Scenario

In this scenario, you're a security analyst investigating traffic to a website.

You'll analyze a network packet capture file that contains traffic data related to a user connecting to an internet site. The ability to filter network traffic using packet sniffers to gather relevant information is an essential skill as a security analyst.

You must filter the data in order to:

- identify the source and destination IP addresses involved in this web browsing session,
- examine the protocols that are used when the user makes the connection to the website, and
- analyze some of the data packets to identify the type of information sent and received by the systems that connect to each other when the network data is captured.

Here is an overview of the key property columns listed for each packet:

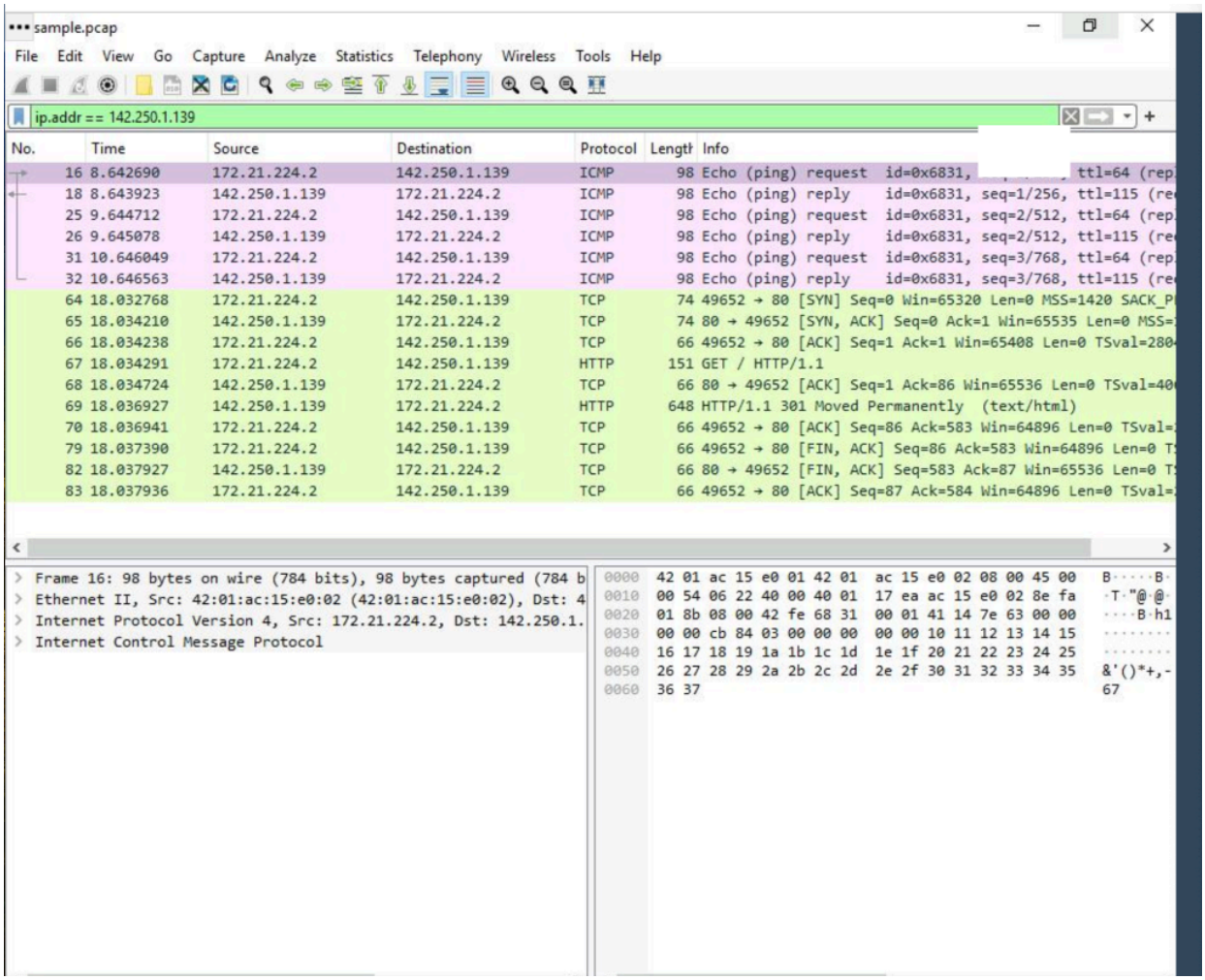
- **No.** : The index number of the packet in this packet capture file
- **Time**: The timestamp of the packet
- **Source**: The source IP address
- **Destination**: The destination IP address
- **Protocol**: The protocol contained in the packet
- **Length**: The total length of the packet

