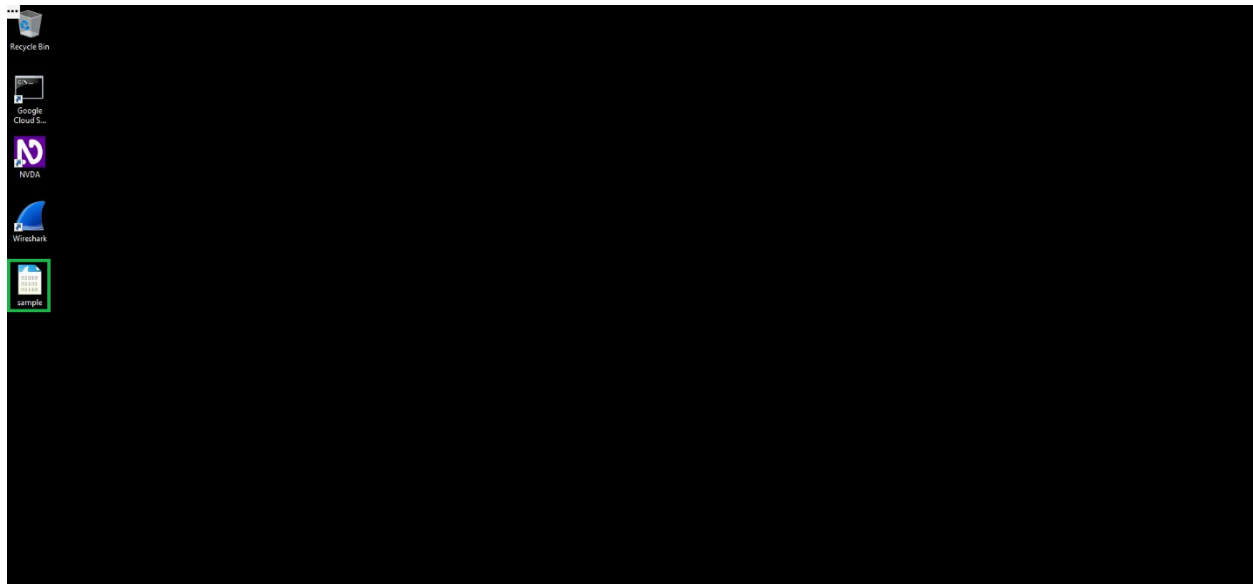


Packet Analysis using Wireshark

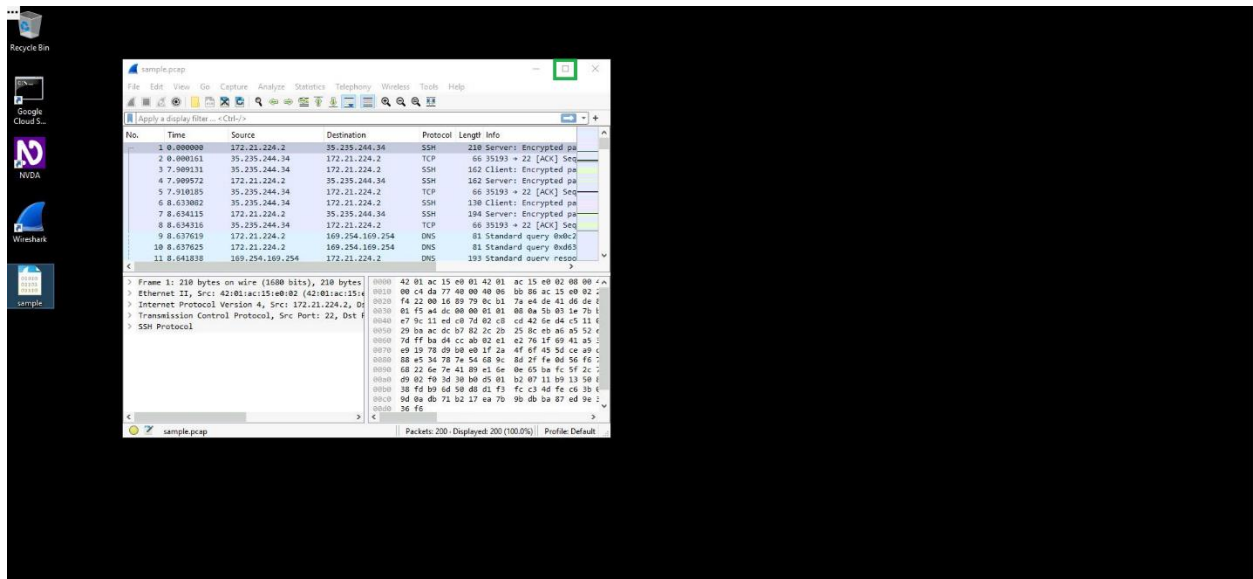
In this project I boot up a Windows VM which has Wireshark installed and go through the various ways that internet traffic can be analyzed using this application. This packet analyzer is particularly user-friendly because, unlike Tcpdump, it has a graphical user interface (GUI).

Exploring data with Wireshark

After booting up the Windows VM, We click on the `.pcap` sample packet capture file that that has been prepared on the desktop.



This then opens up the Wireshark application window which can be maximized for a better experience.



The key property columns listed for each packet can be examined:

- **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 information about the data in the packet (the payload) as interpreted by Wireshark

Packets are also color-coded based on protocol type. These coloring rules are customizable and changes can be both temporary (valid until the application is closed) or permanent.

Applying Wireshark filters and inspecting packets

The first filter that could be applied to the data is *ip.addr ==*. In this example we will type “ip.addr == 142.250.1.139” into the “**Apply a display filter ...**” text box.

