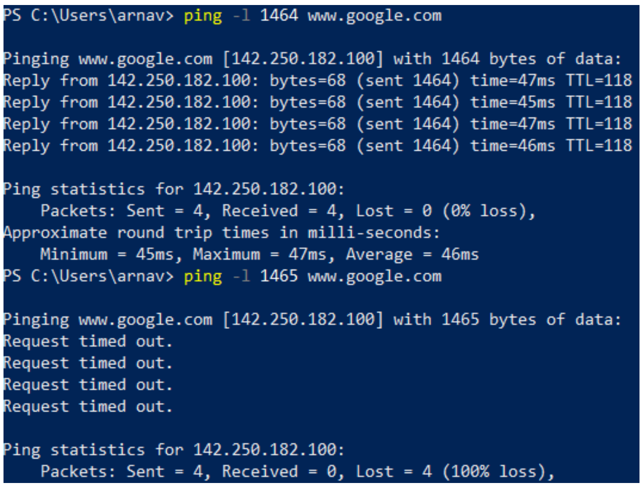
(b)

Figure 1.1: `ping` command to `www.iitd.ac.in`: (a) `-f` ON, (b) `-f` OFF



```
PS C:\Users\arnav> ping -l 1464 -f www.google.com

Pinging www.google.com [142.250.182.100] with 1464 bytes of data:
Reply from 142.250.182.100: bytes=68 (sent 1464) time=44ms TTL=118
Reply from 142.250.182.100: bytes=68 (sent 1464) time=48ms TTL=118
Reply from 142.250.182.100: bytes=68 (sent 1464) time=48ms TTL=118
Reply from 142.250.182.100: bytes=68 (sent 1464) time=48ms TTL=118

Ping statistics for 142.250.182.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 44ms, Maximum = 48ms, Average = 47ms
PS C:\Users\arnav> ping -l 1465 -f www.google.com

Pinging www.google.com [142.250.182.100] with 1465 bytes of data:
Reply from 192.168.1.1: Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.

Ping statistics for 142.250.182.100:
    Packets: Sent = 4, Received = 1, Lost = 3 (75% loss),
```

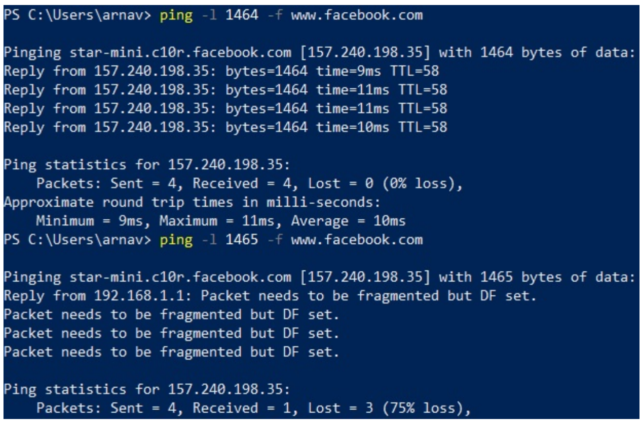
(a)



```
PS C:\Users\arnav> ping -l 1464 www.google.com

Pinging www.google.com [142.250.182.100] with 1464 bytes of data:
Reply from 142.250.182.100: bytes=68 (sent 1464) time=47ms TTL=118
Reply from 142.250.182.100: bytes=68 (sent 1464) time=45ms TTL=118
Reply from 142.250.182.100: bytes=68 (sent 1464) time=47ms TTL=118
Reply from 142.250.182.100: bytes=68 (sent 1464) time=46ms TTL=118

Ping statistics for 142.250.182.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 45ms, Maximum = 47ms, Average = 46ms
PS C:\Users\arnav> ping -l 1465 www.google.com

Pinging www.google.com [142.250.182.100] with 1465 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 142.250.182.100:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

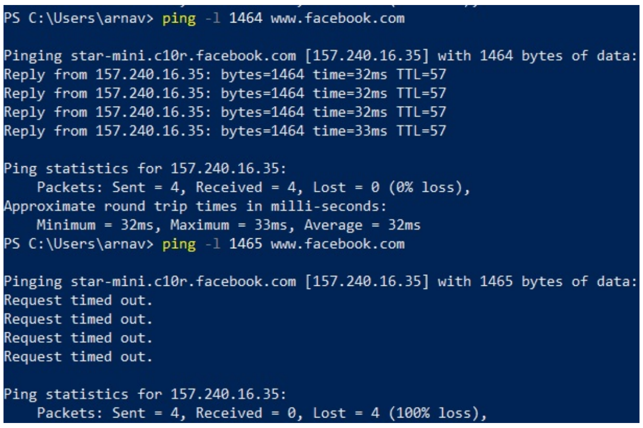
(b)

Figure 1.2: `ping` command to `www.google.com`: (a) `-f` ON, (b) `-f` OFF



```
PS C:\Users\arnav> ping -l 1464 -f www.facebook.com

Pinging star-mini.c10r.facebook.com [157.240.198.35] with 1464 bytes of data:
Reply from 157.240.198.35: bytes=1464 time=9ms TTL=58
Reply from 157.240.198.35: bytes=1464 time=11ms TTL=58
Reply from 157.240.198.35: bytes=1464 time=11ms TTL=58
Reply from 157.240.198.35: bytes=1464 time=10ms TTL=58

Ping statistics for 157.240.198.35:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 9ms, Maximum = 11ms, Average = 10ms
PS C:\Users\arnav> ping -l 1465 -f www.facebook.com

Pinging star-mini.c10r.facebook.com [157.240.198.35] with 1465 bytes of data:
Reply from 192.168.1.1: Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.
Packet needs to be fragmented but DF set.

Ping statistics for 157.240.198.35:
    Packets: Sent = 4, Received = 1, Lost = 3 (75% loss),
```

(a)



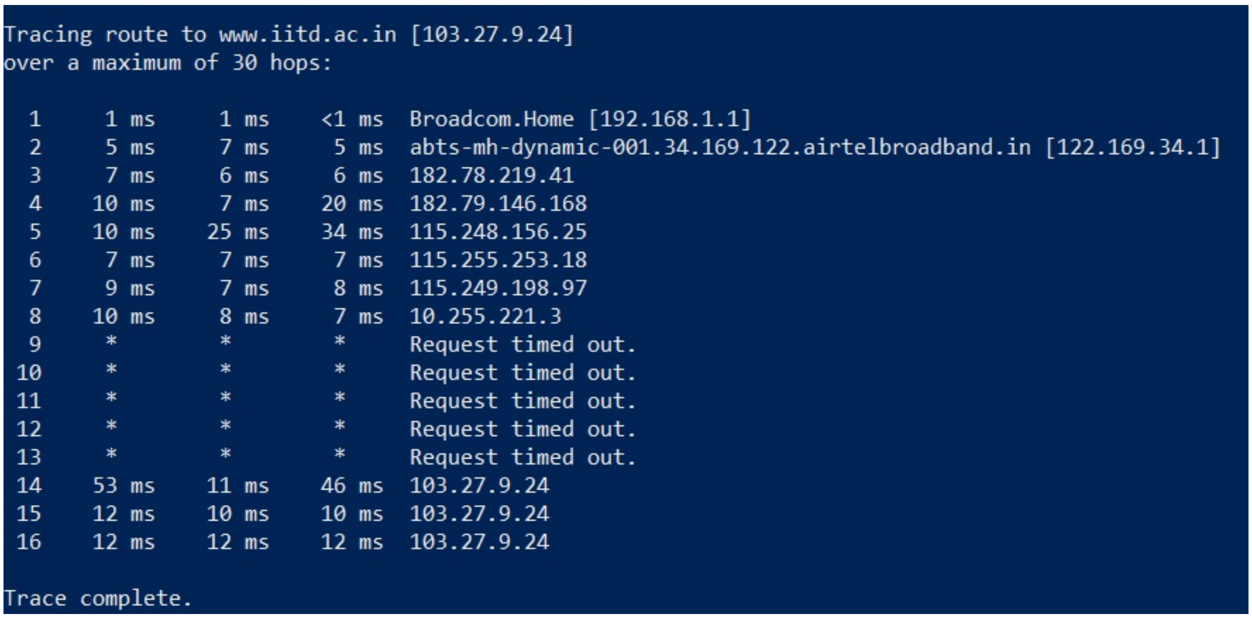
```
PS C:\Users\arnav> ping -l 1464 www.facebook.com

