Cyber Security

Mruganshi Gohel

B20CS014

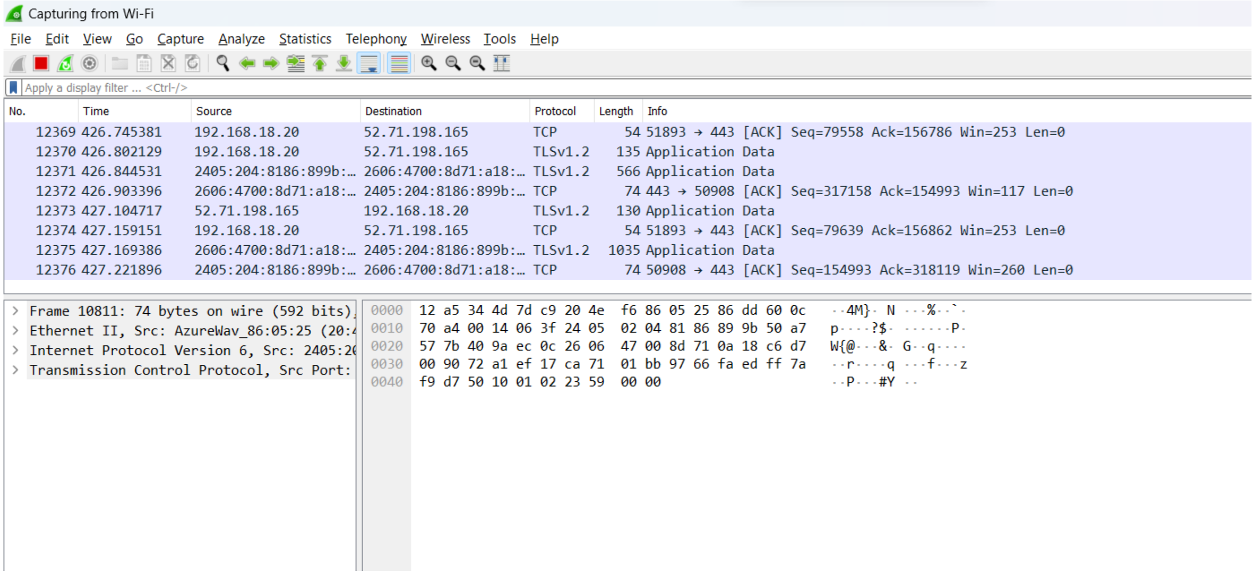
1. **Start packet capture in Wireshark on your wireless interface. What do you observe?**

   to see packet capture in Wireshark on my wireless interface, Firstly I will open wireshark application and under the capture, I can see different interfaces i.e LANx, Ethernet, Bluetooth Network, Wi-fi, etc. Now to see on the wireless interface click on wi-fi that is a wireless interface and we will be able to see all the packet transfers happening on my browser.

   there are seven columns :

   - No. - this shows the packet number, which is the order in which the packet has been captured.
   - Time - this shows the time at which the packet has been captured
   - Source - This shows the IP or MAC address of the machine from where the packet has been sent.
   - Destination - this shows the IP or MAC address of the machine to where the packet has to be sent.
   - Protocol - this shows the protocol that is used in transferring packets from source to destination.
   - Length - Length of the packet in bytes
   - Info - Additional information about packet

here is the output of packet capture on wireless interface.



Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

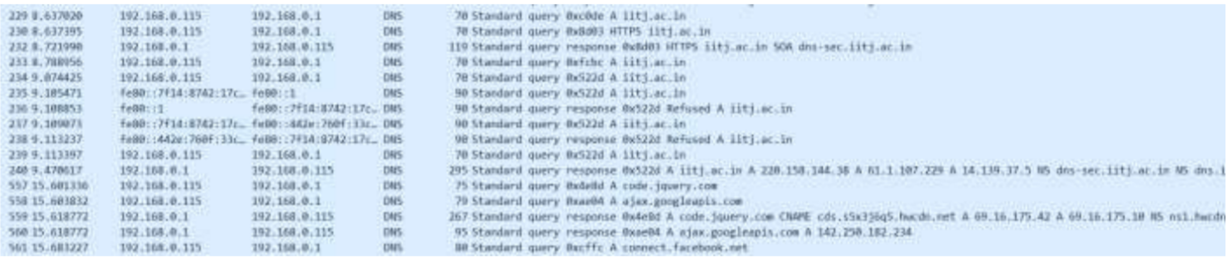
```
No.     Time        Source                Destination           Protocol  Length Info
  12369 426.745381  192.168.18.20         52.71.198.165         TCP        54 51893 → 443 [ACK] Seq=79558 Ack=156786 Win=253 Len=0
  12370 426.802129  192.168.18.20         52.71.198.165         TLSv1.2   135 Application Data
  12371 426.844531  2405:204:8186:899b:…  2606:4700:8d71:a18:…  TLSv1.2   566 Application Data
  12372 426.903396  2606:4700:8d71:a18:…  2405:204:8186:899b:…  TCP        74 443 → 50908 [ACK] Seq=317158 Ack=154993 Win=117 Len=0
  12373 427.104717  52.71.198.165         192.168.18.20         TLSv1.2   130 Application Data
  12374 427.159151  192.168.18.20         52.71.198.165         TCP        54 51893 → 443 [ACK] Seq=79639 Ack=156862 Win=253 Len=0
  12375 427.169386  2606:4700:8d71:a18:…  2405:204:8186:899b:…  TLSv1.2  1035 Application Data
  12376 427.221896  2405:204:8186:899b:…  2606:4700:8d71:a18:…  TCP        74 50908 → 443 [ACK] Seq=154993 Ack=318119 Win=260 Len=0
```

```
> Frame 10811: 74 bytes on wire (592 bits),
> Ethernet II, Src: AzureWav_86:05:25 (20:4
> Internet Protocol Version 6, Src: 2405:20
> Transmission Control Protocol, Src Port:
```

```
0000  12 a5 34 4d 7d c9 20 4e  f6 86 05 25 86 dd 60 0c   ··4M}· N ···%··`·
0010  70 a4 00 14 06 3f 24 05  02 04 81 86 89 9b 50 a7   p····?$· ······P·
0020  57 7b 40 9a ec 0c 26 06  47 00 8d 71 0a 18 c6 d7   W{@···&· G··q····
0030  00 90 72 a1 ef 17 ca 71  01 bb 97 66 fa ed ff 7a   ··r····q ···f···z
0040  f9 d7 50 10 01 02 23 59  00 00                     ··P···#Y ··
```

