

# Sentinel

## Networking - 5 Layers and Wireshark

DEFENDING OUR DIGITAL WAY OF LIFE

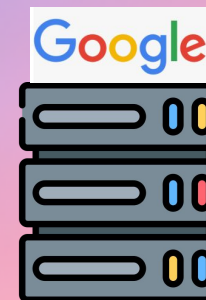
# From our last lesson....

- We talked about:  
Protocols  
Ports  
HTTP

# Let's send a message



**Hello!**



But how does it actually look?

# Any problems you can think of?

- How are we going to physically represent the data?
- How is it going to arrive at the other computer?
- What if the other computer is in Spain?
  - Or on Mars?



# Analogy - sending a real package

- I want to send a package abroad
- I put it in a parcel with the destination address
- The courier service puts it in another box (courier's package) with more information
- The international shipping service puts it in a container



# Analogy - sending a real package

- The container is shipped across the sea
- The international shipping service on the other side unloads it from the container
- The courier service unboxes the courier package and delivers the parcel
- The parcel arrives at its destination



# The encapsulation principle

- Each layer on the sender's side gets something from the previous layer, and wraps it in extra package
  - The courier service puts my small parcel in a big package with extra address details
  - The international shipping company puts the package in a container
- Each layer on the receiver's side opens the relevant package, and passes its content to the next layer (in reverse order)
  - The international shipping company unloads the package from a container
  - The courier service takes out the parcel from the courier package and delivers it



# The abstraction principle

- Each layer on the way is unaware of the other layers
  - The sender and the recipient don't care about the big container
  - The courier service doesn't know what's in the small parcel
  - The international shipping company doesn't care about the what's in the package
- Each layer could be replaced without affecting the others
  - The package could be sent by airplane instead of shipping, but I don't have to know it



# So how is it working in sending a message between computers?

- The process is very similar.
- Let's see

# Header and payload

- Payload - the actual data we want to send
- Header - metadata, more information we need to send it properly
- In this case, the message “Hello!” is the payload



**Hello!**



# Ports

- We need to specify the source and destination ports!
- So we take the payload, and add before it more information - the source and destination ports



Src port: 1234  
Dst port: 80

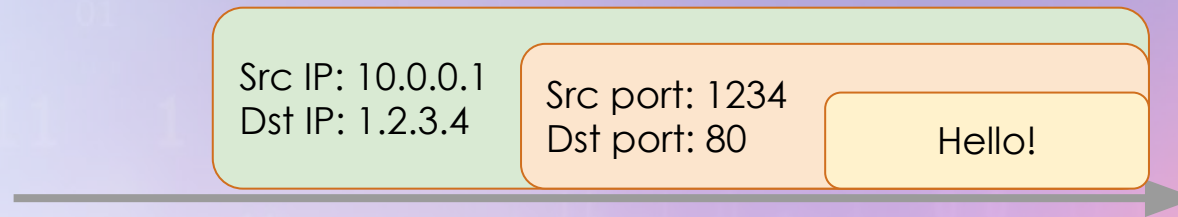
Hello!





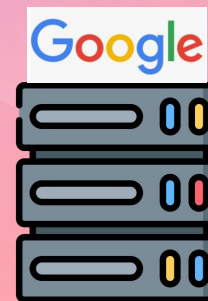
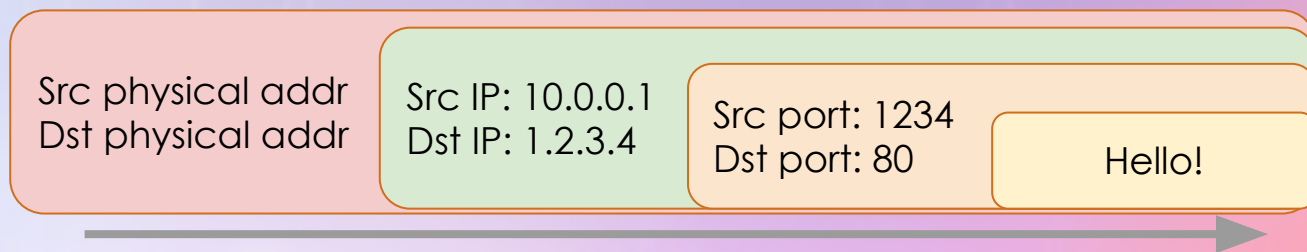
# IPs

- We also need to specify the source and destination IPs!
- So now our payload is the src/dst port information + the HTTP request
- We take this payload and add a header



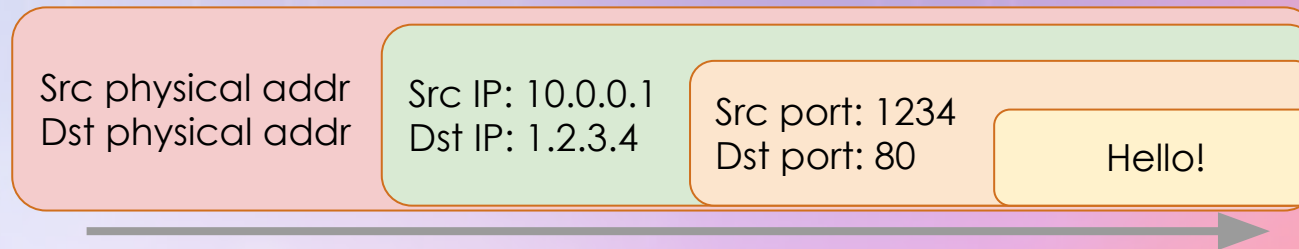
# Physical address

- Before our computer sends the information, it has to know the physical address of the device it sends the message to
- If we are at home, it will be our home Wi-Fi router
- So this is the last header we add



