Project 1 - HTTP, DNS, and TCP

Due on 03-3-2024 Midnight

Submit your answers in a PDF file

Name: Sai Kiran Belana

NetID: skb23007

Course: 5299

Suggested Resources:

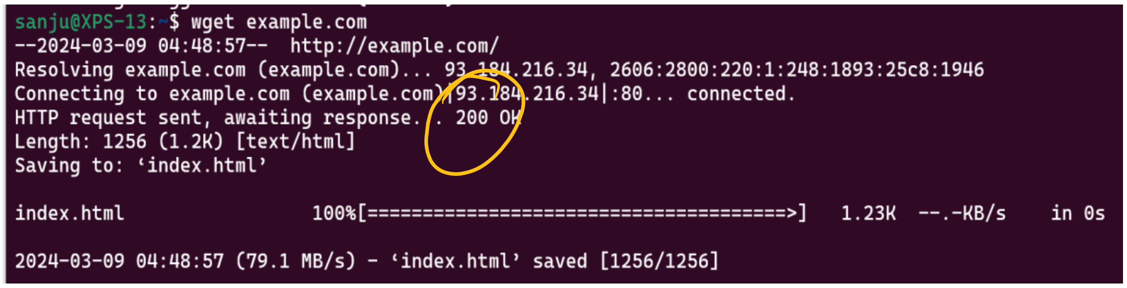
https://www.ietf.org/rfc/rfc2616.txt
https://www.ietf.org/rfc/rfc1035.txt
https://linux.die.net/man/
http://www.tcpipguide.com/free/

HTTP Questions

1. [7 pts] Choose 5 HTTP status codes and describe each one.
   Answer: **200 – success code**
   If you make a request to a web page and the server handles the request and sends a successful response, then it is a 200 Status code.
   301 – Moved Permanently
   It indicates that the requested page has been permanently moved to a new URL provided by the location header in response. Clients are expected to update their links to the new URL.
   404 – Page not found.
   It means the requested resource is not found. The page doesn't exists, returned by server.
   500 -- Internal Server Error
   An error message indicating that the server encountered an unexpected condition that prevented it from fulfilling the request.
   503 – Service Unavailable
   It indicates that the server is not ready to handle the request, usually because it is overloaded or down for maintenance.

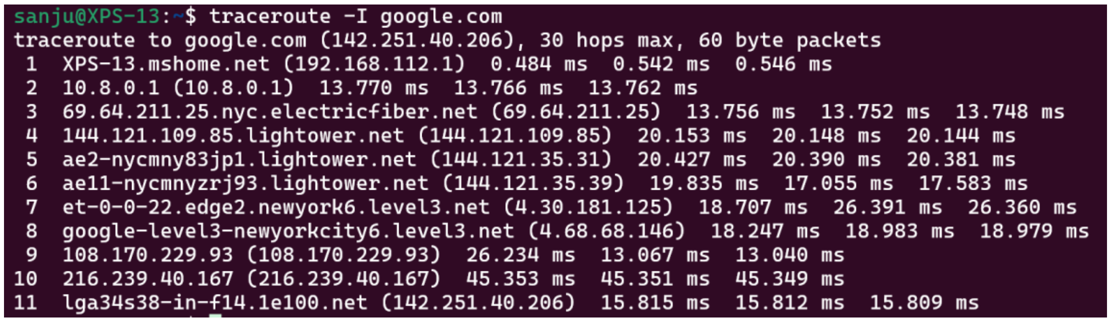
2. [7 pts] List the 8 HTTP 1.1 methods and explain what they do.
   Answer: ***wget*** and ***traceroute*** are two commonly known command line tools for testing and debugging. Answer the following questions by using your Mininet VM's terminal or the Unix timeshare.

3. [7 pts] Use *wget* on *example.com* to view the last modified date of the webpage. What was the HTTP return status given and what command was used to do this? (The command should not download the file! Hint: Look into the wget man page.)
   Answer: We receive a response with status code of 200 when we use `wget`.



```
sanju@XPS-13:~$ wget example.com
--2024-03-09 04:48:57--  http://example.com/
Resolving example.com (example.com)... 93.184.216.34, 2606:2800:220:1:248:1893:25c8:1946
Connecting to example.com (example.com)|93.184.216.34|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1256 (1.2K) [text/html]
Saving to: ‘index.html’

index.html                100%[====================================>]   1.23K  --.-KB/s    in 0s

2024-03-09 04:48:57 (79.1 MB/s) - ‘index.html’ saved [1256/1256]
```

4. [7 pts] Run traceroute -I google.com, show your screenshot and illustrate the meaning of this command.
   Answer:



```
sanju@XPS-13:~$ traceroute -I google.com
traceroute to google.com (142.251.40.206), 30 hops max, 60 byte packets
 1  XPS-13.mshome.net (192.168.112.1)  0.484 ms  0.542 ms  0.546 ms
 2  10.8.0.1 (10.8.0.1)  13.770 ms  13.766 ms  13.762 ms
 3  69.64.211.25.nyc.electricfiber.net (69.64.211.25)  13.756 ms  13.752 ms  13.748 ms
 4  144.121.109.85.lightower.net (144.121.109.85)  20.153 ms  20.148 ms  20.144 ms
 5  ae2-nycmny83jp1.lightower.net (144.121.35.31)  20.427 ms  20.390 ms  20.381 ms
 6  ae11-nycmnyzrj93.lightower.net (144.121.35.39)  19.835 ms  17.055 ms  17.583 ms
 7  et-0-0-22.edge2.newyork6.level3.net (4.30.181.125)  18.707 ms  26.391 ms  26.360 ms
 8  google-level3-newyorkcity6.level3.net (4.68.68.146)  18.247 ms  18.983 ms  18.979 ms
 9  108.170.229.93 (108.170.229.93)  26.234 ms  13.067 ms  13.040 ms
10  216.239.40.167 (216.239.40.167)  45.353 ms  45.351 ms  45.349 ms
11  lga34s38-in-f14.1e100.net (142.251.40.206)  15.815 ms  15.812 ms  15.809 ms
```

   Traceroute is used for tracing the URL's end to end connection. We can see the intermediate servers for finding the final web server which sends the response for google.com. This command displays the no. of hops for finding the end server(google.com). For all the intermediatory servers/systems, we can see the IP address, time taken to process the request to the next server and so on.

DNS Questions

5. [7 pts] In your own words describe what a DNS resource record (RR) is. Now using the command line tool *nslookup* find the *MX* resource record of *uconn.edu*. What does this resource record mean?
   Answer: A DNS Resource Record is an entry in a DNS database – a sort of directory for the internet – that specifies information about a domain name. Each record contains details about the domain, and different types of records for different purposes.
   DNS Resource Record Format:

(name, value, type, ttl)

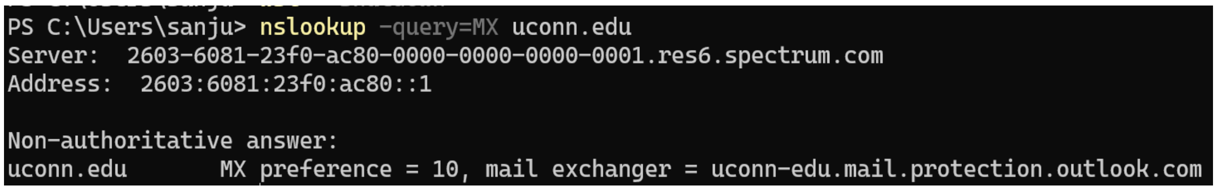
name is hostname, **value** is the IP address, **type** is the record type such as mail server and **ttl** is time to live.

Some Examples for DNS records:

- type=A: maps a domain name to its corresponding Ipv4 address.
- type=CNAME: Points a domain name to another domain name (like aliasing)
- type=MX: Directs email to mail servers for the domain.
- type=NS: Specifies the servers that are authoritative for a domain zone.

The command used for checking the MX record used for uconn.edu is.

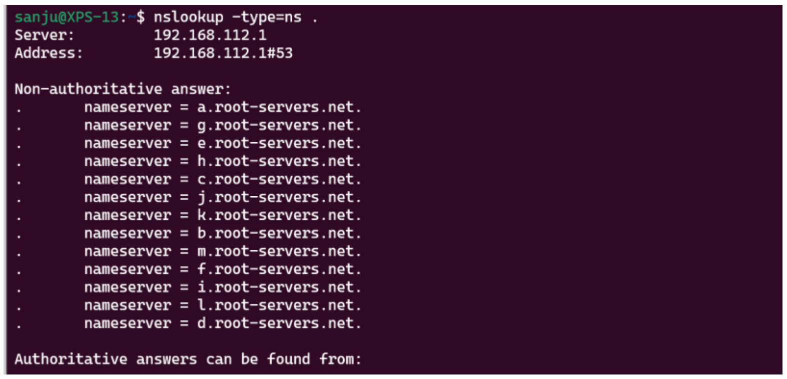
```
nslookup -query=MX uconn.edu
```



```
PS C:\Users\sanju> nslookup -query=MX uconn.edu
Server:  2603-6081-23f0-ac80-0000-0000-0000-0001.res6.spectrum.com
Address:  2603:6081:23f0:ac80::1

Non-authoritative answer:
uconn.edu       MX preference = 10, mail exchanger = uconn-edu.mail.protection.outlook.com
```

We can observe that the Server is the DNS server the query has sent to, which is likely the internet Service provider (Spectrum).
Address: This is the IPV6 address of the DNS server that responded to the query.
No-authoritative answer:
This indicates that the answer was provided by a server that is not the authoritative source for the DNS zone of `uconn.edu`. It might be from a cached source rather than the domain's primary DNS Server.
MX =preference = 10: This is the priority score for the mail server
Mail exchanger= uconn-edu.mail.protection.outlook.com: Specifies the actual mail server that handles email for **uconn.edu**. The domain listed here is part of Microsoft's email protection service, indicating that `uconn.edu` uses Microsoft's email services like Office 365 for handling its email traffic.

