

# Networks and Internet Applications

## Practical Exercise 1 – PR1

Nicolas D'Alessandro Calderon

### Practical exercise 1: Watch the network and you will find out what is going on there

In this practice, we will analyze some protocols that are usually present in everyday communications we make using computer networks. For this reason, it will be very useful to install a packet analyzer, which, visually, will allow us to understand what is happening at a glance. Analyzers are programs that capture packets that send and receive our computer's network card. In section **Annex: Wireshark preliminary guidelines** you have instructions for installing and testing how the packet analyzer that we will use in the subject: Wireshark, works.

Once you have installed the tool required to analyze what happens in the network, we need a common study scenario for us: Internet browsing. In order to solve the following exercises, you should run Wireshark and start capturing packets. Once you have done this, you have to open a browser and go to the following web site:

<http://www.edu4java.com/es/web/web1.html>

At this point, you can stop capturing Wireshark packets and save the capture into a file to follow with the work afterwards. To use it, you just have to open Wireshark and open the capture file stored in disk.

### Qualification

This practice has four parts that should be done in sequential order (first part 1, then part 2, ...). The final qualification depends on the delivered parts:

- Part 1. Link and network layers - Maximum qualification C-*
- Part 2. Transport layer - Maximum qualification C+*
- Part 3. Application layer - Maximum qualification B*
- Part 4. Network security - Maximum qualification A*

In order to obtain the aforementioned qualification, you have to do all exercises of the indicated part. If there is any exercise or section not done or incomplete, it will mean not obtaining the corresponding qualification.

**Note:** Upload your responses in the subject's classroom, **both the pdf file with the responses to the questions of all parts and the file captured by Wireshark**. Please do not use any compression program to submit in a single file. Name the files with your full name.

# Index

First part (maximum qualification: C-): Link and network layers .....	3
First Part Answers .....	4
Exercise 1 .....	4
Exercise 2 .....	5
Exercise 3 .....	8
Second part (maximum qualification: C+): Transport layer .....	9
Second Part Answers .....	10
Exercise 1 .....	10
Exercise 2 .....	12
Exercise 3 .....	13
Third part (maximum qualification: B): Application layer .....	15
Third Part Answers .....	16
Exercise 1 .....	16
Exercise 2 .....	19
Exercise 3 .....	20
Exercise 4 .....	23
Fourth part (maximum qualification: A): Network security .....	25
Fourth Part Answers .....	26
Exercise 1 .....	26
Exercise 2 .....	28
Exercise 3 .....	29
Exercise 4 .....	31
Exercise 5 .....	32

## First part (maximum qualification: C-): Link and network layers

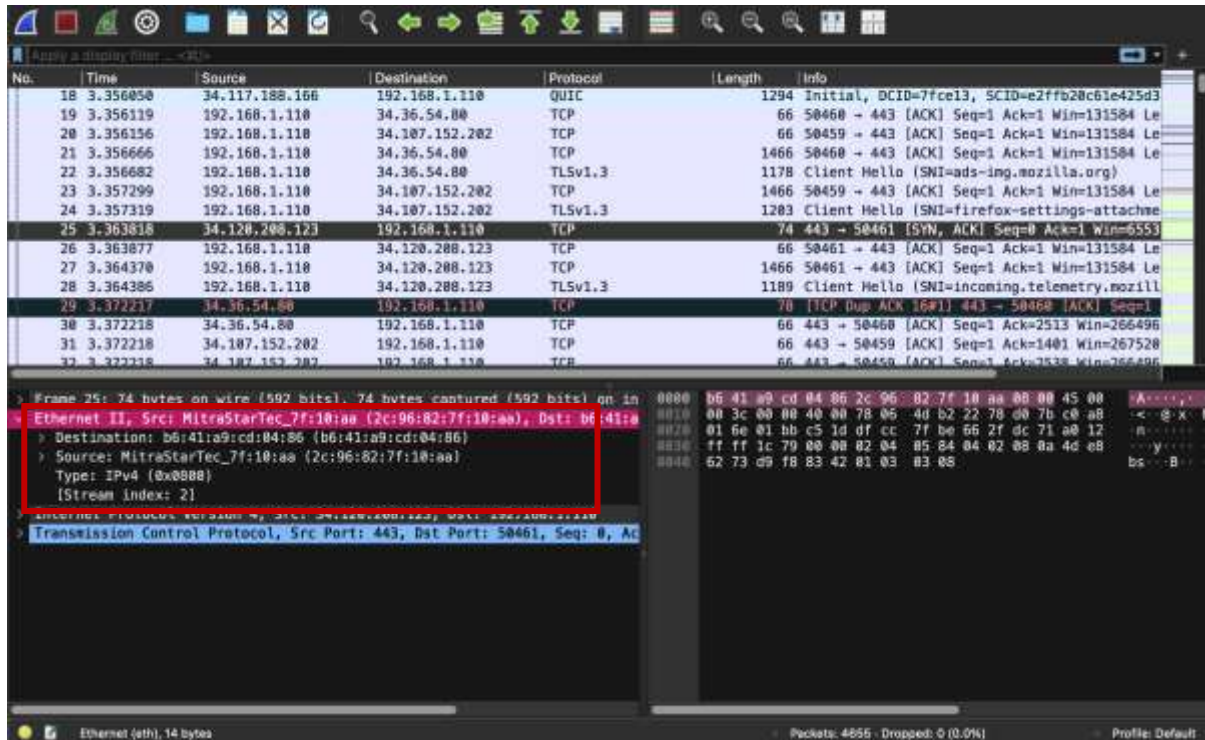
Internet protocols stack is structured in several layers. In this first part, you have to work with the lowest levels and related to the network cards: Link and network layers. We work with MAC and IP addresses, identifying the network card and the device, respectively.

As support material to make this part of the practice, we propose you the following sections from the Computer Networking book (8th edition), especially sections concerning protocol headers:

- **6.4** Switched Local Area Networks pp. 516-517
  - **6.4.2** Ethernet
  - **4.3** The Internet Protocol (IP) pp. 361-363
  - **4.3.1** IPv4 Datagram Format
1. Click over a specific frame from the capture you have done with Wireshark. Check that it includes the **Ethernet** header with data from the link layer. Show via a screenshot the Ethernet header contents. Respond to the following questions:
    - a) What are source and destination addresses? Who do you think they belong to?
    - b) Who assigns these addresses?
    - c) What does type field mean?
    - d) Which function has the CRC field? Respond in a theoretical way.
    - e) What is the purpose of the preamble in an Ethernet frame?
    - f) What data does this Ethernet frame carry?
  2. In the previous frame, notice that the detail of an IP header is also included. Show via a screenshot the IP header contents. Respond to the following questions:
    - a) What are source and destination addresses? Who do you think they belong to?
    - b) Why the packet has an identifier?
    - c) Which flags are active in this packet?
    - d) Explain what the TTL value means in the analyzed packet.
    - e) Why is a checksum field needed again in the IP protocol?
    - f) Go to menu Statistics > IPv4 statistics > IP protocol types. Which is the protocol sending more packets? Why?
  3. Finally, without applying any filters, go to the statistics menu Statistics > Protocol Hierarchy. Show in a screenshot the results and comment them, relating them with packet encapsulation and de-encapsulation, a fundamental pillar of network communications.

## First Part Answers

### Exercise 1



No.	Time	Source	Destination	Protocol	Length	Info
18	3.356050	34.117.188.166	192.168.1.110	QUIC	1294	Initial, DCID=7fce13, SCID=e2ffb20c61e425d3
19	3.356119	192.168.1.110	34.36.54.80	TCP	66	50460 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
20	3.356156	192.168.1.110	34.107.152.202	TCP	66	50459 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
21	3.356666	192.168.1.110	34.36.54.80	TCP	1466	50460 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
22	3.356682	192.168.1.110	34.36.54.80	TLSv1.3	1178	Client Hello (SNI=ads-ing.mozilla.org)
23	3.357299	192.168.1.110	34.107.152.202	TCP	1466	50459 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
24	3.357319	192.168.1.110	34.107.152.202	TLSv1.3	1203	Client Hello (SNI=firefox-settings-attache
25	3.363818	34.120.200.123	192.168.1.110	TCP	74	443 → 50461 [SYN, ACK] Seq=0 Ack=1 Win=6553
26	3.363877	192.168.1.110	34.120.200.123	TCP	66	50461 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
27	3.364370	192.168.1.110	34.120.200.123	TCP	1466	50461 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
28	3.364386	192.168.1.110	34.120.200.123	TLSv1.3	1189	Client Hello (SNI=incoming.telemetry.mozill
29	3.372217	34.36.54.80	192.168.1.110	TCP	70	[TCP Dup ACK 16#1] 443 → 50460 [ACK] Seq=1
30	3.372218	34.36.54.80	192.168.1.110	TCP	66	443 → 50460 [ACK] Seq=1 Ack=2513 Win=266496
31	3.372218	34.107.152.202	192.168.1.110	TCP	66	443 → 50459 [ACK] Seq=1 Ack=1401 Win=267528
32	3.372218	34.107.152.202	192.168.1.110	TCP	66	443 → 50459 [ACK] Seq=1 Ack=1401 Win=267528

Frame 25: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on in  
 Ethernet II, Src: MitraStarTec\_7f:10:aa (2c:96:82:7f:10:aa), Dst: b6:41:a9:cd:04:86  
 Destination: b6:41:a9:cd:04:86 (b6:41:a9:cd:04:86)  
 Source: MitraStarTec\_7f:10:aa (2c:96:82:7f:10:aa)  
 Type: IPv4 (0x0800)  
 [Stream index: 2]  
 Transmission Control Protocol, Src Port: 443, Dst Port: 50461, Seq: 0, Ac

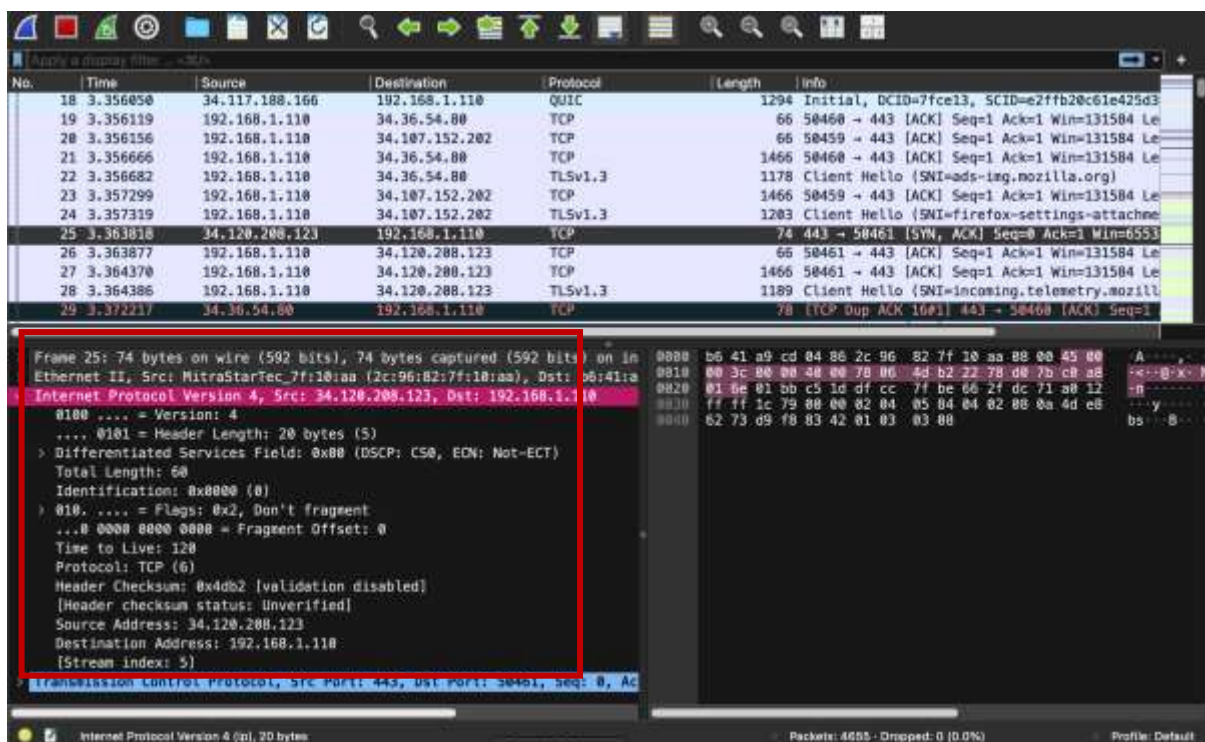
- a)
- The **source** is the MAC address or physical address, a 12-digit hexadecimal number that is the unique identifier assigned to a device's network interface and is the device that sent the packet. In this example **Source: MitraStarTec\_7f:10:aa (2c:96:82:7f:10:aa)**
  - The **destination** is the MAC address corresponding to the device receiving the packer (in this case my computer MAC address **Destination: b6:41:a9:cd:04:86 (b6:41:a9:cd:04:86)**).

```
nicolasdalessandro --bash-- 80x24
(base) Nicolass-MacBook-Pro-9:~ nicolasdalessandro$ ifconfig en0 | grep ether
ether b6:41:a9:cd:04:86
(base) Nicolass-MacBook-Pro-9:~ nicolasdalessandro$
```

- b) These MAC addresses are **assigned by the manufacturer of the network card**, and as we mentioned, every device has a unique MAC address that helps with its identification on a network.