Pinging star-mini.c10r.facebook.com [157.240.16.35] with 1464 bytes of data:
Reply from 157.240.16.35: bytes=1464 time=32ms TTL=57
Reply from 157.240.16.35: bytes=1464 time=32ms TTL=57
Reply from 157.240.16.35: bytes=1464 time=32ms TTL=57
Reply from 157.240.16.35: bytes=1464 time=33ms TTL=57

Ping statistics for 157.240.16.35:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 32ms, Maximum = 33ms, Average = 32ms
PS C:\Users\arnav> ping -l 1465 www.facebook.com

Pinging star-mini.c10r.facebook.com [157.240.16.35] with 1465 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 157.240.16.35:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

(b)

Figure 1.3: `ping` command to `www.facebook.com`: (a) `-f` ON, (b) `-f` OFF

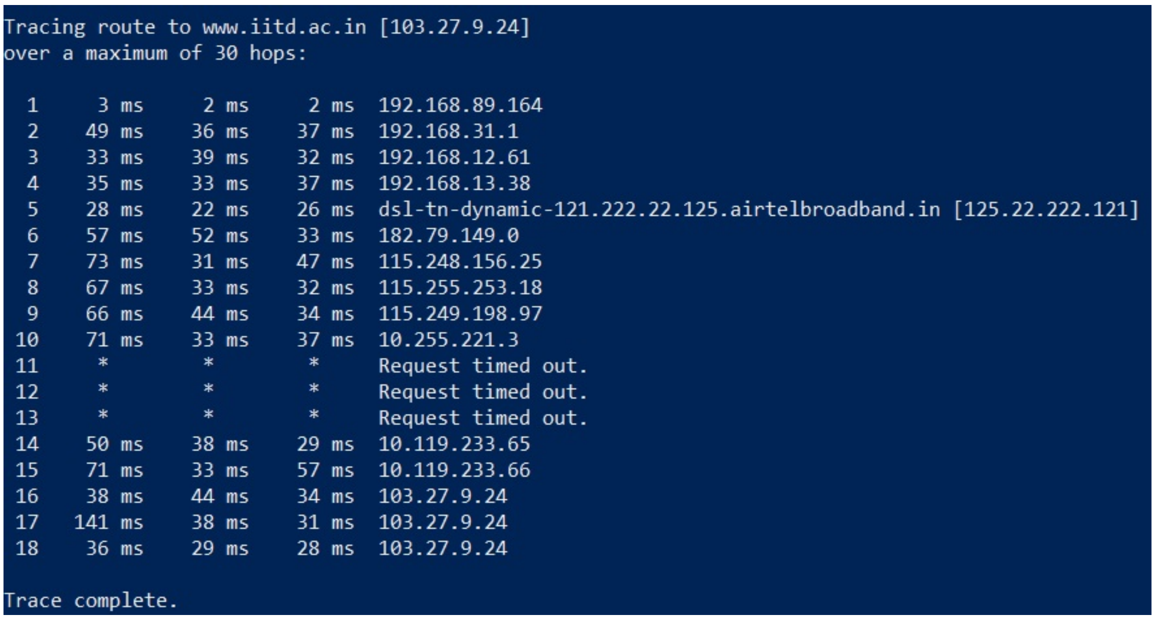
d Tool: **tracert**

I ran traceroute for `www.iitd.ac.in` using two different ISPs. I present my findings below:



```
Tracing route to www.iitd.ac.in [103.27.9.24]
over a maximum of 30 hops:

  1     1 ms     1 ms    <1 ms  Broadcom.Home [192.168.1.1]
  2     5 ms     7 ms     5 ms  abts-mh-dynamic-001.34.169.122.airtelbroadband.in [122.169.34.1]
  3     7 ms     6 ms     6 ms  182.78.219.41
  4    10 ms     7 ms    20 ms  182.79.146.168
  5    10 ms    25 ms    34 ms  115.248.156.25
  6     7 ms     7 ms     7 ms  115.255.253.18
  7     9 ms     7 ms     8 ms  115.249.198.97
  8    10 ms     8 ms     7 ms  10.255.221.3
  9     *        *        *     Request timed out.
 10     *        *        *     Request timed out.
 11     *        *        *     Request timed out.
 12     *        *        *     Request timed out.
 13     *        *        *     Request timed out.
 14    53 ms    11 ms    46 ms  103.27.9.24
 15    12 ms    10 ms    10 ms  103.27.9.24
 16    12 ms    12 ms    12 ms  103.27.9.24

Trace complete.
```

Figure 1.4: Console output for command `tracert www.iitd.ac.in` (ISP BHARTI)



```
Tracing route to www.iitd.ac.in [103.27.9.24]
over a maximum of 30 hops:

  1     3 ms     2 ms     2 ms  192.168.89.164
  2    49 ms    36 ms    37 ms  192.168.31.1
  3    33 ms    39 ms    32 ms  192.168.12.61
  4    35 ms    33 ms    37 ms  192.168.13.38
  5    28 ms    22 ms    26 ms  dsl-tn-dynamic-121.222.22.125.airtelbroadband.in [125.22.222.121]
  6    57 ms    52 ms    33 ms  182.79.149.0
  7    73 ms    31 ms    47 ms  115.248.156.25
  8    67 ms    33 ms    32 ms  115.255.253.18
  9    66 ms    44 ms    34 ms  115.249.198.97
 10    71 ms    33 ms    37 ms  10.255.221.3
 11     *        *        *     Request timed out.
 12     *        *        *     Request timed out.
 13     *        *        *     Request timed out.
 14    50 ms    38 ms    29 ms  10.119.233.65
 15    71 ms    33 ms    57 ms  10.119.233.66
 16    38 ms    44 ms    34 ms  103.27.9.24
 17   141 ms    38 ms    31 ms  103.27.9.24
 18    36 ms    29 ms    28 ms  103.27.9.24

Trace complete.
```

Figure 1.5: Console output for command `tracert www.iitd.ac.in` (ISP Bharti Airtel)

d.1 Observations:

For ISP BHARTI (d.1)

- **Maximum number of hops:** 30
- **Waiting time:** 4000 ms
- **Number of packets per hop:** 3
- **Private IP addresses:** 192.168.1.1 (Hop #1), 10.255.221.3 (Hop #8)
- **Routers that do not respond:** Hop #9 to Hop #13 (both inclusive)

For ISP Bharti Airtel (d.2)

- **Maximum number of hops:** 30
- **Waiting time:** 4000 ms
- **Number of packets per hop:** 3
- **Private IP addresses:** 192.168.89.164 192.168.31.1 192.168.12.61 192.168.13.38 (Hop #1 to Hop #4), 10.255.221.3 (Hop #10), 10.119.233.65 10.119.233.66 (Hop #14 to Hop #15)
- **Routers that do not respond:** Hop #11 to Hop #13 (both inclusive)

d.2 Remarks:

Q. If some paths default to IPv6, then how you can force traceroute to use IPv4?
Ans. On Windows/Linux, we can use `-4` flag to force traceroute to use IPv4.