6. [7 pts] What does the command *nslookup -type=ns .* do*?* Explain its output. (Note: the . is part of the command!)
Answer:



```
sanju@XPS-13:~$ nslookup -type=ns .
Server:         192.168.112.1
Address:        192.168.112.1#53

Non-authoritative answer:
.       nameserver = a.root-servers.net.
.       nameserver = g.root-servers.net.
.       nameserver = e.root-servers.net.
.       nameserver = h.root-servers.net.
.       nameserver = c.root-servers.net.
.       nameserver = j.root-servers.net.
.       nameserver = k.root-servers.net.
.       nameserver = b.root-servers.net.
.       nameserver = m.root-servers.net.
.       nameserver = f.root-servers.net.
.       nameserver = i.root-servers.net.
.       nameserver = l.root-servers.net.
.       nameserver = d.root-servers.net.

Authoritative answers can be found from:
```

The *nslookup* command is querying the current namespaces that are known for our local system. We can see that it printed some `Non-authoriative answer`. These are the root nameservers which are used for querying the domains when we hit any URL in the browsers.

TCP Questions

7. [10 pts] How can multiple application services running on a single machine with a single IP address be uniquely identified?
   Answer: A single IP address in a machine is enough for running multiple applications since we use **PORT #** to identify each application process uniquely.
   - Each application service is identified by a unique port number.
   - This enables a machine to communicate b/w multiple services/processes.

8. [9 pts] What is the purpose of the window mechanism in TCP?
   Answer: The window mechanism in TCP serves two main purposes:
   Flow Control and Congestion Control.
   Flow Control:
   It prevents the sender from overwhelming the receiver's buffer by limiting the amount of unacked data that can be in transit.
   Congestion Control:
   It helps to avoid overwhelming the network itself by adjust the rate if data flow based on current state of the network, attempting to prevent package loss due to congestion. This is achieved through various phases like slow start, congestion avoidance, fast retransmit and fast recovery.

9. [9 pts] What is an MTU? What happens when a packet is larger than the MTU?
   Answer: MTU is the Maximum Transmission Unit, which is the largest packet size or frame that can be sent in a packet/frame. MTU is measured in bytes and varies by the physical medium and network layer.

When a packet is larger than the MTU of the network over which it needs to be sent, it must be divided into smaller fragments and this process is known as fragmentation.
NOTE: Fragmentation can occur at the sender side or at an intermediate router.

Fragmented packets are reassembled back into their original size at the destination. But fragmentation can lead to inefficiency and increased latency. So, it is best to discover and use the small MTY along the path to avoid it.

HTTP, DNS, and TCP:

Suggested Resources:

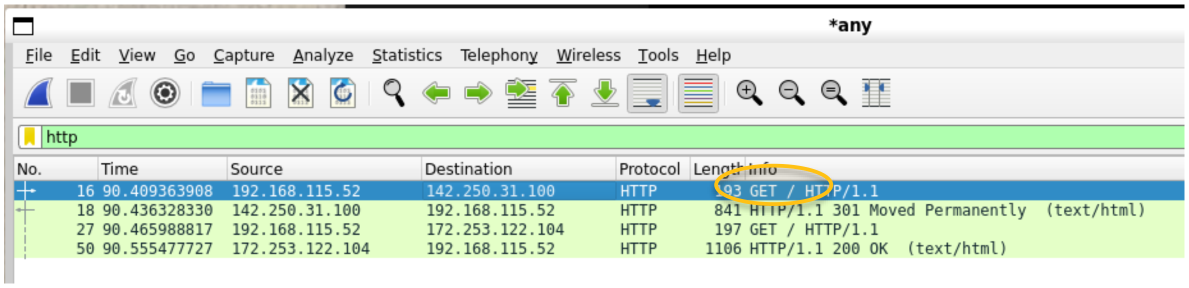
http://packetbomb.com/understanding-the-tcptrace-time-sequence-graph-in-wireshark/
https://wiki.linuxfoundation.org/networking/netem

Part 1: HTTP

In this section, we will observe how the HTTP protocol operates. We will do this by using the Mininet VM. Begin by opening Wireshark and listening on the 'any' interface.

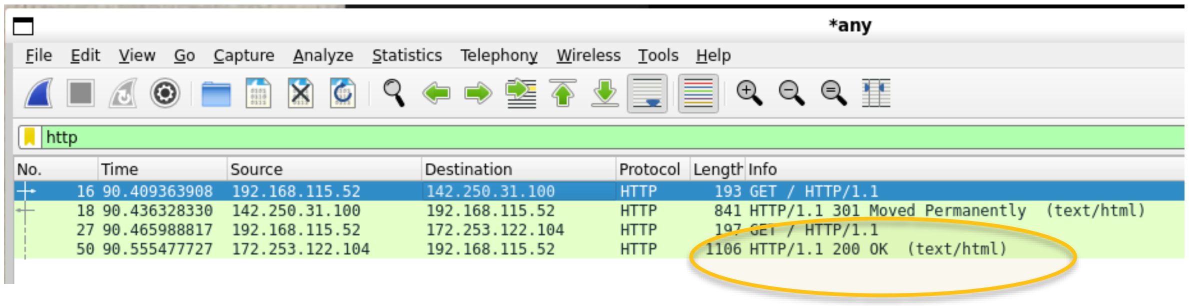
Open Chromium and navigate to http://httpbin.org.

1. [10 pts] Find the HTTP packet that corresponds to the initial request that your computer made. Take a screenshot of this packet. What HTTP method did your computer use to make this request?
   Answer: My computer is using HTTP protocol of version 1.1. The info describes that I'm using a ***GET*** method for requesting the base HTML page at destination (142.250.31.100).


*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.   Time          Source           Destination      Protocol Length Info
   16 90.409363908  192.168.115.52   142.250.31.100   HTTP      193 GET / HTTP/1.1
   18 90.436328330  142.250.31.100   192.168.115.52   HTTP      841 HTTP/1.1 301 Moved Permanently  (text/html)
   27 90.465988817  192.168.115.52   172.253.122.104  HTTP      197 GET / HTTP/1.1
   50 90.555477727  172.253.122.104  192.168.115.52   HTTP     1106 HTTP/1.1 200 OK  (text/html)
```

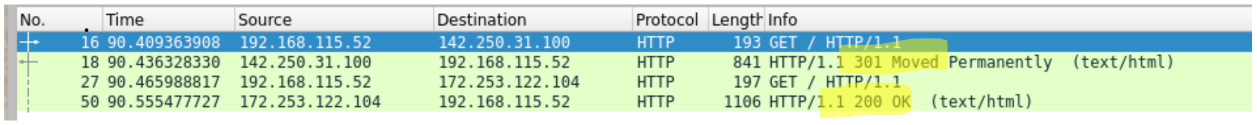
2. [10 pts] Find the HTTP packet that corresponds to the initial response the server made to your request. Take a screenshot of this packet. What HTTP status code did the server return? What is the content type of the response the server is sending back?
   Answer: The Server has sent a response code of **200 OK** which means it is a success for the previous query received from my computer. The content sent by the server to the response is an HTML file.


*any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
http
No.   Time          Source           Destination      Protocol Length Info
   16 90.409363908  192.168.115.52   142.250.31.100   HTTP      193 GET / HTTP/1.1
   18 90.436328330  142.250.31.100   192.168.115.52   HTTP      841 HTTP/1.1 301 Moved Permanently  (text/html)
   27 90.465988817  192.168.115.52   172.253.122.104  HTTP      197 GET / HTTP/1.1
   50 90.555477727  172.253.122.104  192.168.115.52   HTTP     1106 HTTP/1.1 200 OK  (text/html)
```

Using Chromium and navigate to http://uconn.edu.

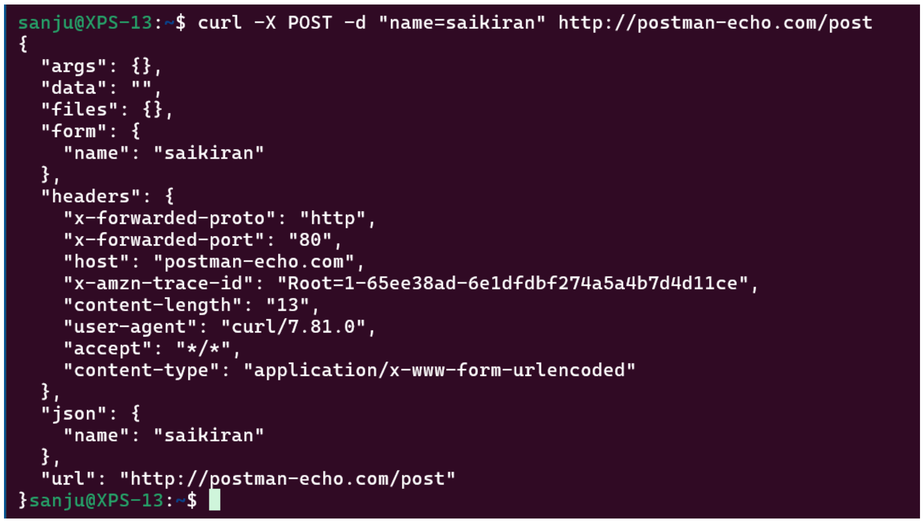
3. [10 pts] Find the HTTP packets that correspond to the initial request and response that your computer made. Take a screenshot of these packets. What's different? Explain.
   Answer:
   - The initial *request* made on my computer (192.168.115.52) when I do a google.com search is a GET method which is requesting a base HTML page.
   - I have received a *response* of 301 from 142.259.21.100 where the site does not exist and has been redirected to a different server.
   - So, my next *request* is a GET method again which has request for the base html to 172.253.122.104.
   - Finally, I got a response of 200 status code from 172.253.122.104 with the html page.



```
No.   Time          Source           Destination      Protocol Length Info
   16 90.409363908  192.168.115.52   142.250.31.100   HTTP      193 GET / HTTP/1.1
   18 90.436328330  142.250.31.100   192.168.115.52   HTTP      841 HTTP/1.1 301 Moved Permanently  (text/html)
   27 90.465988817  192.168.115.52   172.253.122.104  HTTP      197 GET / HTTP/1.1
   50 90.555477727  172.253.122.104  192.168.115.52   HTTP     1106 HTTP/1.1 200 OK  (text/html)
```

Using Chromium (or any other Linux utility you are comfortable with), find a way to make a HTTP packet with a method other than GET.