# Who adds the headers?

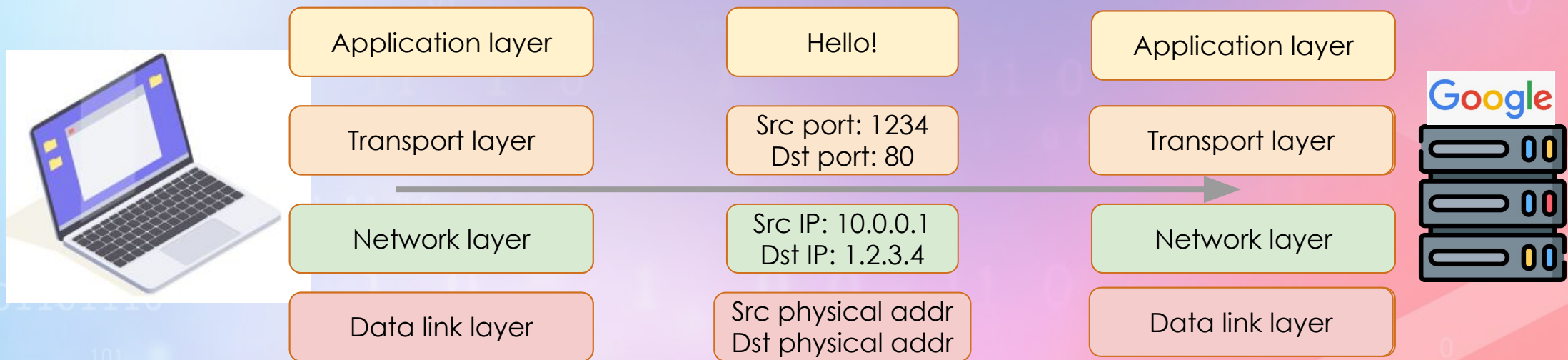
- We don't want to add ourselves all of this information to each message we send
- We want our computer (operating system) to handle it for us





# The special programs that add our headers

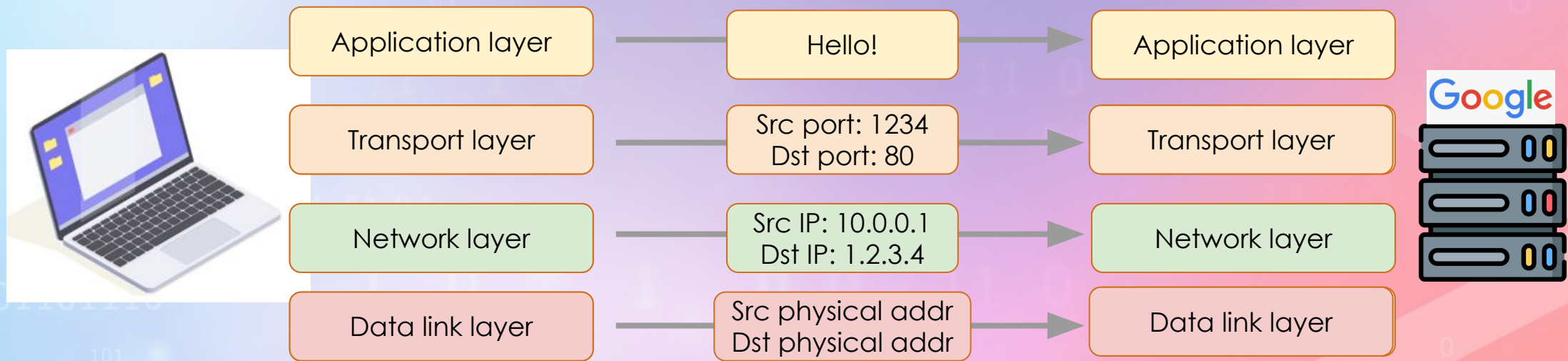
- For each one of these headers, there is a special program that knows how to read and write them, and what to do with this information.
- Let's give them nicer names



# The abstraction principle

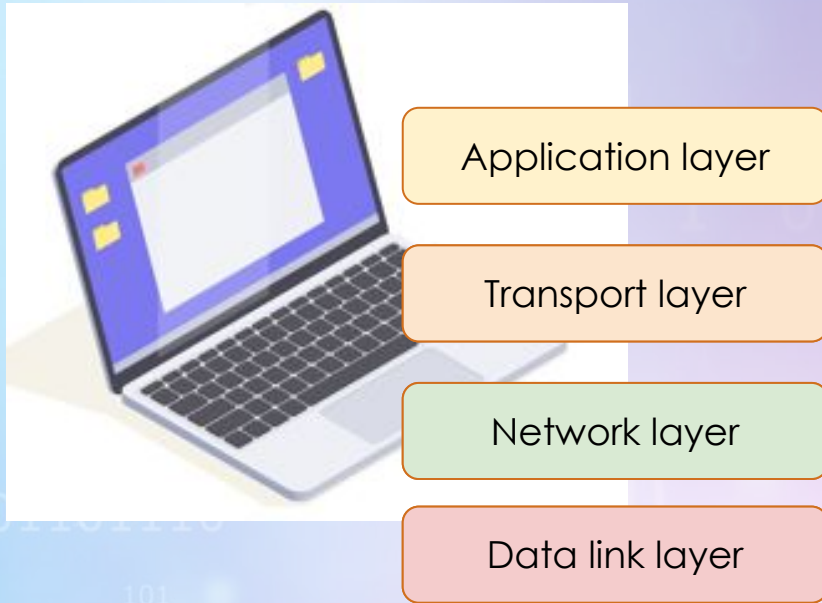
- Each layer is only aware of the equivalent layer on the other side
- For example: the application layer is thinking that it is talking “directly” with the application layer on the other side.

Just like in the “package in the mail” example



# So how is it actually working?

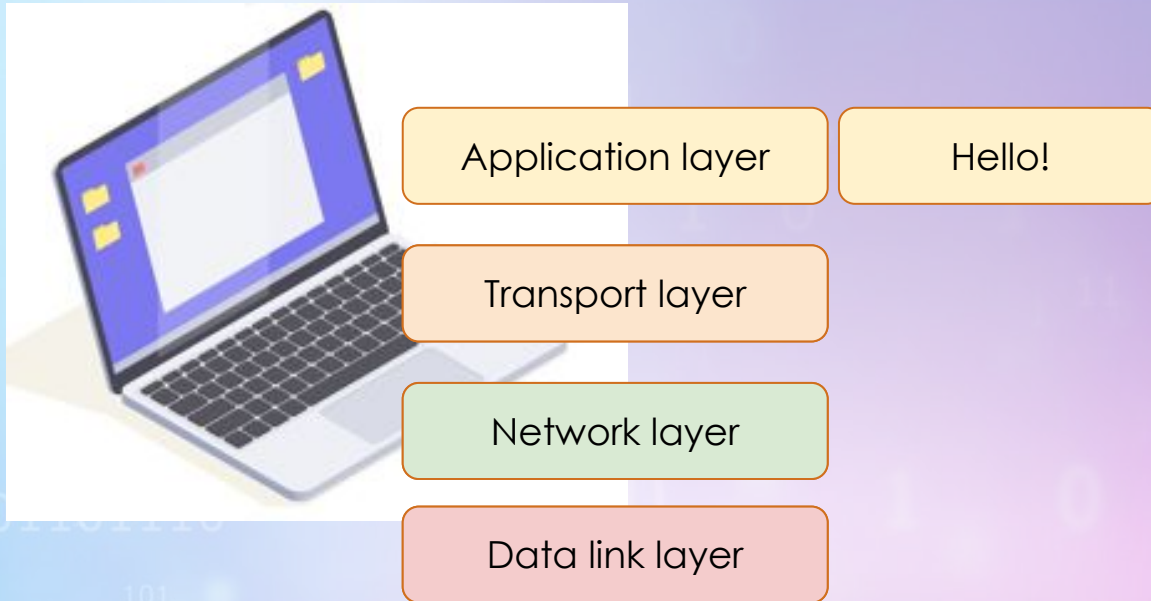
- Let's see what happens when the client wants to send a message.