The screenshot shows the Wireshark interface with the filter `ip.addr == 142.250.1.139` applied. The packet list displays the following packets:

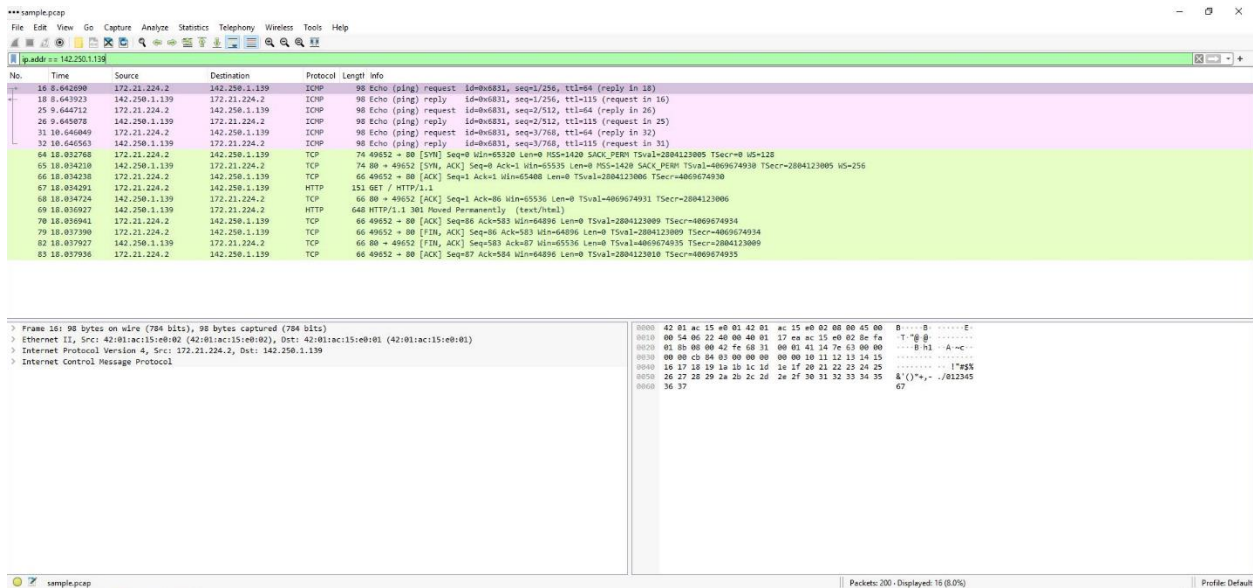
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.21.224.2	35.235.244.34	SSH	210	Server: Encrypted packet (len=144)
2	0.000151	35.235.244.34	172.21.224.2	TCP	66	35193 → 22 [ACK] Seq=101 Win=1050 Len=0 TSval=3147819177 TSecr=1526931867
3	7.900131	35.235.244.34	172.21.224.2	SSH	162	Client: Encrypted packet (len=96)
4	7.900972	172.21.224.2	35.235.244.34	SSH	162	Server: Encrypted packet (len=96)
5	7.910805	35.235.244.34	172.21.224.2	TCP	66	35193 → 22 [ACK] Seq=97 Win=1050 Len=0 TSval=3147827087 TSecr=1526938976
6	8.633882	35.235.244.34	172.21.224.2	SSH	130	Client: Encrypted packet (len=64)
7	8.634115	172.21.224.2	35.235.244.34	SSH	194	Server: Encrypted packet (len=128)
8	8.634316	35.235.244.34	172.21.224.2	TCP	66	35193 → 22 [ACK] Seq=101 Win=1050 Len=0 TSval=3147827811 TSecr=1526939781
9	8.637619	172.21.224.2	169.254.169.254	DNS	81	Standard query 0xd638 AAAA opensource.google.com
10	8.637625	172.21.224.2	169.254.169.254	DNS	81	Standard query 0xd638 AAAA opensource.google.com
11	8.641838	169.254.169.254	172.21.224.2	DNS	193	Standard query response 0xb530 AAAA opensource.google.com AAAA 2087:fb0b:4001:c24:64 AAAA 2087:fb0b:4001:c24:71 AAAA 2087:fb0b:4001:c24:165
12	8.641878	169.254.169.254	172.21.224.2	DNS	177	Standard query response 0xb520 A opensource.google.com A 142.250.1.139 A 142.250.1.135 A 142.250.1.102 A 142.250.1.115 A 142.250.1.180 A 142.250.1.181
13	8.642416	172.21.224.2	35.235.244.34	SSH	194	Server: Encrypted packet (len=128)
14	8.642508	172.21.224.2	35.235.244.34	SSH	130	Server: Encrypted packet (len=64)
15	8.642588	35.235.244.34	172.21.224.2	TCP	66	35193 → 22 [ACK] Seq=101 Win=1050 Len=0 TSval=3147827819 TSecr=1526939789
16	8.642690	172.21.224.2	142.250.1.139	ICMP	98	Echo (ping) request 1d=0xb831, seq=1/250, ttl=64 (reply in 18)
17	8.642735	35.235.244.34	172.21.224.2	TCP	66	35193 → 22 [ACK] Seq=101 Win=1050 Len=0 TSval=3147827819 TSecr=1526939789
18	8.643923	142.250.1.139	172.21.224.2	ICMP	98	Echo (ping) reply 1d=0xb831, seq=1/250, ttl=115 (request in 16)
19	8.644093	172.21.224.2	169.254.169.254	DNS	86	Standard query 0xb549 PTR 139.1.250.142.in-addr.arpa
20	8.647339	169.254.169.254	172.21.224.2	DNS	120	Standard query response 0xb549 PTR 139.1.250.142.in-addr.arpa PTR jw-in-f139.1e180.net
21	8.647314	172.21.224.2	35.235.244.34	SSH	210	Server: Encrypted packet (len=144)

The packet details pane for the selected packet (No. 1) shows the following structure:

- Frame 1: 210 bytes on wire (1680 bits), 210 bytes captured (1680 bits) on interface 0
- Ethernet II, Src: 42:01:ac:15:e0:02 (42:01:ac:15:e0:02), Dst: 42:01:ac:15:e0:01 (42:01:ac:15:e0:01)
- Internet Protocol Version 4, Src: 172.21.224.2, Dst: 35.235.244.34
- Transmission Control Protocol, Src Port: 22, Dst Port: 35193, Seq: 1, Ack: 1, Len: 144
- SSH Protocol