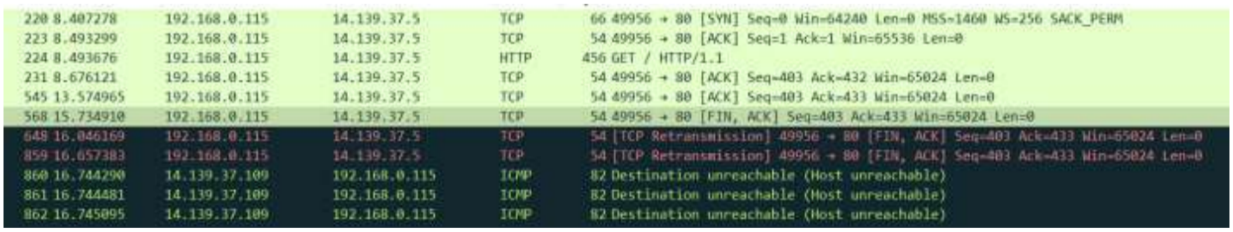
2. **Now visit a local website, say www.iitj.ac.in. Subsequently, stop the packet capture and record your observations. Are you able to see the DNS request? What about TCP and HTTP? What is the IP address of the IITJ server? Are you able to see different HTTP requests/responses? Please justify your answer with relevant screenshots.**

   We can observe that when we access the IITJ website, our localhost first sends a DNS request to the IITJ server.



```
229 8.637020   192.168.0.115          192.168.0.1            DNS   70 Standard query 0xc0de A iitj.ac.in
230 8.637395   192.168.0.115          192.168.0.1            DNS   70 Standard query 0x8d03 HTTPS iitj.ac.in
232 8.721990   192.168.0.1            192.168.0.115          DNS  119 Standard query response 0x8d03 HTTPS iitj.ac.in SOA dns-sec.iitj.ac.in
233 8.788056   192.168.0.115          192.168.0.1            DNS   70 Standard query 0xfcbc A iitj.ac.in
234 9.074425   192.168.0.115          192.168.0.1            DNS   70 Standard query 0x522d A iitj.ac.in
235 9.105471   fe80::7f14:8742:17c…   fe80::1                DNS   90 Standard query 0x522d A iitj.ac.in
236 9.108853   fe80::1                fe80::7f14:8742:17c…   DNS   90 Standard query response 0x522d Refused A iitj.ac.in
237 9.109073   fe80::7f14:8742:17c…   fe80::442e:760f:33c…   DNS   90 Standard query 0x522d A iitj.ac.in
238 9.113237   fe80::442e:760f:33c…   fe80::7f14:8742:17c…   DNS   90 Standard query response 0x522d Refused A iitj.ac.in
239 9.113397   192.168.0.115          192.168.0.1            DNS   70 Standard query 0x522d A iitj.ac.in
240 9.470617   192.168.0.1            192.168.0.115          DNS  295 Standard query response 0x522d A iitj.ac.in A 220.158.144.38 A 61.1.107.229 A 14.139.37.5 NS dns-sec.iitj.ac.in NS dns.i
557 15.601336  192.168.0.115          192.168.0.1            DNS   75 Standard query 0xde8d A code.jquery.com
558 15.603832  192.168.0.115          192.168.0.1            DNS   79 Standard query 0xae04 A ajax.googleapis.com
559 15.618772  192.168.0.1            192.168.0.115          DNS  267 Standard query response 0x4e8d A code.jquery.com CNAME cds.s5x3j6q5.hwcdn.net A 69.16.175.42 A 69.16.175.10 NS ns1.hwcdn
560 15.618772  192.168.0.1            192.168.0.115          DNS   95 Standard query response 0xae04 A ajax.googleapis.com A 142.250.182.234
561 15.681227  192.168.0.115          192.168.0.1            DNS   80 Standard query 0xcffc A connect.facebook.net
```

