NETWORKS AND PROTOCOLS

Lab 04 - Wireshark - ARP / Introduction to scapy

Context

When higher layer protocols communicate with each other, the data flows through the layers of the OSI (Open Systems Interconnection) model and is encapsulated in a layer 2 frame. The composition of the frames depends on the type of media access. For example, if the upper layer protocols are TCP and IP and the media access is Ethernet, the encapsulation of layer 2 frames is Ethernet II. This is generally the case for a Local Area Network (LAN) environment.

When studying Layer 2 concepts, it is useful to analyse frame header information. In the first part of this tutorial, you will look at the fields in an Ethernet II frame. In the second part, you will use Wireshark to capture and analyse Ethernet II frame header fields for local and remote traffic.

Introduction

The Ethernet II protocol, also known as 'Ethernet Frame Version 2', is a defined standard for network communication used in Ethernet technologies. It is defined by the IEEE 802.3 standard and is one of the most commonly used frame formats for transmitting data on local area networks (LANs).

A few simple principles

- All stations are equal with regard to the network: there is no master equipment controlling the network.
- The access method used is distributed between all the connected equipment.
- The transmission mode is alternating bidirectional: signals travel in both directions, but not simultaneously.
- A machine can be connected or removed from the network without disrupting the operation of the whole.

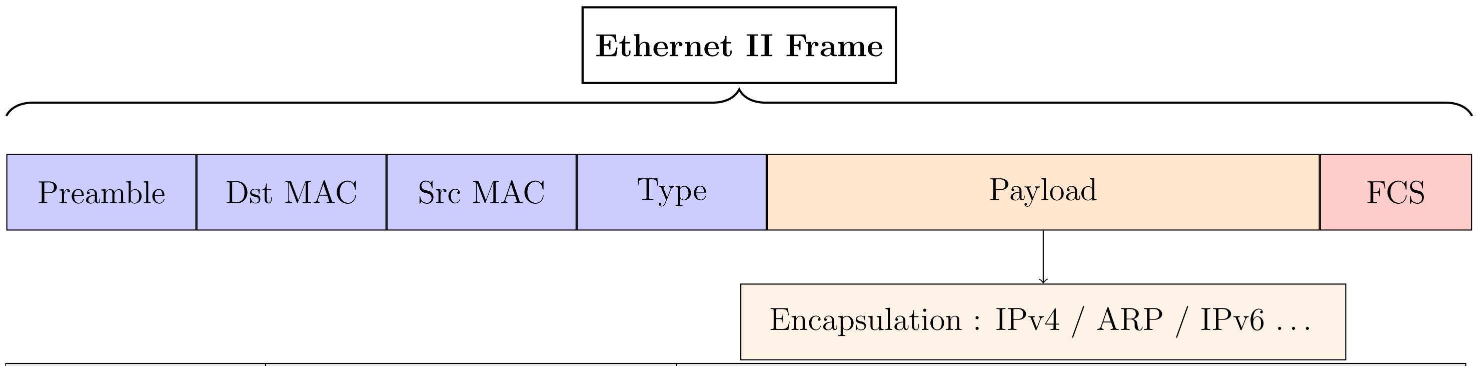
Task 1: Understanding the header fields in an Ethernet II frame

In the first part, you will examine the header fields and contents of an Ethernet II frame. A Wireshark capture will be used to examine the contents of these fields.

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.

An Ethernet II frame is a set of structured data used to transport information from one device to another. It is composed as follows:

Step 1:



Field	Length (Bytes)	Description
Preamble	Not displayed in the capture	This field contains synchronisation bits processed by the network card.
Destination MAC	6	Destination MAC address, identifying the device which is to receive the frame.
MAC source	6	Source MAC address, identifying the device sending the frame.
Type	2	Field identifying the upper layer protocol (e.g. `IPv4 = 0x0800`, `ARP = 0x0806`).
Payload	Variable (46-1500)	Useful content of the frame, often an IP or ARP packet. The minimum size is 46 bytes.
FCS (Checksum)	4	Frame Check Sequence, used to detect errors during transmission.