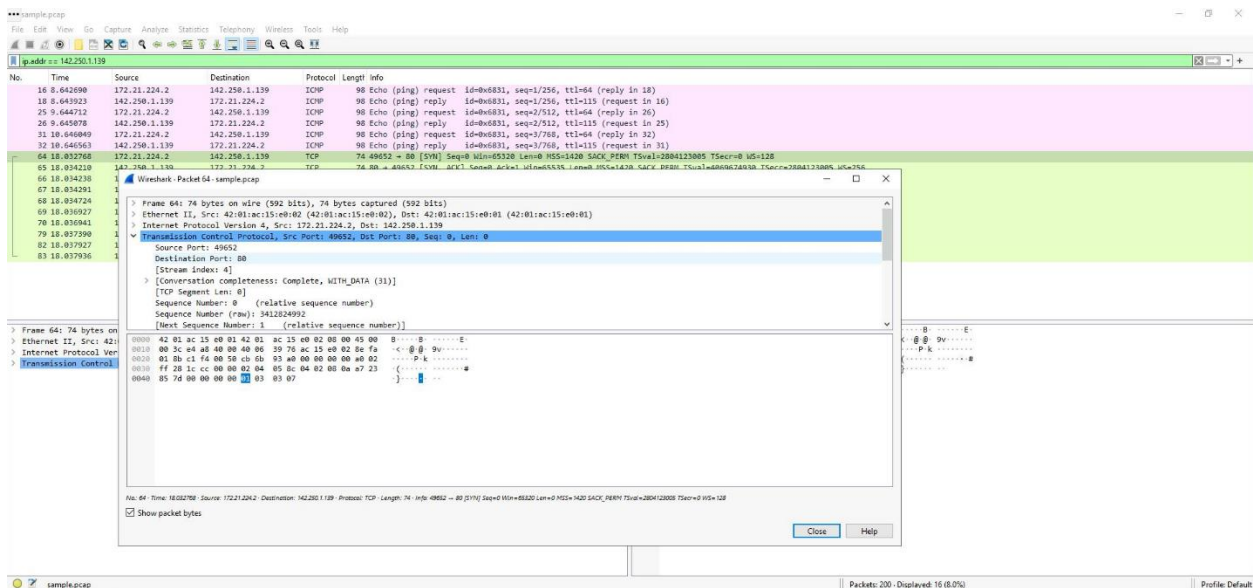
The packet bytes pane shows the raw data in hexadecimal and ASCII.

This results in a list of only the packets which have the same source or destination IP address as the one that was entered after the double equal sign.



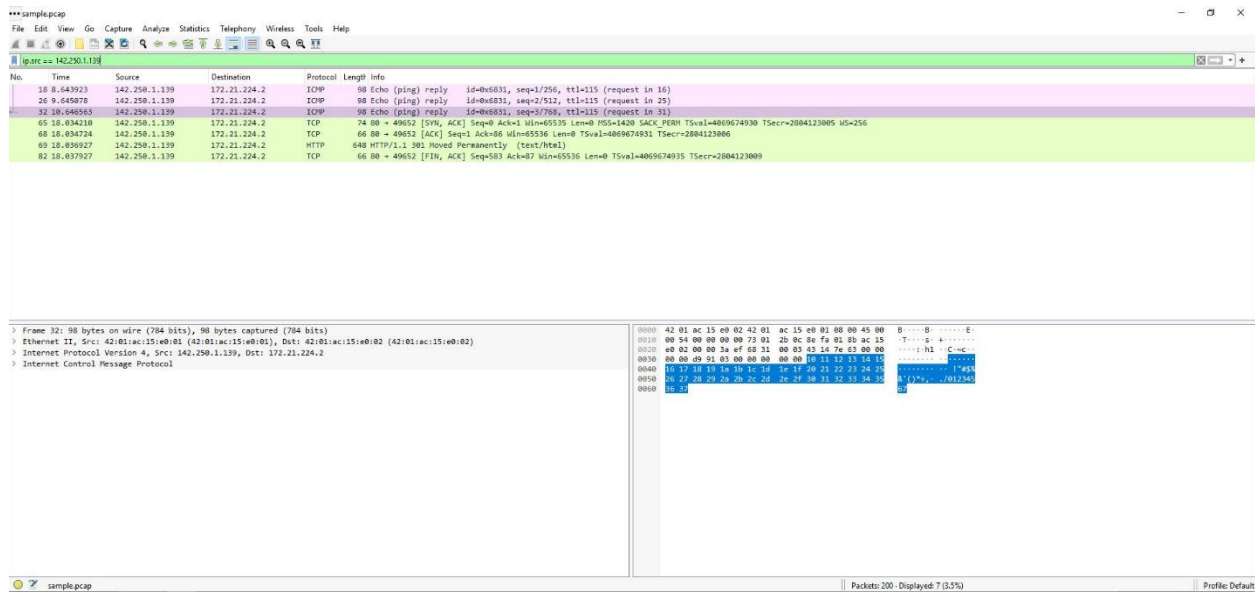
Now only ICMP and TCP traffic is visible (HTTP runs over TCP, which is why it also shows up and is under the same color rule).

If, for example, the first TCP packet is selected and double-clicked, the TCP destination port of the packet can be seen under the **Transmission Control Protocol** subtree. In this case the destination port is 80.