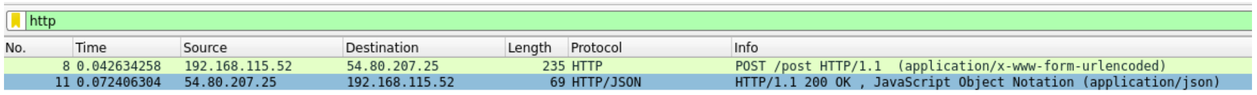
4. [10 pts] Take a screenshot of your packet and explain what you did to create it.
   Answer: I am using ***curl,*** a command line utility for Linux for querying the URL. It supports all the HTTP methods such as GET, POST, PUT, DELETE, etc.



```
sanju@XPS-13:~$ curl -X POST -d "name=saikiran" http://postman-echo.com/post
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "name": "saikiran"
  },
  "headers": {
    "x-forwarded-proto": "http",
    "x-forwarded-port": "80",
    "host": "postman-echo.com",
    "x-amzn-trace-id": "Root=1-65ee38ad-6e1dfdbf274a5a4b7d4d11ce",
    "content-length": "13",
    "user-agent": "curl/7.81.0",
    "accept": "*/*",
    "content-type": "application/x-www-form-urlencoded"
  },
  "json": {
    "name": "saikiran"
  },
  "url": "http://postman-echo.com/post"
}sanju@XPS-13:~$ 
```

In the screenshot, I'm using the POST method for submitting my name to postman-echo.com. By running the command, I get an output as the form I mentioned with a JSON object of my details (name=saikiran). It includes the headers, Json, and URL used.

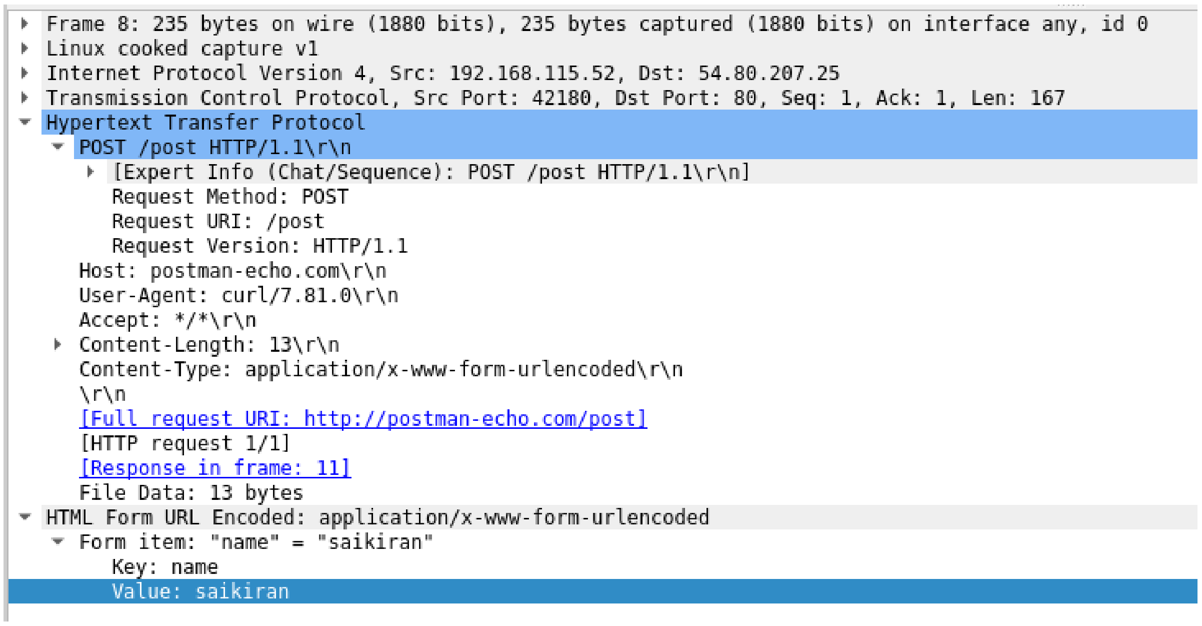
Wireshark:



```
http
No.   Time          Source           Destination      Length Protocol   Info
    8 0.042634258   192.168.115.52   54.80.207.25        235 HTTP       POST /post HTTP/1.1  (application/x-www-form-urlencoded)
   11 0.072406304   54.80.207.25     192.168.115.52       69 HTTP/JSON  HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
```

We can see the Protocol used it HTTP and the info says we used POST method for /post route.

Detailed information of the packet is here.



```
▸ Frame 8: 235 bytes on wire (1880 bits), 235 bytes captured (1880 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 192.168.115.52, Dst: 54.80.207.25
▸ Transmission Control Protocol, Src Port: 42180, Dst Port: 80, Seq: 1, Ack: 1, Len: 167
▾ Hypertext Transfer Protocol
  ▾ POST /post HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): POST /post HTTP/1.1\r\n]
      Request Method: POST
      Request URI: /post
      Request Version: HTTP/1.1
    Host: postman-echo.com\r\n
    User-Agent: curl/7.81.0\r\n
    Accept: */*\r\n
  ▸ Content-Length: 13\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    \r\n
    [Full request URI: http://postman-echo.com/post]
    [HTTP request 1/1]
    [Response in frame: 11]
    File Data: 13 bytes
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▾ Form item: "name" = "saikiran"
      Key: name
      Value: saikiran
```

The form has key of name and value of saikiran.

Part 2: DNS

In this section, we will observe how the DNS protocol operates. We will do this by using the Mininet VM. Begin by opening Wireshark and listening on the 'any' interface.

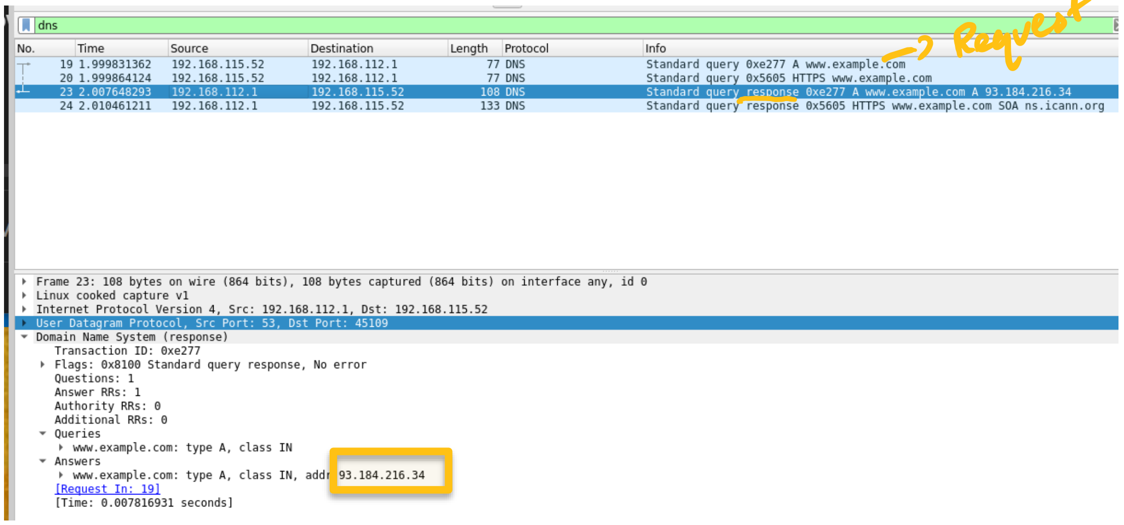
Open Chromium and navigate to www.example.com.

5. [10 pts] Were any steps taken by your computer before the web page was loaded? If so, using your captured packets in Wireshark, find the packets that allowed your computer to successfully load http://www.example.com. Take a screenshot of these packets and explain why you think these are the correct packets. What's the IP address of www.example.com?

   Answer: While navigating to www.example.com, we can find the packets for loading the URL.

