Week 1 Assignment
WireShark

Q1: List up to 10 different protocols that appear in the protocol column in the unfiltered packet-listing window.

A: A protocol is a set of instructions or rules for transmitting data between electronic devices, such as computers.

1. Ethernet: The base layer protocol for most wired local area networks can be used mostly in the data link layer
2. IPv4: Internet Protocol version 4 responsible for addressing and routing packets across networks
3. TCP: Transmission control protocol ensured reliable and orderly data transmission
4. UDP: User Datagram protocol, a connectionless protocol that provides a simple way to exchange data packets without guarantees of reliability
5. HTTP: Hypertext transfer protocol, used for transferring webpages and other resources on the world wide web
6. DNS: Domain Name System, responsible for transferring human-readable domain names into IP addresses
7. ARP: Address Resolution Protocol, used to map a MAC address to an IP address in the local network
8. ICMP: Internet Control Message Protocol, used for sending error messages and operational information about network conditions
9. TLS: Transport Layer Security, a cryptographic protocol it provides secure communication over a computer network
10. SSH: Secure shell, a cryptographic network protocol used for secure remote access to devices on unsecured networks.

Q2: How long did it take from when the HTTP (or TLS) GET message was sent until the HTTP OK reply was received? Include the screenshot.

A: HTTP GET: An HTTP GET message is a request sent by a client (typically a web browser) to a web server to retrieve a specific resource, such as a web page, image, CSS file, or script.

HTTP OK: An HTTP OK message, often referred to as an HTTP 200 OK response, is a response sent by a web server to a client (such as a web browser) after successfully fulfilling an HTTP request.

Time taken to receive an HTTP OK message after an HTTP GET request for a website is approximately 0.08706 sec.
Refer to the 35 and 40^{th} columns of the following screenshot.

No.	Time	Source	Destination	Protocol	Length	Info
35	6.2947…	10.1.36.156	34.104.35.123	HTTP	525	GET /edgedl/diffgen-puffin/gcmjkmgdlgnkkc
37	6.3378…	34.104.35.123	10.1.36.156	HTTP	692	HTTP/1.1 416 Requested range not satisfia
38	6.3398…	10.1.36.156	34.104.35.123	HTTP	505	HEAD /edgedl/diffgen-puffin/gcmjkmgdlgnkk
40	6.3818…	34.104.35.123	10.1.36.156	HTTP	667	HTTP/1.1 200 OK
41	6.4187…	10.1.36.156	34.104.35.123	HTTP	525	GET /edgedl/diffgen-puffin/gcmjkmgdlgnkkc
43	6.4817…	34.104.35.123	10.1.36.156	HTTP	731	HTTP/1.1 416 Requested range not satisfia
44	6.4832…	10.1.36.156	34.104.35.123	HTTP	505	HEAD /edgedl/diffgen-puffin/gcmjkmgdlgnkk
46	6.5272…	34.104.35.123	10.1.36.156	HTTP	706	HTTP/1.1 200 OK
47	6.5612…	10.1.36.156	34.104.35.123	HTTP	525	GET /edgedl/diffgen-puffin/gcmjkmgdlgnkkc
49	6.5996…	34.104.35.123	10.1.36.156	HTTP	731	HTTP/1.1 416 Requested range not satisfia
50	6.6012…	10.1.36.156	34.104.35.123	HTTP	505	HEAD /edgedl/diffgen-puffin/gcmjkmgdlgnkk
51	6.6454…	34.104.35.123	10.1.36.156	HTTP	667	HTTP/1.1 200 OK
52	6.6714…	10.1.36.156	34.104.35.123	HTTP	525	GET /edgedl/diffgen-puffin/gcmjkmgdlgnkkc
53	6.7207…	34.104.35.123	10.1.36.156	HTTP	731	HTTP/1.1 416 Requested range not satisfia
54	6.7228…	10.1.36.156	34.104.35.123	HTTP	505	HEAD /edgedl/diffgen-puffin/gcmjkmgdlgnkk

Q3: What is the Internet address (IP address) of www.gmail.com? What is the Internet address of your computer? Include a screenshot and describe where you got the data to answer this question.

A: IP: IP stands for "Internet Protocol," and an IP address is like a digital address for a device on the internet. It's a unique combination of numbers that helps computers locate and communicate with each other. Just like the address on an envelope, an IP address tells the Internet where to send data.