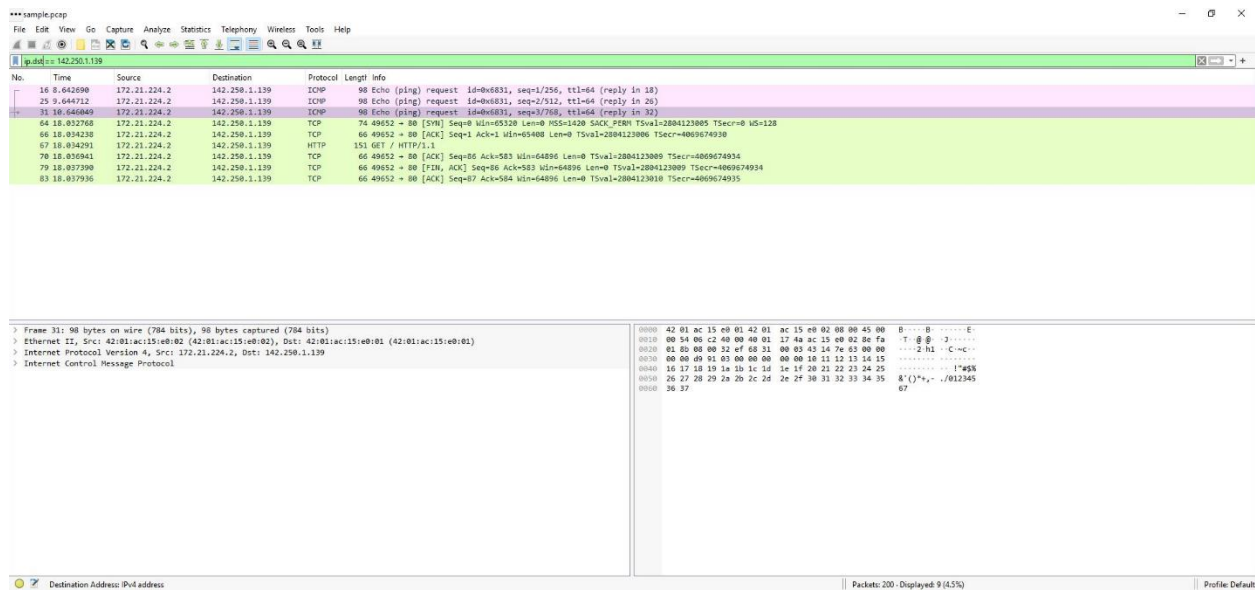


Selecting packets

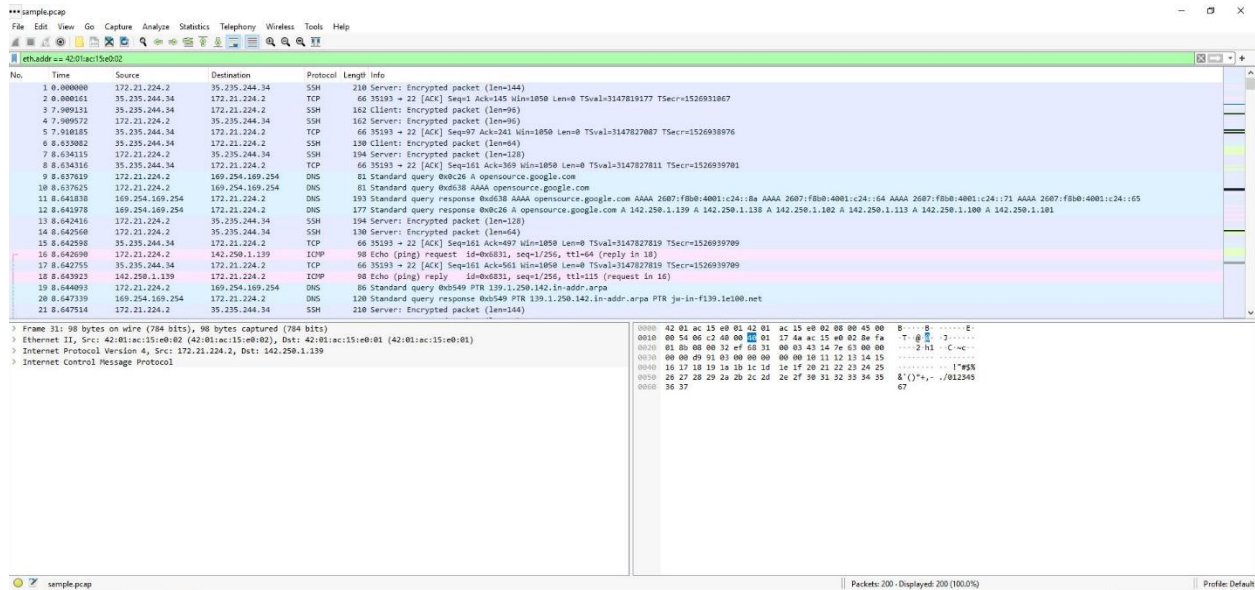
For more granular search results we could specifically filter by source or destination IP address. Applying the “ip.src == 142.250.1.139” display filter returns only entries for packets that came from the IP 142.250.1.139.