In this screenshot we can see that the different standard query of iitj.ac.in is made from our local host to the iitj server, by observing these queries we can find the destination IP address.



```
220 8.407278    192.168.0.115   14.139.37.5     TCP    66 49956 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
223 8.493299    192.168.0.115   14.139.37.5     TCP    54 49956 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
224 8.493676    192.168.0.115   14.139.37.5     HTTP  456 GET / HTTP/1.1
231 8.676121    192.168.0.115   14.139.37.5     TCP    54 49956 → 80 [ACK] Seq=403 Ack=432 Win=65024 Len=0
545 13.574965   192.168.0.115   14.139.37.5     TCP    54 49956 → 80 [ACK] Seq=403 Ack=433 Win=65024 Len=0
568 15.734910   192.168.0.115   14.139.37.5     TCP    54 49956 → 80 [FIN, ACK] Seq=403 Ack=433 Win=65024 Len=0
648 16.046169   192.168.0.115   14.139.37.5     TCP    54 [TCP Retransmission] 49956 → 80 [FIN, ACK] Seq=403 Ack=433 Win=65024 Len=0
859 16.657383   192.168.0.115   14.139.37.5     TCP    54 [TCP Retransmission] 49956 → 80 [FIN, ACK] Seq=403 Ack=433 Win=65024 Len=0
860 16.744290   14.139.37.109   192.168.0.115   ICMP   82 Destination unreachable (Host unreachable)
861 16.744481   14.139.37.109   192.168.0.115   ICMP   82 Destination unreachable (Host unreachable)
862 16.745095   14.139.37.109   192.168.0.115   ICMP   82 Destination unreachable (Host unreachable)
```

In this screenshot we can see the destination IP address of iitj.ac.in
Destination IP address of iitj.ac.in = 14.139.37.5

Also, we can see the HTTP request sent from local-host to the iitj server, which ultimately results in the retrieval of the iitj.ac.in page. We can see that a GET request is being sent, and the HTTP request's version is also visible.

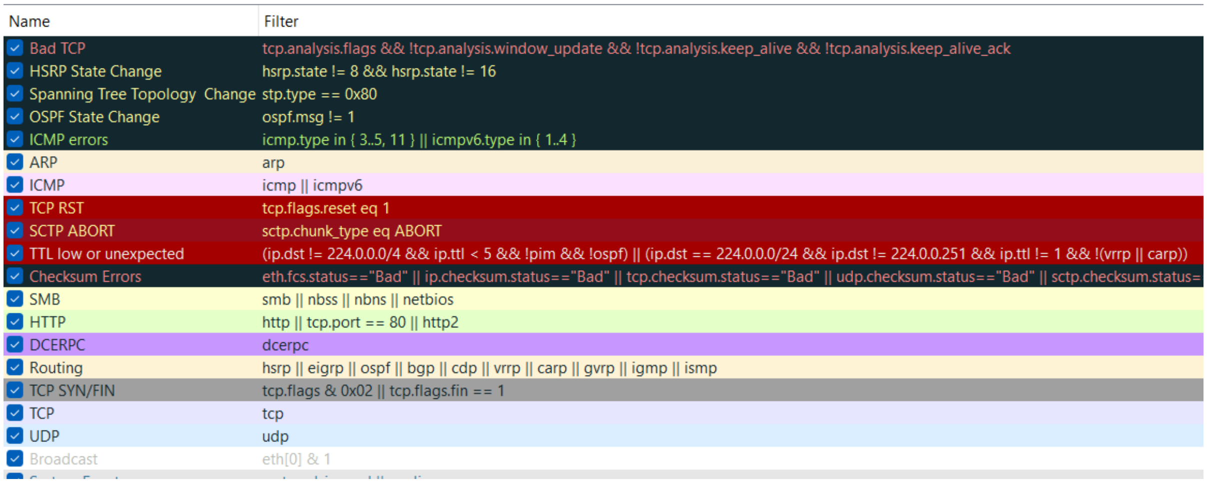
We can also see that the HTTP answer comes from the server, which then delivers the client the requested packets.

3. **What does a packet highlight in `black' color signify?**

there are multiple reasons why the packet is highlighted in black color.

1. Bad TCP
2. HSRP state change
3. spanning tree topology change
4. OSPF state change
5. ICMP errors
6. checksum errors

this can be found via color rules that is an option in view option of wireshark . screenshot is attached below



Name	Filter
☑ Bad TCP	tcp.analysis.flags && !tcp.analysis.window_update && !tcp.analysis.keep_alive && !tcp.analysis.keep_alive_ack
☑ HSRP State Change	hsrp.state != 8 && hsrp.state != 16
☑ Spanning Tree Topology Change	stp.type == 0x80
☑ OSPF State Change	ospf.msg != 1
☑ ICMP errors	icmp.type in { 3..5, 11 } || icmpv6.type in { 1..4 }
☑ ARP	arp
☑ ICMP	icmp || icmpv6
☑ TCP RST	tcp.flags.reset eq 1
☑ SCTP ABORT	sctp.chunk_type eq ABORT
☑ TTL low or unexpected	(ip.dst != 224.0.0.0/4 && ip.ttl < 5 && !pim && !ospf) || (ip.dst == 224.0.0.0/24 && ip.dst != 224.0.0.251 && ip.ttl != 1 && !(vrrp || carp))
☑ Checksum Errors	eth.fcs.status=="Bad" || ip.checksum.status=="Bad" || tcp.checksum.status=="Bad" || udp.checksum.status=="Bad" || sctp.checksum.status=
☑ SMB	smb || nbss || nbns || netbios
☑ HTTP	http || tcp.port == 80 || http2
☑ DCERPC	dcerpc
☑ Routing	hsrp || eigrp || ospf || bgp || cdp || vrrp || carp || gvrp || igmp || ismp
☑ TCP SYN/FIN	tcp.flags & 0x02 || tcp.flags.fin == 1
☑ TCP	tcp
☑ UDP	udp
☑ Broadcast	eth[0] & 1