Q. What changes can you make to traceroute request for some of the missing routers to reply?
Ans. There can be multiple reasons as to why some routers do not **seem** to reply. Sometimes the reason has got to do something with ISP and/or the router's personal configuration (for example, a router may be configured not to send back ICMP Code 11 (TTL exceeded) packet, if it is beneficial from a security aspect etc.). Nevertheless, I suggest some changes that can be done to **traceroute** request, which can *probably* make some of the missing routers to reply:

- **Waiting Time:** Try changing the waiting time of the traceroute request using `-w` flag. Sometimes due to high network congestion, a router may take some time to acknowledge the ICMP packet (low priority), and respond to it. If this time is greater than the default waiting time (4 s on Windows), then the request will time-out and no packet will be received. Hence, increasing the waiting time can help mitigate this issue.
- **Packet Count:** Try changing the number of probe packets sent per hop using `-q` flag (Linux). Similar to reasons above, a single packet may experience extreme congestion and/or loss (very rare), and therefore may not be indicative of the actual situation. Hence, sending multiple probe packets can help in getting response from the corresponding router.
- **Protocol Used:** Generally, Linux uses UDP packets for probing. Since, UDP is unreliable, firewalls can be setup to intercept and discard such packets, leading to packet loss and no response from router. In such cases, using reliable protocols like ICMP and TCP can help. Packet protocol can be changed by using `-I` (ICMP) and `-T` (TCP) flags respectively. TCP is even more reliable than ICMP (as some firewalls may block ICMP packets as well), however, using TCP packets does not necessarily give an edge over ICMP, as the response from router is almost always going to be an ICMP packet (Code 0, 11 etc.).

Chapter 2

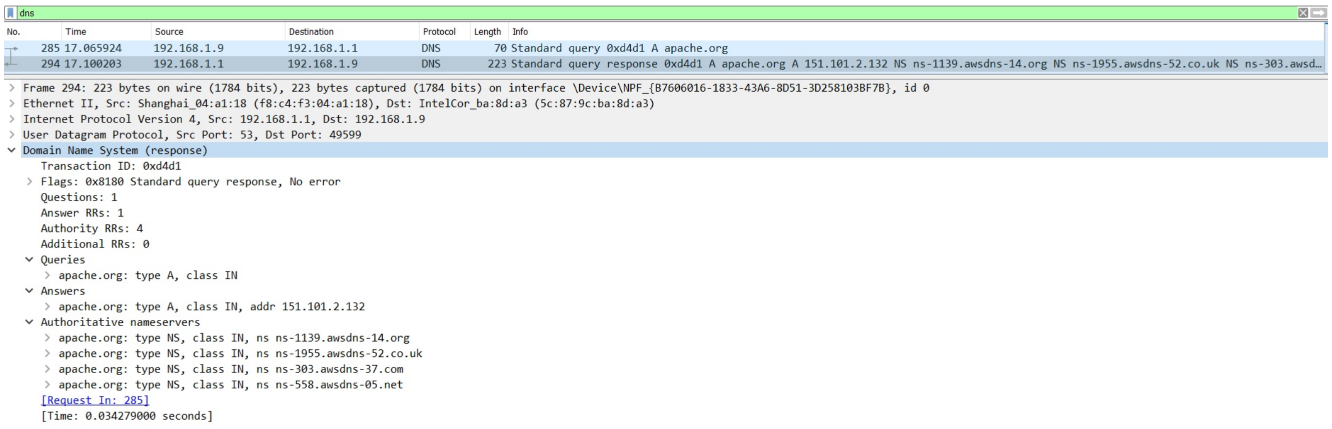
Packet Analysis (Wireshark)

For all parts below, local DNS and browser cache was flushed before visiting website and sniffing packets (using Wireshark).

a DNS Filter

I applied DNS filter on packets and noted down the request-response time for `type=A` DNS query.

Time between server response and DNS query request `= 0.034279000 seconds` $\approx$ `34 ms`

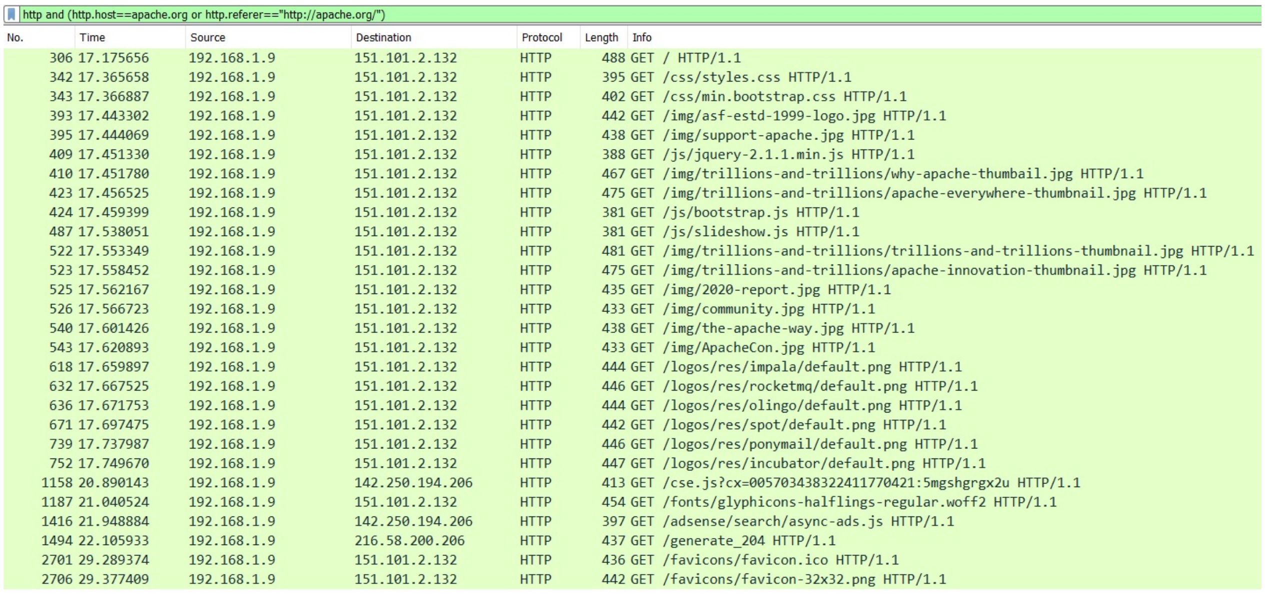


```
dns
No.   Time       Source        Destination   Protocol Length Info
  285 17.065924  192.168.1.9   192.168.1.1   DNS          70 Standard query 0xd4d1 A apache.org
  294 17.100203  192.168.1.1   192.168.1.9   DNS         223 Standard query response 0xd4d1 A apache.org A 151.101.2.132 NS ns-1139.awsdns-14.org NS ns-1955.awsdns-52.co.uk NS ns-303.awsd…

> Frame 294: 223 bytes on wire (1784 bits), 223 bytes captured (1784 bits) on interface \Device\NPF_{B7606016-1833-43A6-8D51-3D258103BF7B}, id 0
> Ethernet II, Src: Shanghai_04:a1:18 (f8:c4:f3:04:a1:18), Dst: IntelCor_ba:8d:a3 (5c:87:9c:ba:8d:a3)
> Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.9
> User Datagram Protocol, Src Port: 53, Dst Port: 49599
v Domain Name System (response)
    Transaction ID: 0xd4d1
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 4
    Additional RRs: 0
  v Queries
    > apache.org: type A, class IN
  v Answers
    > apache.org: type A, class IN, addr 151.101.2.132
  v Authoritative nameservers
    > apache.org: type NS, class IN, ns ns-1139.awsdns-14.org
    > apache.org: type NS, class IN, ns ns-1955.awsdns-52.co.uk
    > apache.org: type NS, class IN, ns ns-303.awsdns-37.com
    > apache.org: type NS, class IN, ns ns-558.awsdns-05.net
    [Request In: 285]
    [Time: 0.034279000 seconds]
```