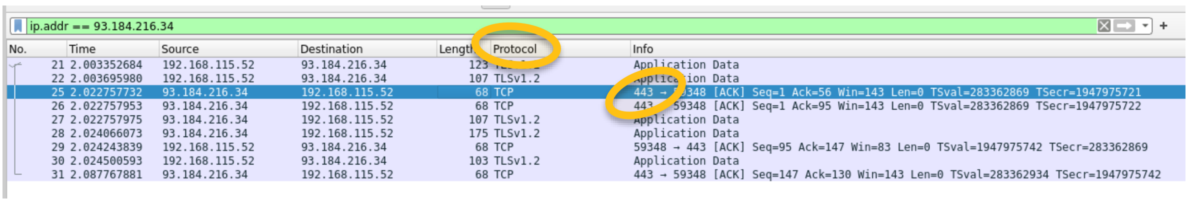
   Using Chromium & Wireshark:

   - Firstly, we can use the filter `**dns**` for querying the URL which returns the IP address(93.184.216.34) for the URL.



```
dns
No.  Time          Source           Destination      Length Protocol  Info
  19 1.999831362   192.168.115.52   192.168.112.1       77 DNS        Standard query 0xe277 A www.example.com
  20 1.999864124   192.168.115.52   192.168.112.1       77 DNS        Standard query 0x5605 HTTPS www.example.com
  23 2.007648293   192.168.112.1    192.168.115.52     108 DNS        Standard query response 0xe277 A www.example.com A 93.184.216.34
  24 2.010461211   192.168.112.1    192.168.115.52     133 DNS        Standard query response 0x5605 HTTPS www.example.com SOA ns.icann.org

▸ Frame 23: 108 bytes on wire (864 bits), 108 bytes captured (864 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 192.168.112.1, Dst: 192.168.115.52
▸ User Datagram Protocol, Src Port: 53, Dst Port: 45109
▾ Domain Name System (response)
    Transaction ID: 0xe277
  ▸ Flags: 0x8100 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▸ www.example.com: type A, class IN
  ▾ Answers
    ▸ www.example.com: type A, class IN, addr 93.184.216.34
    [Request In: 19]
    [Time: 0.007816931 seconds]
```

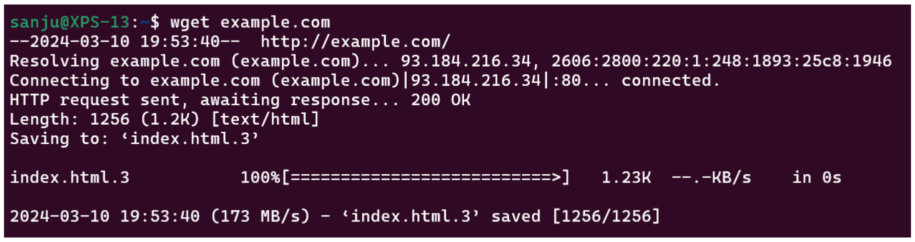
   - Once we have the IP address from DNS response, we can use Wireshark output to show only the traffic to and from this IP address using the filter `ip.addr == 93.184.216.34`.



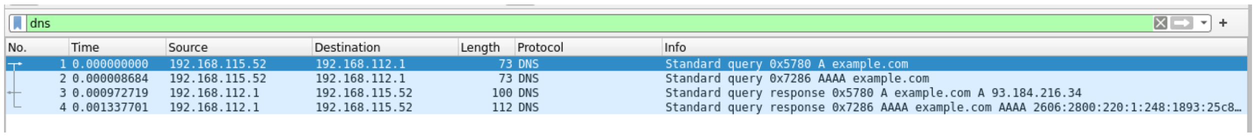
```
ip.addr == 93.184.216.34
No.  Time          Source           Destination      Length Protocol  Info
  21 2.003352684   192.168.115.52   93.184.216.34      123 TLSv1.2    Application Data
  22 2.003695980   192.168.115.52   93.184.216.34      107 TLSv1.2    Application Data
  25 2.022757732   93.184.216.34    192.168.115.52      68 TCP        443 → 59348 [ACK] Seq=1 Ack=56 Win=143 Len=0 TSval=283362869 TSecr=1947975721
  26 2.022757953   93.184.216.34    192.168.115.52      68 TCP        443 → 59348 [ACK] Seq=1 Ack=95 Win=143 Len=0 TSval=283362869 TSecr=1947975722
  27 2.022757975   93.184.216.34    192.168.115.52     107 TLSv1.2    Application Data
  28 2.024066073   93.184.216.34    192.168.115.52     175 TLSv1.2    Application Data
  29 2.024243839   192.168.115.52   93.184.216.34       68 TCP        59348 → 443 [ACK] Seq=95 Ack=147 Win=83 Len=0 TSval=1947975742 TSecr=283362869
  30 2.024500593   192.168.115.52   93.184.216.34      103 TLSv1.2    Application Data
  31 2.087767881   93.184.216.34    192.168.115.52      68 TCP        443 → 59348 [ACK] Seq=147 Ack=130 Win=143 Len=0 TSval=283362934 TSecr=1947975742
```

- We can observe that we use TCP protocol for the IP address for loading/querying data from this URL.
- To identify the packets related to HTTP, we can use filter as **http** or **tcp.port == 80** which shows the HTTP traffic for the URL.
- We can also notice that we are using HTTPS (port: 443) instead of HTTP(port 80) we can query with **tcp.port==443**
- Since we are using a TCP response, we can also observe the handshake happening b/w my system and the server before loading the response.

Using wget & Wireshark:

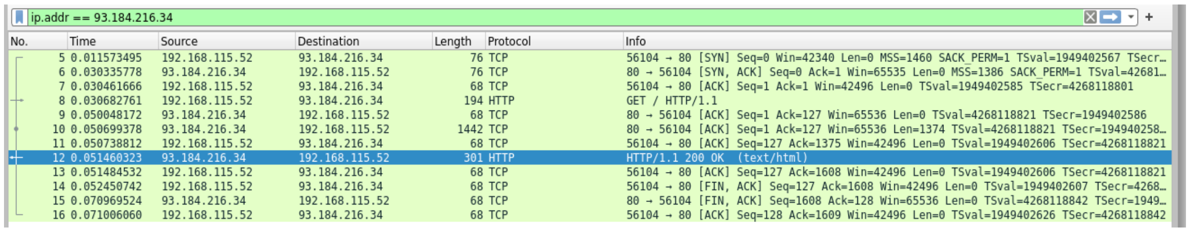


```
sanju@XPS-13:~$ wget example.com
--2024-03-10 19:53:40--  http://example.com/
Resolving example.com (example.com)... 93.184.216.34, 2606:2800:220:1:248:1893:25c8:1946
Connecting to example.com (example.com)|93.184.216.34|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1256 (1.2K) [text/html]
Saving to: ‘index.html.3’

index.html.3           100%[=========================>]   1.23K  --.-KB/s    in 0s

2024-03-10 19:53:40 (173 MB/s) - ‘index.html.3’ saved [1256/1256]
```



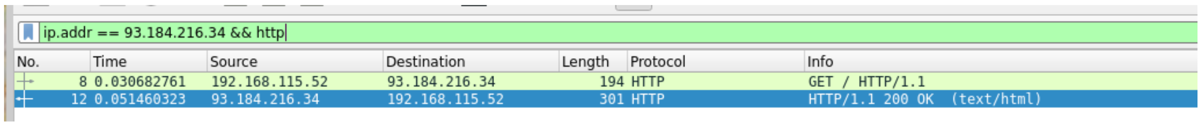
```
dns
No.   Time         Source          Destination     Length Protocol  Info
    1 0.000000000  192.168.115.52  192.168.112.1       73 DNS       Standard query 0x5780 A example.com
    2 0.000008684  192.168.115.52  192.168.112.1       73 DNS       Standard query 0x7286 AAAA example.com
    3 0.000972719  192.168.112.1   192.168.115.52     100 DNS       Standard query response 0x5780 A example.com A 93.184.216.34
    4 0.001337701  192.168.112.1   192.168.115.52     112 DNS       Standard query response 0x7286 AAAA example.com AAAA 2606:2800:220:1:248:1893:25c8…
```

- Type **dns** in the filter to see the DNS response and we get the IP address 93.184.216.34
- Next, we can use \`ip.addr==93.184.216.34\` in filter to see packets making request for example.com URL.



```
ip.addr == 93.184.216.34
No.   Time         Source          Destination     Length Protocol  Info
    5 0.011573495  192.168.115.52  93.184.216.34       76 TCP       56104 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=1949402567 TSecr…
    6 0.030335778  93.184.216.34   192.168.115.52      76 TCP       80 → 56104 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1386 SACK_PERM=1 TSval=42681…
    7 0.030461666  192.168.115.52  93.184.216.34       68 TCP       56104 → 80 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=1949402585 TSecr=4268118801
    8 0.030682761  192.168.115.52  93.184.216.34      194 HTTP      GET / HTTP/1.1
    9 0.050048172  93.184.216.34   192.168.115.52      68 TCP       80 → 56104 [ACK] Seq=1 Ack=127 Win=65536 Len=0 TSval=4268118821 TSecr=1949402586
   10 0.050699378  93.184.216.34   192.168.115.52    1442 TCP       80 → 56104 [ACK] Seq=1 Ack=127 Win=65536 Len=1374 TSval=4268118821 TSecr=194940258…
   11 0.050738812  192.168.115.52  93.184.216.34       68 TCP       56104 → 80 [ACK] Seq=127 Ack=1375 Win=42496 Len=0 TSval=1949402606 TSecr=4268118821
   12 0.051460323  93.184.216.34   192.168.115.52     301 HTTP      HTTP/1.1 200 OK  (text/html)
   13 0.051484532  192.168.115.52  93.184.216.34       68 TCP       56104 → 80 [ACK] Seq=127 Ack=1608 Win=42496 Len=0 TSval=1949402606 TSecr=4268118821
   14 0.052450742  192.168.115.52  93.184.216.34       68 TCP       56104 → 80 [FIN, ACK] Seq=127 Ack=1608 Win=42496 Len=0 TSval=1949402607 TSecr=4268…
   15 0.070969524  93.184.216.34   192.168.115.52      68 TCP       80 → 56104 [FIN, ACK] Seq=1608 Ack=128 Win=65536 Len=0 TSval=4268118842 TSecr=1949…
   16 0.071006060  192.168.115.52  93.184.216.34       68 TCP       56104 → 80 [ACK] Seq=128 Ack=1609 Win=42496 Len=0 TSval=1949402626 TSecr=4268118842
```

- Now we can see the GET method used for HTTP by querying with \`ip.addr == 93.184.216.34 && http\`

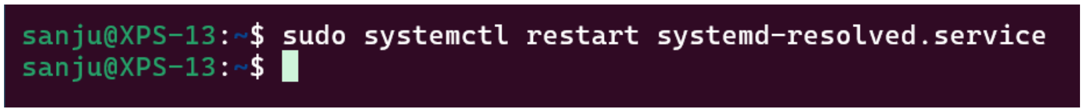


```
ip.addr == 93.184.216.34 && http
No.   Time         Source          Destination     Length Protocol  Info
    8 0.030682761  192.168.115.52  93.184.216.34      194 HTTP      GET / HTTP/1.1
   12 0.051460323  93.184.216.34   192.168.115.52     301 HTTP      HTTP/1.1 200 OK  (text/html)
```

- We can see we got a response of status code 200 which means it's a success of page load.