- c) This field shows what kind of protocol is inside the frame, in our example is 0x0800 and this means that the frame has an IPv4 packet.
- d) The **Cyclic Redundancy Check** is for error detection in the Ethernet frame. This is because its being checked if the data in the frame was corrupted or changed: if the value doesn't match the receiver will not accept the frame.
- e) The **preamble** is a small sequence of 64 bits that are at the beginning of the Ethernet frame. It is for helping in synchronize devices because they understand when the data starts (last two bits). This allows the receiving device to prepare for incoming data.
- f) This packet contains information like address and protocols. In this example is an IPv4 packet with data using the UDP protocol.

## Exercise 2



The image shows a Wireshark packet capture analysis. The top pane displays a list of captured packets. The bottom pane shows the detailed view of a selected packet (Frame 25), which is an Internet Protocol Version 4 (IPv4) packet. The packet details are as follows:

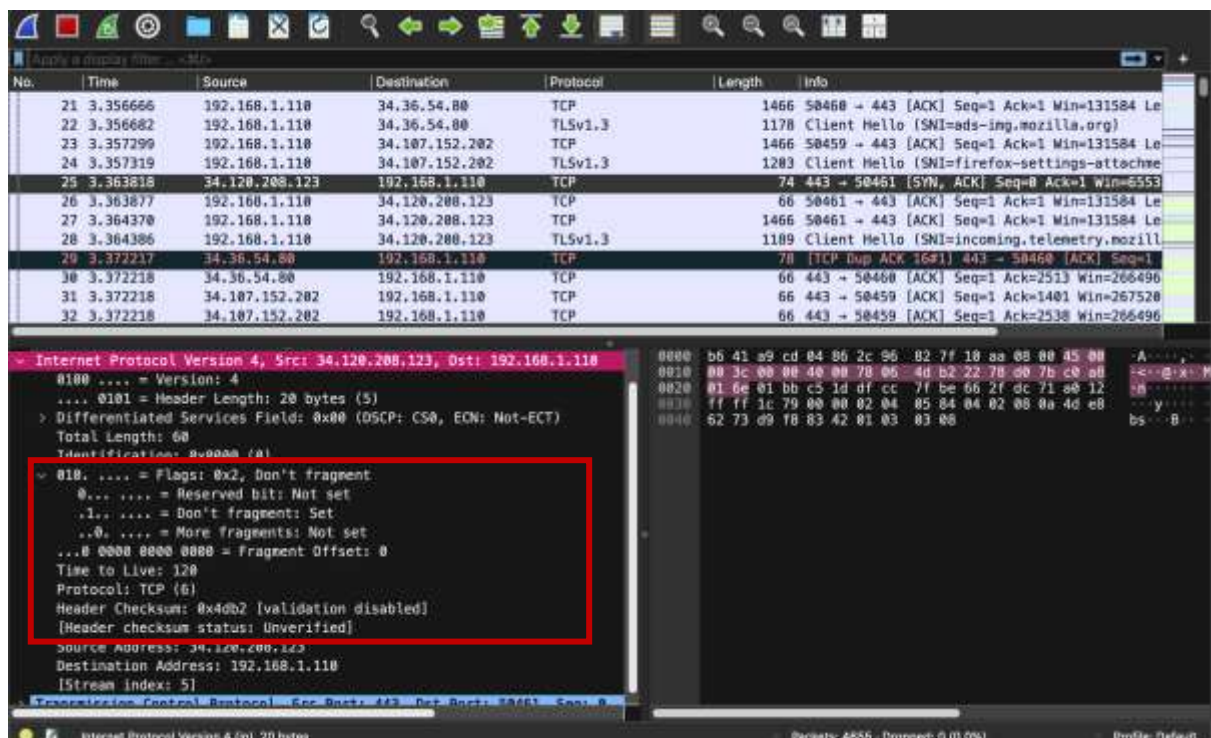
- Frame 25: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
- Ethernet II, Src: MitraStarTec\_7f:10:aa (2c:96:82:7f:10:aa), Dst: 192.168.1.110
- Internet Protocol Version 4, Src: 34.120.208.123, Dst: 192.168.1.110
  - 0100 .... = Version: 4
  - .... 0101 = Header Length: 20 bytes (5)
  - > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  - Total Length: 60
  - Identification: 0x0000 (0)
  - > 0100 .... = Flags: 0x2, Don't fragment
  - ...0 0000 0000 0000 = Fragment Offset: 0
  - Time to Live: 120
  - Protocol: TCP (6)
  - Header Checksum: 0x4db2 [validation disabled]
  - [Header checksum status: Unverified]
  - Source Address: 34.120.208.123
  - Destination Address: 192.168.1.110
  - [Stream index: 5]

- a)
  - The **source** IP address (a string of number that are the unique identifier assigned to a device's connected to a computer network using the Internet Protocol for Communication), is the server where the information was sent. In this example **34.120.208.123**.
  - The **destination** IP address corresponds to the device receiving the resources (in this case my computer IP address **192.168.1.110**).



IP address	192.168.1.110
Router	192.168.1.1

- b) The identifier helps if the packet needs to be divided into small pieces. All these pieces will contain the same identifier allowing to easily put them together.
- c) In our example we have the **“Don't Fragment”** flag active (0x2). This means that we can't separate the package into smaller pieces.



The image shows a Wireshark packet capture analysis. The packet list pane displays a table of captured packets. The packet details pane shows the structure of the selected packet, and the packet bytes pane shows the raw data.

No.	Time	Source	Destination	Protocol	Length	Info
21	3.356666	192.168.1.110	34.36.54.80	TCP	1466	50460 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
22	3.356682	192.168.1.110	34.36.54.80	TLsv1.3	1178	Client Hello [SNI=ads-img.mozilla.org]
23	3.357299	192.168.1.110	34.107.152.202	TCP	1466	50459 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
24	3.357319	192.168.1.110	34.107.152.202	TLsv1.3	1283	Client Hello [SNI=firefox-settings-attache
25	3.363818	34.120.208.123	192.168.1.110	TCP	74	443 → 50461 [SYN, ACK] Seq=0 Ack=1 Win=6553
26	3.363877	192.168.1.110	34.120.208.123	TCP	66	50461 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
27	3.364370	192.168.1.110	34.120.208.123	TCP	1466	50461 → 443 [ACK] Seq=1 Ack=1 Win=131584 Le
28	3.364386	192.168.1.110	34.120.208.123	TLsv1.3	1189	Client Hello [SNI=incoming.telemetry.mozill
29	3.372217	34.36.54.80	192.168.1.110	TCP	70	[TCP Dup ACK 16#1] 443 → 50460 [ACK] Seq=1
30	3.372218	34.36.54.80	192.168.1.110	TCP	66	443 → 50460 [ACK] Seq=1 Ack=2513 Win=266496
31	3.372218	34.107.152.202	192.168.1.110	TCP	66	443 → 50459 [ACK] Seq=1 Ack=1401 Win=267520
32	3.372218	34.107.152.202	192.168.1.110	TCP	66	443 → 50459 [ACK] Seq=1 Ack=2538 Win=266496

The packet details pane shows the structure of the selected packet (No. 29). The packet is an Internet Protocol Version 4 packet from Source 34.120.208.123 to Destination 192.168.1.110. The packet length is 60 bytes. The packet details pane shows the structure of the selected packet, and the packet bytes pane shows the raw data.

```

Internet Protocol Version 4, Src: 34.120.208.123, Dst: 192.168.1.110
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x0000 (0)
    010. .... = Flags: 0x2, Don't fragment.
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
      ...0 0000 0000 = Fragment Offset: 0
      Time to Live: 120
      Protocol: TCP (6)
      Header Checksum: 0x4db2 [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 34.120.208.123
      Destination Address: 192.168.1.110
      [Stream index: 5]
  Transmission Control Protocol, Src Port: 443, Dst Port: 50461, Seq: 0
  Internet Protocol Version 4 (ip), 20 bytes
  
```

- d) **Time to Live** is 120 meaning that the packet can pass on 120 routers before being discarded. This number is decreasing -1 every time the packet passes in a router avoiding infinite loops.
- e) The **Checksum** is needed for detecting errors in the header of the IP. If there's any mismatch this means that the data is corrupted, and the package is ignored.

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
IPv4 Statistics/IP Protocol Types	4637				0.3102	100%	10.6400	8.040
UDP	2583				0.1728	55.70%	7.6100	8.041
TCP	2047				0.1369	44.14%	3.2400	8.023
NONE	7				0.0005	0.15%	0.0400	6.738

Display filter:  Apply

Copy Save as... Close

- f) I was at the moment of the capture watching the YouTube video available on the page, and probably that's the reason why **the UDP protocol has the higher number of 55.70%**. The protocol **TCP has 44,14%** (this is the most used protocol for reliable connections). Finally, packets with NON protocol are 0.15% of this capture.



[JAVA](#)
[ANDROID](#)
[JUEGOS](#)
[SQL](#)
[HTM](#)

Conceptos web o www

World Wide Web

Cliente HTTP

documento HTML

Servidor HTTP

**World Wide Web, www, w3 o La Web**



**Web 1.0 World Wide Web, www, w3 o La Web**

Una página web está escrita en lenguaje **HTML**. Tiene un nombre o identificador único en todo Internet. Este identificador se llama **URL**.

**URL "Uniform Resource Locator"**

### Exercise 3

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End
Frame	100.0	4655	100.0	3749494	2006 k	0	0
Ethernet	100.0	4655	1.7	65208	34 k	0	0
Realtek Layer 2 Protocols	0.1	4	0.0	184	98	4	184
Internet Protocol Version 6	0.2	11	0.0	464	248	0	0
User Datagram Protocol	0.0	1	0.0	8	4	0	0
Data	0.0	1	0.0	4	2	1	4
Internet Control Message Protocol v6	0.2	10	0.0	316	169	10	316
Internet Protocol Version 4	99.6	4637	2.5	92768	49 k	0	0
User Datagram Protocol	55.5	2583	0.6	20664	11 k	0	0
QUIC IETF	53.7	2499	64.5	2419080	1294 k	2499	231
Domain Name System	1.7	80	0.2	6070	3248	80	607
Data	0.1	4	0.0	816	436	4	816
Transmission Control Protocol	44.0	2047	1.8	66668	35 k	1170	387
Transport Layer Security	18.0	839	23.2	871286	466 k	839	851
Hypertext Transfer Protocol	0.8	38	0.4	14099	7544	20	725
Portable Network Graphics	0.2	9	3.0	112900	60 k	9	112
Media Type	0.0	1	0.0	1150	615	1	115
Line-based text data	0.2	8	3.9	146784	78 k	8	146
Internet Group Management Protocol	0.2	7	0.0	56	29	7	56
Address Resolution Protocol	0.1	3	0.0	84	44	3	84
802.1Q Virtual LAN	0.1	6	0.0	34	18	0	0

On this image we can understand how the encapsulation and de-encapsulation with the multiple layers works observing:

- **Encapsulation:** When we make the request, the data goes by multiple layers with different information (for example headers, etc), in our image:
  - On the top we have all packet that are Ethernet frames at Layer2 and inside almost all of the packages are IPv4 packets.
  - This Ipv4 packages have TCP and UDP (multiple protocols).
  - In summary:
    - Layer 2: Ethernet frames
    - Layer 3: 99,6% Ipv4 Packages
    - Layer 4: 44.0% TCP
      - TLS 18%
      - Data
    - Layer 4: 55,5% UDP
      - QUIC IETF 53.7% (*\*\*\*maybe the YouTube video impact this number*)
      - DNS 1.7%
      - Data 0.1%
- **De-encapsulation:** When the device receives the information, the process is the opposite of what was mentioned above. Each layer removes its headers to ensure the data is processed correctly, and only the relevant application receives the data.



## **Second part (maximum qualification: C+): Transport layer**

In this second part of the practice, we will deepen on the next layer of the Internet stack: Transport layer. There are two protocols offering different services at this layer: UDP (User Datagram Protocol) and TCP (Transport Control Protocol). You can work with the same packet capture you already used in the previous section.