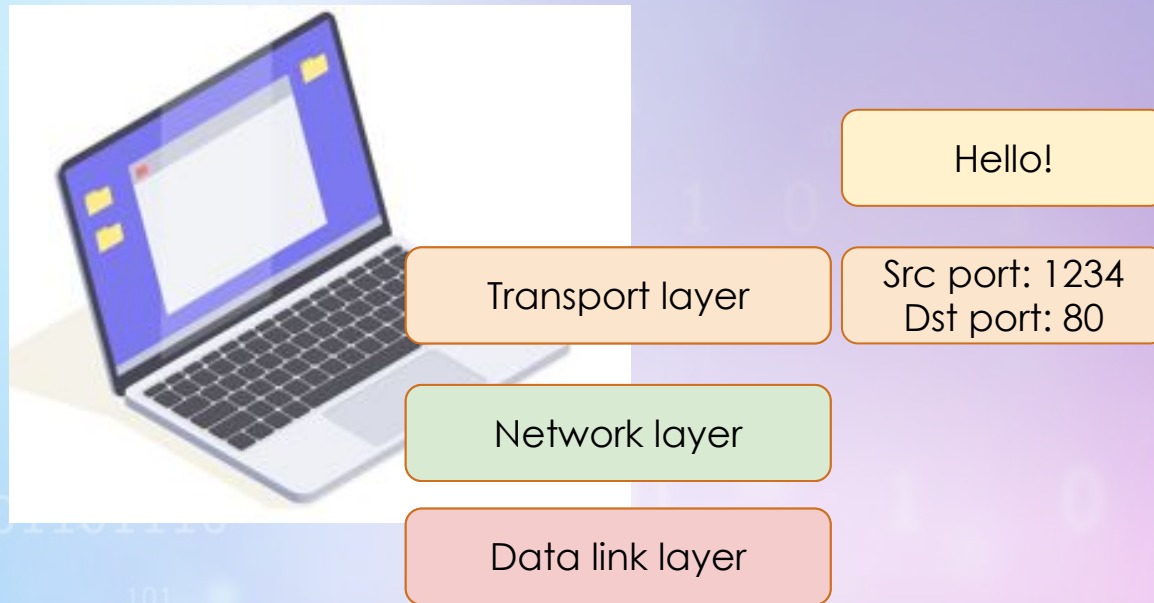
# Application layer - client

- The top layer is the application layer. It can be a web client, a mail client, a chat client, any type of application that sends and receives messages.
- Let's say our application wants to send just the message "Hello!"



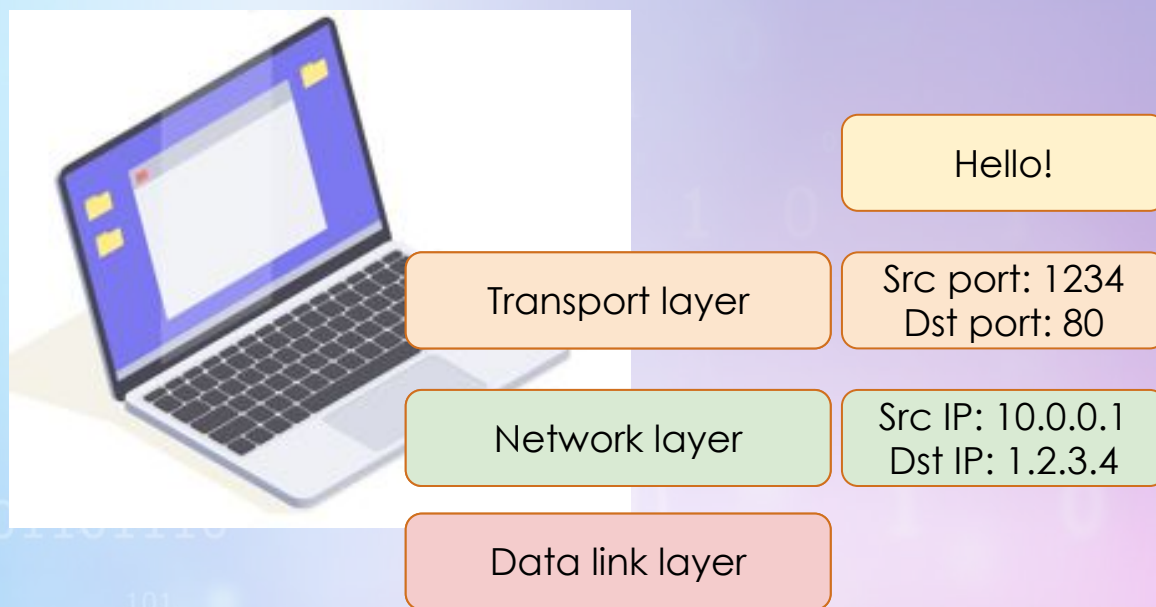
# Transport layer-client

- The application layer gives the data to the transport layer, which adds the correct port numbers