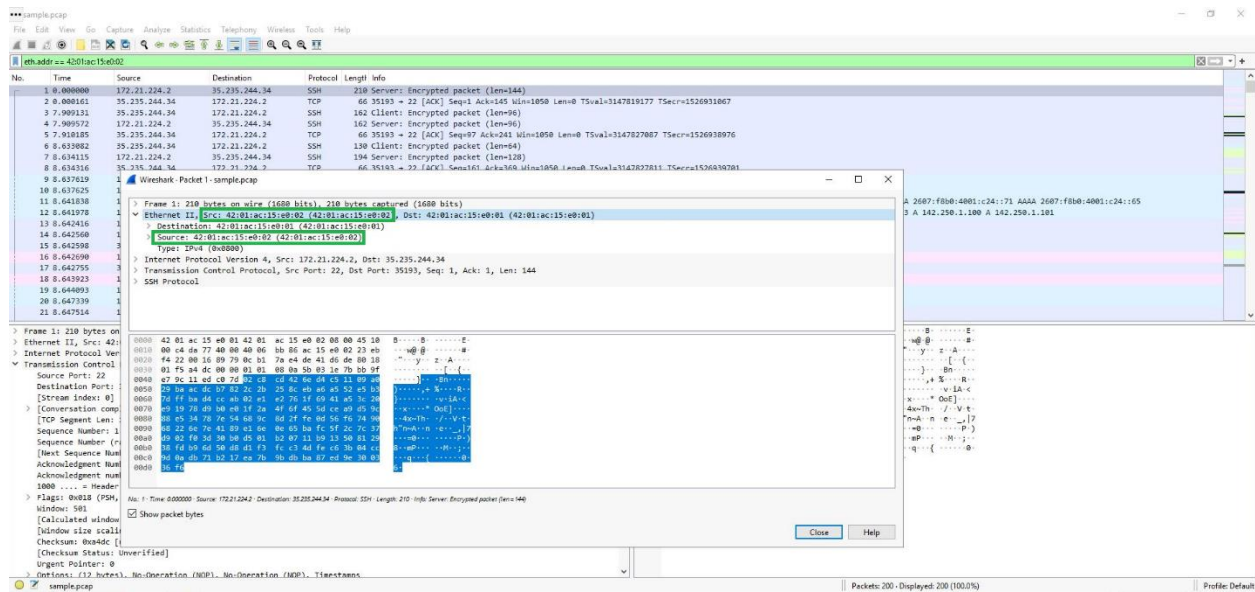
Likewise, applying the “`ip.dst == 142.250.1.139`” display filter will return only entries for packets that were sent to the IP 142.250.1.139.



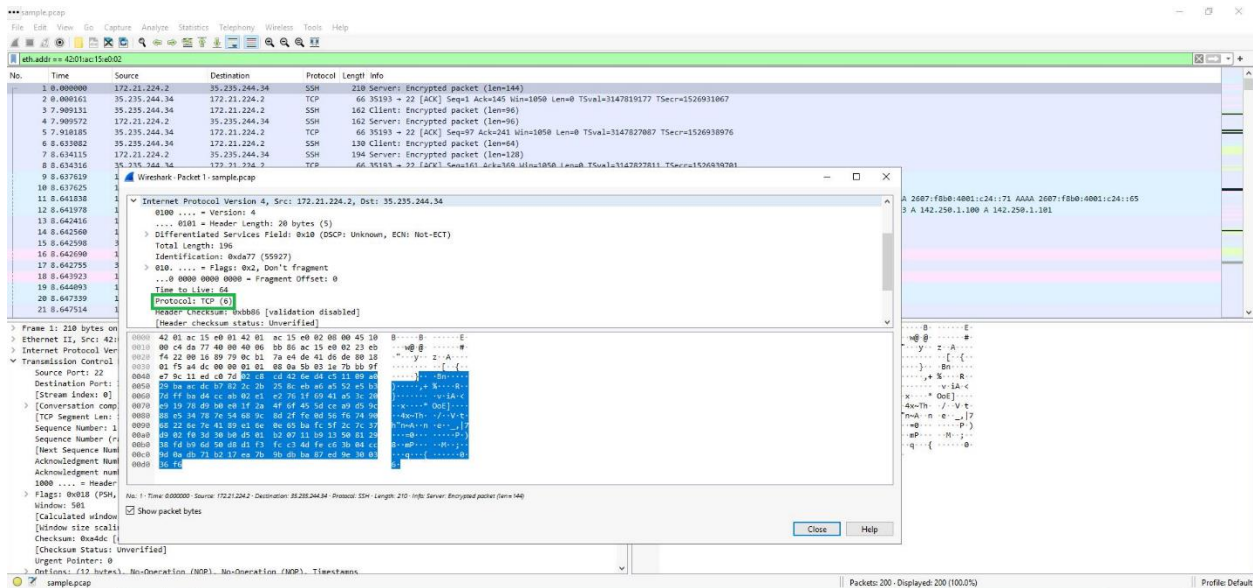
Results can also be filtered based a specific MAC address. Typing “`eth.addr == 42:01:ac:15:e0:02`” into the search box will result in all traffic related to this MAC address, regardless of the protocols which are involved.



Here, if we open the first packet in the results and expand the **Ethernet II** subtree, we can see that the specified MAC address is the source of the packet.



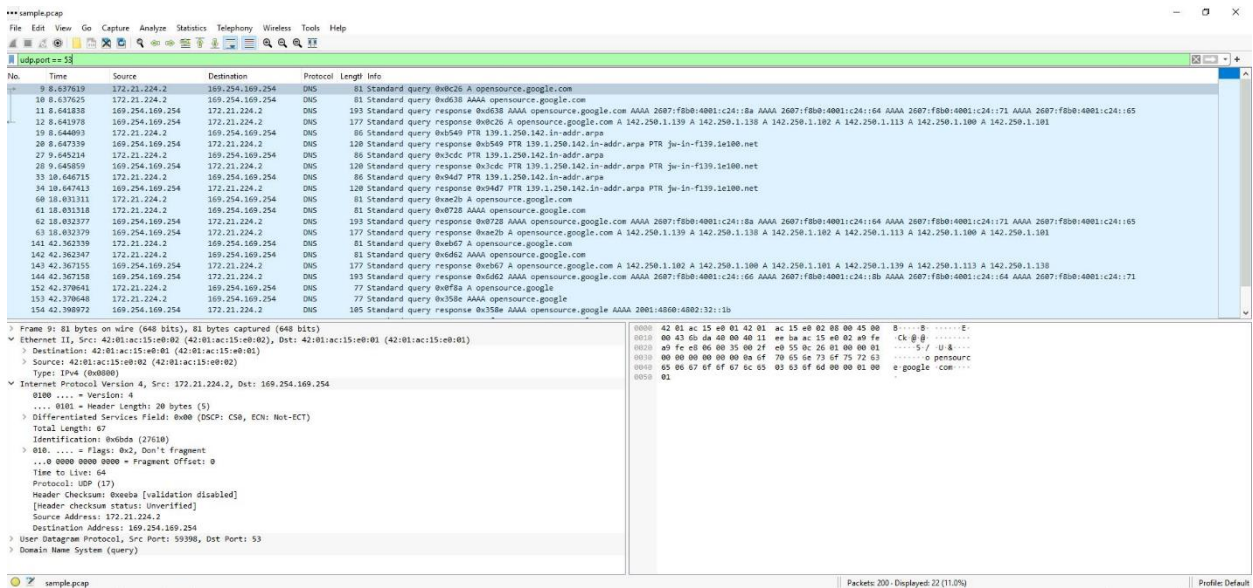
Additionally, expanding the **Internet Protocol Version 4** subtree reveals the internal protocol contained in the packet.