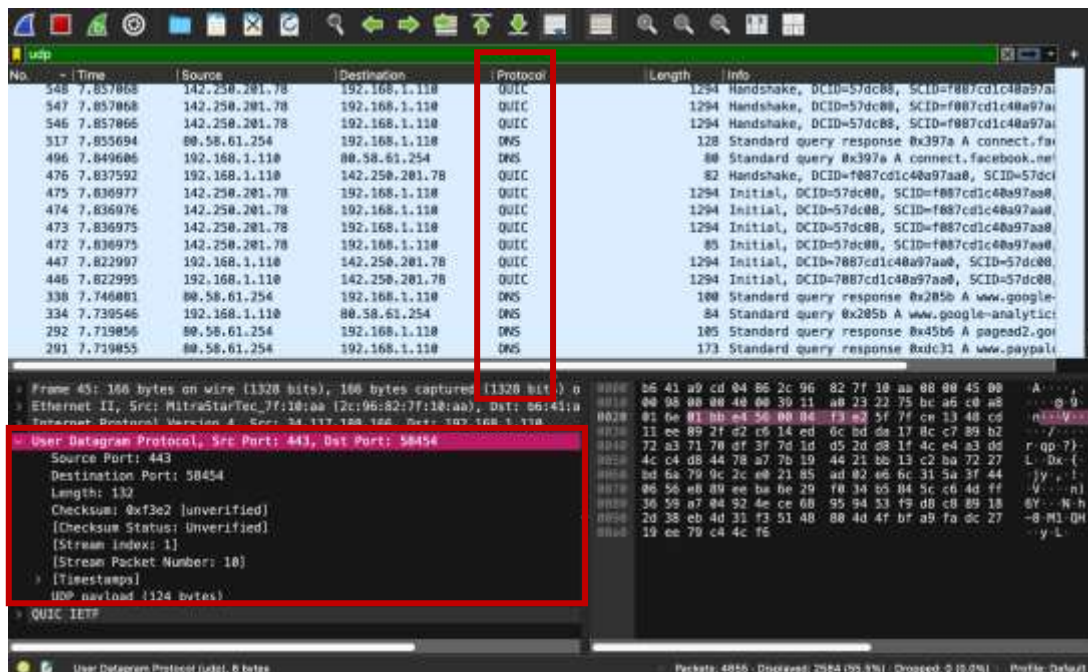
As support material for doing this second part of the practice, we propose you the following sections of the Computer Networking book (8th edition), especially sections concerning protocol headers:

- **3.3 Connectionless Transport: UDP** p. 228
  - **3.3.1 UDP Segment Structure**
  - **3.5 Connection-Oriented Transport: TCP** pp. 260-265
  - **3.5.2 TCP Segment Structure**
- 1) First of all, you have to analyze the UDP protocol. You can filter this protocol in Wireshark and you will only see packets from this protocol. Show via a screenshot the header fields and respond to the following questions:
    - a) Which protocol/s from the application layer has/have generated these packets?
    - b) Why do you think UDP is used instead of TCP?
    - c) Why do values have source and destination ports? What does it mean?
    - d) Why is the checksum field required?
    - e) How is it calculated?
    - f) What is the value of the length field? Check with hexadecimal values of the UDP packet that this is the value indeed.
  - 2) Next, we have to check how the TCP protocol works. Go to the statistics menu Statistics > Flow Graph, and select only TCP type, checking the option Flow type > TCP flow. Show in a screenshot the phases where the connection is established and finished. Explain the process. What is the flag PSH used for?
  - 3) Go to the main packets list. You can filter using TCP protocol to see only packets concerning this protocol. Select a packet and show in a screenshot the TCP header fields content and respond to the following questions:
    - a) Which is the sequence number? What is it used for?
    - b) And the ACK number?
    - c) Which values have source and destination ports? What does it mean?
    - d) Which flags are active in this packet?
    - e) Explain what the values relative to the window mean.
    - f) Which data does this TCP packet carry?

## Second Part Answers

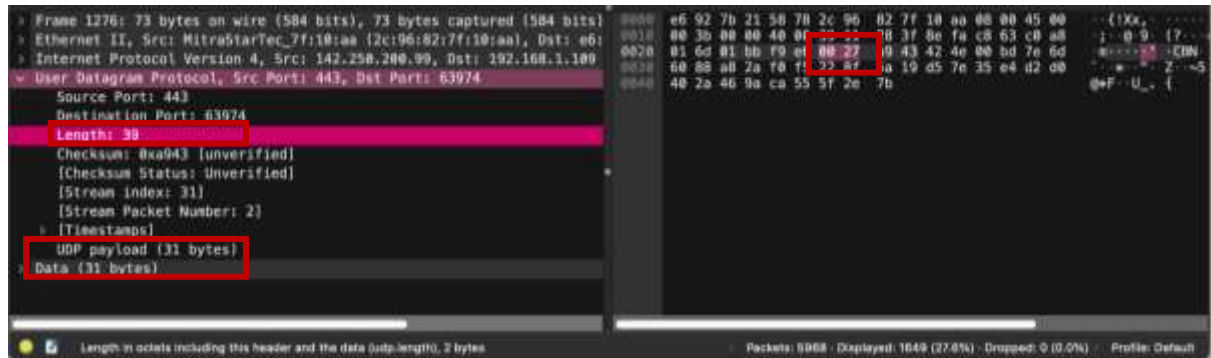
### Exercise 1

- a) The protocols from the application layer that generated these packets are QUIC and DNS



- b) UDP is used instead of TCP because it is faster, it has also lower latency and they can tolerate packet loss. In services like the YouTube video I was playing while capturing these packages, it is good for improving video loading. The DNS also uses UDP because it needs to send queries fast without waiting for a connection.
- c)
- The **source port** is assigned by the sender and is for identifying which application or service is sending the information.
  - The **destination port** is to identify where the packet should be delivered.
  - **These ports** are for allowing multiple applications to use the network simultaneously.
- d) The **checksum field** is for verifying the integrity of the data, if there's any mismatch this means that the data is corrupted, and the package is ignored. UDP doesn't accept retransmissions mechanisms as TCP.
- e) To calculate this field, we have:
- Pseudo header that is not transmitted but it is included in the calculation of the checksum
  - UDP header and data concatenated and if the total length is odd a 0 is added at the end to make it even.
  - The words divided in 16 bits.
  - Compute the one's complement of the 16 bits.
  - Taking the one's complement of that sum.

- f) In one specific selected packet (Frame 1276), the length field value is 39 bytes (0027 in the Hexadecimal window). That is the UDP header of 8 bytes and 31 bytes of data, as confirmed in the packet details with **"UDP payload (31 bytes)"**.



## Exercise 2



In this flow graph we can see both establish and termination of connections on a TCP communication. We learn from the theory that we have the different types of “handshake ways” in TCP for establish and terminate connections:

1. *Three-way handshake (establish connection)*
  - i. **SYN**: The client sends a request to open the connection.
  - ii. **SYN-ACK**: The server responds with **acknowledgment**.
  - iii. **ACK**: Client confirms and now the connection is established.
2. *Four-way handshake (terminate connection)*
  - i. **FIN**: Initiates with one part with a **FIN** packet.
  - ii. **ACK**: The receiver responds with **acknowledgment**.
  - iii. **FIN**: The receiver sends its own **FIN**
  - iv. **ACK**: Sender acknowledges the connection and closes it.

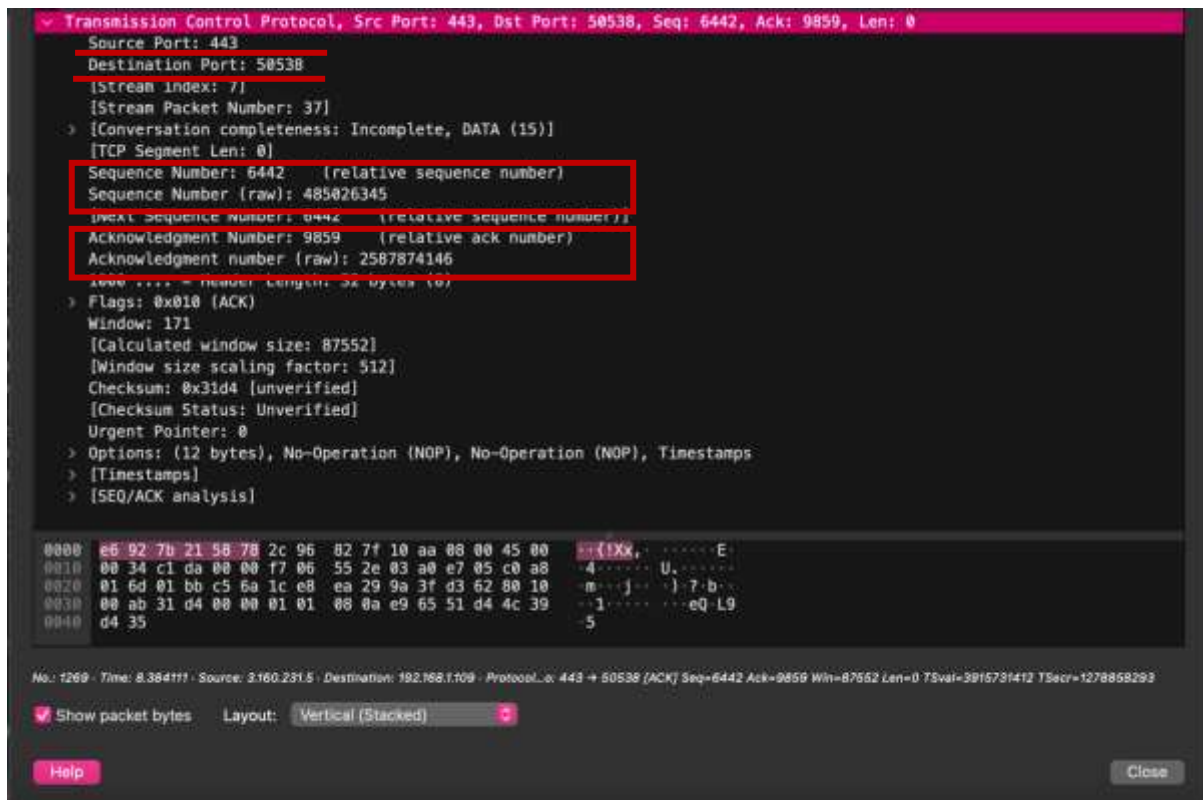
In our image we can see the communication between IP 192.168.1.110 (my computer IP), IP 134.107.152.202 and 34.36.54.80 and the destination is mainly the port HTTPS 443.

We can see a **complete three-way handshake** in the green section:

- A **SYN** packet is sent at time **3.330803**.
- A **SYN-ACK** response is received at time **3.356048**.
- An **ACK** is sent back at time **3.356116**, completing the handshake.

The **PSH push flag** is for telling the receiver to process the data immediately instead of waiting for more data to arrive. This is useful for applications like **chat, streaming, or any real-time data transfer**.

### Exercise 3



```

Transmission Control Protocol, Src Port: 443, Dst Port: 50538, Seq: 6442, Ack: 9859, Len: 0
  Source Port: 443
  Destination Port: 50538
  [Stream index: 7]
  [Stream Packet Number: 37]
  [Conversation completeness: Incomplete, DATA (15)]
  [TCP Segment Len: 0]
  Sequence Number: 6442 (relative sequence number)
  Sequence Number (raw): 485026345
  [Next sequence number: 6442 (relative sequence number)]
  Acknowledgment Number: 9859 (relative ack number)
  Acknowledgment number (raw): 2587874146
  [Data offset: 5 - Header length: 20 bytes (0)]
  Flags: 0x010 (ACK)
  Window: 171
  [Calculated window size: 87552]
  [Window size scaling factor: 512]
  Checksum: 0x31d4 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  [Timestamps]
  [SEQ/ACK analysis]

0000 e6 92 7b 21 58 78 2c 96 82 7f 10 aa 08 00 45 00  {IXX,.....E
0010 00 34 c1 da 00 00 f7 06 55 2e 03 a0 e7 05 c0 a8  4.....U.....
0020 01 6d 01 bb c5 6a 1c e8 ea 29 9a 3f d3 62 80 10  m...j...}.7.b...
0030 00 ab 31 d4 00 00 01 01 08 0a e9 65 51 d4 4c 39  --1.....eQ.L9
0040 d4 35                                           5
  
```

No.: 1269 · Time: 8.384111 · Source: 3.160.231.5 · Destination: 192.168.1.109 · Protocol: 443 → 50538 [ACK] Seq=6442 Ack=9859 Win=87552 Len=0 TSval=3915731412 TSecr=1278858293

☒ Show packet bytes Layout: Vertical (Stacked)

Help Close

- The sequence number is **6442 (relative)** or **485026345 (raw)**.  
This number helps to order of data bytes in packages because the receiver can re-build the data properly based on that information. This number is also used to detect missing segments.
- The ACK number is **9589 (relative)** or **2587874146 (raw)**.  
As I have explained in the part 2, the ACK number “acknowledges” the receipt of data up to this byte position from the other side of the connection. It means that it tells the sender which byte was receiving correctly and confirming the delivery.
- The **source port is 443 (standard for HTTPS)** and the **destination port is 50538 (random and chosen by my device)**. So, this means that my computer is having a communication with a web server using HTTPS.



```
Sequence Number (raw): 489026345
[Next Sequence Number: 6442 (relative sequence number)]
Acknowledgment Number: 9859 (relative ack number)
Acknowledgment number (raw): 2587874146
1000 ..... = Header length: 32 bytes (8)

= Flags: 0x010 (ACK)
000 ..... = Reserved: Not set
...0 ..... = Accurate ECN: Not set
...0 ..... = Congestion Window Reduced: 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: 171
[Calculated window size: 87552]
[Window size scaling factor: 512]
Checksum: 0x1000 (unverified)
[Checksum status: Unverified]
Urgent pointer: 0

Acknowledgment (tcp.flags.ack, 1 bit)
```

- d) In our example, the **ACK flag** is active. This means the packet is **acknowledging** data received from the other side.
- e) The **windows value** is **171** with a **calculated window size 87552 scaled by factor 512**. In TCP, the windows size indicates how much data the receiver can accept. So, in our example we can see that 87.552 bytes (171 x 512) is how much data can be sent before waiting for acknowledgement.
- f) A TCP package can carry different type of information:
  1. The **header** which is always present.
  2. If it is part of the “handshake” will have the **connection establishment** (example flags SYN-ACK).
  3. And the **acknowledge of confirmation** ACK.