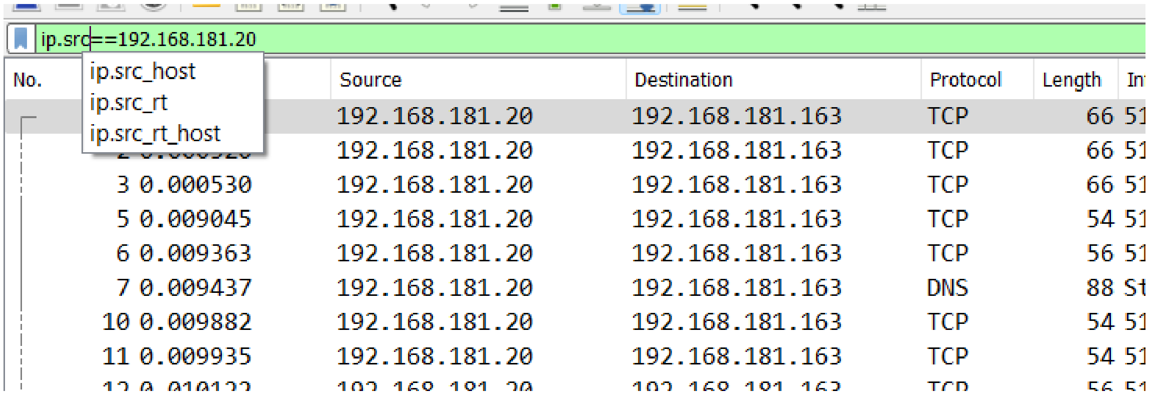
here each black color signifies that what could be the reason for black color for a packet.

4. **explore five different filters in wireshark.**

   here are 5 different filters that I have used

   1. **Filter by Source IP Address:**

   To filter packets by a specific source IP address, use the filter "ip.src == [IP Address]". This will display only the packets that have the specified IP address as the source.

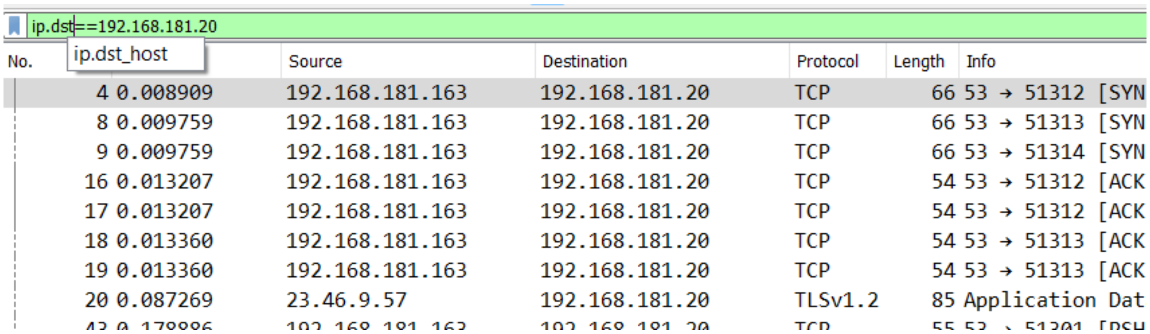


ip.src==192.168.181.20

No.		Source	Destination	Protocol	Length	In
		192.168.181.20	192.168.181.163	TCP	66	51
		192.168.181.20	192.168.181.163	TCP	66	51
3	0.000530	192.168.181.20	192.168.181.163	TCP	66	51
5	0.009045	192.168.181.20	192.168.181.163	TCP	54	51
6	0.009363	192.168.181.20	192.168.181.163	TCP	56	51
7	0.009437	192.168.181.20	192.168.181.163	DNS	88	St
10	0.009882	192.168.181.20	192.168.181.163	TCP	54	51
11	0.009935	192.168.181.20	192.168.181.163	TCP	54	51