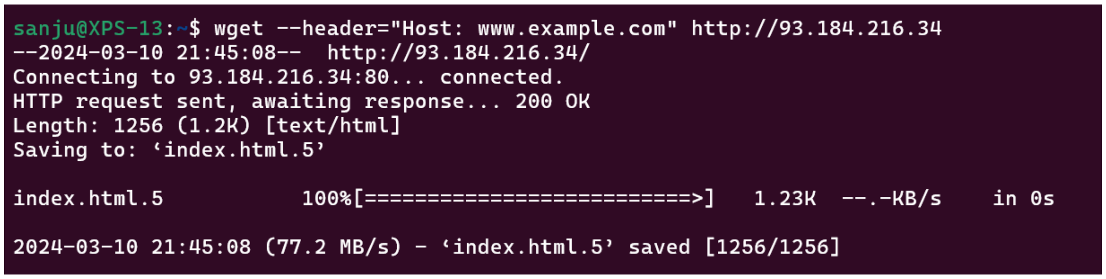
6. [10 pts] Open a terminal window. Execute the command to flush your DNS cache: sudo /etc/init.d/networking restart
   Using wget, download the same content of www.example.com with its IP address you discovered in question 5, without sending DNS requests.

What command did you use to accomplish that? Take a screenshot of related packets and explain why you think these are the correct packets.
Answer: Since, I couldn't find /etc/init.d/networking file, I ran the following command for flushing the DNS cache.



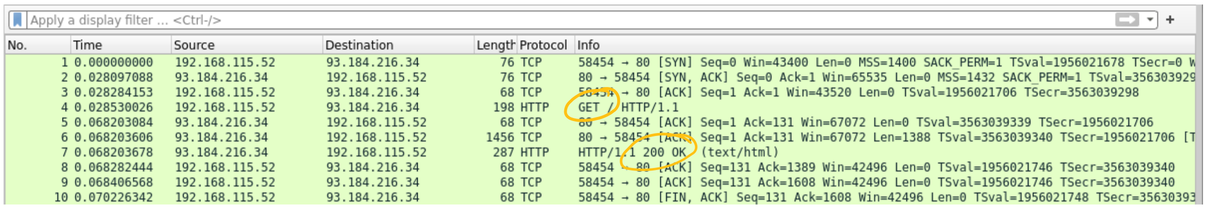
```
sanju@XPS-13:~$ sudo systemctl restart systemd-resolved.service
sanju@XPS-13:~$ 
```

For downloading the content from www.example.com with its IP address, I used the following command.
wget --header="Host: www.example.com" http://93.184.216.34



```
sanju@XPS-13:~$ wget --header="Host: www.example.com" http://93.184.216.34
--2024-03-10 21:45:08--  http://93.184.216.34/
Connecting to 93.184.216.34:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1256 (1.2K) [text/html]
Saving to: ‘index.html.5’

index.html.5           100%[=========================>]   1.23K  --.-KB/s    in 0s

2024-03-10 21:45:08 (77.2 MB/s) - ‘index.html.5’ saved [1256/1256]
```

In Wireshark, we can see the packets where some of them are using TCP and some are using HTTP protocols. Two HTTP Protocols are used, one for GET method and the second one is a response of status code 200.



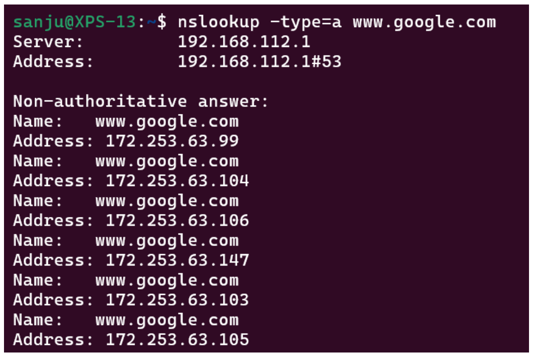
Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Length	Protocol	Info
1	0.000000000	192.168.115.52	93.184.216.34	76	TCP	58454 → 80 [SYN] Seq=0 Win=43400 Len=0 MSS=1400 SACK_PERM=1 TSval=1956021678 TSecr=0 W
2	0.028097088	93.184.216.34	192.168.115.52	76	TCP	80 → 58454 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1432 SACK_PERM=1 TSval=356303929
3	0.028284153	192.168.115.52	93.184.216.34	68	TCP	58454 → 80 [ACK] Seq=1 Ack=1 Win=43520 Len=0 TSval=1956021706 TSecr=3563039298
4	0.028530026	192.168.115.52	93.184.216.34	198	HTTP	GET / HTTP/1.1
5	0.068203084	93.184.216.34	192.168.115.52	68	TCP	80 → 58454 [ACK] Seq=1 Ack=131 Win=67072 Len=0 TSval=3563039339 TSecr=1956021706
6	0.068203606	93.184.216.34	192.168.115.52	1456	TCP	80 → 58454 [ACK] Seq=1 Ack=131 Win=67072 Len=1388 TSval=3563039340 TSecr=1956021706 [T
7	0.068203678	93.184.216.34	192.168.115.52	287	HTTP	HTTP/1.1 200 OK (text/html)
8	0.068282444	192.168.115.52	93.184.216.34	68	TCP	58454 → 80 [ACK] Seq=131 Ack=1389 Win=42496 Len=0 TSval=1956021746 TSecr=3563039340
9	0.068406568	192.168.115.52	93.184.216.34	68	TCP	58454 → 80 [ACK] Seq=131 Ack=1608 Win=42496 Len=0 TSval=1956021746 TSecr=3563039340
10	0.070226342	192.168.115.52	93.184.216.34	68	TCP	58454 → 80 [FIN, ACK] Seq=131 Ack=1608 Win=42496 Len=0 TSval=1956021748 TSecr=35630393

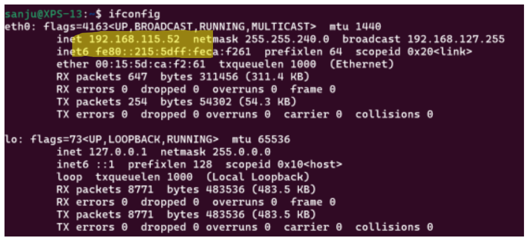
Open a terminal window. Using nslookup, find the A records for www.google.com.

7. [10 pts] Take a screenshot of the packets corresponding to your request, and the response from the server. If the request was resolved, what is the IP address you were given for www.google.com?
Answer: I have tried the nslookup command on both Ubuntu VM and Windows PowerShell. The following images are the outputs.

Ubuntu: The response I got is from my windows machine(cache) which has the IP address of 192.168.112.1. S



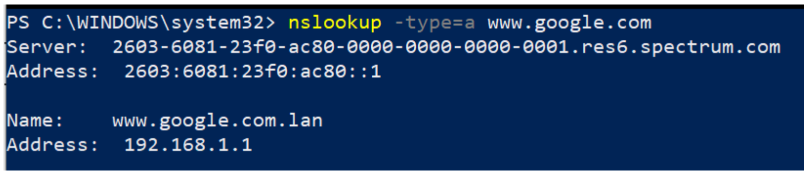
```
sanju@XPS-13:~$ nslookup -type=a www.google.com
Server:         192.168.112.1
Address:        192.168.112.1#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.253.63.99
Name:   www.google.com
Address: 172.253.63.104
Name:   www.google.com
Address: 172.253.63.106
Name:   www.google.com
Address: 172.253.63.147
Name:   www.google.com
Address: 172.253.63.103
Name:   www.google.com
Address: 172.253.63.105
```



```
sanju@XPS-13:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1440
        inet 192.168.115.52  netmask 255.255.240.0  broadcast 192.168.127.255
        inet6 fe80::215:5dff:feca:f261  prefixlen 64  scopeid 0x20<link>
        ether 00:15:5d:ca:f2:61  txqueuelen 1000  (Ethernet)
        RX packets 647  bytes 311456 (311.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 254  bytes 54302 (54.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8771  bytes 483536 (483.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8771  bytes 483536 (483.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Note: my ubuntu IP is 192.168.115.52

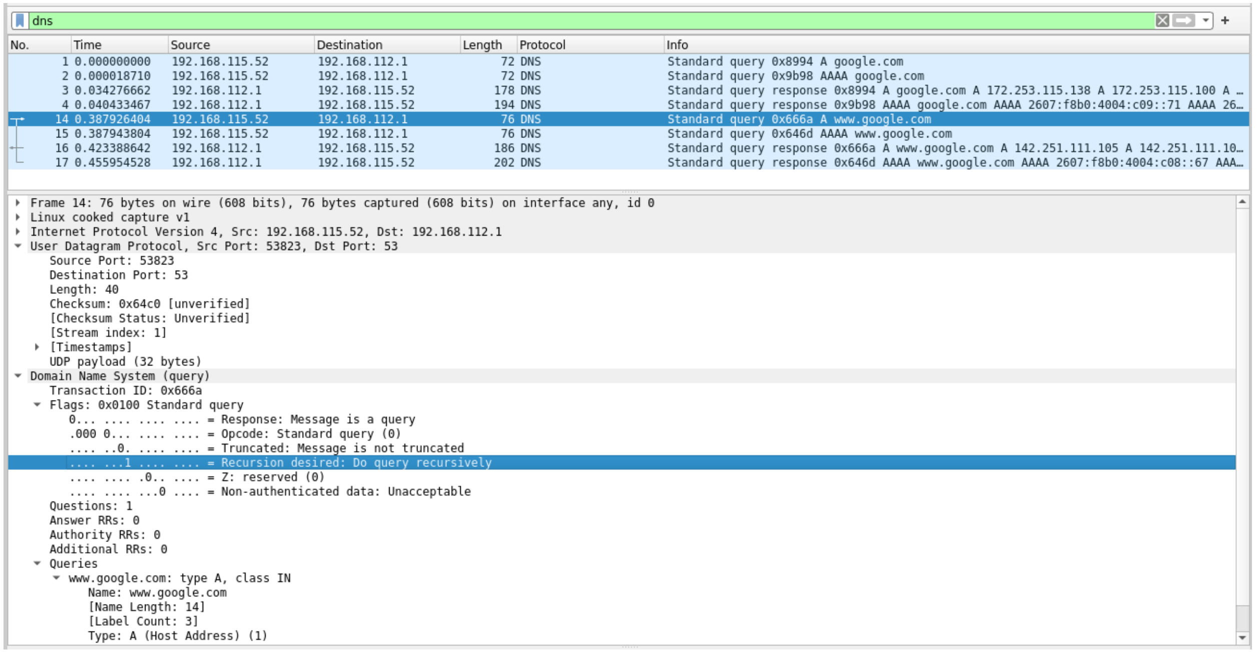
PowerShell (windows 10):