In our example, the TCP package has no data **Len: 0**, so is a pure acknowledgment packet with no content.

```
TCP 66 443 → 50538 [ACK] Seq=6442 Ack=9859 Win=87552 Len=0 (Sval=...
```

## Third part (maximum qualification: B): Application layer

Following the Internet stack structure, the last layer is the one interacting with applications executed by the user, directly or indirectly: Application Layer. At this layer we can find several protocols, but in this part, we will focus on two of them: DNS (Domain Name Server) and HTTP (Hypertext Transfer Protocol).

As support material for doing this third part of the practice, we propose you the following sections of the Computer Networking book (8th edition):

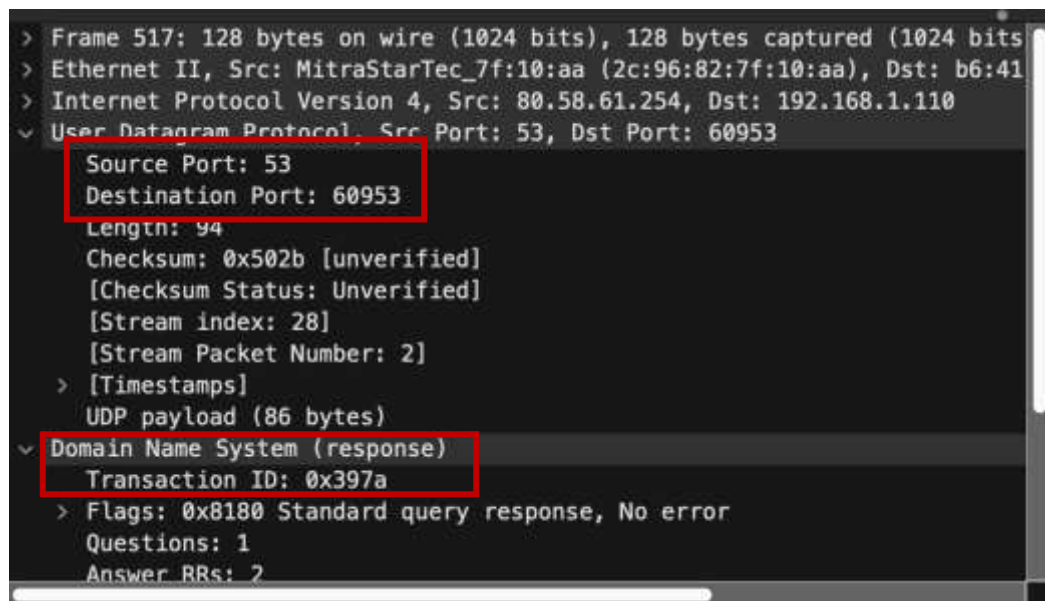
- **2.4 DNS—The Internet's Directory Service** pp. 161-164
  - **2.4.3 DNS Records and Messages**
  - **2.2 The Web and HTTP** pp. 131-138
  - **2.2.3 HTTP Message Format**
  - **2.2.4 User-Server Interaction: Cookies**
1. First of all, you have to analyze a DNS header from a request and a response, so you can filter in Wireshark by DNS protocol and you will see only packets from this protocol. Show a screenshot with the header fields and respond to the following questions:
    - a) Which source and destination ports are used?
    - b) What is the transaction identifier for?
    - c) What do flags mean?
    - d) In the request, which domain is requested?
    - e) Which kind and class this domain is? What does it mean?
    - f) Can there be multiple records (RR, resource records) in the response?
  2. Now analyze an **HTTP** header from a GET request. Show a screenshot with the header fields and respond to the following questions:
    - a) Which source and destination ports are used?
    - b) Which version of HTTP protocol is?
    - c) Are persistent connections used?
    - d) Which language is used? Why is it provided?
    - e) Which kind of content could be processed?
    - f) Which coding formats are used?
  3. Analyze an **HTTP** header of a successful GET response. Show a screenshot with the header fields and respond to the following questions:
    - a) What does this response contain?
    - b) Which kind of content contains?
    - c) Is the content fragmented? How do we know?
    - d) Which fields are related to dates? What do they mean?
    - e) Are cookies used? How do they work?
    - f) Describe what the fields related to caches mean.

4. Go to the Wireshark statistics menu Statistics > HTTP > Packet counter. Show a screenshot where the unsuccessful responses received are seen (they have value Count > 0). Describe what each error code means and how many packets of each packet have been received. Afterwards, select an error code and look for it on the main packet capture screen. Show a screenshot with the header fields with a brief explanation.

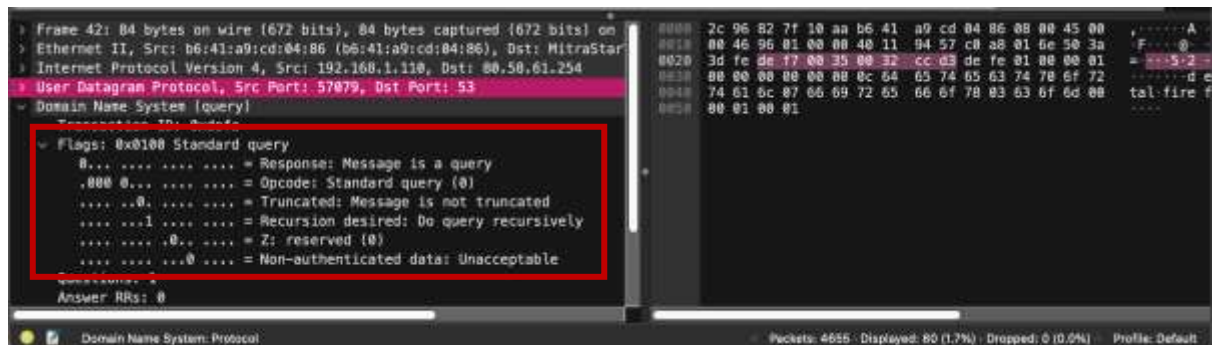
### Third Part Answers

#### Exercise 1

- a) The **Source Port is 60953** which is chosen random by the client and the **Destination port is 53** which is the standard port used by DNS servers:



- b) The **transaction ID** is the unique number that is assigned to a DNS request, and it serves for several purposes, but mainly it is used for request-response matching, meaning that when a DNS client sends a query, having this unique transaction ID, the server returns the same ID in the response. It is also useful when we have multiple concurrent queries, since this number helps to differentiate between response to multiple queries.



```

Frame 42: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on
Ethernet II, Src: b6:41:a9:cd:04:86 (b6:41:a9:cd:04:86), Dst: MitraStar
Internet Protocol Version 4, Src: 192.168.1.110, Dst: 80.50.61.254
User Datagram Protocol, Src Port: 57079, Dst Port: 53
Domain Name System (query)
  Transaction ID: 0x0000
  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
  Answer RRs: 0
  Question RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  Packets: 4655 · Displayed: 80 (1.7%) · Dropped: 0 (0.0%) · Profile: Default

```

c)

- **Response: Message is a query (0)** means that this is a request (not a response).
- **Opcode: Standard query (0)** means this is a normal DNS request.
- **Truncated: Message is not truncated (0)** means no truncation.
- **Recursion Desired (RD bit 1)** means that the client wants that the server to perform recursive resolution.
- **Z: reserved (0)** means that there are three bits that are reserved for future use.
- **Non-authenticated data: Unacceptable (0)** means that the client wants authenticated data.

- d) The domain requested is **www-geo.nike.com.akadns.net**

```
> Frame 87: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on in
> Ethernet II, Src: b6:41:a9:cd:04:86 (b6:41:a9:cd:04:86), Dst: MitraStarTe
> Internet Protocol Version 4, Src: 192.168.1.110, Dst: 80.58.61.254
> User Datagram Protocol, Src Port: 65223, Dst Port: 53
v Domain Name System (query)
  Transaction ID: 0xe376
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > www-geo.nike.com.akadns.net: type A, class IN
    [Response In: 100]
```

- e) As we can observe in the above image we have **type A, class IN**:

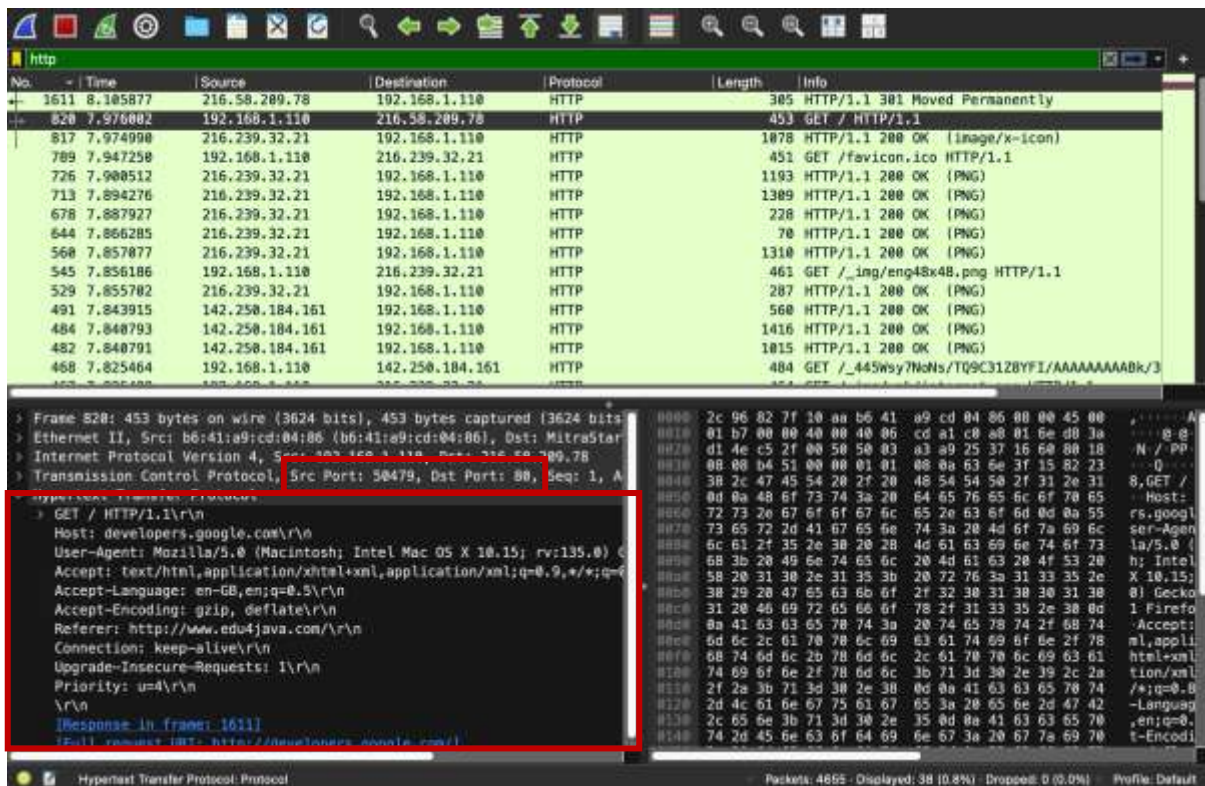
- The **type A** means that the query is asking for an IPv4 address
- The **class IN** means that the domain is in the public internet.

- f) Yes, there can be multiple resource records in the response. Example:

- **A records** as our example, asking for IPv4 addresses resolution
- **AAAA records** for IPv6 addresses)
- **CNAME or Canonical Name** that creates an alias for another domain name.
- **MX or Mail Exchange** In our case, the response included an **A record with the IP address** of the requested domain.



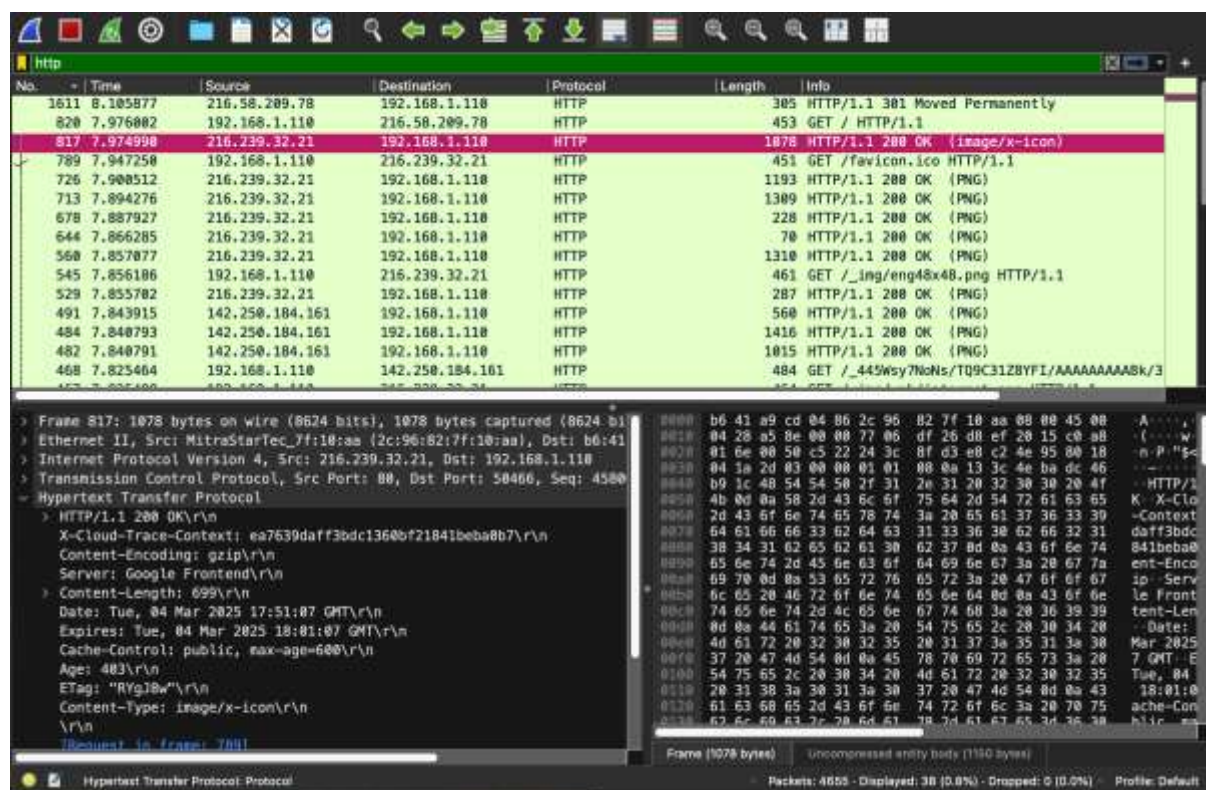
## Exercise 2



The image shows a Wireshark packet capture of an HTTP GET request. The packet list shows a GET request to developers.google.com. The packet details pane shows the request structure, including Host, User-Agent, Accept, Accept-Encoding, Referer, Connection: keep-alive, Upgrade-Insecure-Requests, Priority, and a full request line. The packet bytes pane shows the raw data.