   2. **Filter by destination IP address**

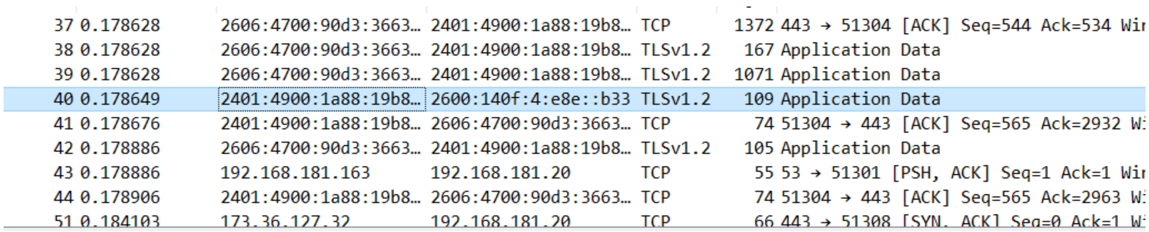
   To filter packets by a specific destination IP address, use the filter "ip.dst == [IP Address]". This will display only the packets that have the specified IP address as the destination.



```
ip.dst==192.168.181.20
        ip.dst_host
No.                  Source            Destination      Protocol  Length Info
      4 0.008909     192.168.181.163   192.168.181.20   TCP       66 53 → 51312 [SYN
      8 0.009759     192.168.181.163   192.168.181.20   TCP       66 53 → 51313 [SYN
      9 0.009759     192.168.181.163   192.168.181.20   TCP       66 53 → 51314 [SYN
     16 0.013207     192.168.181.163   192.168.181.20   TCP       54 53 → 51312 [ACK
     17 0.013207     192.168.181.163   192.168.181.20   TCP       54 53 → 51312 [ACK
     18 0.013360     192.168.181.163   192.168.181.20   TCP       54 53 → 51313 [ACK
     19 0.013360     192.168.181.163   192.168.181.20   TCP       54 53 → 51313 [ACK
     20 0.087269     23.46.9.57        192.168.181.20   TLSv1.2   85 Application Dat
     43 0.178886     192.168.181.163   192.168.181.20   TCP       55 53 → 51301 [PSH
```

3. filter by protocol

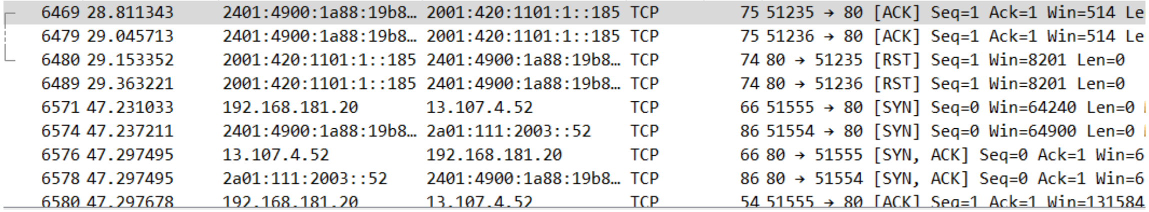
To filter packets by a specific protocol, use the filter “[Protocol]". This will display only the packets that have the specified protocol.



```
37 0.178628   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TCP     1372 443 → 51304 [ACK] Seq=544 Ack=534 Wi
38 0.178628   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TLSv1.2  167 Application Data
39 0.178628   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TLSv1.2 1071 Application Data
40 0.178649   2401:4900:1a88:19b8… 2600:140f:4:e8e::b33 TLSv1.2  109 Application Data
41 0.178676   2401:4900:1a88:19b8… 2606:4700:90d3:3663… TCP       74 51304 → 443 [ACK] Seq=565 Ack=2932 Wi
42 0.178886   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TLSv1.2  105 Application Data
43 0.178886   192.168.181.163      192.168.181.20       TCP       55 53 → 51301 [PSH, ACK] Seq=1 Ack=1 Wi
44 0.178906   2401:4900:1a88:19b8… 2606:4700:90d3:3663… TCP       74 51304 → 443 [ACK] Seq=565 Ack=2963 Wi
51 0.184103   173.36.127.32        192.168.181.20       TCP       66 443 → 51308 [SYN, ACK] Seq=0 Ack=1 Wi
```

4. filter by port number

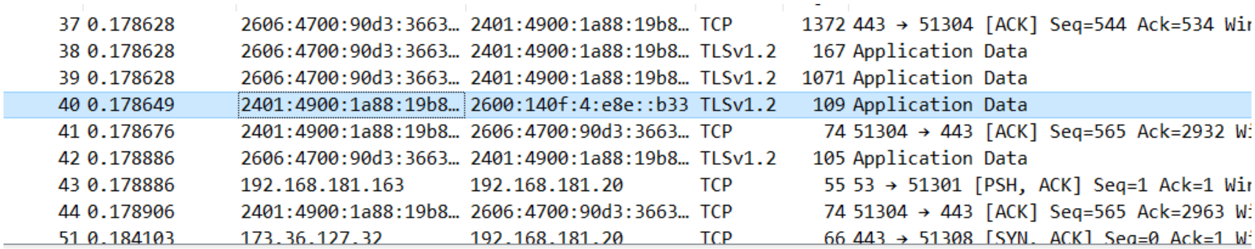
To filter packets by a specific port number, use the filter "tcp.port == [Port Number]". This will display only the packets that have the specific port number.



```
6469 28.811343  2401:4900:1a88:19b8… 2001:420:1101:1::185 TCP  75 51235 → 80 [ACK] Seq=1 Ack=1 Win=514 Le
6479 29.045713  2401:4900:1a88:19b8… 2001:420:1101:1::185 TCP  75 51236 → 80 [ACK] Seq=1 Ack=1 Win=514 Le
6480 29.153352  2001:420:1101:1::185 2401:4900:1a88:19b8… TCP  74 80 → 51235 [RST] Seq=1 Win=8201 Len=0
6489 29.363221  2001:420:1101:1::185 2401:4900:1a88:19b8… TCP  74 80 → 51236 [RST] Seq=1 Win=8201 Len=0
6571 47.231033  192.168.181.20       13.107.4.52          TCP  66 51555 → 80 [SYN] Seq=0 Win=64240 Len=0
6574 47.237211  2401:4900:1a88:19b8… 2a01:111:2003::52    TCP  86 51554 → 80 [SYN] Seq=0 Win=64900 Len=0
6576 47.297495  13.107.4.52          192.168.181.20       TCP  66 80 → 51555 [SYN, ACK] Seq=0 Ack=1 Win=6
6578 47.297495  2a01:111:2003::52    2401:4900:1a88:19b8… TCP  86 80 → 51554 [SYN, ACK] Seq=0 Ack=1 Win=6
6580 47.297678  192.168.181.20       13.107.4.52          TCP  54 51555 → 80 [ACK] Seq=1 Ack=1 Win=131584
```