Explanation of frame fields

- **Preamble** is a special sequence added before the frame header to help synchronise communications between devices on a network. This field is present in all Ethernet frames,

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.

including those using the Ethernet II format, although it is not explicitly defined in the Ethernet II standard, as it is considered to be part of the physical layer.

It consists of two main parts:

- Preamble proper: A sequence of 7 bytes, 10101010 (in binary) repeated 7 times, i.e. a total of 56 bits.
- SFD (Start Frame Delimiter): A separate final byte signalling the start of the frame, 10101011 (in binary).

The preamble is located before the destination MAC address in an Ethernet frame. Although it is an integral part of the frame, it is considered to belong to the physical layer and is not generally transmitted at link layer network software level.is a unique physical identifier assigned to each network card (or network interface) when it is manufactured. It plays a crucial role in communications on local area networks (LANs) and enables connected devices to be uniquely identified.

- **Dest & Src MAC address** is a unique physical identifier assigned to each network card (or network interface) when it is manufactured. It plays a crucial role in communications on local area networks (LANs) and enables connected devices to be uniquely identified. Its structure contains:
  - Length: A MAC address is made up of 48 bits, generally represented in the form of 12 hexadecimal characters.
  - Format: It is often written in six pairs separated by ':', '-', or without separator: Example: 00:1A:2B:3C:4D:5E or 00-1A-2B-3C-4D-5E

  It is divided into two main parts:
  - **OUI (Organizationally Unique Identifier):** The first 24 bits (first 3 bytes), it identifies the manufacturer or organisation that produced the device. Example: 00:1A:2B corresponds to Ayecom Technology Co., Ltd. More details on the officie IEEE OUI website: `https://standards-oui.ieee.org/`
  - **Interface identifier :** The next 24 bits (last 3 bytes). Unique number assigned by the manufacturer to identify the particular device.

- **EtherType** field contains a hexadecimal value to indicate the type of upper layer protocol in the data field. Many higher layer protocols are supported by Ethernet II. Here are some examples of values:
  - `0x0800`: IPv4
  - `0x0806`: ARP
  - `0x86DD`: IPv6
  - `0x8847`: MPLS Unicast
  - `0x8848`: MPLS Multicast
  - `0x0805`: X.25

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.

- **Payload** contains the useful data transmitted over the network. It is in this field that the information from a higher layer protocol (such as IP, ARP, etc.) is encapsulated for transport from one device to another. Its size is between 46 and 1500 bytes.
  - If the data is less than 46 bytes, padding is added to reach the minimum size required by the Ethernet protocol.
  - If the data to be transmitted exceeds 1500 bytes, it must be fragmented into several frames.
- **Frame check sequence** used by the network card to identify errors during transmission. The value is calculated by the sending device, encompassing the frame addresses, type and data field. It is checked by the receiver.

Task 2: Use Wireshark to capture and analyse Ethernet frames and ARP Packet

In the second part, you will use Wireshark to capture local and remote Ethernet frames. You will then examine the information contained in the frame header fields.