Figure 2.1: Wireshark Output (DNS filter) and request/response details (`apache.org`)

b HTTP Filter

I applied HTTP filter on packets and noted down the HTTP requests being made to `apache.org` Web server and other referenced Web servers. A total of **28 HTTP requests** were generated by my browser (Microsoft Edge). This tells me that a Web page consists of many objects (or files). The base HTML defines the structure of the Web page and sequencing of Web objects (with URL) in page. Hence, HTML file is the first object which is fetched by the browser from the server. Next, the browser starts parsing the HTML file of a possibly complex Web page and makes request to the server as and when (pipelined GETs) it comes across URL of an unavailable object (like image, video, CSS/JS file etc.). Also, the browser does not wait for reception of all objects before it renders the Web page, rather it renders the Web objects as and when they are received (and ready). The request order will depend on how the browser parses the HTML file. Request order for current example: CSS → Images/JS → Logos → Fonts → Favicon (see screenshot)



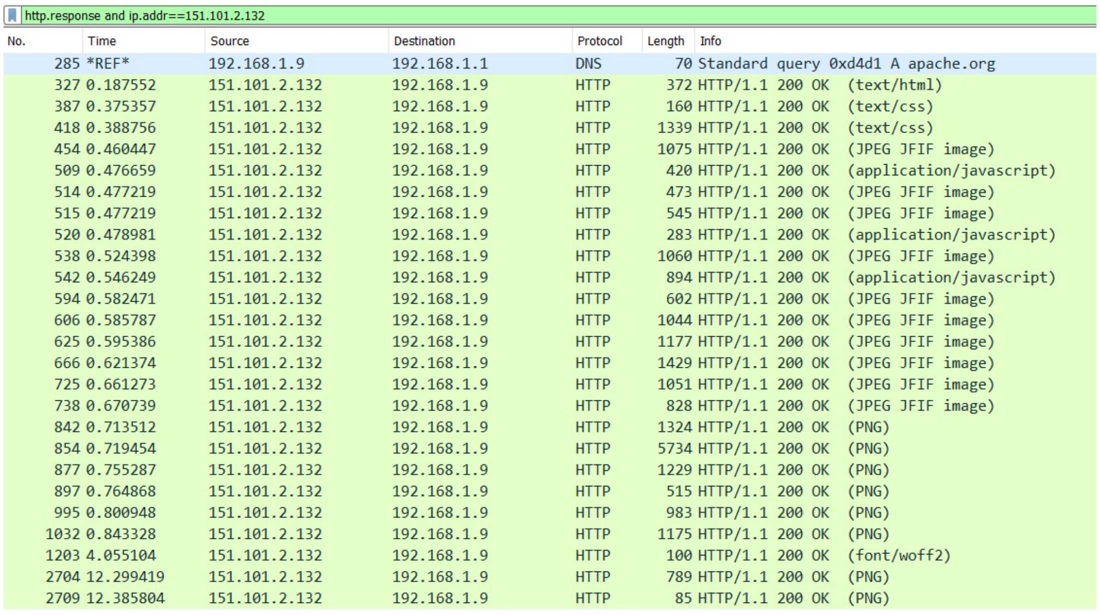
```
http and (http.host==apache.org or http.referer=="http://apache.org/")
No.   Time        Source         Destination      Protocol  Length Info
  306 17.175656   192.168.1.9    151.101.2.132    HTTP      488 GET / HTTP/1.1
  342 17.365658   192.168.1.9    151.101.2.132    HTTP      395 GET /css/styles.css HTTP/1.1
  343 17.366887   192.168.1.9    151.101.2.132    HTTP      402 GET /css/min.bootstrap.css HTTP/1.1
  393 17.443302   192.168.1.9    151.101.2.132    HTTP      442 GET /img/asf-estd-1999-logo.jpg HTTP/1.1
  395 17.444069   192.168.1.9    151.101.2.132    HTTP      438 GET /img/support-apache.jpg HTTP/1.1
  409 17.451330   192.168.1.9    151.101.2.132    HTTP      388 GET /js/jquery-2.1.1.min.js HTTP/1.1
  410 17.451780   192.168.1.9    151.101.2.132    HTTP      467 GET /img/trillions-and-trillions/why-apache-thumbail.jpg HTTP/1.1
  423 17.456525   192.168.1.9    151.101.2.132    HTTP      475 GET /img/trillions-and-trillions/apache-everywhere-thumbnail.jpg HTTP/1.1
  424 17.459399   192.168.1.9    151.101.2.132    HTTP      381 GET /js/bootstrap.js HTTP/1.1
  487 17.538051   192.168.1.9    151.101.2.132    HTTP      381 GET /js/slideshow.js HTTP/1.1
  522 17.553349   192.168.1.9    151.101.2.132    HTTP      481 GET /img/trillions-and-trillions/trillions-and-trillions-thumbnail.jpg HTTP/1.1
  523 17.558452   192.168.1.9    151.101.2.132    HTTP      475 GET /img/trillions-and-trillions/apache-innovation-thumbnail.jpg HTTP/1.1
  525 17.562167   192.168.1.9    151.101.2.132    HTTP      435 GET /img/2020-report.jpg HTTP/1.1
  526 17.566723   192.168.1.9    151.101.2.132    HTTP      433 GET /img/community.jpg HTTP/1.1
  540 17.601426   192.168.1.9    151.101.2.132    HTTP      438 GET /img/the-apache-way.jpg HTTP/1.1
  543 17.620893   192.168.1.9    151.101.2.132    HTTP      433 GET /img/ApacheCon.jpg HTTP/1.1
  618 17.659897   192.168.1.9    151.101.2.132    HTTP      444 GET /logos/res/impala/default.png HTTP/1.1
  632 17.667525   192.168.1.9    151.101.2.132    HTTP      446 GET /logos/res/rocketmq/default.png HTTP/1.1
  636 17.671753   192.168.1.9    151.101.2.132    HTTP      444 GET /logos/res/olingo/default.png HTTP/1.1
  671 17.697475   192.168.1.9    151.101.2.132    HTTP      442 GET /logos/res/spot/default.png HTTP/1.1
  739 17.737987   192.168.1.9    151.101.2.132    HTTP      446 GET /logos/res/ponymail/default.png HTTP/1.1
  752 17.749670   192.168.1.9    151.101.2.132    HTTP      447 GET /logos/res/incubator/default.png HTTP/1.1
 1158 20.890143   192.168.1.9    142.250.194.206  HTTP      413 GET /cse.js?cx=005703438322411770421:5mgshgrgx2u HTTP/1.1
 1187 21.040524   192.168.1.9    151.101.2.132    HTTP      454 GET /fonts/glyphicons-halflings-regular.woff2 HTTP/1.1
 1416 21.948884   192.168.1.9    142.250.194.206  HTTP      397 GET /adsense/search/async-ads.js HTTP/1.1
 1494 22.105933   192.168.1.9    216.58.200.206   HTTP      437 GET /generate_204 HTTP/1.1
 2701 29.289374   192.168.1.9    151.101.2.132    HTTP      436 GET /favicons/favicon.ico HTTP/1.1
 2706 29.377409   192.168.1.9    151.101.2.132    HTTP      442 GET /favicons/favicon-32x32.png HTTP/1.1
```

Figure 2.2: Wireshark Output (HTTP filter) and request details (`apache.org`)

c Total Download Time

In order to determine the total download time of the webpage (`http://apache.org`), I marked the first DNS query (for `apache.org`) as a reference in Wireshark (by clicking on the request and pressing `Ctrl + T`). After that, I noted down the timestamp of the last Web object (favicon) received from the server. This timestamp denotes the total download time by definition.