5. filter by time range

To filter packets by a specific time range, use the filter "frame.time >= [Start Time] and frame.time <= [End Time]"



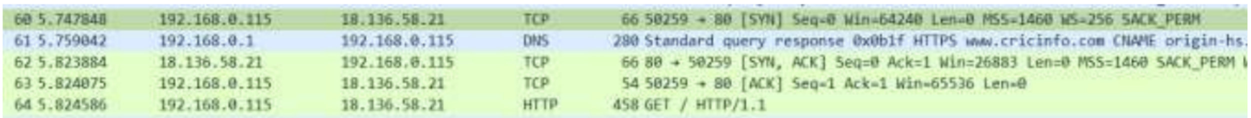
```
37 0.178628   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TCP      1372 443 → 51304 [ACK] Seq=544 Ack=534 Wi
38 0.178628   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TLSv1.2   167 Application Data
39 0.178628   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TLSv1.2  1071 Application Data
40 0.178649   2401:4900:1a88:19b8… 2600:140f:4:e8e::b33 TLSv1.2   109 Application Data
41 0.178676   2401:4900:1a88:19b8… 2606:4700:90d3:3663… TCP        74 51304 → 443 [ACK] Seq=565 Ack=2932 Wi
42 0.178886   2606:4700:90d3:3663… 2401:4900:1a88:19b8… TLSv1.2   105 Application Data
43 0.178886   192.168.181.163      192.168.181.20       TCP        55 53 → 51301 [PSH, ACK] Seq=1 Ack=1 Wi
44 0.178906   2401:4900:1a88:19b8… 2606:4700:90d3:3663… TCP        74 51304 → 443 [ACK] Seq=565 Ack=2963 Wi
51 0.184103   173.36.127.32        192.168.181.20       TCP        66 443 → 51308 [SYN, ACK] Seq=0 Ack=1 Wi
```

5. **What is the filter command for listing all outgoing traffic?**

the filter command for listing all outgoing traffic is **http.request**. this will show all packets that have a specific IP address and subnet as the source address. which indicates all outgoing traffic from the device with that IP address or subnet. but we can use this filter only when the source address that matches the specified subnet or IP address.

6. **Start a new packet capture to now visit an external website, say <u>www.cricinfo.com</u>.**
   Can you show the 3-way TCP handshake happening? Can you see your IITJ proxy in
   between? What is its IP address?

It demonstrates how the SYN signal is first delivered from the source to the destination, followed by the server producing the ACK signal [SYN, ACK], and lastly sending the ACK signal back to the source.



```
60 5.747848   192.168.0.115   18.136.58.21    TCP    66 50259 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
61 5.759042   192.168.0.1     192.168.0.115   DNS   280 Standard query response 0x0b1f HTTPS www.cricinfo.com CNAME origin-hs.
62 5.823884   18.136.58.21    192.168.0.115   TCP    66 80 → 50259 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460 SACK_PERM
63 5.824075   192.168.0.115   18.136.58.21    TCP    54 50259 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
64 5.824586   192.168.0.115   18.136.58.21    HTTP  458 GET / HTTP/1.1
```

(no iitj proxy server as outside camus)

7. **Why does DNS follow the UDP stream while HTTP follows the TCP stream?**

Different transport layer protocols are used by HTTP and DNS to convey data. Whereas HTTP employs the Transmission Control Protocol (TCP), DNS makes use of the User

Datagram Protocol (UDP) (TCP).

Being a connectionless protocol, UDP does not require the establishment of a connection in order to transmit data. Each UDP packet is autonomous and contains all the information required for delivery. Because of this, it is better suited for brief, straightforward queries and answers, such those used in DNS.

TCP, on the other hand, is a protocol focused on connections. Before data can be transmitted, a link needs to be made between the sender and recipient. Throughout the data transfer, this connection is kept open, and packets are transmitted in the correct order and with delivery assurance. This makes it more appropriate for requests and replies that are longer and more complicated, like those used in HTTP.

Streams are automatically created when Wireshark records network data. These streams are based on the source and destination IP addresses and ports, as well as the transport protocol employed. Because DNS packets are frequently brief and straightforward, they don't need the consistency and orderliness that TCP offers. As a result, Wireshark views all DNS packets as belonging to the same stream and interprets each one as a separate UDP packet.

On the other hand, HTTP packets frequently involve numerous packets and need stability and sequencing. As a result, Wireshark considers them as a TCP stream and displays each packet as part of the same stream.

8. **Run your socket program (both server and client) and show the TCP communication**
   happening at different ports.

for this question single client and server handshaking program I have written.