- The **source port** is **50479** (again, randomly chosen) and the **destination port** is **80** which is the standard for HTTP.
- The HTTP version that is used in this case is **HTTP/1.1**.
- Yes, because we can observe that the request includes the **Connection: keep-alive** which means that the connection keep open for multiple requests.
- The language of our example is **Accept-Language: en-GB,en;q=0.5**, this means that the preferred language for receiving the response is english with UK preference. The **q** parameter indicates the relative preference and it's a value from 0 to 1. So in this case is not the highest preference but an intermediate. This means that it is preferred british english but if not available it can accept generic english
- In this example the request has the **Accept:text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8** parameter which means that it can accept any type of content with a preference order: **text/html** and **application/xhtml+xml** is the preferred content types (highest priority). **application/xml;q=0.9** XML content is also acceptable, but less priority (q=0.9). **\*/\*;q=0.8** it can accept any other content type, but with the lowest priority (q=0.8).
- This example has the **Accept-Encoding: gzip, deflate** parameter which means that the browser can accept compressed responses.

### Exercise 3



Wireshark packet capture showing HTTP traffic. The top pane displays a list of packets. Packet 817 is selected, showing an HTTP 200 OK response for a favicon. The middle pane shows the details of the selected packet, including the request line, status line, and various headers. The bottom pane shows the raw packet data in hexadecimal and ASCII.

Frame 817: 1078 bytes on wire (8624 bits), 1078 bytes captured (8624 bits) on interface en0, id 0

Ethernet II, Src: MitraStarTec\_7f:10:aa (2c:96:82:7f:10:aa), Dst: b6:41:a9:cd:04:86 (b6:41:a9:cd:04:86)

Internet Protocol Version 4, Src: 216.239.32.21, Dst: 192.168.1.110

Transmission Control Protocol, Src Port: 80, Dst Port: 50466, Seq: 45806, Ack: 1102, Len: 1012

Hypertext Transfer Protocol

HTTP/1.1 200 OK\r\n

X-Cloud-Trace-Context: ea7639daff3bdc1360bf21841beba0b7\r\n

Content-Encoding: gzip\r\n

Server: Google Frontend\r\n

Content-Length: 699\r\n

Date: Tue, 04 Mar 2025 17:51:07 GMT\r\n

Expires: Tue, 04 Mar 2025 18:01:07 GMT\r\n

Cache-Control: public, max-age=600\r\n

Age: 403\r\n

ETag: "RYgJ8w"\r\n

Content-Type: image/x-icon\r\n

\r\n

[Request in frame: 789]

[Time since request: 0.027740000 seconds]

[Request URI: /favicon.ico]

[Full request URI: http://www.edu4java.com/favicon.ico]

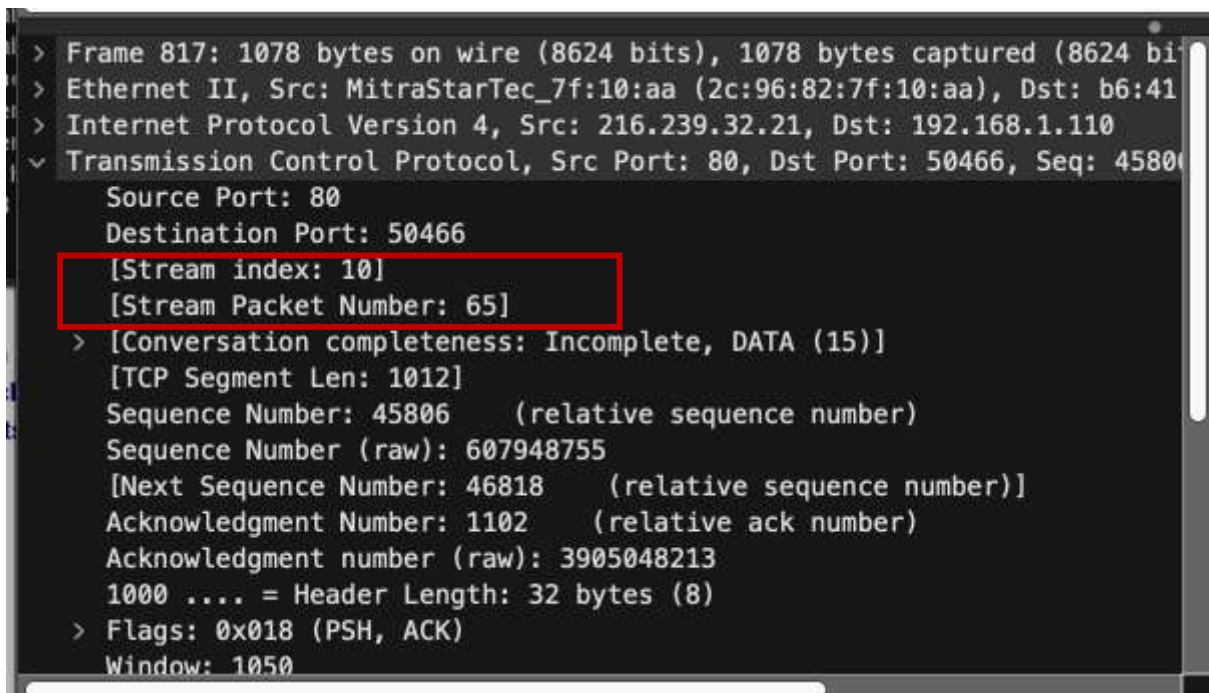
Content-encoded entity body (gzip): 699 bytes -> 1150 bytes

File Data: 1150 bytes

Media Type

Media type: image/x-icon (1150 bytes)

- a) The response contains the **header** with metadata about the response and the **body** with the requested data.
- Header:
    - o **Status Line HTTP/1.1 200 OK**
    - o **Date: Tue, 04 Mar 2025 17:51:07 GMT**
    - o **Server: Google Frontend**
    - o Other headers like Content-Type, Content-Length, Cache-Control, etc. (I will explain in the next exercises).
  - Body:
    - o The response body is a **favicon file** (favicon.ico) that is an **image file** (image/x-icon) and the content is compressed by **Content-Encoding: gzip**
- b) The file requested was a **the above mentioned favicon**. The URI in the image is <http://www.edujava.com/favicon.ico>. The content type **image/x-icon** which means that the request was for a website favicon.
- c) Yes, we can see multiple TCP segments in Wireshark, this means the response was separated in different packets.



```

> Frame 817: 1078 bytes on wire (8624 bits), 1078 bytes captured (8624 bi
> Ethernet II, Src: MitraStarTec_7f:10:aa (2c:96:82:7f:10:aa), Dst: b6:41
> Internet Protocol Version 4, Src: 216.239.32.21, Dst: 192.168.1.110
> Transmission Control Protocol, Src Port: 80, Dst Port: 50466, Seq: 4580
  Source Port: 80
  Destination Port: 50466
  [Stream index: 10]
  [Stream Packet Number: 65]
  > [Conversation completeness: Incomplete, DATA (15)]
  [TCP Segment Len: 1012]
  Sequence Number: 45806 (relative sequence number)
  Sequence Number (raw): 607948755
  [Next Sequence Number: 46818 (relative sequence number)]
  Acknowledgment Number: 1102 (relative ack number)
  Acknowledgment number (raw): 3905048213
  1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
  Window: 1050
  
```

- The **Stream index 10** means that this packet belongs to an ongoing TCP stream.
- **Stream Packet Number 65** means that possibly, other packets are needed to complete all the response.



d)

```
HTTP/1.1 200 OK
X-Cloud-Trace-Context: 96e035ad5b452153dfd24f46a26ac9d7
Content-Encoding: gzip
Server: Google Frontend
Content-Length: 6436
Date: Tue, 04 Mar 2025 17:51:15 GMT
Expires: Tue, 04 Mar 2025 18:01:15 GMT
Cache-Control: public, max-age=600
Age: 395
ETag: "RYgJ8w"
Content-Type: application/x-javascript
```

**Date: Tue, 04 Mar 2025 17:51:07 GMT** means the date when the server sent the response.

e) Yes, we can see the **Set-Cookie** field in a previous response. These are for storing small data pieces on the client's browser side and be able to keep sessions and preferences.

f)

```
HTTP/1.1 200 OK
X-Cloud-Trace-Context: 96e035ad5b452153dfd24f46a26ac9d7
Content-Encoding: gzip
Server: Google Frontend
Content-Length: 6436
Date: Tue, 04 Mar 2025 17:51:15 GMT
Expires: Tue, 04 Mar 2025 18:01:15 GMT
Cache-Control: public, max-age=600
Age: 395
ETag: "RYgJ8w"
Content-Type: application/x-javascript
```

- **Public** the response can be cached by anyone.
- **max-age=600** which means that the response **can be cached for 600 seconds (10 minutes)** before it is considered that need to be revalidated or get it again.

## Exercise 4

Packet Type	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
▼ Total HTTP Packets	38				0.0081	100%	0.1900	7.808
Other HTTP Packets	0				0.0000	0.00%	-	-
▼ HTTP Response Packets	19				0.0040	50.00%	0.1000	7.808
??? : broken	0				0.0000	0.00%	-	-
5xx: Server Error	0				0.0000	0.00%	-	-
4xx: Client Error	0				0.0000	0.00%	-	-
▼ 3xx: Redirection	1				0.0002	5.26%	0.0100	8.106
301 Moved Permanently	1				0.0002	100.00%	0.0100	8.106
▼ 2xx: Success	18				0.0038	94.74%	0.1000	7.808
200 OK	18				0.0038	100.00%	0.1000	7.808
1xx: Informational	0				0.0000	0.00%	-	-
▼ HTTP Request Packets	19				0.0040	50.00%	0.1200	7.735
GET	19				0.0040	100.00%	0.1200	7.735