There are two main types of IP addresses: IPv4 and IPv6. IPv4 addresses look something like this: 192.168.1.1. They consist of four sets of numbers separated by dots. IPv6 addresses are more advanced and look like this: 2001:0db8:85a3:0000:0000:8a2e:0370:7334. They use a longer format because there are many more possible combinations.

So, when you type a website's address (like www.example.com) into your web browser, your computer uses a system called DNS (Domain Name System) to translate that human-friendly address into an IP address.

IPv4 address of www.google.com is 8.8.8.8 and/or 8.8.4.4

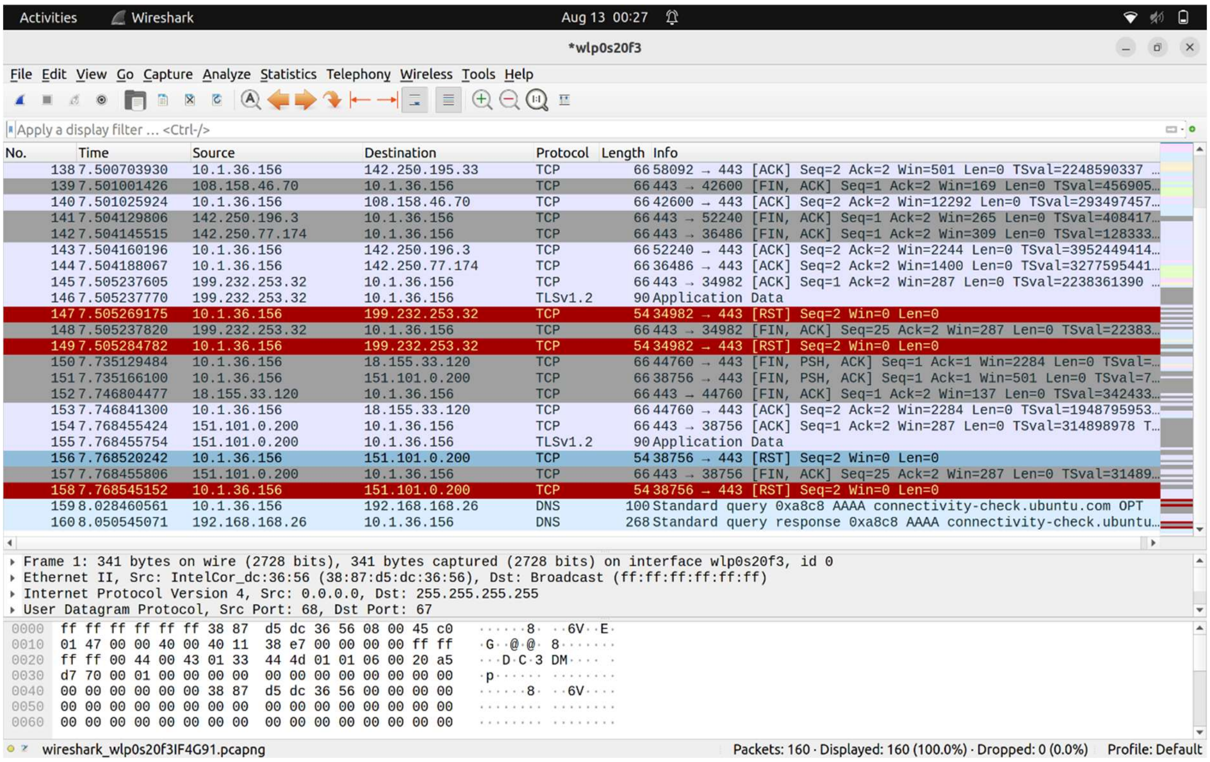
IPv6 address of www.google.com is 2001:4860:4860::8888 and/or 2001:4860:4860::8844