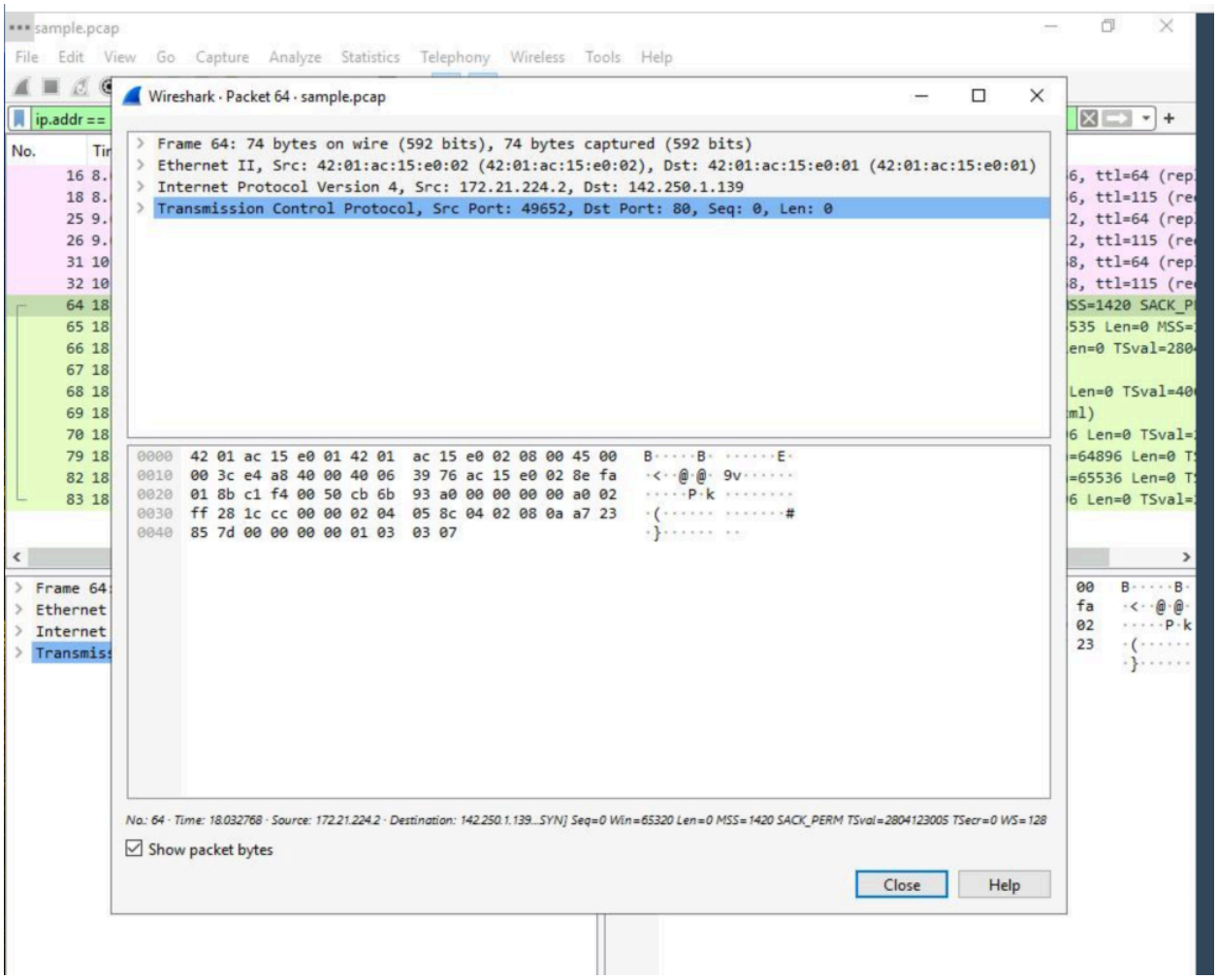
- **Info**: Some infomation about the data in the packet (the payload) as interpreted by Wireshark

You're ready to use Wireshark to inspect network packet data!