```
PS C:\WINDOWS\system32> nslookup -type=a www.google.com
Server:  2603-6081-23f0-ac80-0000-0000-0000-0001.res6.spectrum.com
Address:  2603:6081:23f0:ac80::1

Name:    www.google.com.lan
Address:  192.168.1.1
```

While querying on windows 10 PowerShell, I get the spectrum server address as mentioned in the image.

8. [10 pts] Did your computer want to complete the request recursively? How do you know? Take a screenshot proving your answer.
Answer: Upon observing the packets for DNS query, my computer wants to do the request recursively. I have observed packet #14 to see the Domain Name System Query Flags. It has Recursion desired set to Do Query Recursively and the flag bit is set to **1.**

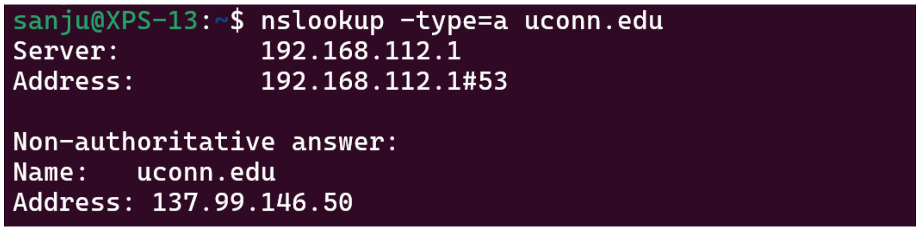


```
No.   Time          Source           Destination      Length Protocol  Info
    1 0.000000000   192.168.115.52   192.168.112.1        72 DNS       Standard query 0x8994 A google.com
    2 0.000018710   192.168.115.52   192.168.112.1        72 DNS       Standard query 0x9b98 AAAA google.com
    3 0.034276662   192.168.112.1    192.168.115.52      178 DNS       Standard query response 0x8994 A google.com A 172.253.115.138 A 172.253.115.100 A …
    4 0.040433467   192.168.112.1    192.168.115.52      194 DNS       Standard query response 0x9b98 AAAA google.com AAAA 2607:f8b0:4004:c09::71 AAAA 26…
   14 0.387926404   192.168.115.52   192.168.112.1        76 DNS       Standard query 0x666a A www.google.com
   15 0.387943804   192.168.115.52   192.168.112.1        76 DNS       Standard query 0x646d AAAA www.google.com
   16 0.423388642   192.168.112.1    192.168.115.52      186 DNS       Standard query response 0x666a A www.google.com A 142.251.111.105 A 142.251.111.10…
   17 0.455954528   192.168.112.1    192.168.115.52      202 DNS       Standard query response 0x646d AAAA www.google.com AAAA 2607:f8b0:4004:c08::67 AAA…
```

```
Frame 14: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: 192.168.115.52, Dst: 192.168.112.1
User Datagram Protocol, Src Port: 53823, Dst Port: 53
    Source Port: 53823
    Destination Port: 53
    Length: 40
    Checksum: 0x64c0 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1]
    [Timestamps]
    UDP payload (32 bytes)
Domain Name System (query)
    Transaction ID: 0x666a
    Flags: 0x0100 Standard query
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...0 .... = Non-authenticated data: Unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
    Queries
        www.google.com: type A, class IN
            Name: www.google.com
            [Name Length: 14]
            [Label Count: 3]
            Type: A (Host Address) (1)
```

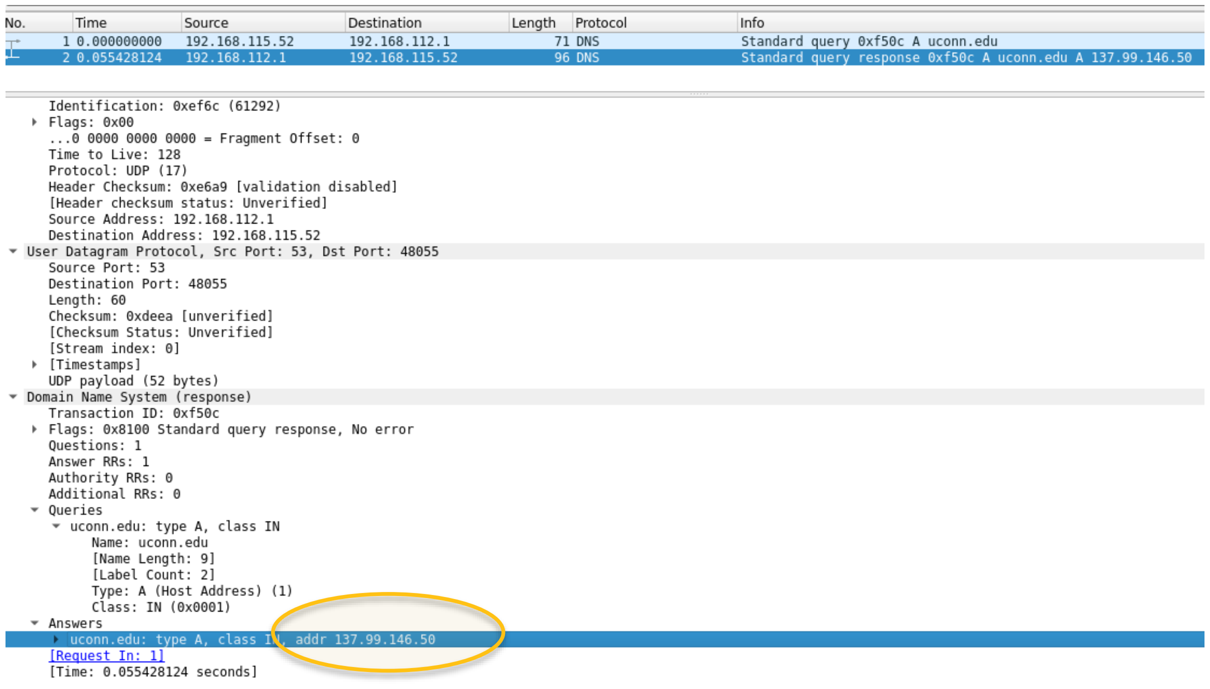
Using nslookup, find the A records for uconn.edu.

9. [10 pts] Take a screenshot of the packets corresponding to your request, and the response from the server. If the request was resolved, what is the IP address you were given for uconn.edu?
Answer: The final IP address for uconn.edu for type A record is **137.99.146.50**



```
sanju@XPS-13:~$ nslookup -type=a uconn.edu
Server:         192.168.112.1
Address:        192.168.112.1#53

Non-authoritative answer:
Name:   uconn.edu
Address: 137.99.146.50
```

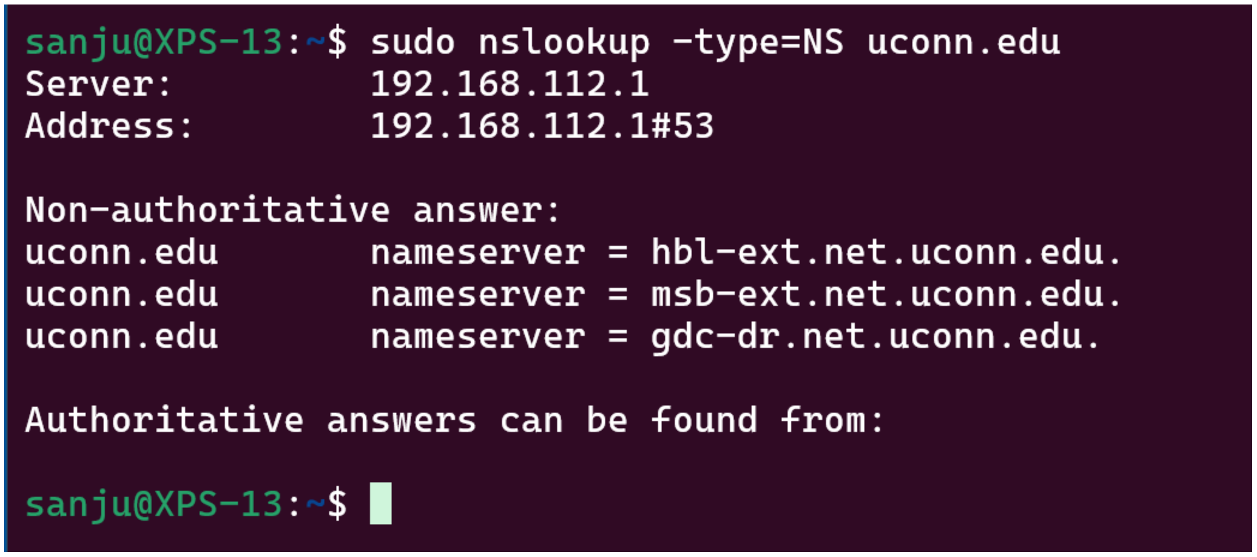
Wireshark packets: Here, we can observe the final IP address is matching in the 2nd packet of type A record.