# Network layer - client

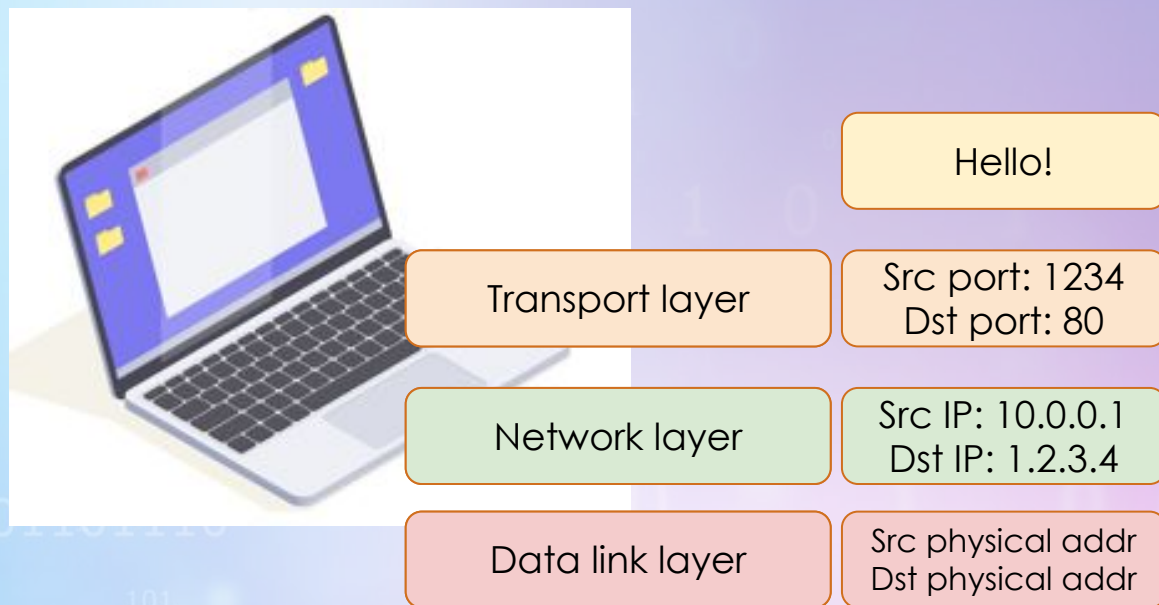
- Then, the transport layer gives it to the network layer, which adds the correct source and destination IPs





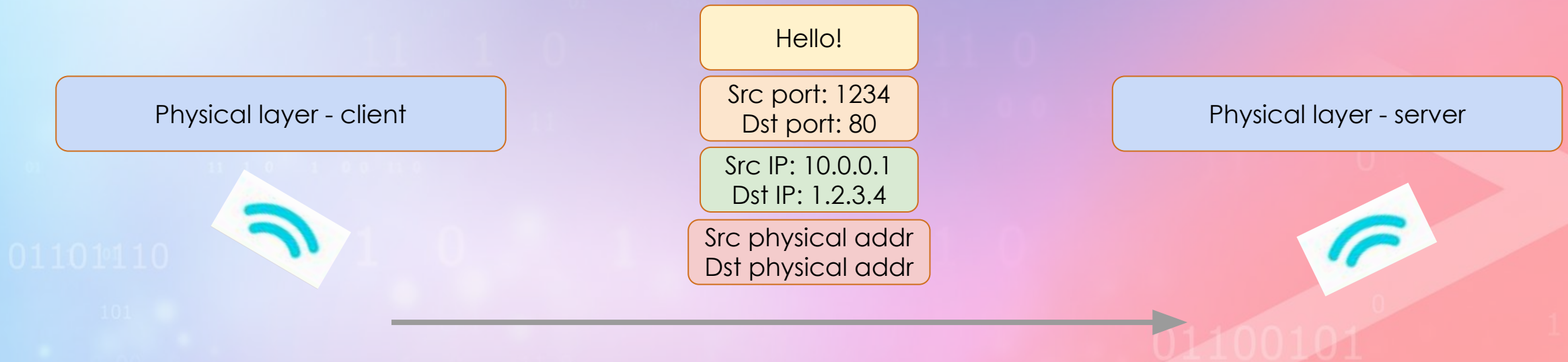
# Data link layer – client

- Then, the network layer gives it to the data link layer, together with the IP of the actual device to which the message should be sent to. The data link layer then adds its physical address



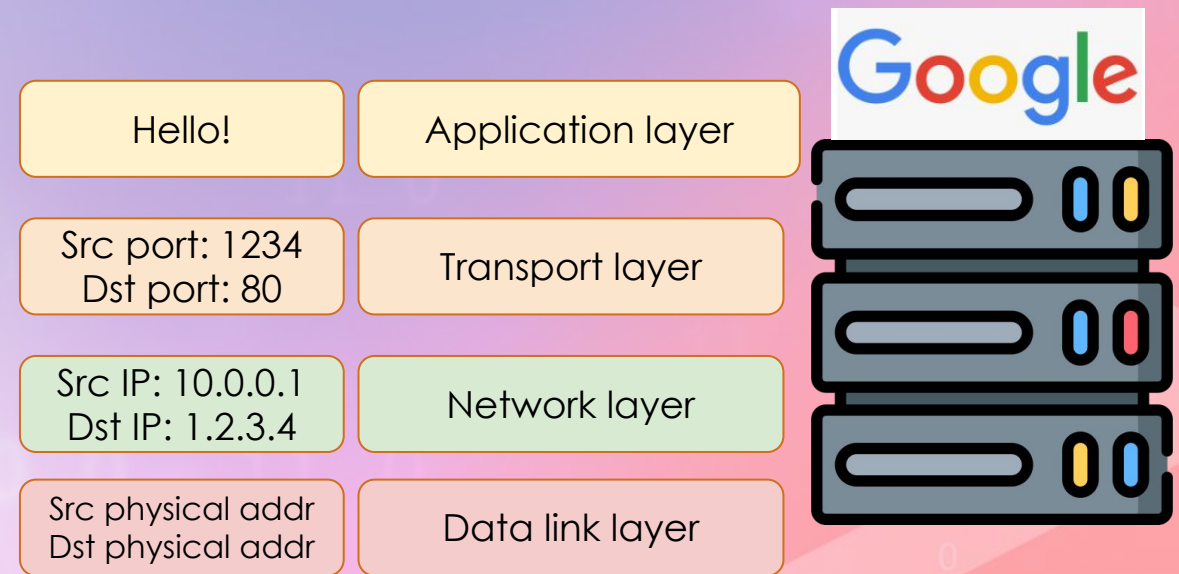
# Physical layer

- Then, the data link layer gives the complete message to the physical layer, which sends it as electronic information on a wire, or wireless radio waves