Total Download Time (Web page) `= 12.385804 seconds` ≈ `12.39 s`



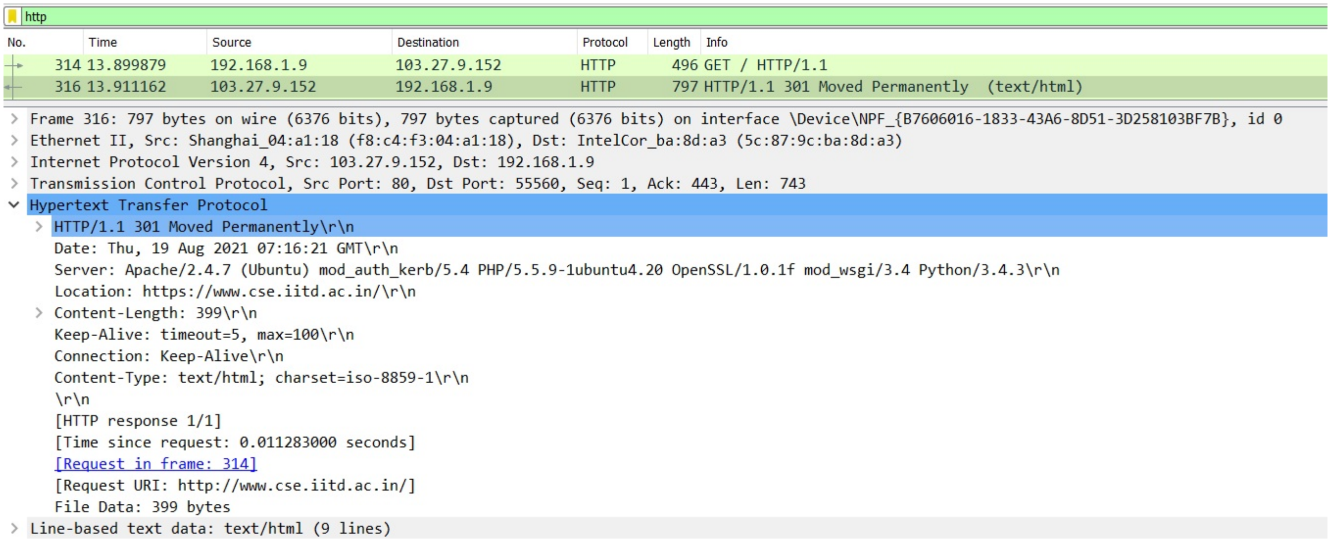
```
http.response and ip.addr==151.101.2.132
No.   Time        Source         Destination    Protocol  Length Info
  285 *REF*       192.168.1.9    192.168.1.1    DNS         70 Standard query 0xd4d1 A apache.org
  327 0.187552    151.101.2.132  192.168.1.9    HTTP       372 HTTP/1.1 200 OK  (text/html)
  387 0.375357    151.101.2.132  192.168.1.9    HTTP       160 HTTP/1.1 200 OK  (text/css)
  418 0.388756    151.101.2.132  192.168.1.9    HTTP      1339 HTTP/1.1 200 OK  (text/css)
  454 0.460447    151.101.2.132  192.168.1.9    HTTP      1075 HTTP/1.1 200 OK  (JPEG JFIF image)
  509 0.476659    151.101.2.132  192.168.1.9    HTTP       420 HTTP/1.1 200 OK  (application/javascript)
  514 0.477219    151.101.2.132  192.168.1.9    HTTP       473 HTTP/1.1 200 OK  (JPEG JFIF image)
  515 0.477219    151.101.2.132  192.168.1.9    HTTP       545 HTTP/1.1 200 OK  (JPEG JFIF image)
  520 0.478981    151.101.2.132  192.168.1.9    HTTP       283 HTTP/1.1 200 OK  (application/javascript)
  538 0.524398    151.101.2.132  192.168.1.9    HTTP      1060 HTTP/1.1 200 OK  (JPEG JFIF image)
  542 0.546249    151.101.2.132  192.168.1.9    HTTP       894 HTTP/1.1 200 OK  (application/javascript)
  594 0.582471    151.101.2.132  192.168.1.9    HTTP       602 HTTP/1.1 200 OK  (JPEG JFIF image)
  606 0.585787    151.101.2.132  192.168.1.9    HTTP      1044 HTTP/1.1 200 OK  (JPEG JFIF image)
  625 0.595386    151.101.2.132  192.168.1.9    HTTP      1177 HTTP/1.1 200 OK  (JPEG JFIF image)
  666 0.621374    151.101.2.132  192.168.1.9    HTTP      1429 HTTP/1.1 200 OK  (JPEG JFIF image)
  725 0.661273    151.101.2.132  192.168.1.9    HTTP      1051 HTTP/1.1 200 OK  (JPEG JFIF image)
  738 0.670739    151.101.2.132  192.168.1.9    HTTP       828 HTTP/1.1 200 OK  (JPEG JFIF image)
  842 0.713512    151.101.2.132  192.168.1.9    HTTP      1324 HTTP/1.1 200 OK  (PNG)
  854 0.719454    151.101.2.132  192.168.1.9    HTTP      5734 HTTP/1.1 200 OK  (PNG)
  877 0.755287    151.101.2.132  192.168.1.9    HTTP      1229 HTTP/1.1 200 OK  (PNG)
  897 0.764868    151.101.2.132  192.168.1.9    HTTP       515 HTTP/1.1 200 OK  (PNG)
  995 0.800948    151.101.2.132  192.168.1.9    HTTP       983 HTTP/1.1 200 OK  (PNG)
 1032 0.843328    151.101.2.132  192.168.1.9    HTTP      1175 HTTP/1.1 200 OK  (PNG)
 1203 4.055104    151.101.2.132  192.168.1.9    HTTP       100 HTTP/1.1 200 OK  (font/woff2)
 2704 12.299419   151.101.2.132  192.168.1.9    HTTP       789 HTTP/1.1 200 OK  (PNG)
 2709 12.385804   151.101.2.132  192.168.1.9    HTTP        85 HTTP/1.1 200 OK  (PNG)
```

Figure 2.3: Wireshark Output (HTTP filter) and Marked REF (DNS query) (`apache.org`)

d HTTP over TLS

d.1 Observations:

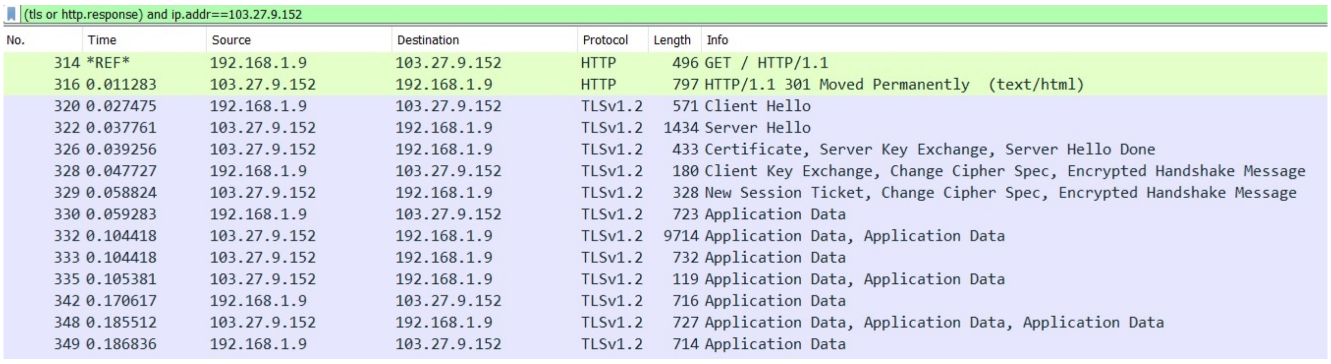
I applied HTTP filter and noted the packet statistics while visiting `http://www.cse.iitd.ac.in`. See screenshot below:



```
http
No.     Time        Source          Destination     Protocol  Length  Info
    314 13.899879   192.168.1.9     103.27.9.152    HTTP         496  GET / HTTP/1.1
    316 13.911162   103.27.9.152    192.168.1.9     HTTP         797  HTTP/1.1 301 Moved Permanently  (text/html)

> Frame 316: 797 bytes on wire (6376 bits), 797 bytes captured (6376 bits) on interface \Device\NPF_{B7606016-1833-43A6-8D51-3D258103BF7B}, id 0
> Ethernet II, Src: Shanghai_04:a1:18 (f8:c4:f3:04:a1:18), Dst: IntelCor_ba:8d:a3 (5c:87:9c:ba:8d:a3)
> Internet Protocol Version 4, Src: 103.27.9.152, Dst: 192.168.1.9
> Transmission Control Protocol, Src Port: 80, Dst Port: 55560, Seq: 1, Ack: 443, Len: 743
v Hypertext Transfer Protocol
  > HTTP/1.1 301 Moved Permanently\r\n
    Date: Thu, 19 Aug 2021 07:16:21 GMT\r\n
    Server: Apache/2.4.7 (Ubuntu) mod_auth_kerb/5.4 PHP/5.5.9-1ubuntu4.20 OpenSSL/1.0.1f mod_wsgi/3.4 Python/3.4.3\r\n
    Location: https://www.cse.iitd.ac.in/\r\n
  > Content-Length: 399\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=iso-8859-1\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.011283000 seconds]
    [Request in frame: 314]
    [Request URI: http://www.cse.iitd.ac.in/]
    File Data: 399 bytes
> Line-based text data: text/html (9 lines)
```

Figure 2.4: Wireshark Output (HTTP filter) and Server response details (`www.cse.iitd.ac.in`)

From server response (Code 301), it can be seen that the Web object that the browser was looking for was actually moved permanently to the address `"https://www.cse.iitd.ac.in/"` (can be verified from the *Location* header field). Hence, the browser then tries to retrieve the new URL. However, the new data transfer will take place over **Transport Layer Security (TLS)**, which is specified in `"https"` (HTTP over TLS). Hence, the data packets will not be visible to the naked HTTP filter in Wireshark, as they are now using extra security measures. Even though the packets consist of encrypted data and their contents cannot be determined by Wireshark, the existence of communication over TLS can be verified using TLS filter. See the screenshot below:



```
(tls or http.response) and ip.addr==103.27.9.152
No.     Time        Source          Destination     Protocol  Length  Info
    314 *REF*       192.168.1.9     103.27.9.152    HTTP         496  GET / HTTP/1.1
    316 0.011283    103.27.9.152    192.168.1.9     HTTP         797  HTTP/1.1 301 Moved Permanently  (text/html)
    320 0.027475    192.168.1.9     103.27.9.152    TLSv1.2      571  Client Hello
    322 0.037761    103.27.9.152    192.168.1.9     TLSv1.2     1434  Server Hello
    326 0.039256    103.27.9.152    192.168.1.9     TLSv1.2      433  Certificate, Server Key Exchange, Server Hello Done
    328 0.047727    192.168.1.9     103.27.9.152    TLSv1.2      180  Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
    329 0.058824    103.27.9.152    192.168.1.9     TLSv1.2      328  New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
    330 0.059283    192.168.1.9     103.27.9.152    TLSv1.2      723  Application Data
    332 0.104418    103.27.9.152    192.168.1.9     TLSv1.2     9714  Application Data, Application Data
    333 0.104418    103.27.9.152    192.168.1.9     TLSv1.2      732  Application Data
    335 0.105381    103.27.9.152    192.168.1.9     TLSv1.2      119  Application Data, Application Data
    342 0.170617    192.168.1.9     103.27.9.152    TLSv1.2      716  Application Data
    348 0.185512    103.27.9.152    192.168.1.9     TLSv1.2      727  Application Data, Application Data, Application Data
    349 0.186836    192.168.1.9     103.27.9.152    TLSv1.2      714  Application Data
```

Figure 2.5: Wireshark Output (HTTP + TLS filter) (`www.cse.iitd.ac.in`)

d.2 Remarks:

In the first case (visiting `apache.org`), the entire communication between the browser and the server was taking place using HTTP, with TCP as the transport layer protocol. Hence, the packets (and their contents) were easily visible on Wireshark under HTTP filter, as the data was unencrypted. However, in the second case (visiting `www.cse.iitd.ac.in`), the majority of communication between the browser and the server was taking place using HTTP over TLS-enhanced TCP. This made the connection more secure as data was being encrypted before being sent over using TCP. Hence, the packets were not visible under the naked HTTP filter. To view the packets, I had to apply TLS filter as well. Nevertheless, since the data was encrypted, contents of the packet were hidden even with TLS filter ON. Hence, HTTP packet sniffing in the first case was easier than in the second case.

Chapter 3

Implementation of Traceroute using Ping (`C++|Python`)

a Implementation Note

I have written **two** programs: **Traceroute Implementation** (`C++`) and **Plotting Script** (`Python`). By default, *traceroute* process creates a sub-process in which the python script is executed and the graph is plotted, but this can be avoided by passing in `-ng` flag to the command line arguments. Refer to `README.md` for more details regarding compilation and usage of `tracert.cpp` file.

a.1 `C++`:

I have used Windows Socket API (`<winsock2.h>`) for the socket interface and ICMP API (`<icmpapi.h>`) for sending/receiving ICMP packets to implement custom `ping` command. For making DNS queries, I used Winsock TCP/IP header (`<ws2tcpip.h>`). I also used IP helper API (`<iphlpapi.h>`) for other IP-related structures that are used in API methods (like structure for IP Address etc.).

a.2 `Python`:

I used `pandas` library and `matplotlib` library for CSV file reading and graph plotting respectively.

b Program Execution

I executed my program to trace the route to `www.iitd.ac.in` and `www.google.com`. Default parameters were used:

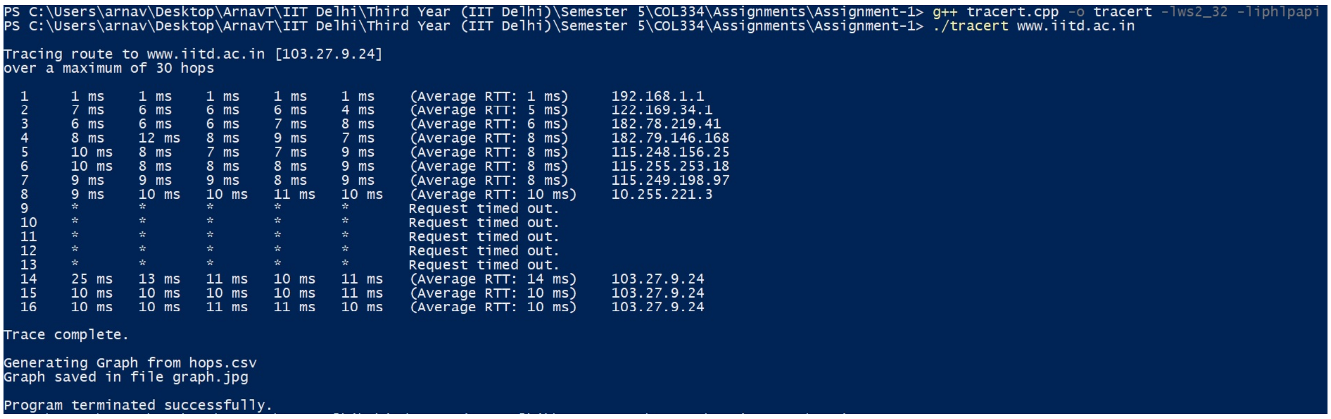
- **Maximum number of hops:** 30
- **Number of packets sent:** 5
- **Waiting time:** 4000 ms
- **Packet size:** 32 bytes

Out of these four parameters, the first **three** parameters can be customized and changed by the user by using `-h`, `-n` and `-w` flags respectively in the command line arguments. Refer to `README.md` for more details regarding compilation and usage of `tracert.cpp`.

c Output

In this section, I have attached the output screenshots and graphs for two possible usage of `tracert.exe` program:

c.1 `./tracert.exe www.iitd.ac.in`



```
PS C:\Users\arnav\Desktop\ArnavT\IIT Delhi\Third Year (IIT Delhi)\Semester 5\COL334\Assignments\Assignment-1> g++ tracert.cpp -o tracert -lws2_32 -liphlpapi
PS C:\Users\arnav\Desktop\ArnavT\IIT Delhi\Third Year (IIT Delhi)\Semester 5\COL334\Assignments\Assignment-1> ./tracert www.iitd.ac.in

Tracing route to www.iitd.ac.in [103.27.9.24]
over a maximum of 30 hops

  1     1 ms    1 ms    1 ms    1 ms    1 ms    (Average RTT: 1 ms)     192.168.1.1
  2     7 ms    6 ms    6 ms    6 ms    4 ms    (Average RTT: 5 ms)     122.169.34.1
  3     6 ms    6 ms    6 ms    7 ms    8 ms    (Average RTT: 6 ms)     182.78.219.41
  4     8 ms    12 ms   8 ms    9 ms    7 ms    (Average RTT: 8 ms)     182.79.146.168
  5     10 ms   8 ms    7 ms    7 ms    9 ms    (Average RTT: 8 ms)     115.248.156.25
  6     10 ms   8 ms    8 ms    8 ms    9 ms    (Average RTT: 8 ms)     115.255.253.18
  7     9 ms    9 ms    9 ms    8 ms    9 ms    (Average RTT: 8 ms)     115.249.198.97
  8     9 ms    10 ms   10 ms   11 ms   10 ms   (Average RTT: 10 ms)    10.255.221.3
  9     *       *       *       *       *       Request timed out.
  10    *       *       *       *       *       Request timed out.
  11    *       *       *       *       *       Request timed out.
  12    *       *       *       *       *       Request timed out.
  13    *       *       *       *       *       Request timed out.
  14    25 ms   13 ms   11 ms   10 ms   11 ms   (Average RTT: 14 ms)    103.27.9.24
  15    10 ms   10 ms   10 ms   10 ms   11 ms   (Average RTT: 10 ms)    103.27.9.24
  16    10 ms   10 ms   11 ms   11 ms   10 ms   (Average RTT: 10 ms)    103.27.9.24

Trace complete.

Generating Graph from hops.csv
Graph saved in file graph.jpg

Program terminated successfully.
```

Figure 3.1: Console Output

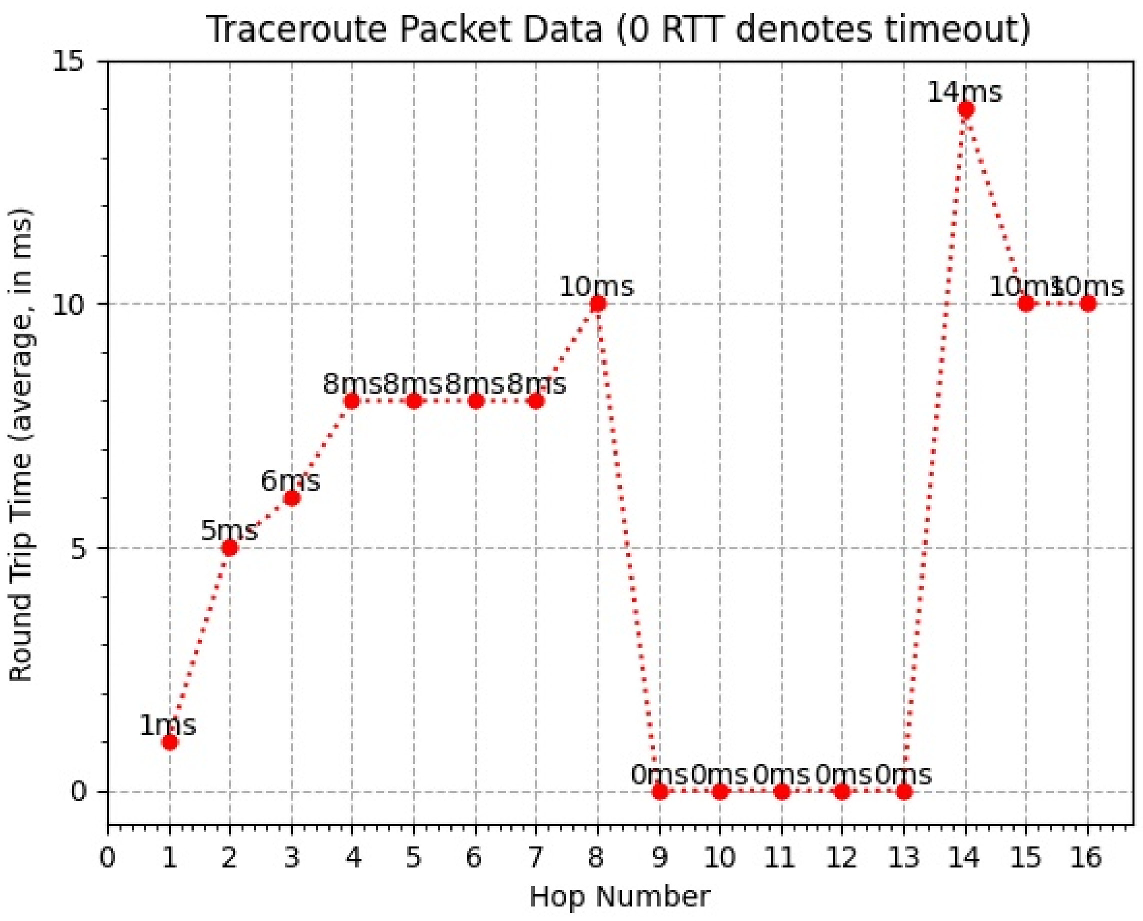
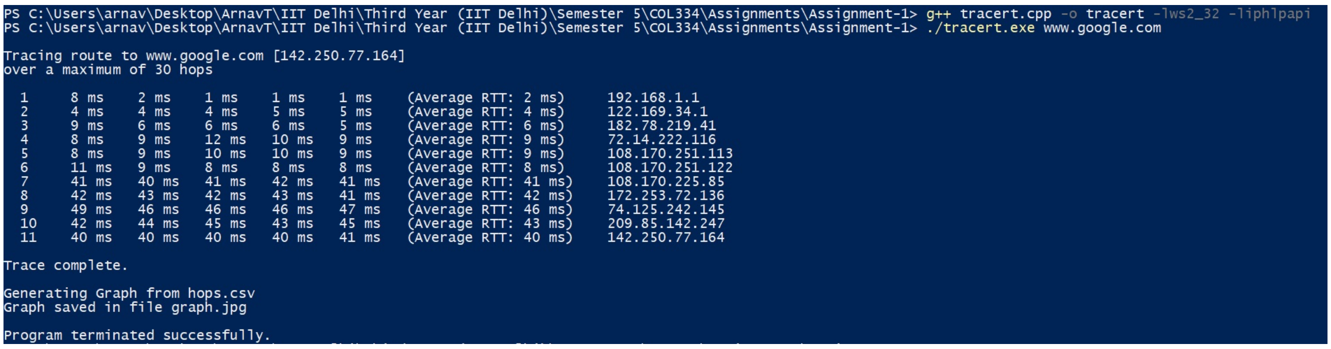