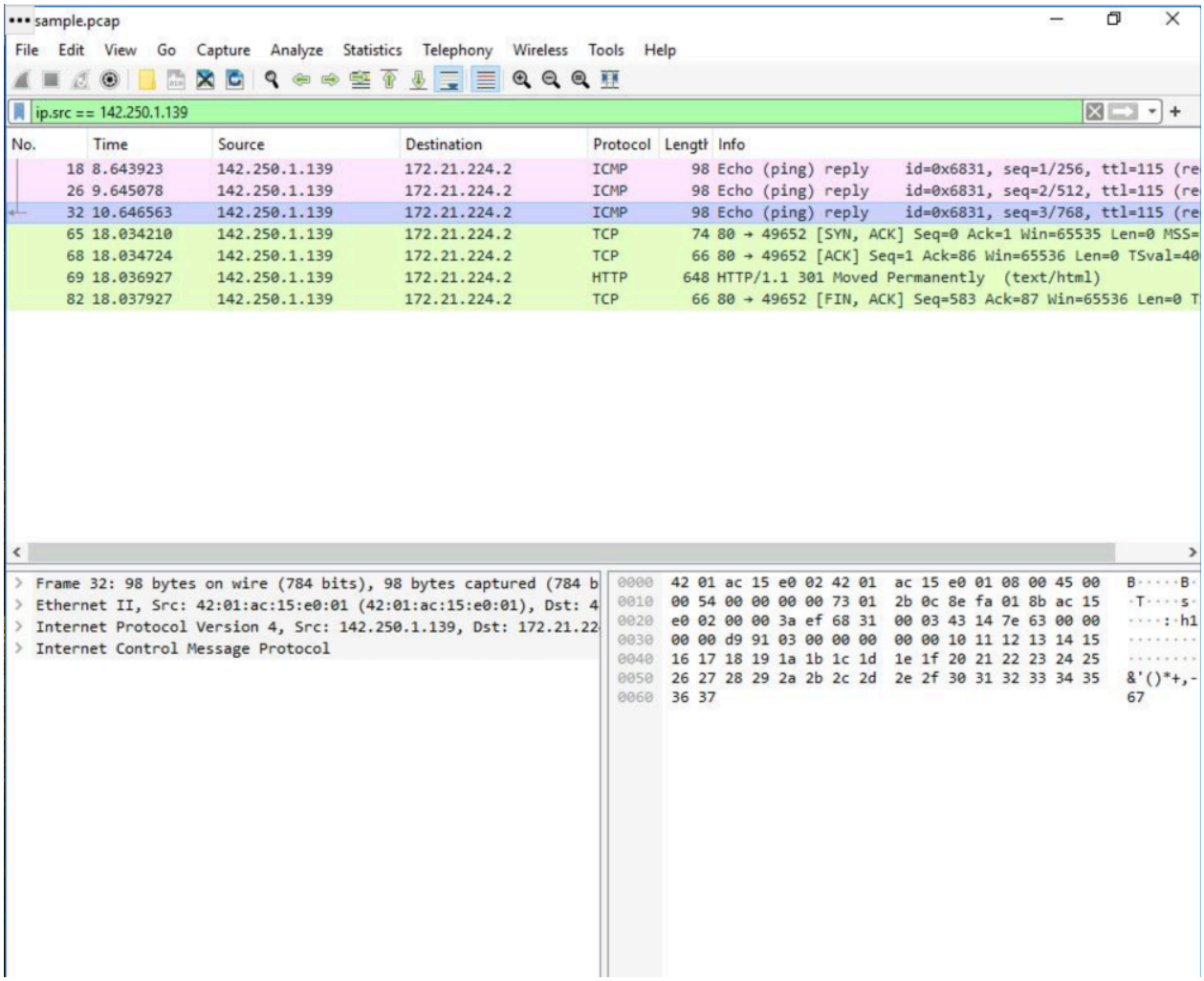
Solutions

1. Identify the source and destination IP addresses involved in this web browsing session.
- On the title bar, type `ip.addr == 142.250.1.139` to filter for traffic associated with a specific IP address. Select the first packet that contains `TCP` on the info field. `addr` means either the source or the destination IP.