```
No.  Time         Source          Destination     Length Protocol Info
   1 0.000000000  192.168.115.52  192.168.112.1       71 DNS      Standard query 0xf50c A uconn.edu
   2 0.055428124  192.168.112.1   192.168.115.52      96 DNS      Standard query response 0xf50c A uconn.edu A 137.99.146.50

    Identification: 0xef6c (61292)
  ▸ Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0xe6a9 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.112.1
    Destination Address: 192.168.115.52
▾ User Datagram Protocol, Src Port: 53, Dst Port: 48055
    Source Port: 53
    Destination Port: 48055
    Length: 60
    Checksum: 0xdeea [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  ▸ [Timestamps]
    UDP payload (52 bytes)
▾ Domain Name System (response)
    Transaction ID: 0xf50c
  ▸ Flags: 0x8100 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ uconn.edu: type A, class IN
        Name: uconn.edu
        [Name Length: 9]
        [Label Count: 2]
        Type: A (Host Address) (1)
        Class: IN (0x0001)
  ▾ Answers
    ▸ uconn.edu: type A, class IN, addr 137.99.146.50
    [Request In: 1]
    [Time: 0.055428124 seconds]
```

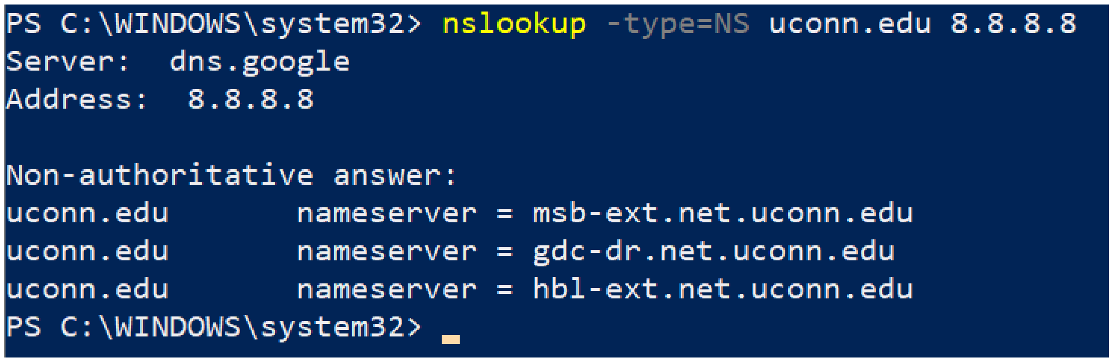
10. [10 pts] What is the authoritative name server for the uconn.edu domain? How do you know? Take a screenshot proving your answer.
Answer: When querying the uconn.edu domain, we get the non-authoritative name servers such as:

hbl-ext.net.uconn.edu. , msb-ext.net.uconn.edu. , gdc-dr.net.uconn.edu.



```
sanju@XPS-13:~$ sudo nslookup -type=NS uconn.edu
Server:         192.168.112.1
Address:        192.168.112.1#53

Non-authoritative answer:
uconn.edu       nameserver = hbl-ext.net.uconn.edu.
uconn.edu       nameserver = msb-ext.net.uconn.edu.
uconn.edu       nameserver = gdc-dr.net.uconn.edu.

Authoritative answers can be found from:

sanju@XPS-13:~$
```

The authoriative answer seems to be empty. When querying the same on Windows PowerShell, we get the same name servers.



```
PS C:\WINDOWS\system32> nslookup -type=NS uconn.edu 8.8.8.8
Server:  dns.google
Address:  8.8.8.8

Non-authoritative answer:
uconn.edu       nameserver = msb-ext.net.uconn.edu
uconn.edu       nameserver = gdc-dr.net.uconn.edu
uconn.edu       nameserver = hbl-ext.net.uconn.edu
PS C:\WINDOWS\system32>
```

Part 3: TCP

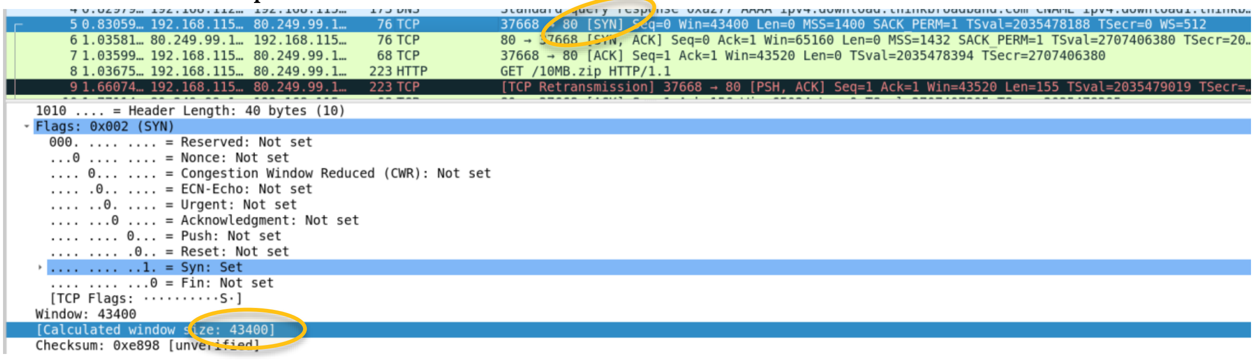
In this section, we will observe how the TCP protocol operates. We will do this by using the Mininet VM. Begin by opening Wireshark and listening on the 'any' interface.

Open a terminal window. Using wget, download the file http://ipv4.download.thinkbroadband.com/10MB.zip

11. [10 pts] Find the packets corresponding with the SYN, SYN-ACK, and ACK that initiated the TCP connection for this file transfer. Take a screenshot of these packets. What was the initial window size that your computer advertised to the server? What was the initial window size that the server advertised to you?
Answer:

The packets corresponding to the TCP connection initialization for the file transfer are:

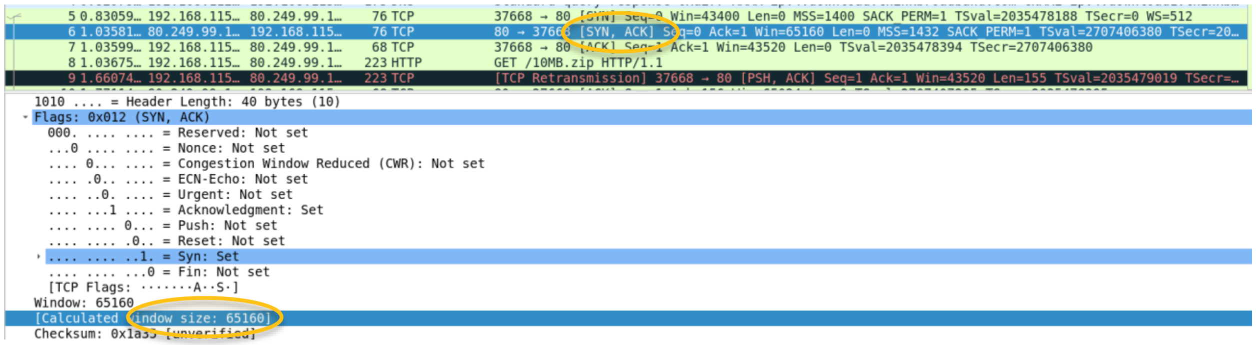
a. SYN: Here, we can see there is SYN with initial windows size of 43400 on my computer.