# Data link layer - server

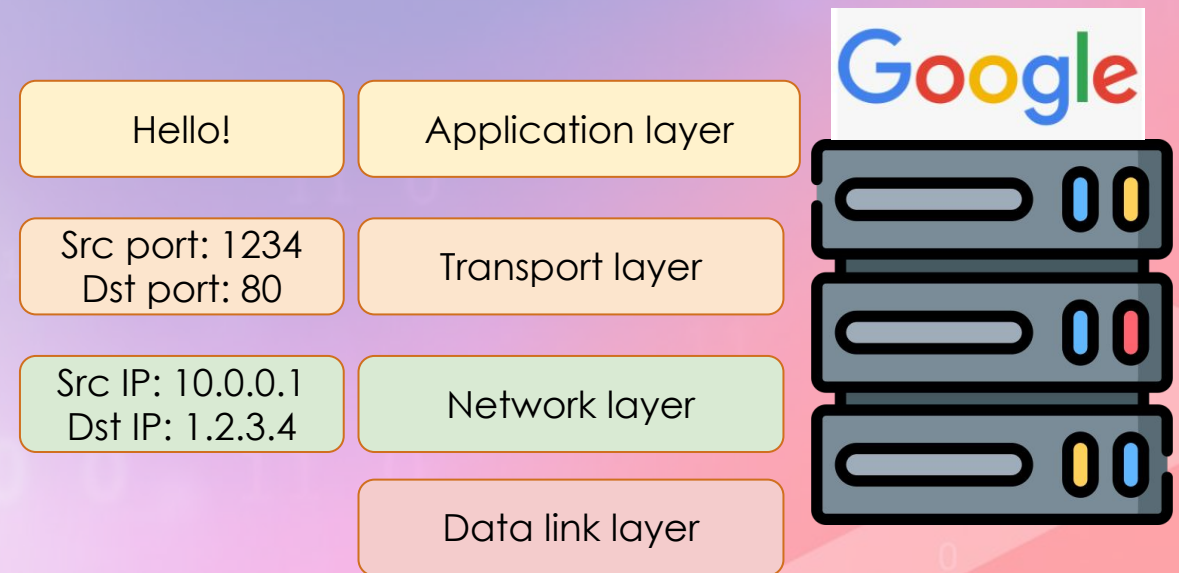
- The message is sent and (eventually) arrives at the server.
- The server's data link layer reads the physical address, and checks if this message is intended for the server





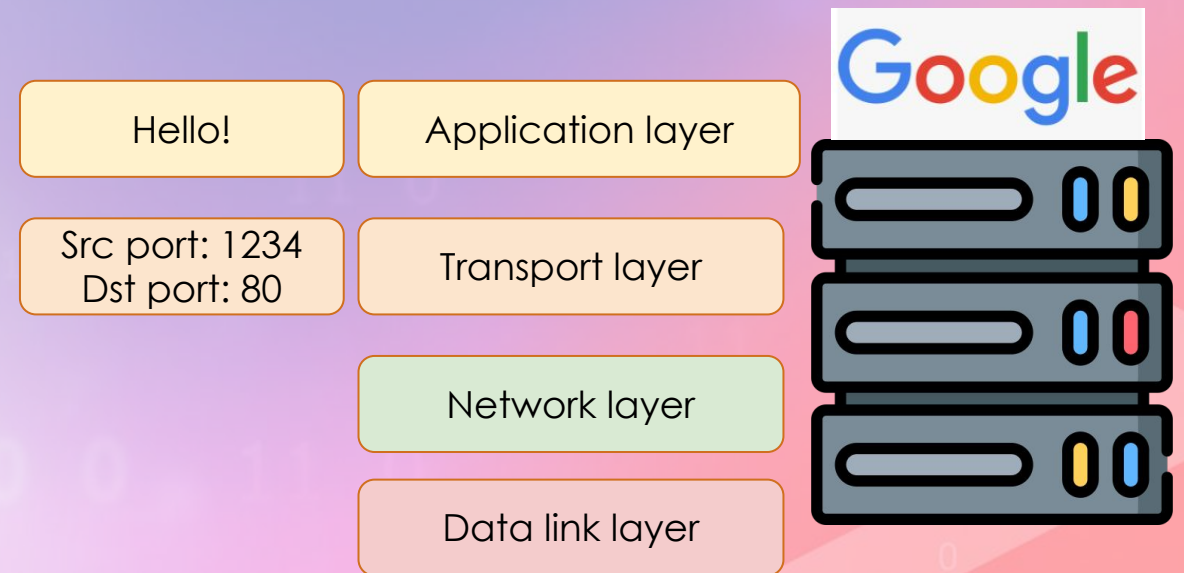
# Network layer – server

- The data link layer then passes the message to the network layer, which checks if the destination IP is indeed the server's IP



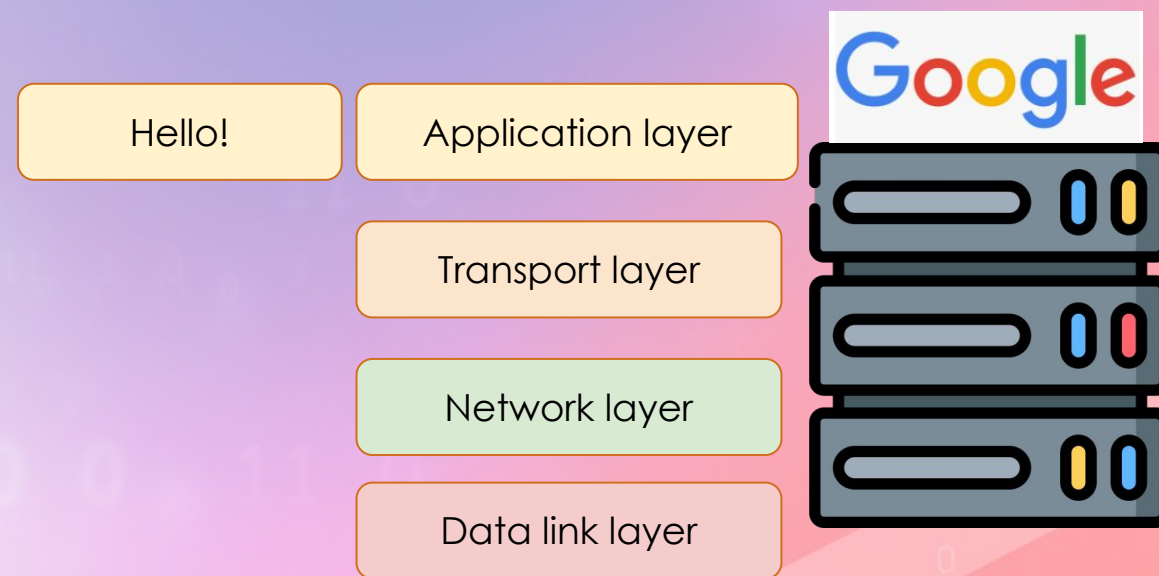
# Transport layer – server

- The network layer passes the message to the transport layer, which checks the destination port number, and passes the message to the correct application