Figure 3.2: Average RTT (in ms) vs Hop Number

c.2 ./tracert.exe www.google.com



```
PS C:\Users\arnav\Desktop\ArnavT\IIT Delhi\Third Year (IIT Delhi)\Semester 5\COL334\Assignments\Assignment-1> g++ tracert.cpp -o tracert -lws2_32 -liphlpapi
PS C:\Users\arnav\Desktop\ArnavT\IIT Delhi\Third Year (IIT Delhi)\Semester 5\COL334\Assignments\Assignment-1> ./tracert.exe www.google.com

Tracing route to www.google.com [142.250.77.164]
over a maximum of 30 hops

  1     8 ms    2 ms    1 ms    1 ms    1 ms    (Average RTT: 2 ms)     192.168.1.1
  2     4 ms    4 ms    4 ms    5 ms    5 ms    (Average RTT: 4 ms)     122.169.34.1
  3     9 ms    6 ms    6 ms    6 ms    5 ms    (Average RTT: 6 ms)     182.78.219.41
  4     8 ms    9 ms    12 ms   10 ms   9 ms    (Average RTT: 9 ms)     72.14.222.116
  5     8 ms    9 ms    10 ms   10 ms   9 ms    (Average RTT: 9 ms)     108.170.251.113
  6     11 ms   9 ms    8 ms    8 ms    8 ms    (Average RTT: 8 ms)     108.170.251.122
  7     41 ms   40 ms   41 ms   42 ms   41 ms   (Average RTT: 41 ms)    108.170.225.85
  8     42 ms   43 ms   42 ms   43 ms   41 ms   (Average RTT: 42 ms)    172.253.72.136
  9     49 ms   46 ms   46 ms   46 ms   47 ms   (Average RTT: 46 ms)    74.125.242.145
  10    42 ms   44 ms   45 ms   43 ms   45 ms   (Average RTT: 43 ms)    209.85.142.247
  11    40 ms   40 ms   40 ms   40 ms   41 ms   (Average RTT: 40 ms)    142.250.77.164

Trace complete.

Generating Graph from hops.csv
Graph saved in file graph.jpg

Program terminated successfully.
```

Figure 3.3: Console Output

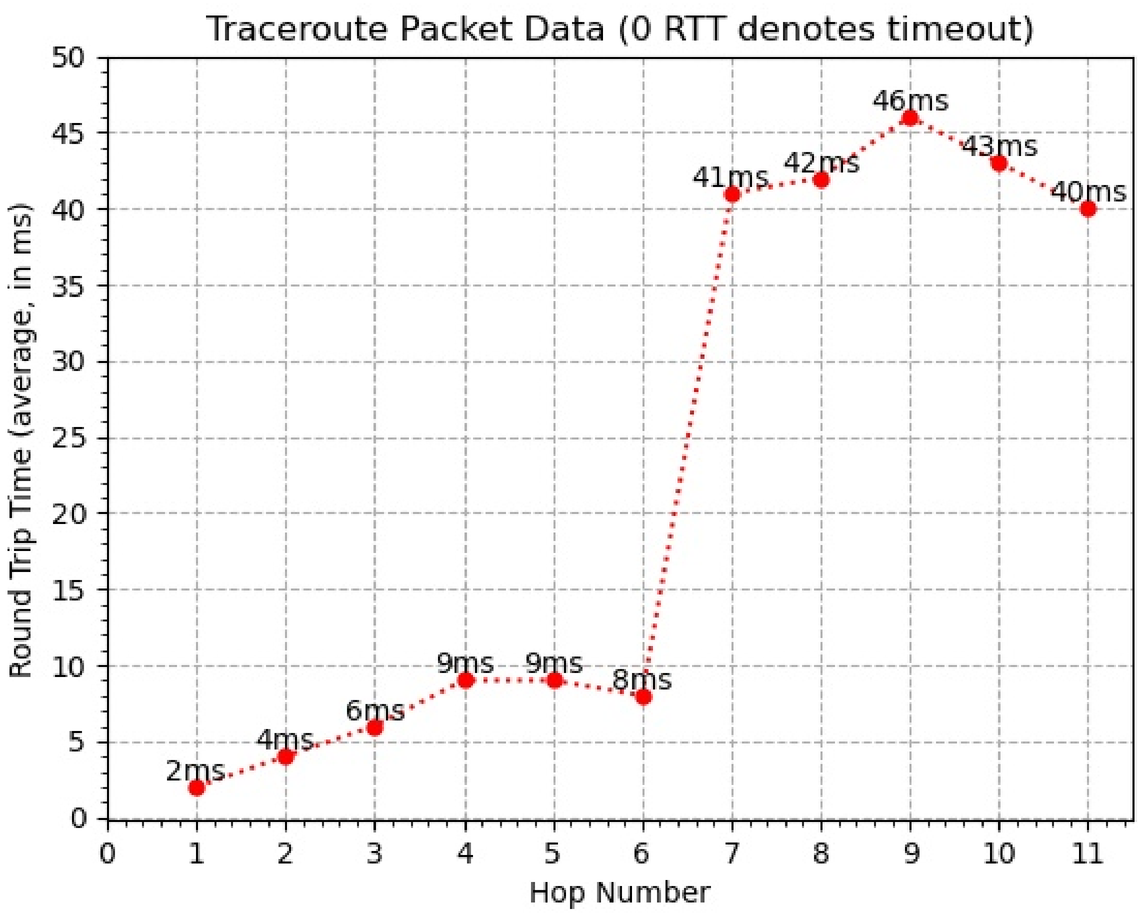


Figure 3.4: Average RTT (in ms) vs Hop Number