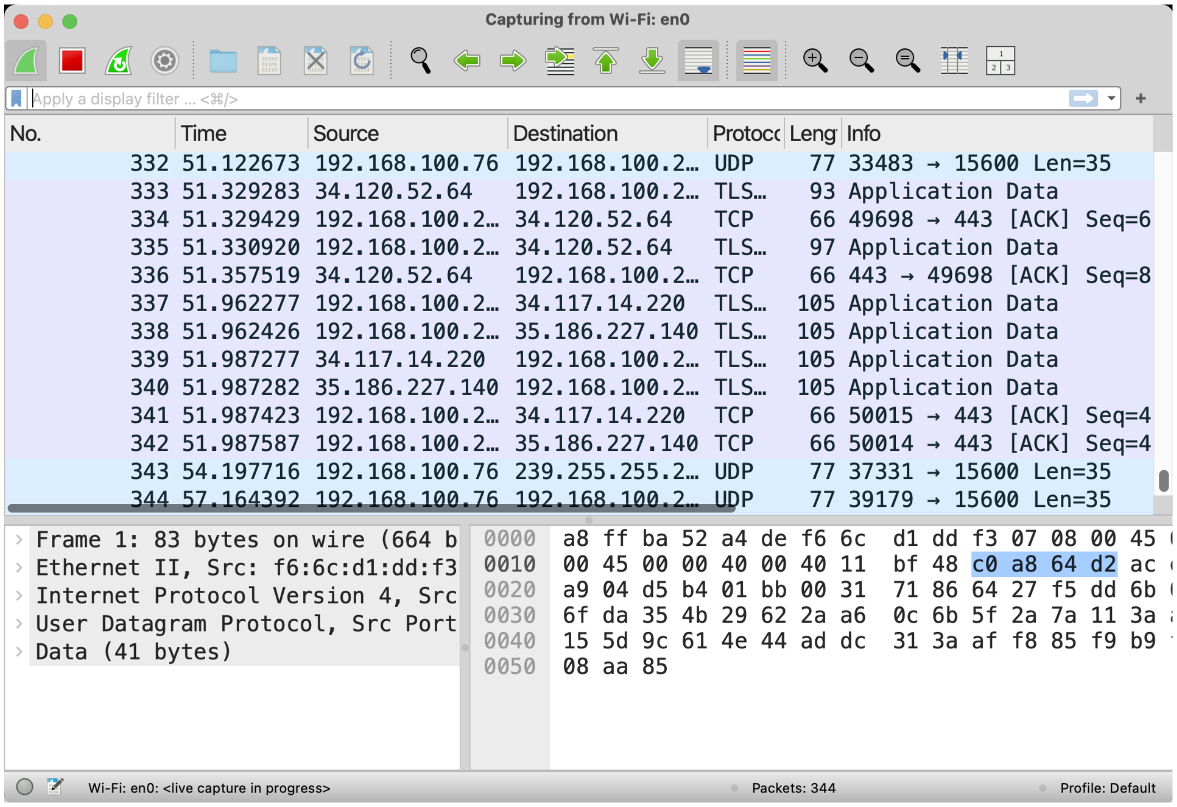
Step 1: Determine the IP address of the default gateway on your computer.

Open a command prompt window and enter the command `ipconfig` or `ifconfig`. What is the IP address of the computer's default gateway?

....

Step 2: Emptying the ARP table

During this phase, you will run wireshark to analyse the frames while deleting your ARP table.


Capturing from Wi-Fi: en0
Apply a display filter ... <⌘/>
Wi-Fi: en0: <live capture in progress>
Packets: 344
Profile: Default