# Application layer - server

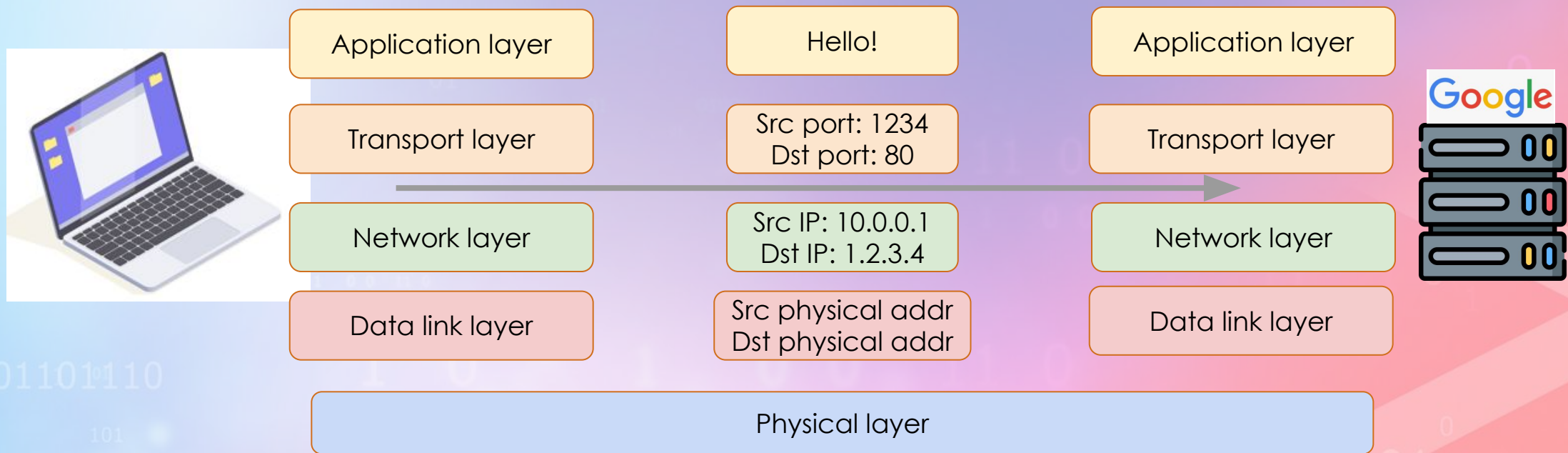
- The application reads the data and decides what to do with it - depending whether it's a web server, a mail server or any other server type





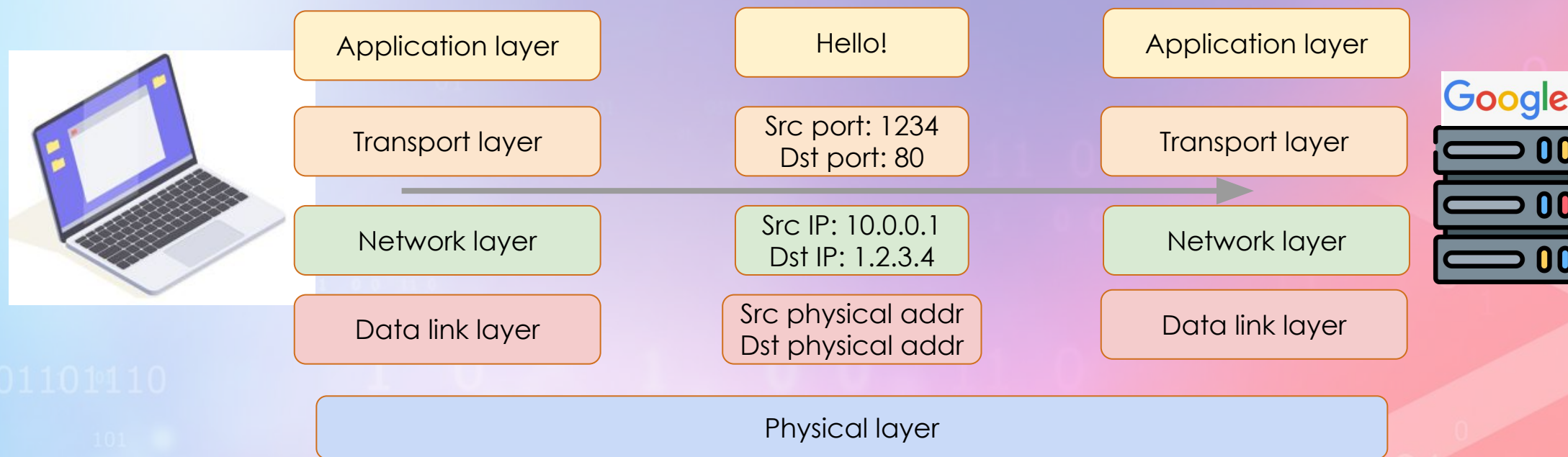
# The Layers Model

- This whole process is called the layers model.
- Each layer is responsible for a specific part of the communication process



# The encapsulation principle

- Each layer is the payload of the layer below it
- Just like the package sent in the mail (inside a box inside a container...)



# The 5 Layers

- The 5 layers are:

Application layer - in which client/server applications communicate (by protocols like HTTP)

Transport layer - allows multiple applications on the same host to send and receive data, by adding port numbers.

Network layer - responsible for making sure the data arrives at the correct destination, even if it's in a remote network, by adding IP addresses.

Data link layer - responsible for sending the data to the correct physical device (not necessarily the final destination of the data)

Physical layer - physically representing the data - for example, as electronic signals



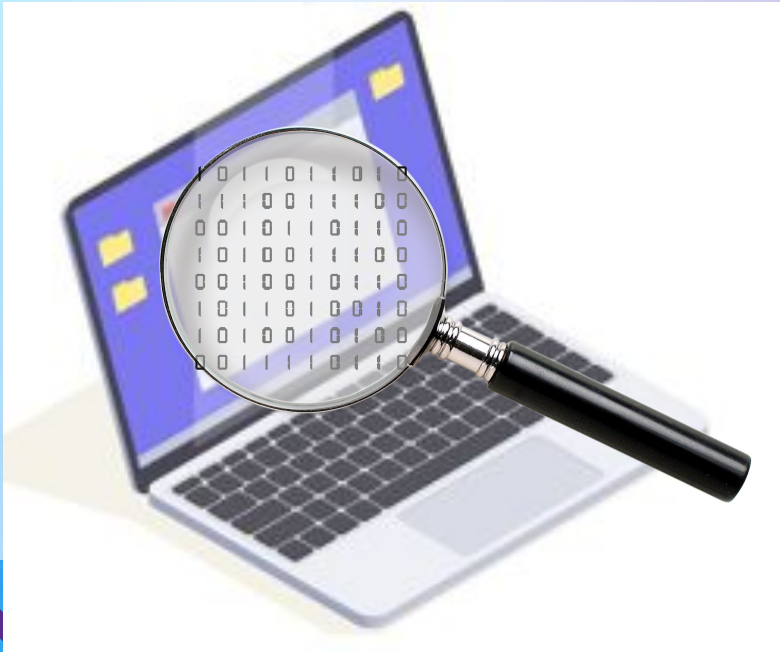
# Where is all the data going through?