Display filter:  Apply

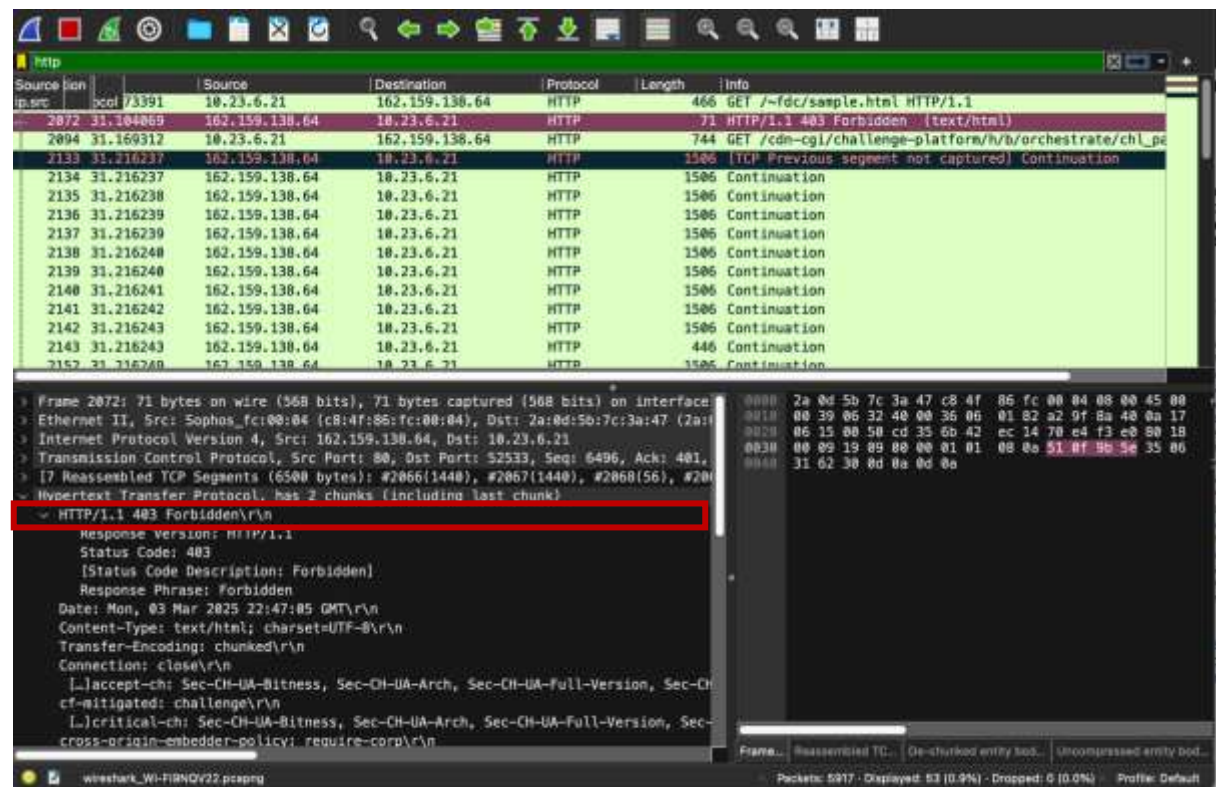
Copy Save as... Close

4. In this image we can observe the HTTP Packet Counter statistics of ur Wireshark capture. Here we can see the break down of the HTTP responses. The responses are into categories depending on the first digit:
- **1xx Information Response** means that the server acknowledges the request but it has not completed it yet (*in our image we have not packages associated with this code*)
  - **2xx Successful Response** means that the request was succesfully processed (*we have 18 packages in our capture and all of them are 200 OK responses*).
  - **3xx Redirections** means that the client is being redirected to other URL (*we have 1 packets which is a 301 moved permanently response*).
  - **4xx Client Errors** means that the client request have error or that the page could not be reached, is unavailable or has bad syntax (*we have no packets of this category*).
  - **5xx Server Errors** means that the server encountered errors while processing the request (*we have o packets in our example*).



### Unsuccessful Responses:

As we mentioned before, there was no error 4XX in our capture, so I force an error and capture other iteration in Wireshark for finding an error packet and checking its information:



This error represent that the server understood the request, but it was not authorized. Possible reasons for this error are:

- If the server request authentication, but the client didn't have valid credentials.
- When the request is blocked because issue with permissions or security policies.
- When the client attempts to access a resource, but the resource is restricted.

## **Fourth part (maximum qualification: A): Network security**

To do the last part of this practice, we have to focus on the interaction that happens when we write our username and password to access an online page where you are registered. You cannot use the UOC campus access page.

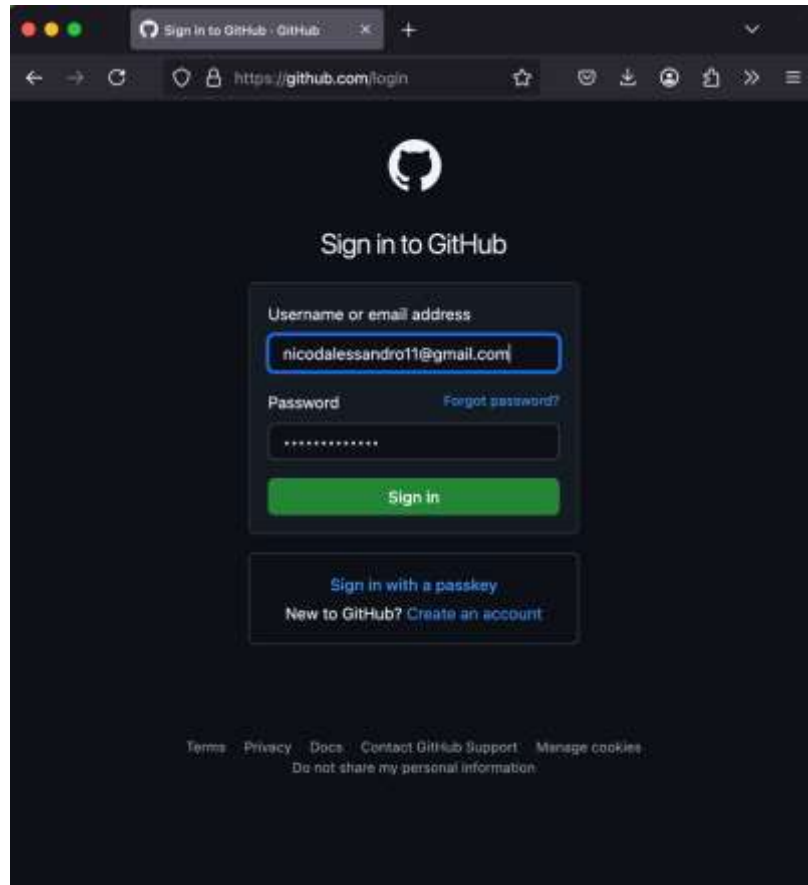
1. Show a screenshot of the URL and the webpage where you write your username and password.
2. Run Wireshark and start capturing packets. Access the webpage with your credentials. Which application layer protocol is used to offer security to this online page? Describe its main features.
3. Focus on a specific packet from this protocol. Show a screenshot with the header fields and explain what you observe.
4. Over which transport layer protocol travels the security protocol used to access the online page?
5. Finally, while Wireshark is capturing packets, connect to the web site:  
[http://gaia.cs.umass.edu/wireshark-labs/protected\\_pages/HTTP-wireshark-file5.html](http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html).

When the web site asks for authentication data, enter the username **wireshark-students** and password **network**. Select http as Wireshark filter. The username and password are visible after line *Authorization: Basic* in a HTTP GET message. Show a screenshot where this information is present.

## Fourth Part Answers

### Exercise 1

- a) For this exercise I will use the GitHub webpage.



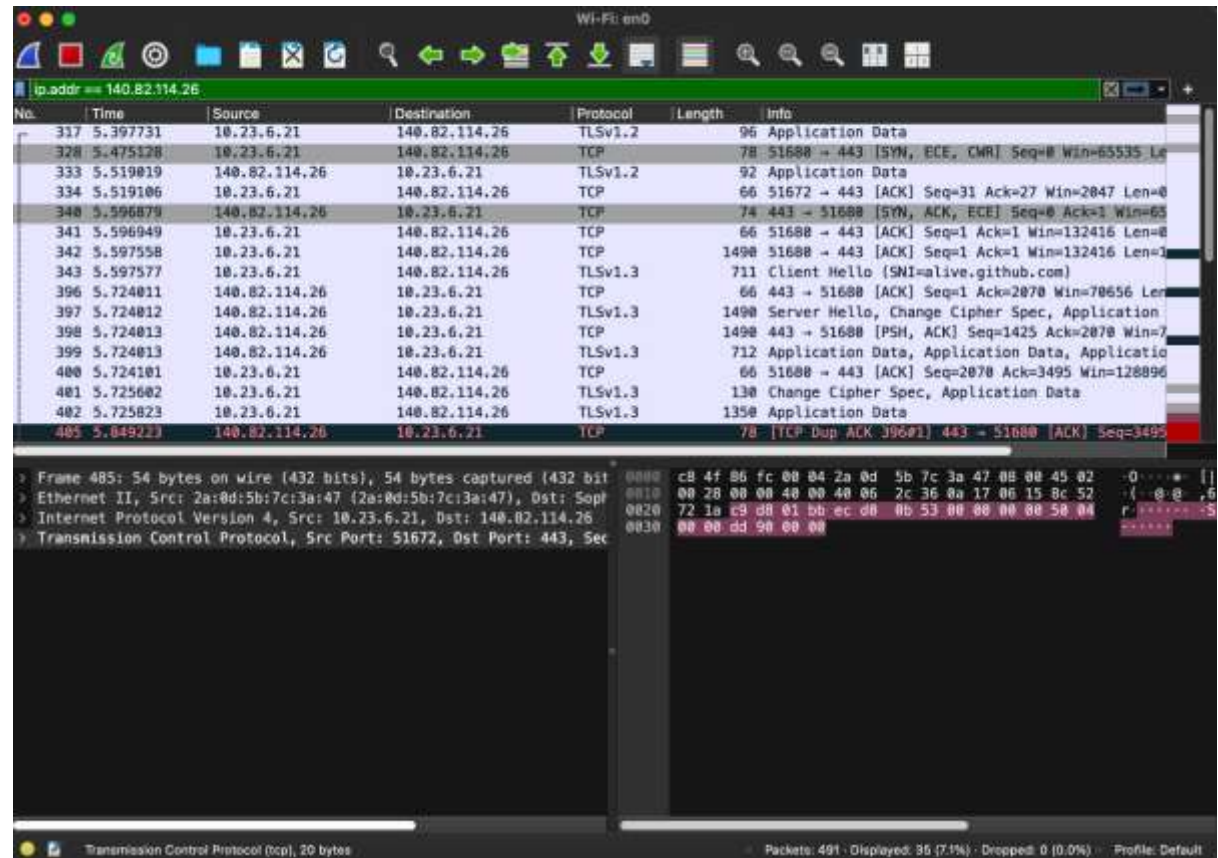
- b) To see the corresponding **IP address** to this URL I have used my terminal and ping [www.github.com](https://www.github.com).

```

nicolasdalessandro — -bash — 80x24
(base) Nicolass-MacBook-Pro-8:~ nicolasdalessandro$ ping www.github.com
PING github.com (140.82.121.4): 56 data bytes
64 bytes from 140.82.121.4: icmp_seq=0 ttl=47 time=40.351 ms
64 bytes from 140.82.121.4: icmp_seq=1 ttl=47 time=49.623 ms
64 bytes from 140.82.121.4: icmp_seq=2 ttl=47 time=44.488 ms
64 bytes from 140.82.121.4: icmp_seq=3 ttl=47 time=56.836 ms
64 bytes from 140.82.121.4: icmp_seq=4 ttl=47 time=48.478 ms
^C

```

- c) Then I used a filter **ip.addr** with the obtained IP to find the packages corresponding to the communication between my device and this URL



Wi-Fi: en0

ip.addr == 140.82.114.26

No.	Time	Source	Destination	Protocol	Length	Info
317	5.397731	10.23.6.21	140.82.114.26	TLSv1.2	96	Application Data
328	5.475128	10.23.6.21	140.82.114.26	TCP	78	51688 → 443 [SYN, ECE, CMR] Seq=0 Win=65535 Len=0
333	5.519019	140.82.114.26	10.23.6.21	TLSv1.2	92	Application Data
334	5.519106	10.23.6.21	140.82.114.26	TCP	66	51672 → 443 [ACK] Seq=31 Ack=27 Win=2047 Len=0
340	5.596879	140.82.114.26	10.23.6.21	TCP	74	443 → 51688 [SYN, ACK, ECE] Seq=0 Ack=1 Win=65535 Len=0
341	5.596949	10.23.6.21	140.82.114.26	TCP	66	51688 → 443 [ACK] Seq=1 Ack=1 Win=132416 Len=0
342	5.597558	10.23.6.21	140.82.114.26	TCP	1490	51688 → 443 [ACK] Seq=1 Ack=1 Win=132416 Len=0
343	5.597577	10.23.6.21	140.82.114.26	TLSv1.3	711	Client Hello (SNI=alive.github.com)
396	5.724011	140.82.114.26	10.23.6.21	TCP	66	443 → 51688 [ACK] Seq=1 Ack=2070 Win=70656 Len=0
397	5.724012	140.82.114.26	10.23.6.21	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application Data
398	5.724013	140.82.114.26	10.23.6.21	TCP	1490	443 → 51688 [PSH, ACK] Seq=1425 Ack=2070 Win=70656 Len=0
399	5.724013	140.82.114.26	10.23.6.21	TLSv1.3	712	Application Data, Application Data, Application Data
400	5.724101	10.23.6.21	140.82.114.26	TCP	66	51688 → 443 [ACK] Seq=2070 Ack=3495 Win=128896 Len=0
401	5.725602	10.23.6.21	140.82.114.26	TLSv1.3	130	Change Cipher Spec, Application Data
402	5.725823	10.23.6.21	140.82.114.26	TLSv1.3	1350	Application Data
485	5.849223	140.82.114.26	10.23.6.21	TCP	78	[TCP Dup ACK 396#1] 443 → 51688 [ACK] Seq=3495 Len=0