Exploring DNS packets

DNS traffic can also be examined with Wireshark. In this part of the project, I will demonstrate how to find DNS packet data such as queries (names of websites that are being looked up) and answers (IP addresses that are being sent back by a DNS server when a name is successfully resolved).

Since DNS traffic is over UDP port 53, the filter that needs to be applied here will be “udp.port == 53”. This lists only DNS queries and responses.



If the first packet is opened, under the **Domain Name System (query)** subtree, if **Queries** is clicked on we see that the name of the website which queries is **opensource.google.com**.

Wireshark · Packet 9 · sample.pcap

Domain Name System (query)

- Transaction ID: 0x0c26
- Flags: 0x0100 Standard query
- Questions: 1
- Answer RRs: 0
- Authority RRs: 0
- Additional RRs: 0
- Queries
 - opensource.google.com: type A, class IN
 - [\[Response In: 12\]](#)

0000	42 01 ac 15 e0 01 42 01 ac 15 e0 02 08 00 45 00	B B E .
0010	00 43 6b da 40 00 40 11 ee ba ac 15 e0 02 a9 fe	. Ck . @ . @
0020	a9 fe e8 06 00 35 00 2f e0 55 0c 26 01 00 00 01 5 . / . U . &
0030	00 00 00 00 00 00 0a 6f 70 65 6e 73 6f 75 72 63 o p e n s o u r c
0040	65 06 67 6f 6f 67 6c 65 03 63 6f 6d 00 00 01 00	e . g o o g l e . c o m
0050	01	.

No.: 9 · Time: 8.637619 · Source: 172.21.224.2 · Destination: 169.254.1... DNS · Length: 81 · Info: Standard query 0x0c26 A opensource.google.com

☒ Show packet bytes

Close Help

If the same is done when examining the 4th packet in the list and we scroll down and double-click on **Answers** under the **Domain Name System (query)**, we can see the name that was queried (**opensource.google.com**) and the addresses that are associated with that name.

Wireshark · Packet 12 · sample.pcap

Type: A (1) (Host Address)
Class: IN (0x0001)

▼ Answers

- > opensource.google.com: type A, class IN, addr 142.250.1.139
- > opensource.google.com: type A, class IN, addr 142.250.1.138
- > opensource.google.com: type A, class IN, addr 142.250.1.102
- > opensource.google.com: type A, class IN, addr 142.250.1.113
- > opensource.google.com: type A, class IN, addr 142.250.1.100
- > opensource.google.com: type A, class IN, addr 142.250.1.101

[\[Request In: 9\]](#)
[Time: 0.004359000 seconds]

0000	42 01 ac 15 e0 02 42 01	ac 15 e0 01 08 00 45 00	B B E .
0010	00 a3 00 00 00 00 40 11	9a 35 a9 fe a9 fe ac 15 @ . 5
0020	e0 02 00 35 e8 06 00 8f	b3 b3 0c 26 81 80 00 01	. . . 5 &
0030	00 06 00 00 00 00 0a 6f	70 65 6e 73 6f 75 72 63 o p e n s o u r c e
0040	65 06 67 6f 6f 67 6c 65	03 63 6f 6d 00 00 01 00	e . g o o g l e . c o m
0050	01 c0 0c 00 01 00 01 00	00 01 2c 00 04 8e fa 01 ,
0060	8b c0 0c 00 01 00 01 00	00 01 2c 00 04 8e fa 01 ,
0070	8a c0 0c 00 01 00 01 00	00 01 2c 00 04 8e fa 01 ,
0080	66 c0 0c 00 01 00 01 00	00 01 2c 00 04 8e fa 01	f ,
0090	71 c0 0c 00 01 00 01 00	00 01 2c 00 04 8e fa 01	q ,
00a0	64 c0 0c 00 01 00 01 00	00 01 2c 00 04 8e fa 01	d ,
00b0	65		e

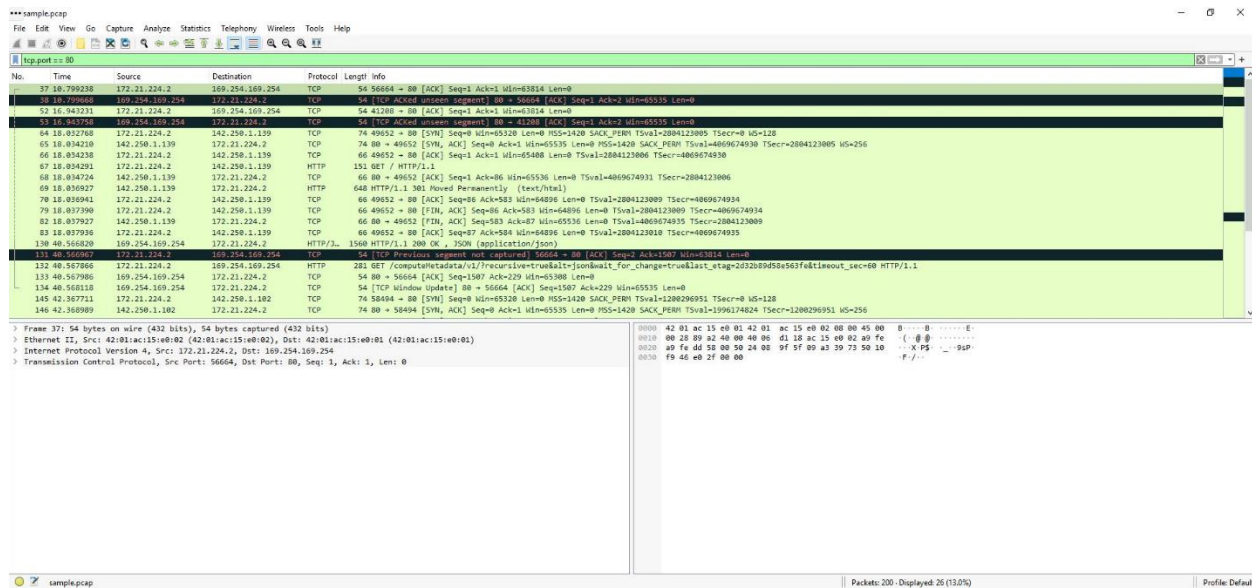
No: 12 · Time: 8.641978 · Source: 169.254.169.254 · Destination: 172.21....250.1.138 A 142.250.1.102 A 142.250.1.113 A 142.250.1.100 A 142.250.1.101