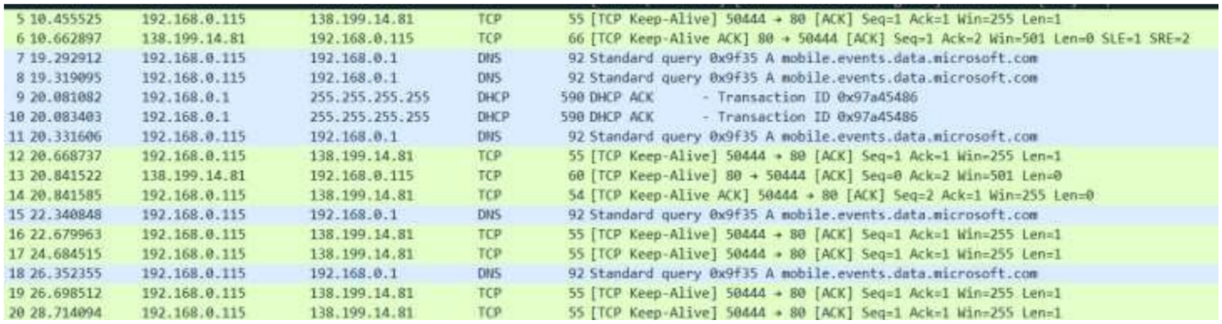
client code:

```
import socket
SERVER_ADDRESS = "localhost"
SERVER_PORT = 1234
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_ADDRESS, SERVER_PORT))
client_socket.sendall(b"Requesting a handshake")
data = client_socket.recv(1024)
print("Received: ", data.decode())
client_socket.close()
```

server code:

```
import socket
SERVER_ADDRESS = "localhost"
SERVER_PORT = 1234
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((SERVER_ADDRESS, SERVER_PORT))
server_socket.listen()
print("Listening for incoming connections...")
client_socket, client_address = server_socket.accept()
print("Connection from", client_address, "has been established.")
client_socket.sendall(b"Handshake successful!")
client_socket.close()
server_socket.close()
```

so firstly server will operate on localhost and client will send a request to the server, after the server accepts the request, the client sends a message to the server.

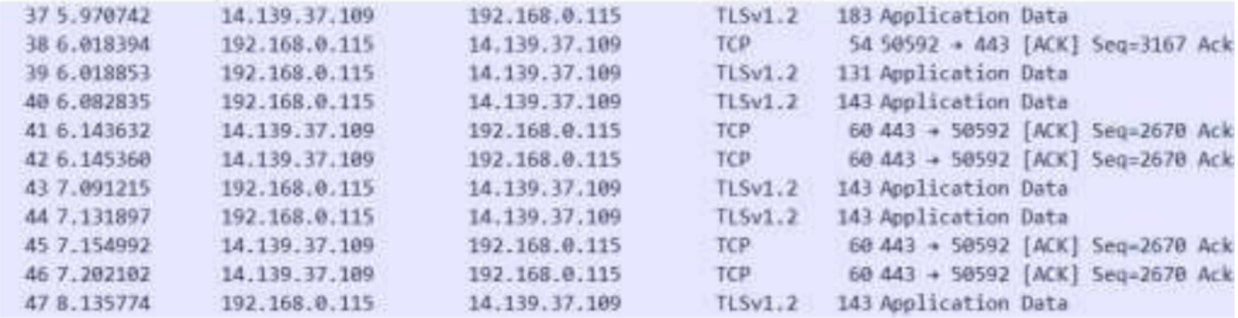


```
 5 10.455525   192.168.0.115   138.199.14.81     TCP    55 [TCP Keep-Alive] 50444 → 80 [ACK] Seq=1 Ack=1 Win=255 Len=1
 6 10.662897   138.199.14.81   192.168.0.115     TCP    66 [TCP Keep-Alive ACK] 80 → 50444 [ACK] Seq=1 Ack=2 Win=501 Len=0 SLE=1 SRE=2
 7 19.292912   192.168.0.115   192.168.0.1       DNS    92 Standard query 0x9f35 A mobile.events.data.microsoft.com
 8 19.319095   192.168.0.115   192.168.0.1       DNS    92 Standard query 0x9f35 A mobile.events.data.microsoft.com
 9 20.081082   192.168.0.1     255.255.255.255   DHCP  590 DHCP ACK      - Transaction ID 0x97a45486
10 20.083403   192.168.0.1     255.255.255.255   DHCP  590 DHCP ACK      - Transaction ID 0x97a45486
11 20.331606   192.168.0.115   192.168.0.1       DNS    92 Standard query 0x9f35 A mobile.events.data.microsoft.com
12 20.668737   192.168.0.115   138.199.14.81     TCP    55 [TCP Keep-Alive] 50444 → 80 [ACK] Seq=1 Ack=1 Win=255 Len=1
13 20.841522   138.199.14.81   192.168.0.115     TCP    60 [TCP Keep-Alive] 80 → 50444 [ACK] Seq=0 Ack=2 Win=501 Len=0
14 20.841585   192.168.0.115   138.199.14.81     TCP    54 [TCP Keep-Alive ACK] 50444 → 80 [ACK] Seq=2 Ack=1 Win=255 Len=0
15 22.340848   192.168.0.115   192.168.0.1       DNS    92 Standard query 0x9f35 A mobile.events.data.microsoft.com
16 22.679963   192.168.0.115   138.199.14.81     TCP    55 [TCP Keep-Alive] 50444 → 80 [ACK] Seq=1 Ack=1 Win=255 Len=1
17 24.684515   192.168.0.115   138.199.14.81     TCP    55 [TCP Keep-Alive] 50444 → 80 [ACK] Seq=1 Ack=1 Win=255 Len=1
18 26.352355   192.168.0.115   192.168.0.1       DNS    92 Standard query 0x9f35 A mobile.events.data.microsoft.com
19 26.698512   192.168.0.115   138.199.14.81     TCP    55 [TCP Keep-Alive] 50444 → 80 [ACK] Seq=1 Ack=1 Win=255 Len=1
20 28.714094   192.168.0.115   138.199.14.81     TCP    55 [TCP Keep-Alive] 50444 → 80 [ACK] Seq=1 Ack=1 Win=255 Len=1
```