- Network interface = an internal hardware component that connects a computer to a network.



# Packet Analyzer

- A **Packet Analyzer** (also known as a Sniffer) is a software that captures and logs the complete traffic that passes through a network interface.
- The leading Packet analyzer today is called **Wireshark**.



# Wireshark

- Using Wireshark, we can view and log all of the traffic that goes in & out of our computer.
- But why is it so interesting?

Analyze network attacks that had occurred (Cyber!)

Watch data that is being sent out without our awareness

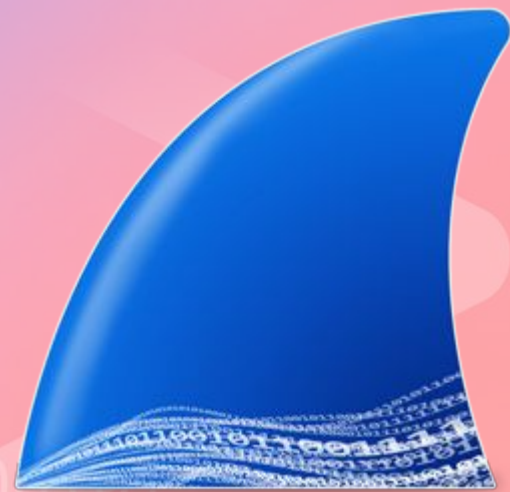
Watch and learn the magic of networks happening with your own eyes :)



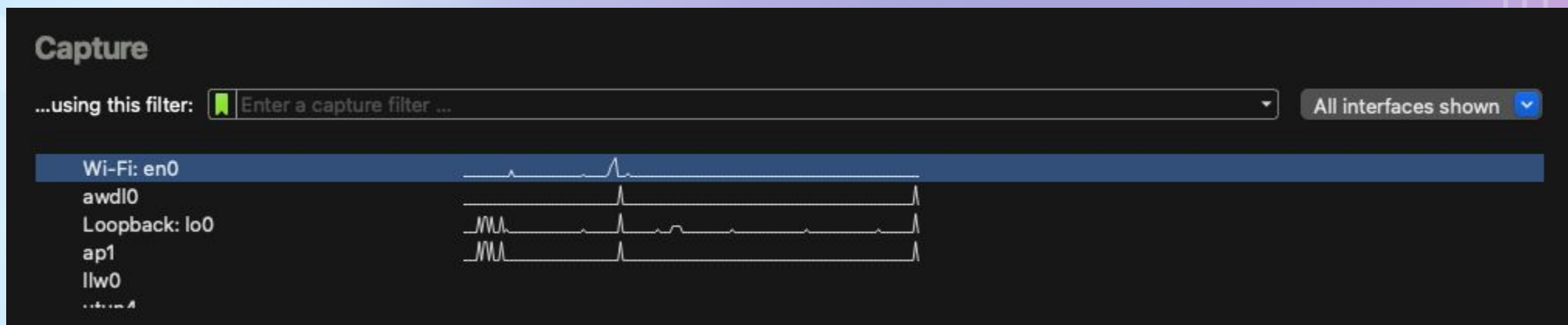


# Warning

- Sniffing may sometimes feel like looking for a needle in a haystack. Don't be afraid! It will all make sense in the end!



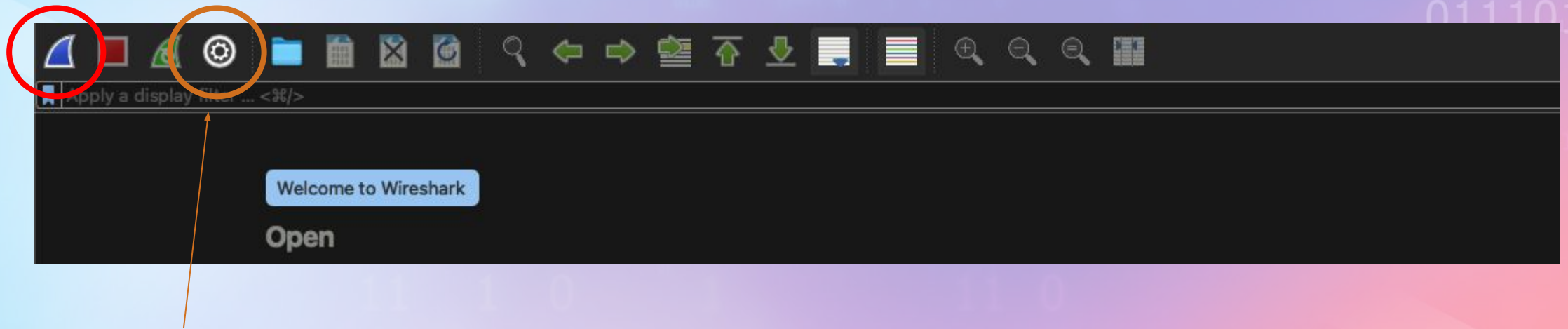
# Selecting a network interface



- You can select more than one (hold Ctrl while selecting)
- Capture filter - will capture only the filtered packets