IPv4 address of my personal computer is 10.1.36.156

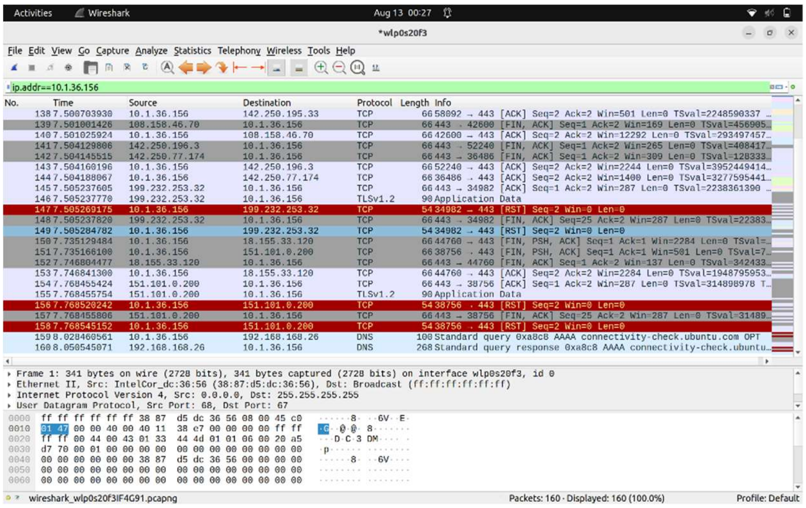
```
Windows PowerShell
   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Unknown adapter Local Area Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter WiFi:

   Connection-specific DNS Suffix  . : srmap.univ
   IPv4 Address. . . . . . . . . . . : 10.1.36.156
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.1.36.254

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
PS C:\Users\91879>
```

Q4: How many packets did you capture (total of all protocols, not just TLS)? Now, use display filters to determine how many packets contain your IP address (Hint: Use ip.addr). Now, reverse the filter to determine how many packets don't contain your IP address. See any problems here? If not, you've already figured out the point of this question, so explain how you did so. If so, how can this problem be fixed? What are the appropriate display filters to use? How does Wireshark warn you of such a problem?

A: When captured normally over the Wi-Fi interface we'll capture only the packets either that were sent from our IP address or received by our IP address.



I captured nearly 160 packets over the interface and when I apply the filter ip.addr==<my_ipaddress> (ip.src=<my_ipaddress> || ip.dst==<my_ipaddress>) display filter, then also I got exactly 160 packets because when capture in normal mode we'll only capture the packets that were either received or sent from our IP address. This because **Network Interface Card** (NIC) in **Manger** mode.



Network Interface Card (NIC): A Network Interface Card (NIC), also known as a network adapter or network interface controller, is a hardware component that allows computers to connect to a computer network. It serves as the bridge between a computer's internal communication and the external network communication. NICs are used to provide computers with the ability to transmit and receive data over various types of networks, such as local area networks (LANs), wide area networks (WANs), and the internet.

Manager mode: It will only allow the NIC to capture the network traffic related to your IP address.

To capture the packets that doesn't contain my IP address I should use
ip.addr!=<my_ipaddress>
I'll get zero packets captured because the NIC was in Manager mode. When we change it to **monitor mode** then we'll be able to capture the packets that will not contain my IP address

Monitor mode: Monitor mode is a specialized mode in wireless networking, particularly in Wi-Fi technology. When a wireless interface is set to monitor mode, it allows the device to capture all wireless frames and packets in the air, regardless of whether they are intended for the monitoring device.

So to capture the packet that doesn't contain my Ip address we should use

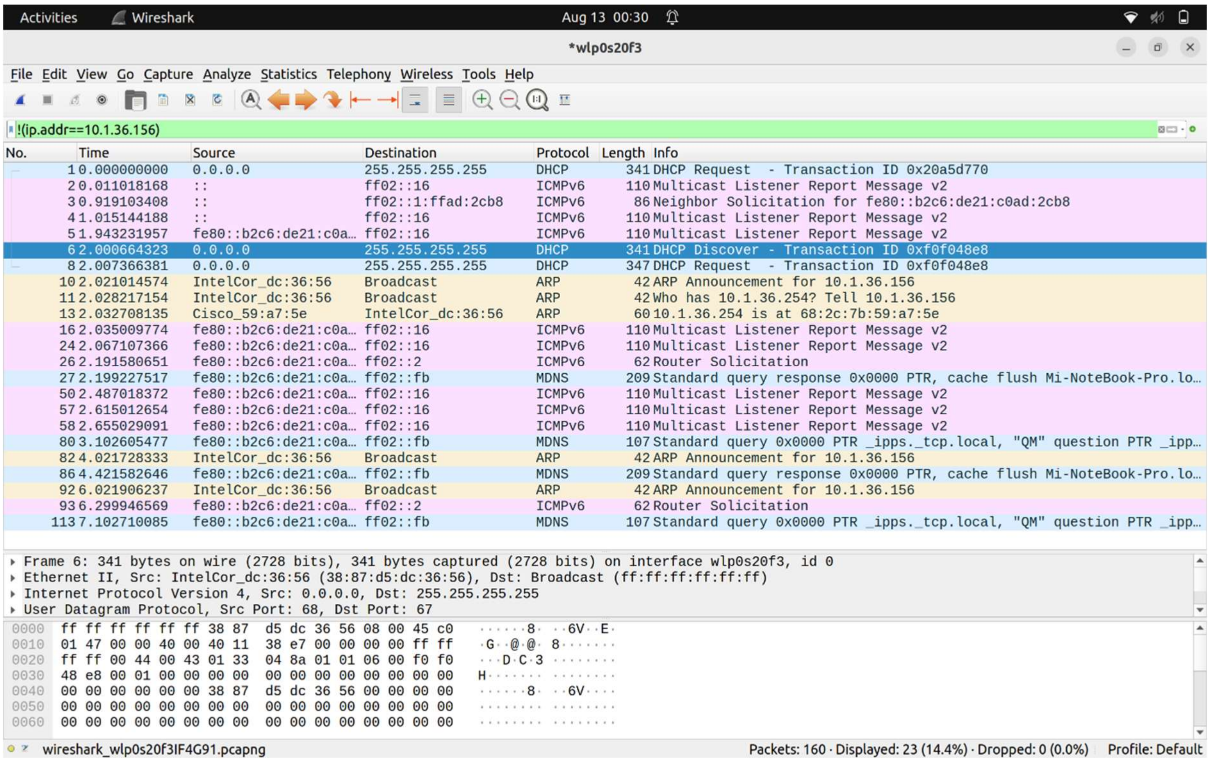
!(ip.addr==<my_ipaddress>) instead of ip.addr!=<my_ipaddress>