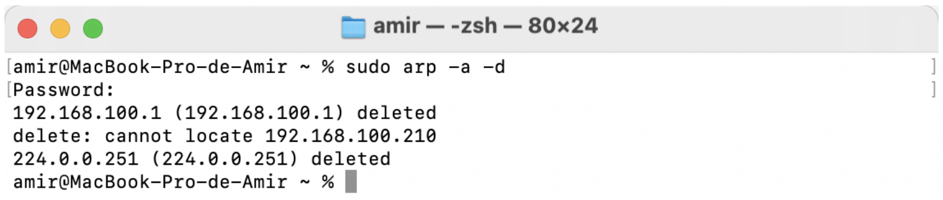
```
No.   Time       Source           Destination      Protocc Leng Info
  332 51.122673  192.168.100.76   192.168.100.2…   UDP       77 33483 → 15600 Len=35
  333 51.329283  34.120.52.64     192.168.100.2…   TLS…      93 Application Data
  334 51.329429  192.168.100.2…   34.120.52.64     TCP       66 49698 → 443 [ACK] Seq=6
  335 51.330920  192.168.100.2…   34.120.52.64     TLS…      97 Application Data
  336 51.357519  34.120.52.64     192.168.100.2…   TCP       66 443 → 49698 [ACK] Seq=8
  337 51.962277  192.168.100.2…   34.117.14.220    TLS…     105 Application Data
  338 51.962426  192.168.100.2…   35.186.227.140   TLS…     105 Application Data
  339 51.987277  34.117.14.220    192.168.100.2…   TLS…     105 Application Data
  340 51.987282  35.186.227.140   192.168.100.2…   TLS…     105 Application Data
  341 51.987423  192.168.100.2…   34.117.14.220    TCP       66 50015 → 443 [ACK] Seq=4
  342 51.987587  192.168.100.2…   35.186.227.140   TCP       66 50014 → 443 [ACK] Seq=4
  343 54.197716  192.168.100.76   239.255.255.2…   UDP       77 37331 → 15600 Len=35
  344 57.164392  192.168.100.76   192.168.100.2…   UDP       77 39179 → 15600 Len=35
```

```
> Frame 1: 83 bytes on wire (664 b
> Ethernet II, Src: f6:6c:d1:dd:f3
> Internet Protocol Version 4, Src
> User Datagram Protocol, Src Port
> Data (41 bytes)
```

```
0000  a8 ff ba 52 a4 de f6 6c  d1 dd f3 07 08 00 45
0010  00 45 00 00 40 00 40 11  bf 48 c0 a8 64 d2 ac
0020  a9 04 d5 b4 01 bb 00 31  71 86 64 27 f5 dd 6b
0030  6f da 35 4b 29 62 2a a6  0c 6b 5f 2a 7a 11 3a
0040  15 5d 9c 61 4e 44 ad dc  31 3a af f8 85 f9 b9
0050  08 aa 85
```

Figure 1: Wireshark live capture

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.



```
amir@MacBook-Pro-de-Amir ~ % sudo arp -a -d
Password:
192.168.100.1 (192.168.100.1) deleted
delete: cannot locate 192.168.100.210
224.0.0.251 (224.0.0.251) deleted
amir@MacBook-Pro-de-Amir ~ %
```

Figure 2: Clearing the ARP table

The command to clear the ARP cache depends on the operating system you are using. Here are the instructions for the main operating systems (Figure 2):

- **Windows:** `netsh interface ip delete arpcache` using the command prompt
- **Linux:** `sudo ip -s -s neigh flush all` using the terminal
- **MacOS:** `sudo arp -a -d` using the terminal

Step 2: ARP traffic capture

Any request launched by the computer connected to a private network goes through the gateway. This is also the connection diagram for a home computer connected to the Internet via a modem. In this case, the modem is the gateway. Remember that the gateway is the local equipment (usually a router) that the machine uses to connect to the Internet.
When using the web browser to load a web page, in order for the request to be sent to the server, the computer needs to know the MAC address of the gateway, so it will use the ARP protocol to find it. The ARP packet exchange captured by Wireshark looks like this:

- In the filter text field, type `arp` or `eth.type == 0x0806` to display only frames containing ARP packets (Figure 3).

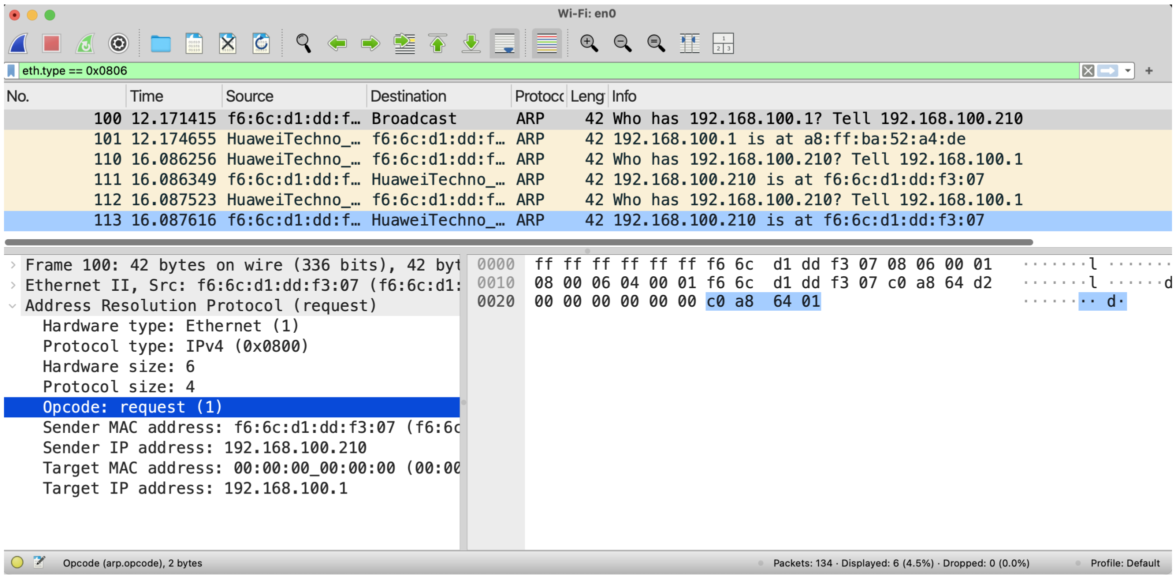
There are two types of ARP packet (distinguished by the Info column in zone 1):