# Starting the Capture



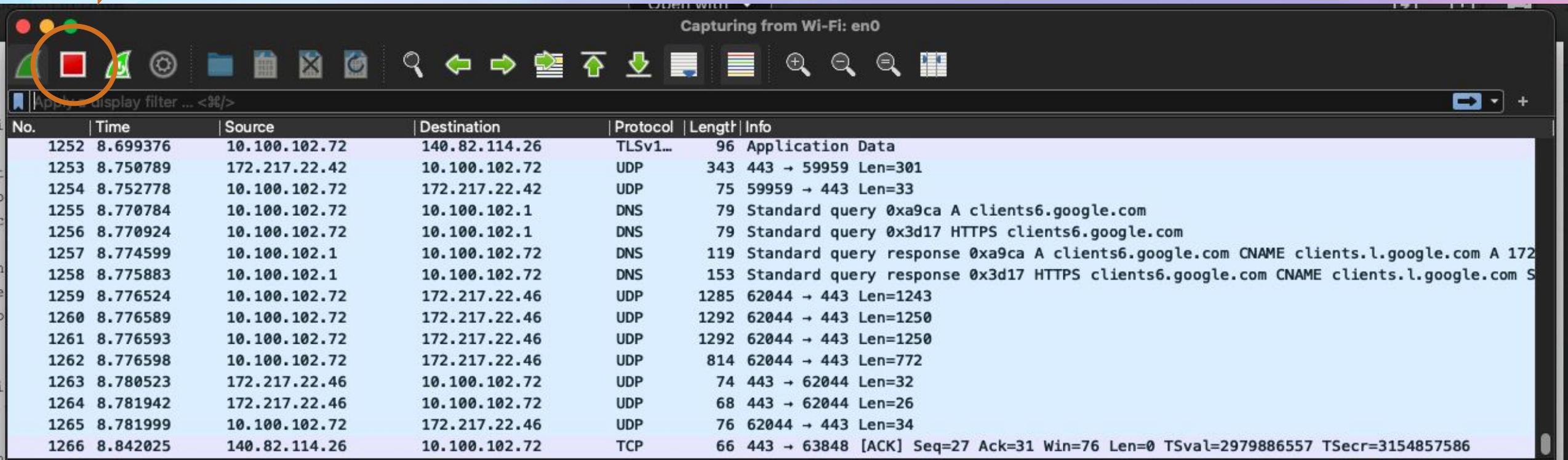
Opening the Capture Options menu (to select interface and capture filter)





# Capturing!

Stopping the capture



Capturing from Wi-Fi: en0

Apply display filter ... <%%/>

No.	Time	Source	Destination	Protocol	Length	Info
1252	8.699376	10.100.102.72	140.82.114.26	TLSv1...	96	Application Data
1253	8.750789	172.217.22.42	10.100.102.72	UDP	343	443 → 59959 Len=301
1254	8.752778	10.100.102.72	172.217.22.42	UDP	75	59959 → 443 Len=33
1255	8.770784	10.100.102.72	10.100.102.1	DNS	79	Standard query 0xa9ca A clients6.google.com
1256	8.770924	10.100.102.72	10.100.102.1	DNS	79	Standard query 0x3d17 HTTPS clients6.google.com
1257	8.774599	10.100.102.1	10.100.102.72	DNS	119	Standard query response 0xa9ca A clients6.google.com CNAME clients.l.google.com A 172
1258	8.775883	10.100.102.1	10.100.102.72	DNS	153	Standard query response 0x3d17 HTTPS clients6.google.com CNAME clients.l.google.com S
1259	8.776524	10.100.102.72	172.217.22.46	UDP	1285	62044 → 443 Len=1243
1260	8.776589	10.100.102.72	172.217.22.46	UDP	1292	62044 → 443 Len=1250
1261	8.776593	10.100.102.72	172.217.22.46	UDP	1292	62044 → 443 Len=1250
1262	8.776598	10.100.102.72	172.217.22.46	UDP	814	62044 → 443 Len=772
1263	8.780523	172.217.22.46	10.100.102.72	UDP	74	443 → 62044 Len=32
1264	8.781942	172.217.22.46	10.100.102.72	UDP	68	443 → 62044 Len=26
1265	8.781999	10.100.102.72	172.217.22.46	UDP	76	62044 → 443 Len=34
1266	8.842025	140.82.114.26	10.100.102.72	TCP	66	443 → 63848 [ACK] Seq=27 Ack=31 Win=76 Len=0 TSval=2979886557 TSecr=3154857586

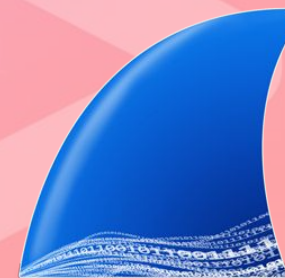


# Display Filter

Wi-Fi: en0

http

No.	Time	Source	Destination	Protocol	Length	Info
6	0.003843	17.253.122.199	10.100.102.72	HTTP	1370	HTTP/1.1 200 OK
392	2.903388	10.100.102.72	13.107.6.158	HTTP	612	GET /captiveportal/generate_204 HTTP/1.1
405	2.991980	13.107.6.158	10.100.102.72	HTTP	309	HTTP/1.1 204 No Content



# Example filters

- By protocol

http

icmp

- By IP

ip.addr == 192.168.1.1

ip.src == 192.168.1.1

ip.dst == 192.168.1.1

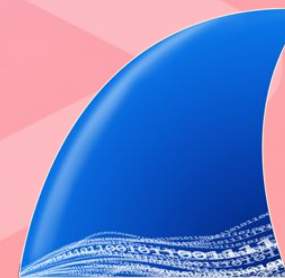
- By port

tcp.port == 80

- Combine filters

ip.addr == 192.168.1.1 and http

tcp.port == 80 or ip.dst == 8.8.8.8



# Analyze a single packet

```
Wireshark · Packet 392 · Wi-Fi: en0

> Frame 392: 612 bytes on wire (4896 bits), 612 bytes captured (4896 bits) on interface en0, id 0
> Ethernet II, Src: Apple_f2:5d:4a (50:1f:c6:f2:5d:4a), Dst: SagemcomBroa_0c:1a:b7 (34:49:5b:0c:1a:b7)
> Internet Protocol Version 4, Src: 10.100.102.72, Dst: 13.107.6.158
> Transmission Control Protocol, Src Port: 63916, Dst Port: 80, Seq: 1, Ack: 1, Len: 558
▼ Hypertext Transfer Protocol
  > GET /captiveportal/generate_204 HTTP/1.1\r\n
    Host: edge-http.microsoft.com\r\n
    Connection: keep-alive\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n

0030  10 00 2b 41 00 00 47 45 54 20 2f 63 61 70 74 69  ..+A..GE T /capti
0040  76 65 70 6f 72 74 61 6c 2f 67 65 6e 65 72 61 74  veportal /generat
0050  65 5f 32 30 34 20 48 54 54 50 2f 31 2e 31 0d 0a  e_204 HT TP/1.1..
0060  48 6f 73 74 3a 20 65 64 67 65 2d 68 74 74 70 2e  Host: ed ge-http.
0070  6d 69 63 72 6f 73 6f 66 74 2e 63 6f 6d 0d 0a 43  microsof t.com..C
0080  6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d  onnectio n: keep-
0090  61 6c 69 76 65 0d 0a 50 72 61 67 6d 61 3a 20 6e  alive..P ragma: n
00a0  6f 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d  a-ache..Cache-C
```





# Wireshark - demo

- Demo time!