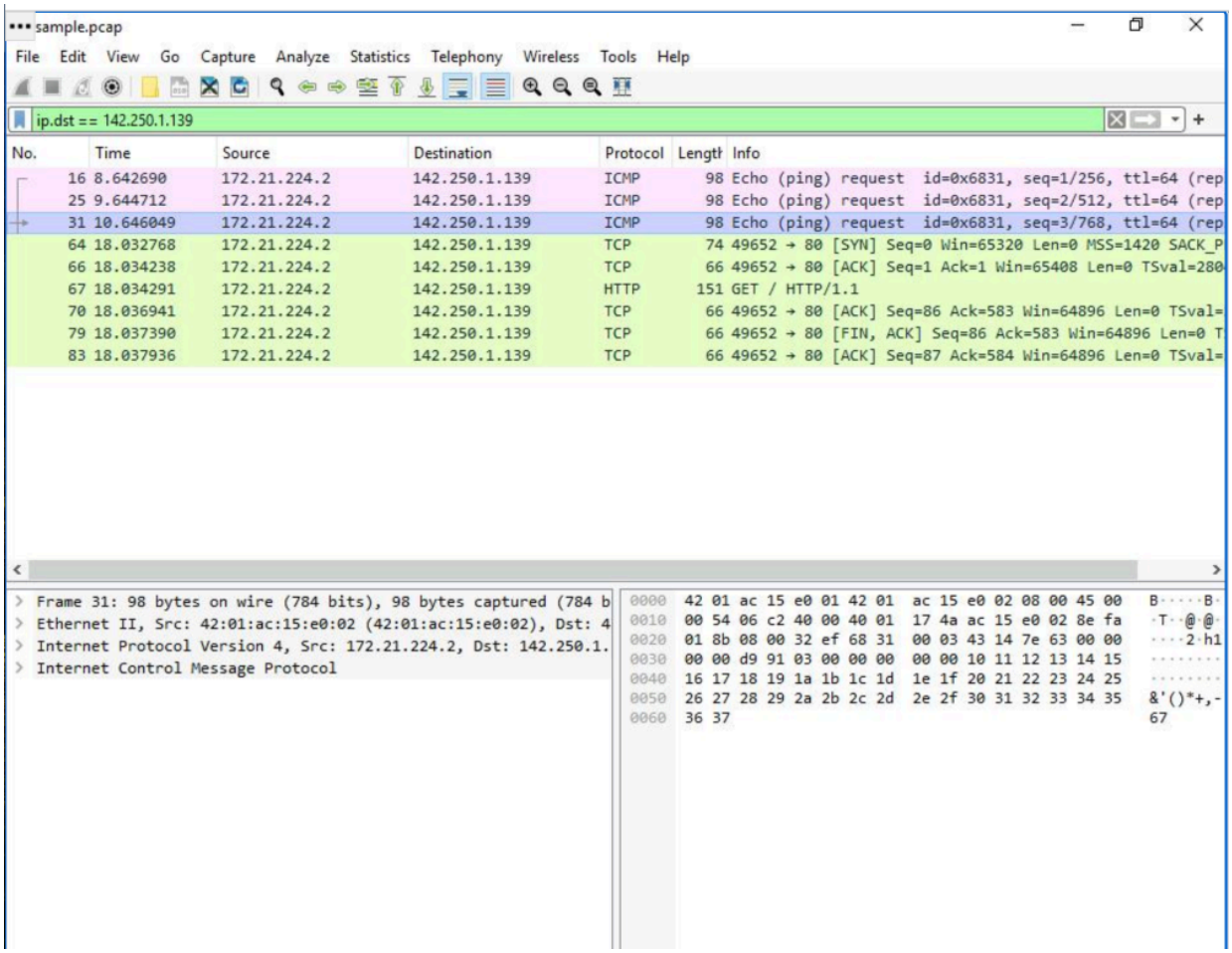




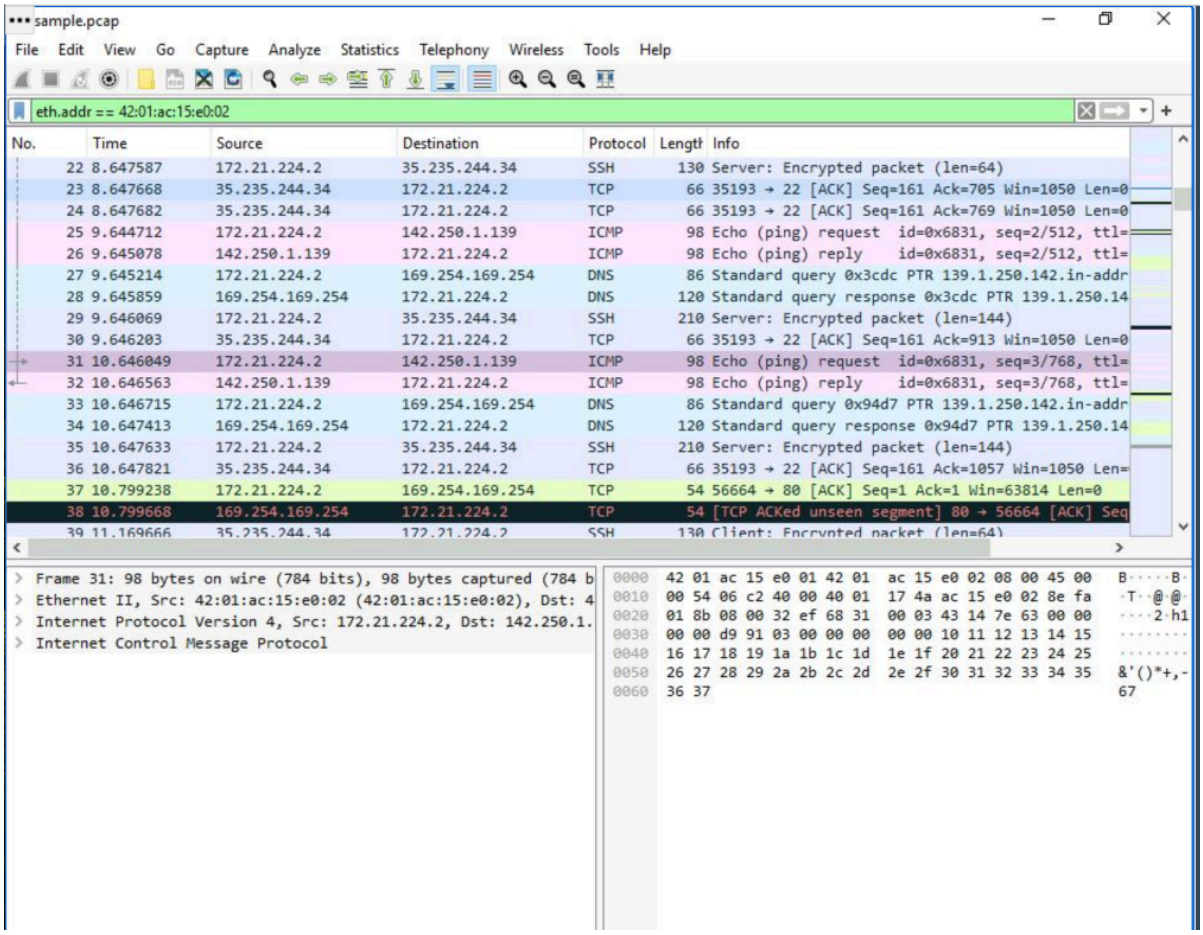
- On the title bar, type `ip.src == 142.250.1.139` to filter for traffic associated with a specific IP address. `src` means it is where the packet comes from.



- On the title bar, type `ip.dst == 142.250.1.139` to filter for traffic associated with a specific IP address. `dst` means it is where the packet goes to.