☒ Show packet bytes

Close Help

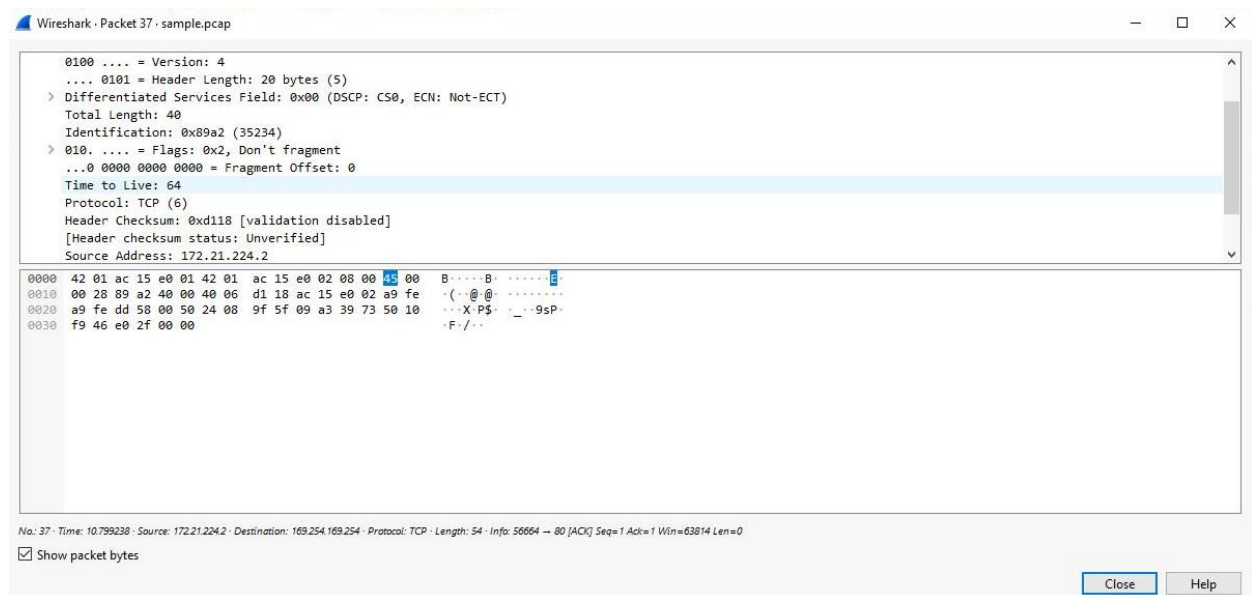
Exploring TCP packets

To examine TCP traffic, we start by applying the display filter “tcp.port == 80”.

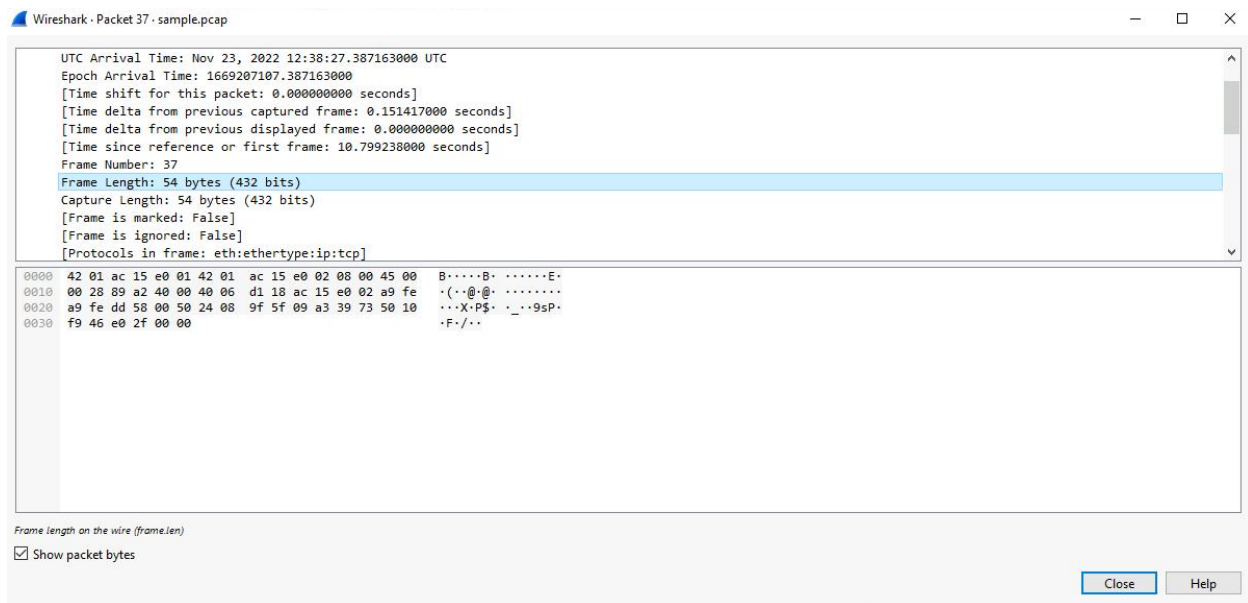


After clicking on the first packet with a destination IP of **169.254.169.254**, we can find various information about the packet such as:

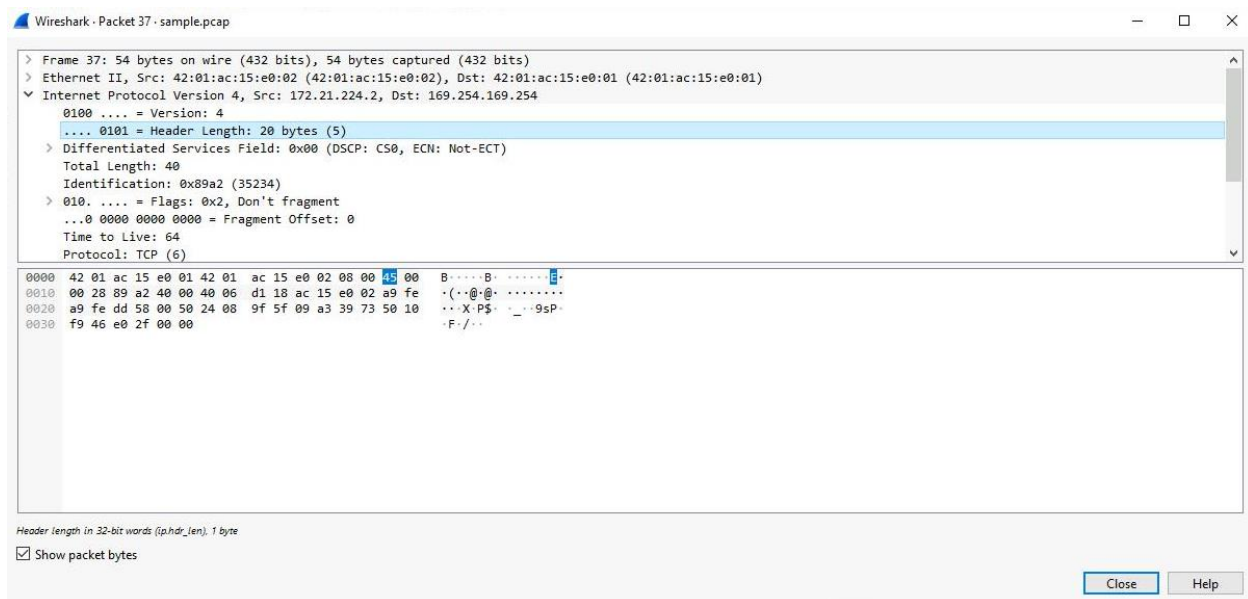
1. The **Time to Live** value which can be found under the **Internet Protocol Version 4** subtree



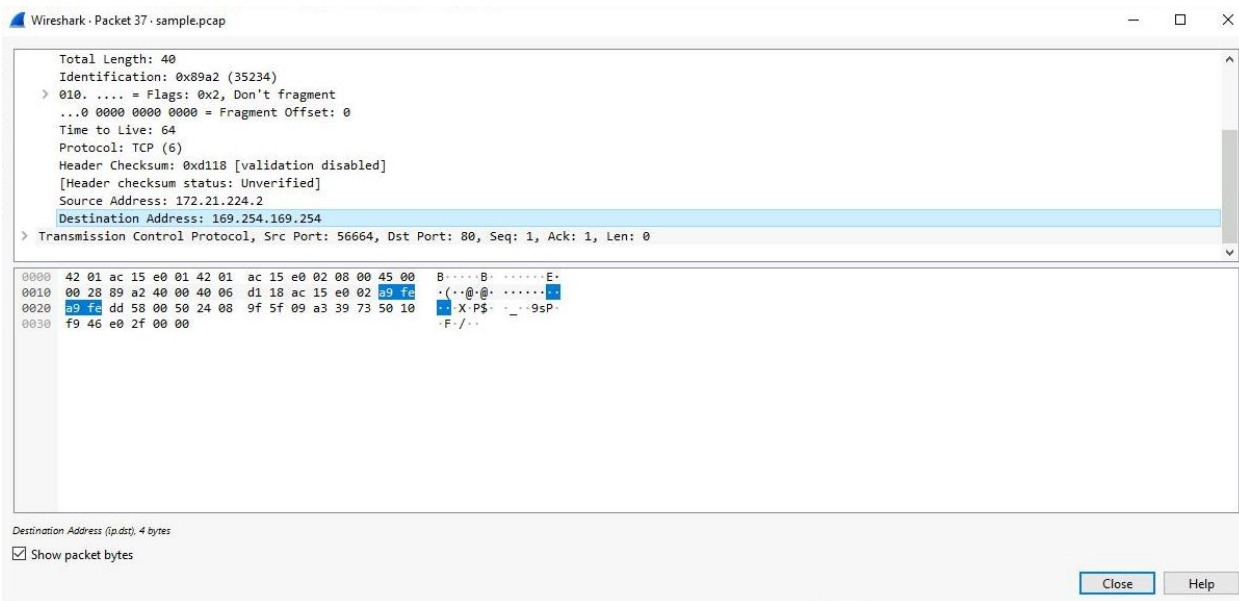
2. The **Frame Length**, found under the **Frame** subtree



3. The **Header Length**, found under the **Internet Protocol Version 4** subtree



4. We can see the Destination IP address here as well by looking under the **Internet Protocol Version 4** subtree



Wireshark also allows us to filter results based on specific text data. This can be done by typing “tcp contains” followed by the word we are searching for written within quotation marks. In this example, we are looking for web requests made with the “curl” command.