Because,

"ip.addr == 192.0.2.1means "match all packets that contain at least one instance of the ip.addr field with the value 192.0.2.1", so it will match packets from 192.0.2.1 and packets to 192.0.2.1.

"!(ip.addr == 192.0.2.1) "means "*don't* match any packets that contain at least one instance of the ip.addr field with the value 192.0.2.1", so it will not match packets from 192.0.2.1 or packets to 192.0.2.1.

"ip.addr != 192.0.2.1" means "match all packets that contain at least one instance of the ip.addr field with a value *other* than 192.0.2.1", so it will match packets from 192.0.2.1 that aren't going to 192.0.2.1, as the destination address will not be equal to 192.0.2.1, and will match packets to 192.0.2.1 that aren't from 192.0.2.1, as the source address will not be equal to 192.0.2.1.



Q5: Use your newly acquired Wireshark skills to capture the process when your browser loads the front page of SRM's website (i.e. https://srmap.edu.in/). How many packets did you capture? Were all of them TLS? How many TLS requests did you make?

A: I used the display filter tcp.port == 443 && ip.addr==3.7.78.115 because, TCP protocol was applied over the given website so we can't find the website by directly adding the filter http so I used the tcp.port ==443 which will give me the packtes received from HTTPS webistes and we can use tcp.port==80 for HTTP websites. Among these websites to only analyze the packtes from srmp.edu.in website I used the ip.addr==3.7.78.115. We can get the ip address of a website by using the **nslookup srmap.edu.in.**

1. All the packets were not TLS only, I received some TCP packets too.
2. The TLS protocol, which stands for Transport Layer Security, is a cryptographic protocol designed to provide secure communication over a computer network. TLS ensures that data transmitted between a client and a server remains private, secure, and tamper-proof. It's primarily used to secure various types of online communication, such as web browsing, email, instant messaging, and more.
3. I found some messages like **Client Hello, Server Hello, Certificate, Key Exchange** etc

In the context of the TLS (Transport Layer Security) protocol, a handshake refers to a series of messages exchanged between a client and a server when establishing a secure connection. The handshake ensures that both the client and server agree on encryption methods, authenticate each other's identities, and exchange cryptographic keys for secure communication.

- **ClientHello:** The client initiates the handshake by sending a ClientHello message to the server. This message includes information such as the TLS version supported by the client, a list of supported cipher suites (encryption algorithms), and other parameters.
- **ServerHello**: The server responds with a ServerHello message, indicating the TLS version and cipher suite it has chosen from the options provided by the client. It may also send its digital certificate, which contains its public key.
- **Certificate:** If the server sends its digital certificate in the ServerHello message, the client receives it. The certificate contains the server's public key and information about the certificate issuer.
- **Key Exchange and Authentication:** Depending on the selected cipher suite, the client and server may perform a key exchange. This step involves exchanging cryptographic information to generate shared secret keys for encryption and authentication. Key exchange methods can include Diffie-Hellman key exchange, RSA encryption, or other techniques.