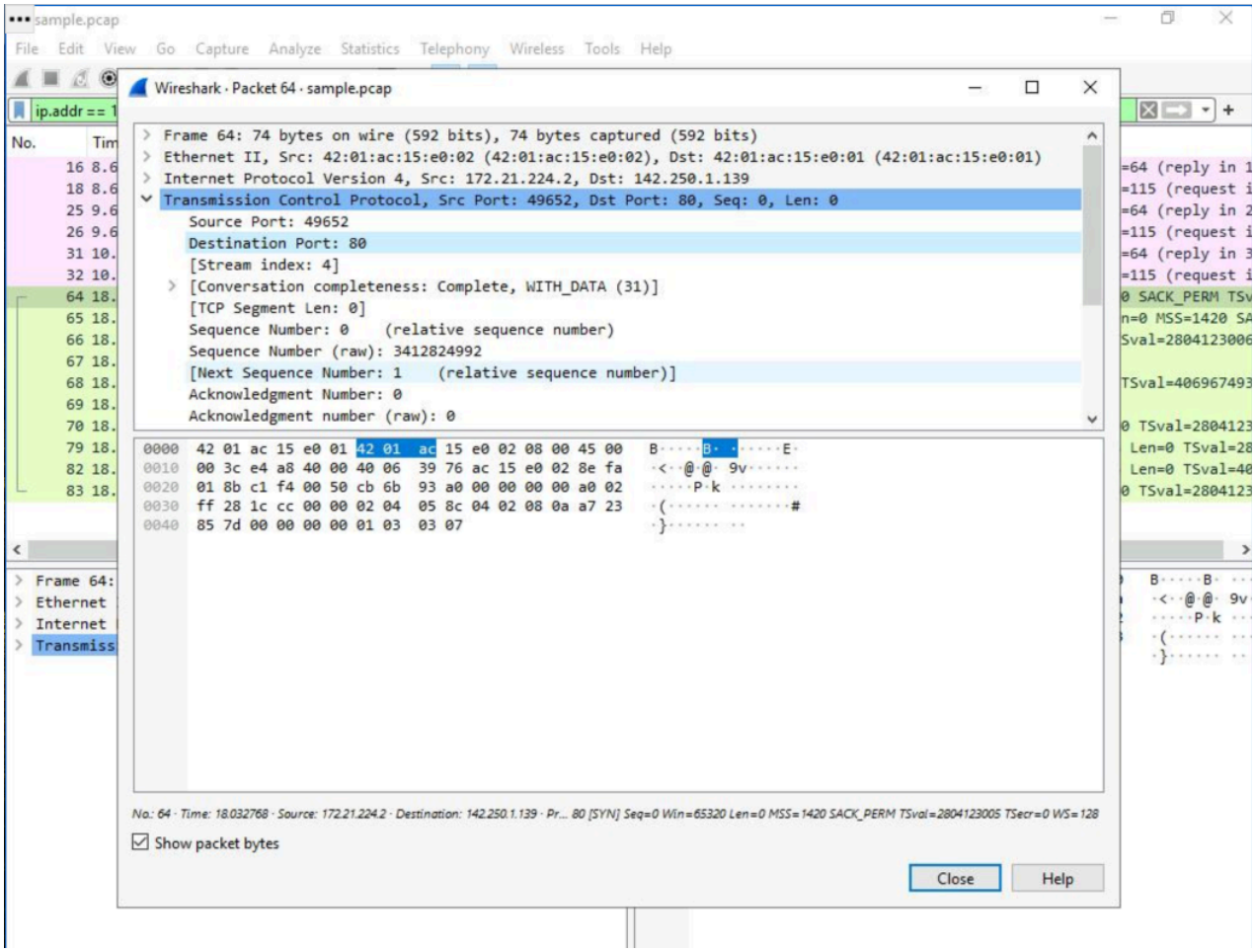


- On the title bar, type `eth.addr == 42:01:ac:15:e0:02` to filter for traffic associated with a specific Ethernet MAC address. `addr` means either the source or the destination IP.

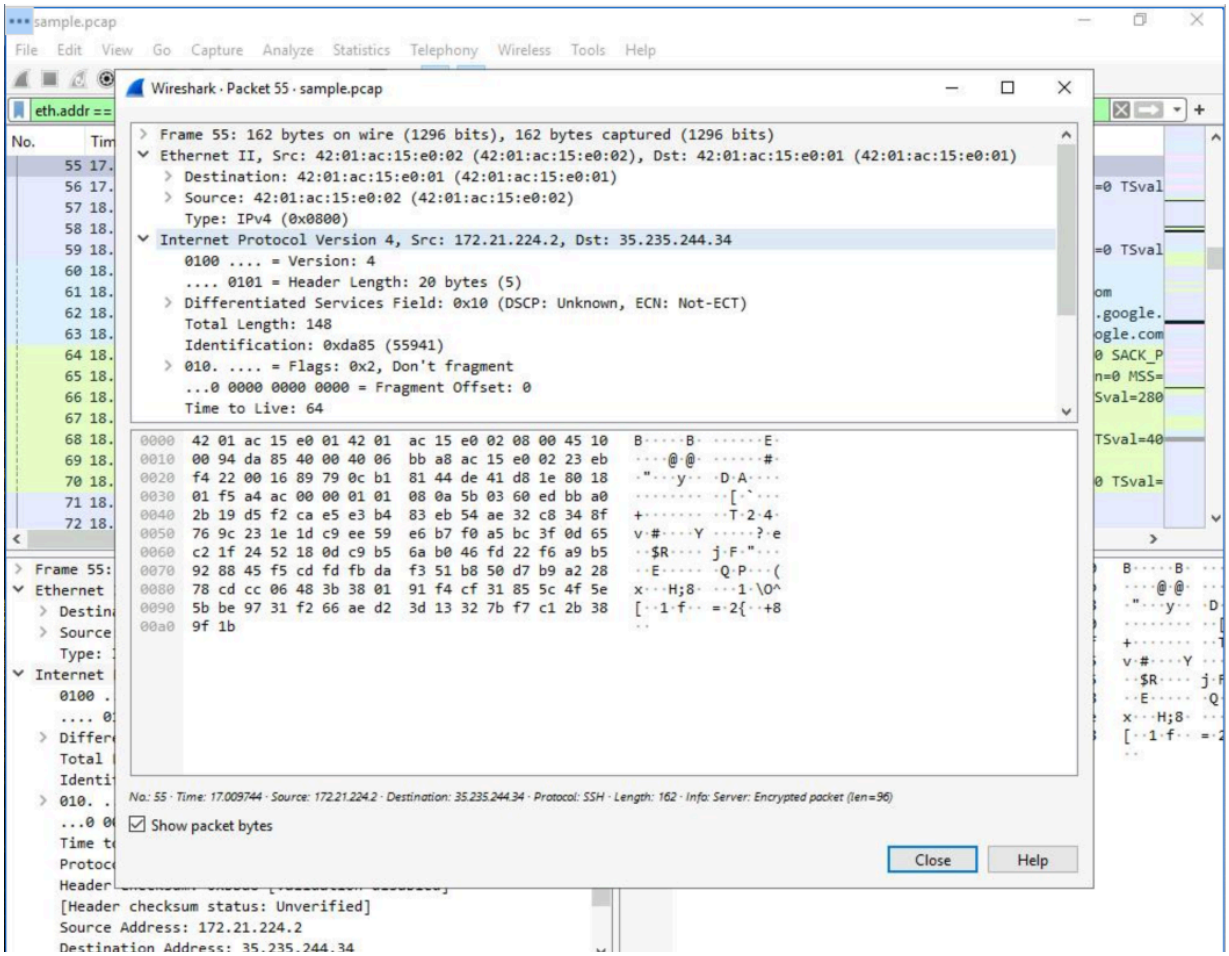


2. Examine the protocols that are used when the user makes the connection to the website.

- The TCP destination port of this TCP packet is 80 when `ip.addr == 142.250.1.139` which contains the initial web request to an HTPP website that will typically be listening on TCP port 80.