```
5 0.83059… 192.168.115… 80.249.99.1…   76 TCP    37668 → 80 [SYN] Seq=0 Win=43400 Len=0 MSS=1400 SACK_PERM=1 TSval=2035478188 TSecr=0 WS=512
6 1.03581… 80.249.99.1… 192.168.115…   76 TCP    80 → 37668 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1432 SACK_PERM=1 TSval=2707406380 TSecr=20…
7 1.03599… 192.168.115… 80.249.99.1…   68 TCP    37668 → 80 [ACK] Seq=1 Ack=1 Win=43520 Len=0 TSval=2035478394 TSecr=2707406380
8 1.03675… 192.168.115… 80.249.99.1…  223 HTTP   GET /10MB.zip HTTP/1.1
9 1.66074… 192.168.115… 80.249.99.1…  223 TCP    [TCP Retransmission] 37668 → 80 [PSH, ACK] Seq=1 Ack=1 Win=43520 Len=155 TSval=2035479019 TSecr=…
```

```
  1010 .... = Header Length: 40 bytes (10)
▾ Flags: 0x002 (SYN)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...0 .... = Acknowledgment: Not set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  ▸ .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ··········S·]
  Window: 43400
  [Calculated window size: 43400]
  Checksum: 0xe898 [unverified]
```

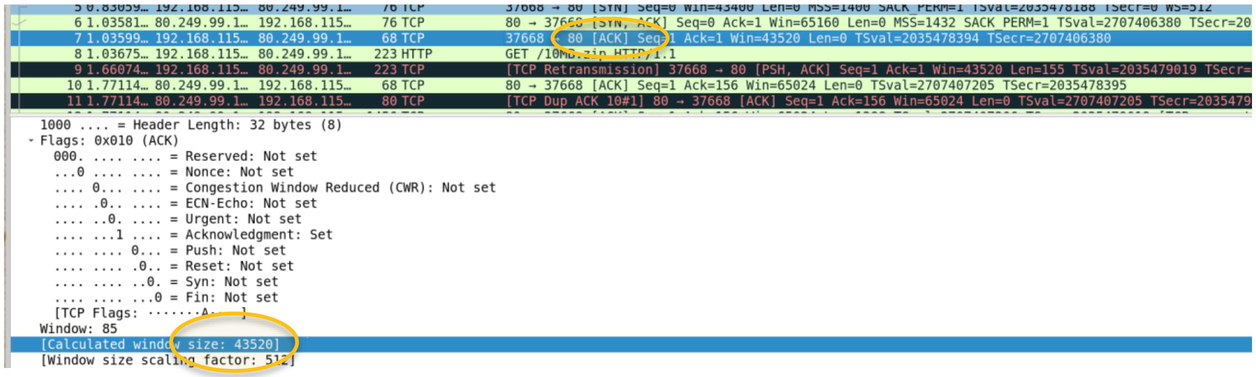
b. SYN-ACK: Here, the SYN-ACK is from Server, the initial window size advertised by server is 65160.



```
5 0.83059… 192.168.115… 80.249.99.1…   76 TCP    37668 → 80 [SYN] Seq=0 Win=43400 Len=0 MSS=1400 SACK_PERM=1 TSval=2035478188 TSecr=0 WS=512
6 1.03581… 80.249.99.1… 192.168.115…   76 TCP    80 → 37668 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1432 SACK_PERM=1 TSval=2707406380 TSecr=20…
7 1.03599… 192.168.115… 80.249.99.1…   68 TCP    37668 → 80 [ACK] Seq=1 Ack=1 Win=43520 Len=0 TSval=2035478394 TSecr=2707406380
8 1.03675… 192.168.115… 80.249.99.1…  223 HTTP   GET /10MB.zip HTTP/1.1
9 1.66074… 192.168.115… 80.249.99.1…  223 TCP    [TCP Retransmission] 37668 → 80 [PSH, ACK] Seq=1 Ack=1 Win=43520 Len=155 TSval=2035479019 TSecr=…
```

```
  1010 .... = Header Length: 40 bytes (10)
▾ Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  ▸ .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·······A··S·]
  Window: 65160
  [Calculated window size: 65160]
  Checksum: 0x1a33 [unverified]
```

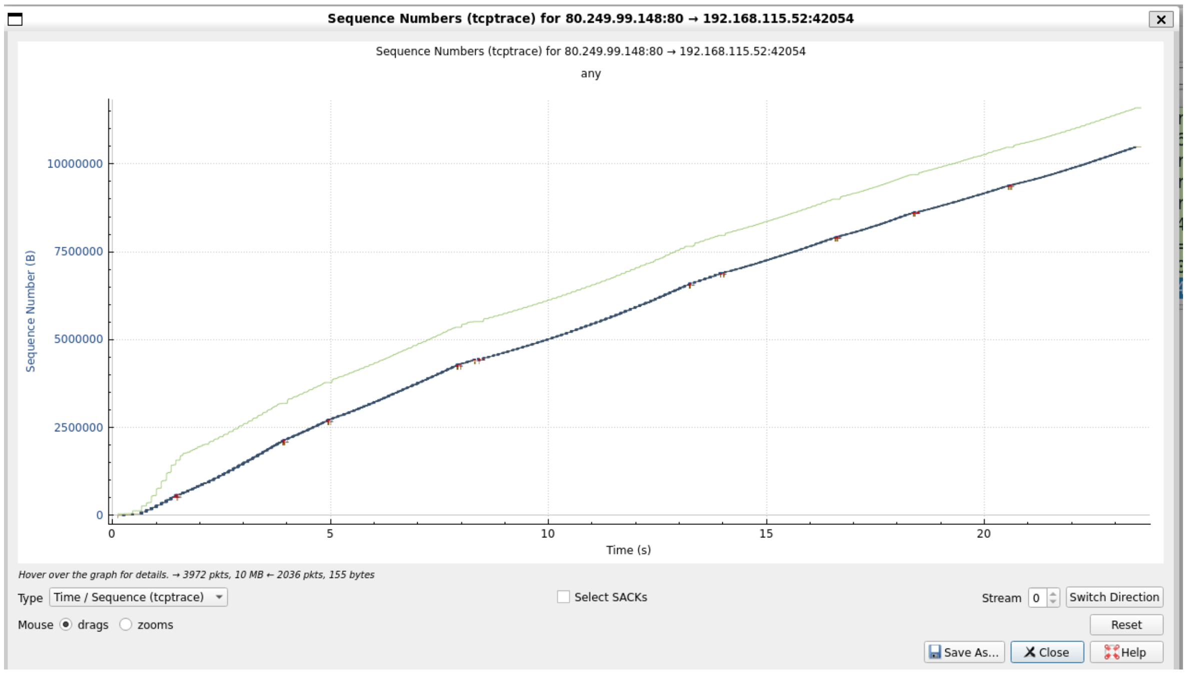
c. ACK: Here, we see the first ACK from my computer to the server with the calculated window size of 43520.



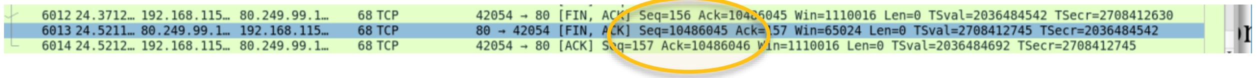
```
 5 0.83059… 192.168.115… 80.249.99.1…   76 TCP    37668 → 80 [SYN] Seq=0 Win=43400 Len=0 MSS=1400 SACK_PERM=1 TSval=2035478188 TSecr=0 WS=512
 6 1.03581… 80.249.99.1… 192.168.115…   76 TCP    80 → 37668 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1432 SACK_PERM=1 TSval=2707406380 TSecr=20
 7 1.03599… 192.168.115… 80.249.99.1…   68 TCP    37668 → 80 [ACK] Seq=1 Ack=1 Win=43520 Len=0 TSval=2035478394 TSecr=2707406380
 8 1.03675… 192.168.115… 80.249.99.1…  223 HTTP   GET /10MB.zip HTTP/1.1
 9 1.66074… 192.168.115… 80.249.99.1…  223 TCP    [TCP Retransmission] 37668 → 80 [PSH, ACK] Seq=1 Ack=1 Win=43520 Len=155 TSval=2035479019 TSecr=
10 1.77114… 80.249.99.1… 192.168.115…   68 TCP    80 → 37668 [ACK] Seq=1 Ack=156 Win=65024 Len=0 TSval=2707407205 TSecr=2035478395
11 1.77114… 80.249.99.1… 192.168.115…   80 TCP    [TCP Dup ACK 10#1] 80 → 37668 [ACK] Seq=1 Ack=156 Win=65024 Len=0 TSval=2707407205 TSecr=2035479
```

```
  1000 .... = Header Length: 32 bytes (8)
▾ Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·······A····]
  Window: 85
  [Calculated window size: 43520]
  [Window size scaling factor: 512]
```

12. [10 pts] Find a packet from the download with a source of the server and a destination of your computer. Create a tcptrace graph with this packet selected. Take a screenshot of the graph and explain what it is showing. Look into the Wireshark documentation if you need assistance making this graph.
Answer: I have created the tcptrace graph with the 10MB downlaoded file from thinkboardband.com



Here, we can observe that the sequence numbers for the packets increase over time. A total of 6014 packets are used with 10MB for 2036 packets. The last Seq # used is 10486045 and we receive an ACK number in the next packet of 10486046.



```
6012 24.3712… 192.168.115… 80.249.99.1…   68 TCP    42054 → 80 [FIN, ACK] Seq=156 Ack=10486045 Win=1110016 Len=0 TSval=2036484542 TSecr=2708412630
6013 24.5211… 80.249.99.1… 192.168.115…   68 TCP    80 → 42054 [FIN, ACK] Seq=10486045 Ack=157 Win=65024 Len=0 TSval=2708412745 TSecr=2036484542
6014 24.5212… 192.168.115… 80.249.99.1…   68 TCP    42054 → 80 [ACK] Seq=157 Ack=10486046 Win=1110016 Len=0 TSval=2036484692 TSecr=2708412745
```

In the next section, we will be simulating loss, the command *tc qdisc* will be needed. When you first use the command you should use *add dev* for the device you plan on changing. It only needs to be set on the sender's side. After adding the device use *change dev*.

Example:

```
sudo tc qdisc add dev eth0 root netem loss 0%
sudo tc qdisc change dev eth0 root netem loss 100%
```

Read through the following paragraph before starting the next step. Open 2 terminals and have the commands typed and ready before you begin. In one terminal, download the

10MB.zip file again. While the download is in progress, change loss to 100%. After a few seconds, change loss to 0%.

13. [10 pts] Find a packet from the download with the source of the server and a destination of your computer. Create a tcptrace graph with this packet selected. Take a screenshot of the graph and explain what it is showing. Using an image editting program, circle the areas where the 0% loss is shown, as well as where TCP is in slow-start and congestion-avoidance.