Frame 485: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0

Ethernet II, Src: 2a:0d:5b:7c:3a:47 (2a:0d:5b:7c:3a:47), Dst: Sophos

Internet Protocol Version 4, Src: 10.23.6.21, Dst: 140.82.114.26

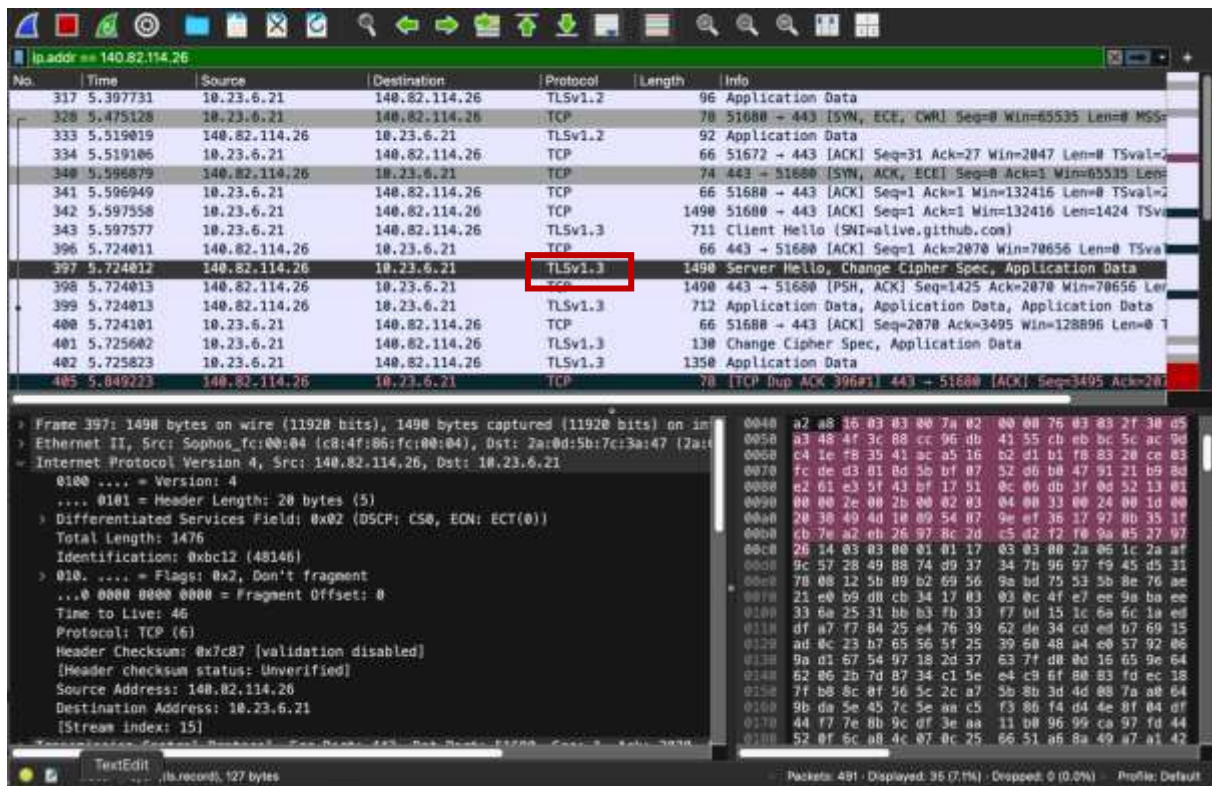
Transmission Control Protocol, Src Port: 51672, Dst Port: 443, Seq: 3495, Win: 0, Len: 0

Transmission Control Protocol (tcp), 20 bytes

Packets: 491 · Displayed: 36 (7.1%) · Dropped: 0 (0.0%) · Profile: Default



## Exercise 2



The application layer protocol that is used in this case is TLS which means Transport Layer Security. Most of the modern websites uses HTTPS instead of HTTP. HTTPS is basically HTTP inside a TLS-encrypted connection.

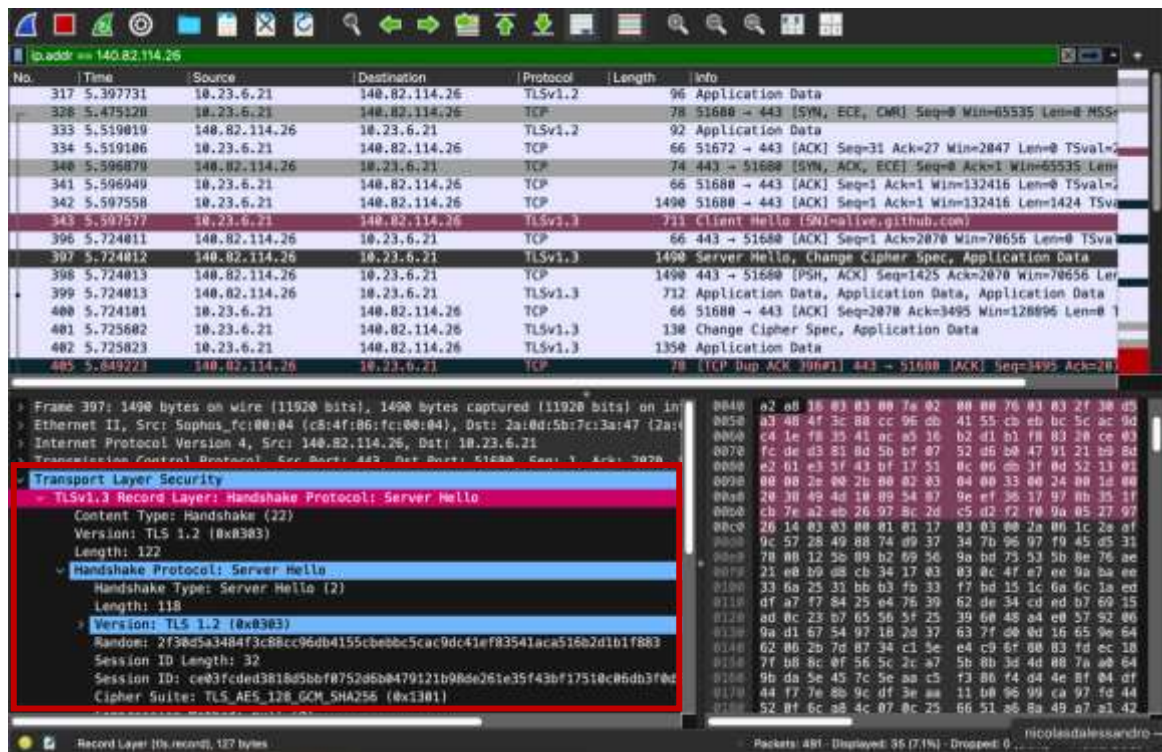
The main Features of the TLS protocol are:

- Encryption**, because the data is being hide (like in our example the username and password) and no one can see it in plain text. To do this, TLS uses asymmetric encryption (public/private keys session keys), and during the handshake the client and server exchange those keys to ensure a secure communication channel, where all the data that is transmitted will be encrypted.
- Authentication**, because it verifies that we are indeed communicating with the expected website and not to a fake one. To do this, TLS requires server to have digital certificates signed by trusted Certificate Authorities. So, in the communication, the client validates the certificate checking its digital signature.
- Integrity**, because it ensures that the information is not changed or modified when is being transmitted. To do this, TLS uses the MAC code already explained in part 1. The sender calculates a MAC (the cryptographic checksum) using a secret key and includes it in the message. The receiver checks the MAC to verify that the data was not changed during the transmission.

So, in our example, the website uses HTTPS, meaning that the login information is inside the TLS application data packets, and hence they are encrypted.



### Exercise 3



Wireshark packet capture showing a TLS handshake. The packet list shows a sequence of packets from 140.82.114.26 to 10.23.6.21. The packet details pane is expanded to show the TLSv1.3 Record Layer: Handshake Protocol: Server Hello. The packet bytes pane shows the raw data of the handshake message.

Frame 397: 1490 bytes on wire (11920 bits), 1490 bytes captured (11920 bits) on interface 0

Ethernet II, Src: Sophos, Prio: 0x00000000 (c8:4f:86:fc:00:00), Dst: 2a:00:5b:7c:3a:47 (2a:00:5b:7c:3a:47)

Internet Protocol Version 4, Src: 140.82.114.26, Dst: 10.23.6.21

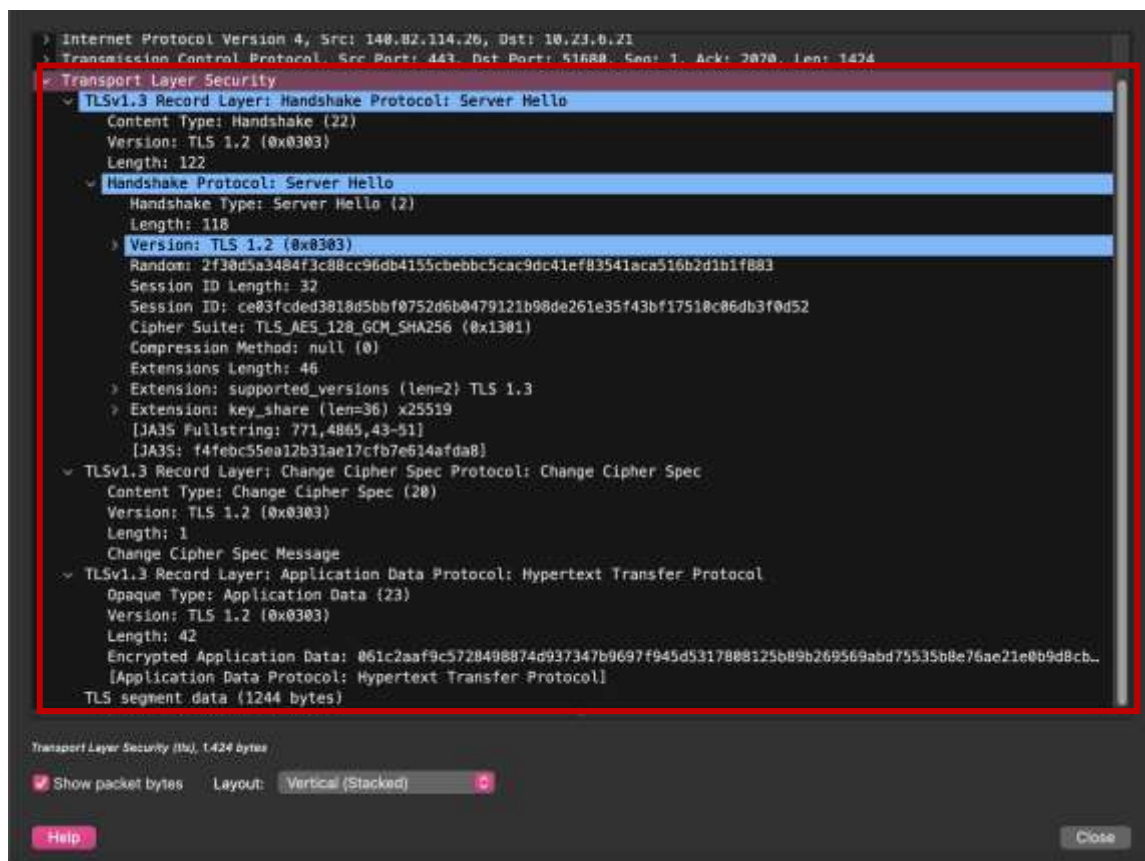
Transmission Control Protocol, Src Port: 443, Dst Port: 51680, Seq: 1, Ack: 7070, Len: 1424

Transport Layer Security

- TLSv1.3 Record Layer: Handshake Protocol: Server Hello
  - Content Type: Handshake (22)
  - Version: TLS 1.2 (0x0303)
  - Length: 122
- Handshake Protocol: Server Hello
  - Handshake Type: Server Hello (2)
  - Length: 118
  - Version: TLS 1.2 (0x0303)
    - Random: 2f38d5a3484f3c88cc96db4155cbebb5c5ac9dc41ef83541aca516b2d1b1f883
    - Session ID Length: 32
    - Session ID: ce03fcded3818d5bbf0752d6b0479121b98de261e35f43bf17510c06db3f0d52
    - Cipher Suite: TLS\_AES\_128\_GCM\_SHA256 (0x1301)
    - Compression Method: null (0)
    - Extensions Length: 46
    - Extension: supported\_versions (len=2) TLS 1.3
    - Extension: key\_share (len=36) x25519
      - [JA3S Fullstring: 771,4865,43-51]
      - [JA3S: f4feb55ea12b31ae17cfb7e614afda8]

Record Layer (16 records), 127 bytes

Packets: 481 - Displayed: 35 (7.1%) - Dropped: 0



Wireshark packet capture showing a TLS handshake. The packet details pane is expanded to show the TLSv1.3 Record Layer: Handshake Protocol: Server Hello. The packet bytes pane shows the raw data of the handshake message.