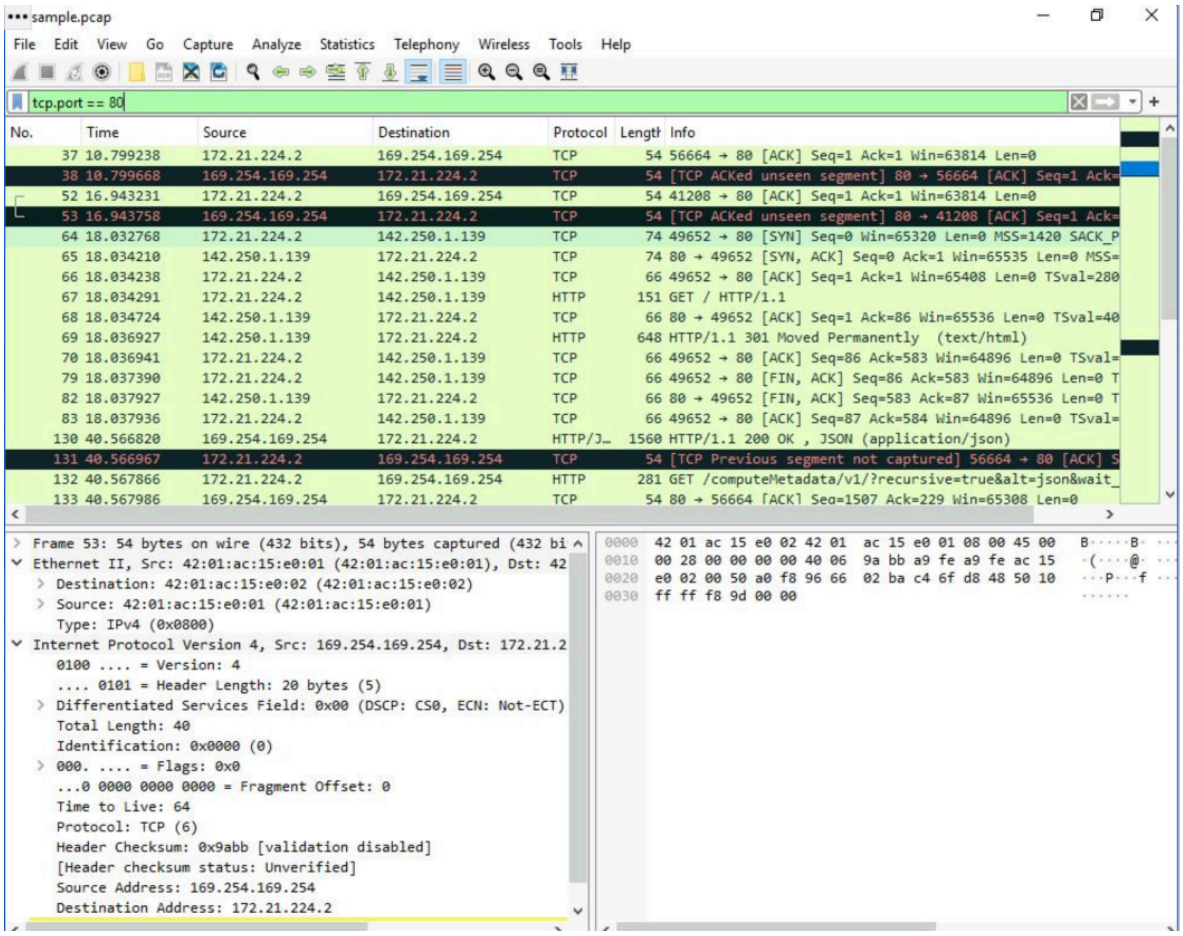


- The protocol destination port is TCP when Etherenet address was `42:01:ac:15:e0:02`. Source address is `172.21.224.2` and the destination address is `35.235.244.34`.