The server has already established a connection with localhost in the first section, and the first two TCP requests for connections with localhost are seen here. The client then requests the server, uses DNS to locate the server's domain name, then uses DHCP to create a connection with the server. When we have set the client destination address as the default, we can see that it is 255.255.255.255. so this way handshaking is done between client and server.

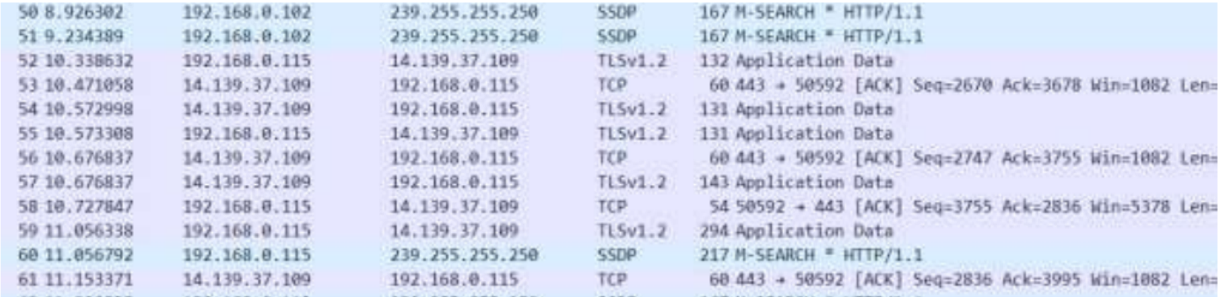
9. **Perform an SSH to your IITJ home folder and show the relevant screenshots captured using Wireshark.**

I will try to contact IITJ home server using my id and password.so I will set request to forticlient so request screenshot is attached below:



```
37 5.970742    14.139.37.109    192.168.0.115    TLSv1.2    183 Application Data
38 6.018394    192.168.0.115    14.139.37.109    TCP         54 50592 → 443 [ACK] Seq=3167 Ack
39 6.018853    192.168.0.115    14.139.37.109    TLSv1.2    131 Application Data
40 6.082835    192.168.0.115    14.139.37.109    TLSv1.2    143 Application Data
41 6.143632    14.139.37.109    192.168.0.115    TCP         60 443 → 50592 [ACK] Seq=2670 Ack
42 6.145360    14.139.37.109    192.168.0.115    TCP         60 443 → 50592 [ACK] Seq=2670 Ack
43 7.091215    192.168.0.115    14.139.37.109    TLSv1.2    143 Application Data
44 7.131897    192.168.0.115    14.139.37.109    TLSv1.2    143 Application Data
45 7.154992    14.139.37.109    192.168.0.115    TCP         60 443 → 50592 [ACK] Seq=2670 Ack
46 7.202102    14.139.37.109    192.168.0.115    TCP         60 443 → 50592 [ACK] Seq=2670 Ack
47 8.135774    192.168.0.115    14.139.37.109    TLSv1.2    143 Application Data
```

screenshot of iitj home server:



```
50 8.926302     192.168.0.102    239.255.255.250    SSDP       167 M-SEARCH * HTTP/1.1
51 9.234389     192.168.0.102    239.255.255.250    SSDP       167 M-SEARCH * HTTP/1.1
52 10.338632    192.168.0.115    14.139.37.109      TLSv1.2    132 Application Data
53 10.471058    14.139.37.109    192.168.0.115      TCP         60 443 → 50592 [ACK] Seq=2670 Ack=3678 Win=1082 Len=
54 10.572998    14.139.37.109    192.168.0.115      TLSv1.2    131 Application Data
55 10.573308    192.168.0.115    14.139.37.109      TLSv1.2    131 Application Data
56 10.676837    14.139.37.109    192.168.0.115      TCP         60 443 → 50592 [ACK] Seq=2747 Ack=3755 Win=1082 Len=
57 10.676837    14.139.37.109    192.168.0.115      TLSv1.2    143 Application Data
58 10.727847    192.168.0.115    14.139.37.109      TCP         54 50592 → 443 [ACK] Seq=3755 Ack=2836 Win=5378 Len=
59 11.056338    192.168.0.115    14.139.37.109      TLSv1.2    294 Application Data
60 11.056792    192.168.0.115    239.255.255.250    SSDP       217 M-SEARCH * HTTP/1.1
61 11.153371    14.139.37.109    192.168.0.115      TCP         60 443 → 50592 [ACK] Seq=2836 Ack=3995 Win=1082 Len=
```

s here we can also see the connection establishment and the packet info along with destination address of home page of iitj.
So this way we can access our IITJ home folder.