Internet Protocol Version 4, Src: 140.82.114.26, Dst: 10.23.6.21

Transmission Control Protocol, Src Port: 443, Dst Port: 51680, Seq: 1, Ack: 7070, Len: 1424

Transport Layer Security

- TLSv1.3 Record Layer: Handshake Protocol: Server Hello
  - Content Type: Handshake (22)
  - Version: TLS 1.2 (0x0303)
  - Length: 122
- Handshake Protocol: Server Hello
  - Handshake Type: Server Hello (2)
  - Length: 118
  - Version: TLS 1.2 (0x0303)
    - Random: 2f38d5a3484f3c88cc96db4155cbebb5c5ac9dc41ef83541aca516b2d1b1f883
    - Session ID Length: 32
    - Session ID: ce03fcded3818d5bbf0752d6b0479121b98de261e35f43bf17510c06db3f0d52
    - Cipher Suite: TLS\_AES\_128\_GCM\_SHA256 (0x1301)
    - Compression Method: null (0)
    - Extensions Length: 46
    - Extension: supported\_versions (len=2) TLS 1.3
    - Extension: key\_share (len=36) x25519
      - [JA3S Fullstring: 771,4865,43-51]
      - [JA3S: f4feb55ea12b31ae17cfb7e614afda8]

Transport Layer Security (16), 1424 bytes

Show packet bytes Layout: Vertical (Stacked)

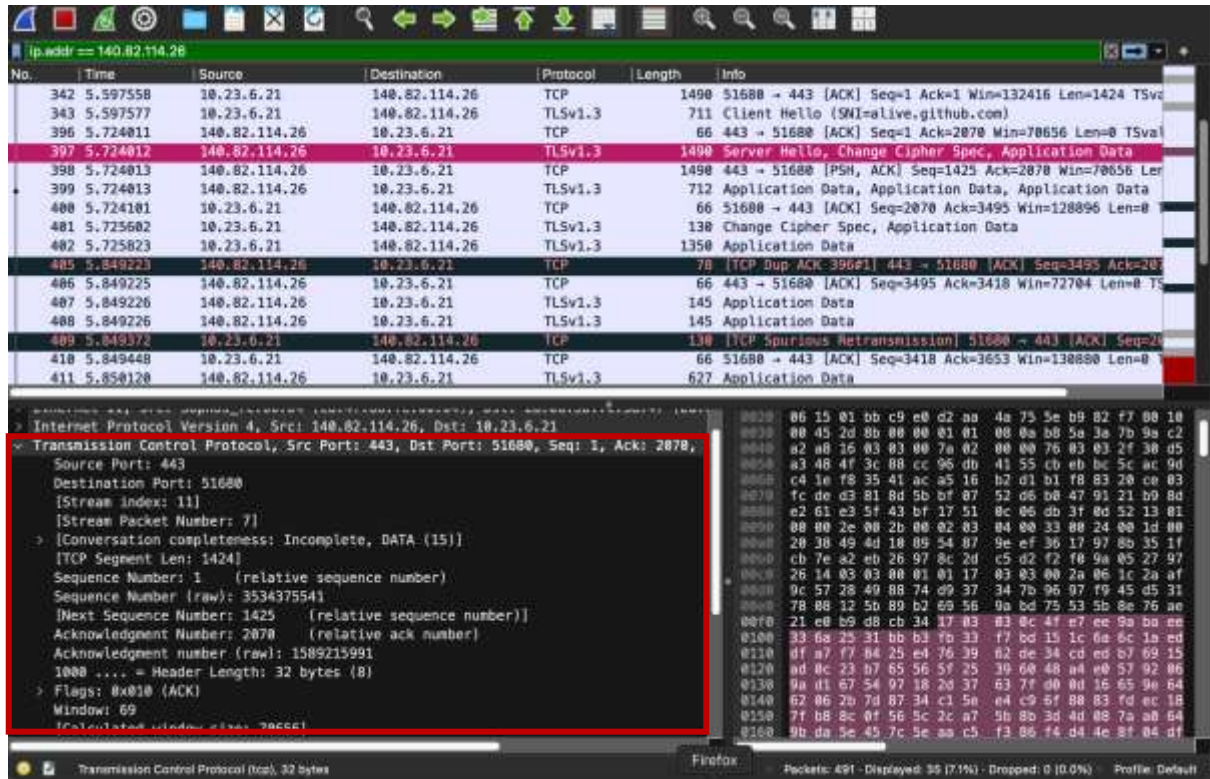
Help Close

Focusing in one specific package, I found this “Server Hello” packet, but as we can see in the image, this packet is part of the TLS handshake process. We saw in previous images different packages like the “Client Hello”, the “Change Cipher Spec” and “Application Data” that are also part of this handshake.

In the selected package and its different layer headers, we can observe these fields. These packets is a good example to demonstrate how the TLS secure the connection before any data is transmitted:

- **Using IPv4 protocol**
- **The Source and Destinations IP:** Communication happens between my devices IP and the GitHub server.
- **The TLS version:** Protocols used are TLS 1.2 or 1.3
- **Security Parameters:**
  - o Random value:  
2f30d5a3484f3c88cc96db4155cbebbc5cac9dc41ef8341aca516b2d1b1f883
  - o Session ID:  
ce03fcded3818d5bbf0752d6b0479121b98de261e35f43bf17510c06db3f0d52
  - o Selected cipher suite: TLS\_AES\_128\_GCM\_SHA256 (0x1301) ( a modern and secure encryption algorithm)
  - o Extensions include **supported\_versions** and **key\_share** for secure key exchange.
- **Handshake context:**
  - o The Server Hello message that we select is the response to a Client Hello that is not shown in this image but we saw it in the list of the first image of the part 4.
  - o The next messages are the Change Cipher Spec and encrypted Application Data
- **Security Confirmation:**
  - o Cipher Spec & Change Cipher Spec: In our case shows that the encryption is active
  - o Application Data: This contains my login request but fully encrypted Because of encryption, I cannot see my **username or password** inside the packet. This confirms that **GitHub uses HTTPS to protect login credentials**.

## Exercise 4



The image shows a Wireshark packet capture of a TLS connection. The top pane displays a list of packets, with packet 397 selected. The bottom pane shows the details of packet 397, which is a TCP segment (Seq: 1, Ack: 2070, Len: 1424) containing TLS data. The packet is from 10.23.6.21 to 140.82.114.26.

No.	Time	Source	Destination	Protocol	Length	Info
342	5.597558	10.23.6.21	140.82.114.26	TCP	1490	51680 → 443 [ACK] Seq=1 Ack=1 Win=132416 Len=1424 TSval=711
343	5.597577	10.23.6.21	140.82.114.26	TLSv1.3	711	Client Hello (SN=alive.github.com)
396	5.724811	140.82.114.26	10.23.6.21	TCP	66	443 → 51680 [ACK] Seq=1 Ack=2070 Win=70656 Len=0 TSval=1490
397	5.724812	140.82.114.26	10.23.6.21	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application Data
398	5.724813	140.82.114.26	10.23.6.21	TCP	1490	443 → 51680 [PSH, ACK] Seq=1425 Ack=2070 Win=70656 Len=712
399	5.724813	140.82.114.26	10.23.6.21	TLSv1.3	712	Application Data, Application Data, Application Data
400	5.724811	10.23.6.21	140.82.114.26	TCP	66	51680 → 443 [ACK] Seq=2070 Ack=3495 Win=128896 Len=0
401	5.725602	10.23.6.21	140.82.114.26	TLSv1.3	130	Change Cipher Spec, Application Data
402	5.725823	10.23.6.21	140.82.114.26	TLSv1.3	1350	Application Data
405	5.849223	140.82.114.26	10.23.6.21	TCP	78	[TCP Dup ACK 396#1] 443 → 51680 [ACK] Seq=3495 Ack=2070
406	5.849225	140.82.114.26	10.23.6.21	TCP	66	443 → 51680 [ACK] Seq=3495 Ack=3418 Win=72704 Len=0 TSval=66
407	5.849226	140.82.114.26	10.23.6.21	TLSv1.3	145	Application Data
408	5.849226	140.82.114.26	10.23.6.21	TLSv1.3	145	Application Data
409	5.849372	10.23.6.21	140.82.114.26	TCP	130	[TCP Spurious Retransmission] 51680 → 443 [ACK] Seq=2070
410	5.849448	10.23.6.21	140.82.114.26	TCP	66	51680 → 443 [ACK] Seq=3418 Ack=3653 Win=138880 Len=0
411	5.850128	140.82.114.26	10.23.6.21	TLSv1.3	627	Application Data

Internet Protocol Version 4, Src: 140.82.114.26, Dst: 10.23.6.21

- Transmission Control Protocol, Src Port: 443, Dst Port: 51680, Seq: 1, Ack: 2070, Len: 1424
  - Source Port: 443
  - Destination Port: 51680
  - [Stream Index: 11]
  - [Stream Packet Number: 7]
  - [Conversation completeness: Incomplete, DATA (15)]
  - [TCP Segment Len: 1424]
  - Sequence Number: 1 (relative sequence number)
  - Sequence Number (raw): 3534375541
  - [Next Sequence Number: 1425 (relative sequence number)]
  - Acknowledgment Number: 2070 (relative ack number)
  - Acknowledgment number (raw): 1589215991
  - 1000 .... = Header Length: 32 bytes (8)
  - Flags: 0x010 (ACK)
  - Window: 69
  - [Calculated window size: 70656]

Transmission Control Protocol (tcp), 32 bytes

Packets: 491 - Displayed: 35 (7.1%) - Dropped: 0 (0.0%) - Profile: Default

As we observe in the previous image the transport layer protocol used by the security protocol TLS is TCP. This is clear in the packet information where we can see **Transmission Control Protocol, Src Port: 443, Dst Port: 51680, Seq: 1, Ack: 2070, Len: 1424**.

This is usually the appropriate choice for TLS because is reliable and ensures that the data arrives in the correct order and with no errors.

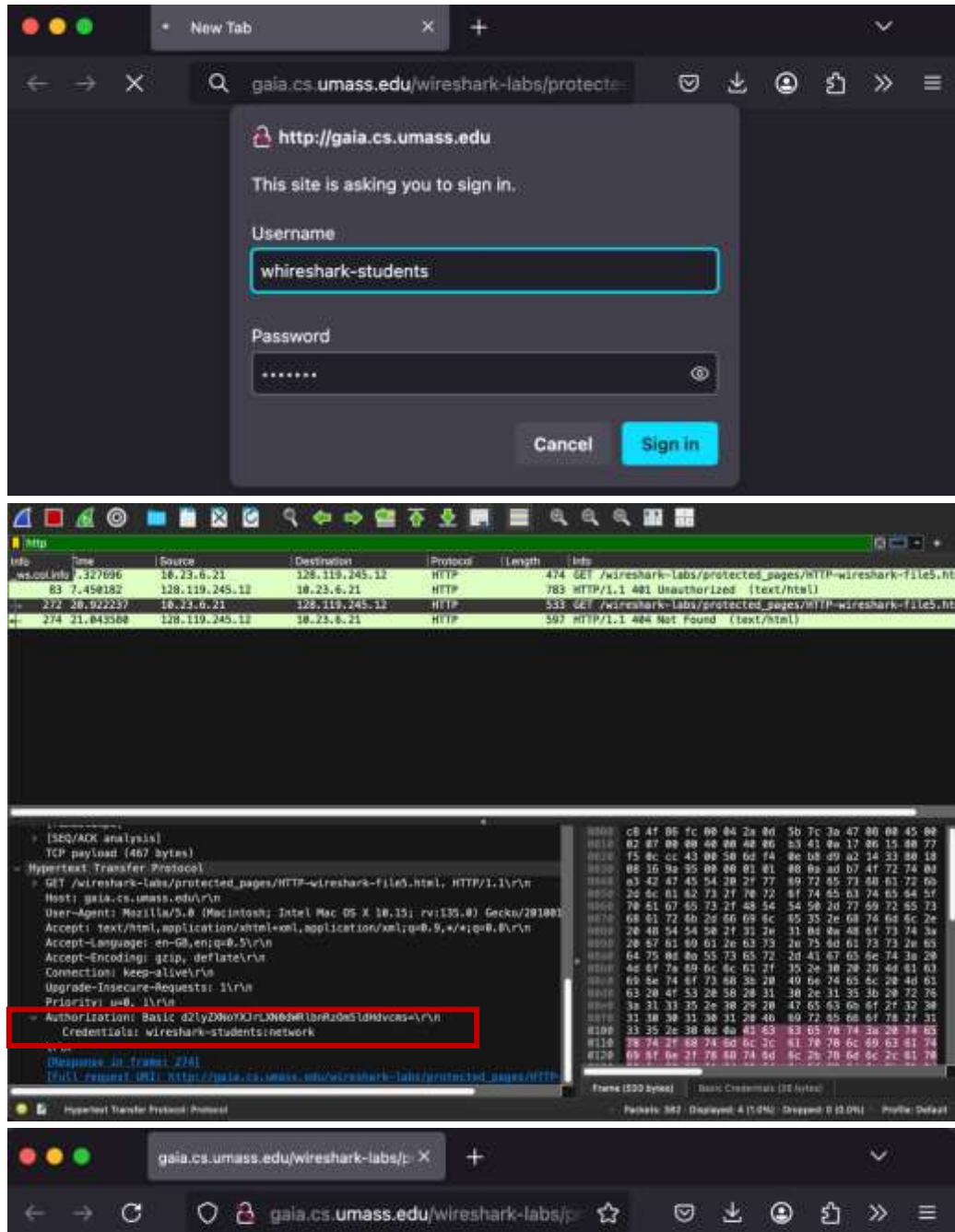
The operation we are checking is the logging, so it is very important to ensure that all the packets arrive correctly (so, UDP will not be the best option).

We can observe also that the port 443 (standard for HTTPS as we already mentioned) is confirming a secure web connection.



## Exercise 5

Following the requested process, we can see how using the HTTP protocol we can observe the credentials used for the logging process:



The screenshot shows a web browser window at `gaia.cs.umass.edu/wireshark-labs/protected` with a login form. The username field is filled with `whireshark-students` and the password field is masked with dots. Below the browser, Wireshark captures the HTTP traffic. The packet list shows an HTTP GET request to `/wireshark-labs/protected_pages/HTTP-wireshark-file5.html`. The packet details pane shows the request headers, including the `Authorization: Basic d2lyZXN0ZXJlbnRlbnR0dS1ldHVsYmcs` header, which is highlighted with a red box. The packet bytes pane shows the raw data of the request.

This page is password protected! If you're seeing this, you've downloaded the page correctly  
Congratulations!