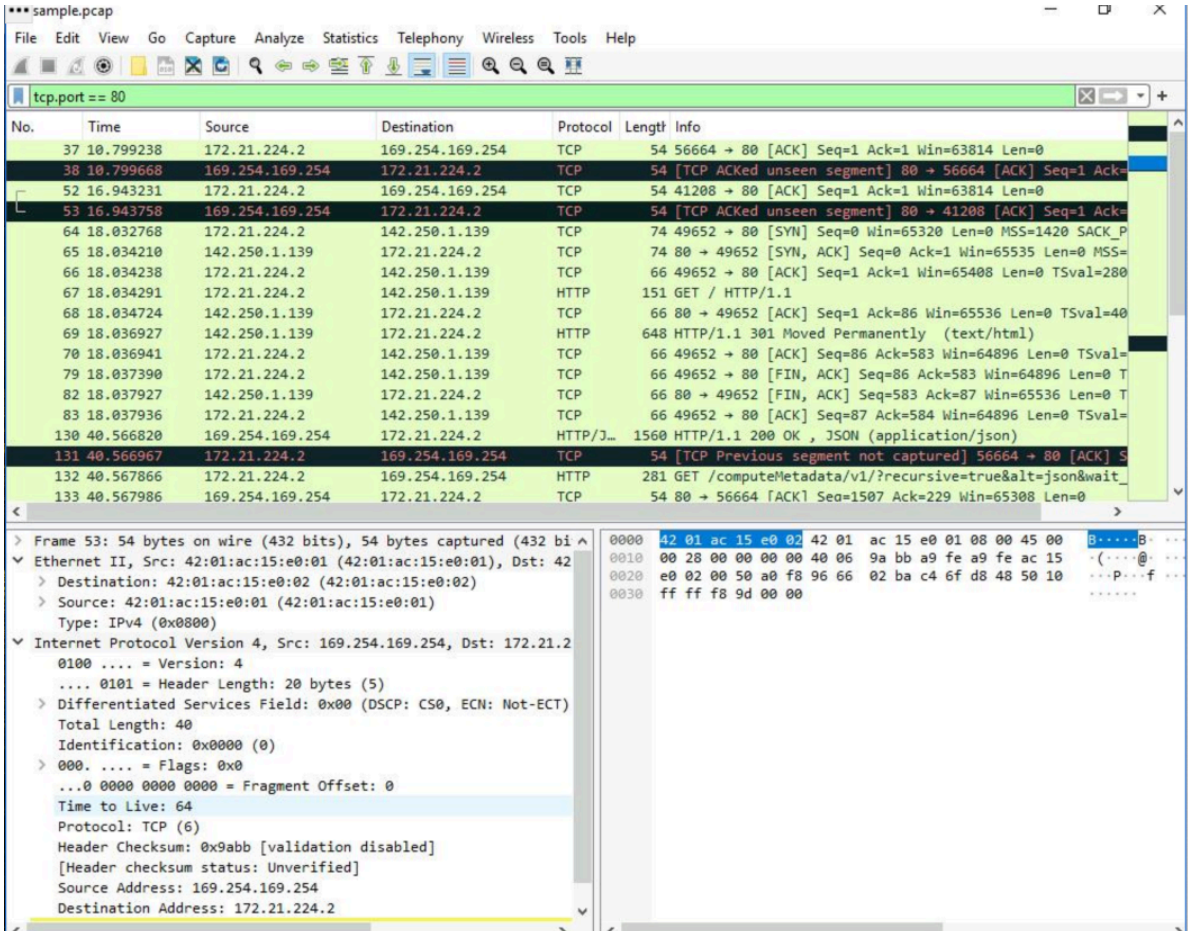


3. Analyze the data packet to identify the type of information sent and received by the systems that connect to each other when the network data is captured.

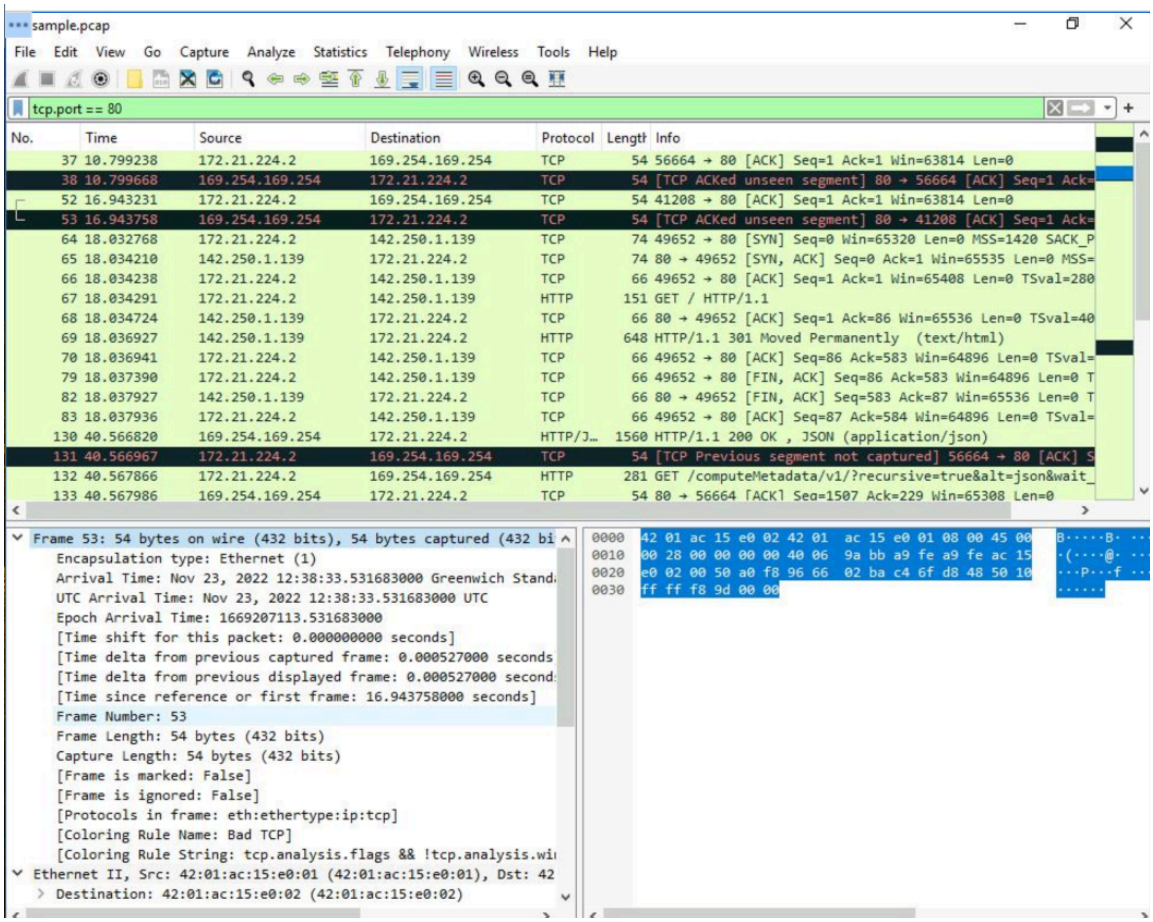
- On the title bar, type `tcp.port == 80` to filter for traffic associated with a specific port number. `tcp.port == 80` means only the tcp port is 80 will be shown.



- When the filter `tcp.port == 80` sets in play, the time to live is 64.
- `Time to Live`: A field in the Internet Protocol (IP) header that indicates the maximum amount of time an IP packet is allowed to exist in the network before it is discarded if it has not reached its destination. TTL is used to prevent packets from circulating indefinitely in the network, which could happen in the case of routing loops. It can be used as a basic security measure to limit how far packets can propagate through the network.



- When the filter `tcp.port == 80` sets in play, the Frame Number is 37 and Frame Length is 54 bytes.
- `Frame Number`: This is essentially the sequence number of a packet within a particular capture. It helps you identify and refer to packets more easily. In your case, a frame number of 53 means it's the 53rd packet captured since the beginning of the capture session. This number is assigned sequentially as packets are captured, starting with the number 1 for the first packet.
- `Frame Length`: This indicates the size of the packet, including all headers and payload, measured in bytes. The frame length of 54 bytes means the total size of the packet is 54 bytes. This size includes everything from the lowest layer (physical layer) up to the highest layer present in the packet that Wireshark can decode. It's useful for understanding the size of the data being transmitted and can help in various analyses, such as identifying potential issues with packet sizes that might indicate fragmentation or other problems.


sample.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 80
No. Time Source Destination Protocol Length Info
37 10.799238 172.21.224.2 169.254.169.254 TCP 54 56664 → 80 [ACK] Seq=1 Ack=1 Win=63814 Len=0
38 10.799668 169.254.169.254 172.21.224.2 TCP 54 [TCP ACKed unseen segment] 80 → 56664 [ACK] Seq=1 Ack=
52 16.943231 172.21.224.2 169.254.169.254 TCP 54 41208 → 80 [ACK] Seq=1 Ack=1 Win=63814 Len=0
53 16.943758 169.254.169.254 172.21.224.2 TCP 54 [TCP ACKed unseen segment] 80 → 41208 [ACK] Seq=1 Ack=
64 18.032768 172.21.224.2 142.250.1.139 TCP 74 49652 → 80 [SYN] Seq=0 Win=65320 Len=0 MSS=1420 SACK_P
65 18.034210 142.250.1.139 172.21.224.2 TCP 74 80 → 49652 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
66 18.034238 172.21.224.2 142.250.1.139 TCP 66 49652 → 80 [ACK] Seq=1 Ack=1 Win=65408 Len=0 TSval=280
67 18.034291 172.21.224.2 142.250.1.139 HTTP 151 GET / HTTP/1.1
68 18.034724 142.250.1.139 172.21.224.2 TCP 66 80 → 49652 [ACK] Seq=1 Ack=86 Win=65536 Len=0 TSval=40
69 18.036927 142.250.1.139 172.21.224.2 HTTP 648 HTTP/1.1 301 Moved Permanently (text/html)
70 18.036941 172.21.224.2 142.250.1.139 TCP 66 49652 → 80 [ACK] Seq=86 Ack=583 Win=64896 Len=0 TSval=
79 18.037390 172.21.224.2 142.250.1.139 TCP 66 49652 → 80 [FIN, ACK] Seq=86 Ack=583 Win=64896 Len=0 T
82 18.037927 142.250.1.139 172.21.224.2 TCP 66 80 → 49652 [FIN, ACK] Seq=583 Ack=87 Win=65536 Len=0 T
83 18.037936 172.21.224.2 142.250.1.139 TCP 66 49652 → 80 [ACK] Seq=87 Ack=584 Win=64896 Len=0 TSval=
130 40.566820 169.254.169.254 172.21.224.2 HTTP/J… 1560 HTTP/1.1 200 OK , JSON (application/json)
131 40.566967 172.21.224.2 169.254.169.254 TCP 54 [TCP Previous segment not captured] 56664 → 80 [ACK] S
132 40.567866 172.21.224.2 169.254.169.254 HTTP 281 GET /computeMetadata/v1/?recursive=true&alt=json&wait_
133 40.567986 169.254.169.254 172.21.224.2 TCP 54 80 → 56664 [ACK] Seq=1507 Ack=229 Win=65308 Len=0
Frame 53: 54 bytes on wire (432 bits), 54 bytes captured (432 bi
Encapsulation type: Ethernet (1)
Arrival Time: Nov 23, 2022 12:38:33.531683000 Greenwich Stand
UTC Arrival Time: Nov 23, 2022 12:38:33.531683000 UTC
Epoch Arrival Time: 1669207113.531683000
[Time shift for this packet: 0.000000000 seconds]
[Time delta from previous captured frame: 0.000527000 seconds
[Time delta from previous displayed frame: 0.000527000 second
[Time since reference or first frame: 16.943758000 seconds]
Frame Number: 53
Frame Length: 54 bytes (432 bits)
Capture Length: 54 bytes (432 bits)
[Frame is marked: False]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:tcp]
[Coloring Rule Name: Bad TCP]
[Coloring Rule String: tcp.analysis.flags && !tcp.analysis.wi
Ethernet II, Src: 42:01:ac:15:e0:01 (42:01:ac:15:e0:01), Dst: 42
Destination: 42:01:ac:15:e0:02 (42:01:ac:15:e0:02)
0000 42 01 ac 15 e0 02 42 01 ac 15 e0 01 08 00 45 00
0010 00 28 00 00 00 00 40 06 9a bb a9 fe a9 fe ac 15
0020 e0 02 00 50 a0 f8 96 66 02 ba c4 6f d8 48 50 10
0030 ff ff f8 9d 00 00