1. Request packet: The Info line of this packet contains 'Who has @IP? ...' (See frame n° 100 in Figure 3).
2. Reply packet: The Info line of this packet contains '@IP is at @MAC', see frame n° 101 in Figure 3.

Now, with your instructor, explore the Ethernet frames that contain the two frames. What is the gateway MAC address ?

. . . .

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.



```
Wi-Fi: en0
eth.type == 0x0806

No.    Time       Source              Destination         Protocol Length Info
   100 12.171415  f6:6c:d1:dd:f…      Broadcast           ARP      42 Who has 192.168.100.1? Tell 192.168.100.210
   101 12.174655  HuaweiTechno_…      f6:6c:d1:dd:f…      ARP      42 192.168.100.1 is at a8:ff:ba:52:a4:de
   110 16.086256  HuaweiTechno_…      f6:6c:d1:dd:f…      ARP      42 Who has 192.168.100.210? Tell 192.168.100.1
   111 16.086349  f6:6c:d1:dd:f…      HuaweiTechno_…      ARP      42 192.168.100.210 is at f6:6c:d1:dd:f3:07
   112 16.087523  HuaweiTechno_…      f6:6c:d1:dd:f…      ARP      42 Who has 192.168.100.210? Tell 192.168.100.1
   113 16.087616  f6:6c:d1:dd:f…      HuaweiTechno_…      ARP      42 192.168.100.210 is at f6:6c:d1:dd:f3:07

> Frame 100: 42 bytes on wire (336 bits), 42 byt
> Ethernet II, Src: f6:6c:d1:dd:f3:07 (f6:6c:d1:
v Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: f6:6c:d1:dd:f3:07 (f6:6c
    Sender IP address: 192.168.100.210
    Target MAC address: 00:00:00_00:00:00 (00:00
    Target IP address: 192.168.100.1

0000  ff ff ff ff ff ff f6 6c  d1 dd f3 07 08 06 00 01   ·······l ·······
0010  08 00 06 04 00 01 f6 6c  d1 dd f3 07 c0 a8 64 d2   ·······l ······d
0020  00 00 00 00 00 00 c0 a8  64 01                     ········ d·

Opcode (arp.opcode), 2 bytes    Packets: 134 · Displayed: 6 (4.5%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 3: ARP filtering

Work requested

Use your wireshark to determine the list of arp requests sent to your network. How many machines require a MAC address?

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.

Task 3: Introduction to Scapy
Scapy is a Python-based network packet manipulation and analysis tool. It allows users to create, send, receive, and analyze network packets at a very low level. Scapy is widely used for network testing, penetration testing, and educational purposes due to its flexibility and ease of use.
To install Scapy, follow the steps appropriate to your operating system. Scapy is available via `pip`, but it may require additional dependencies depending on how you intend to use it (for example, for sniffing or sending packets).

- Make sure you have Python installed (version 3.7 or higher recommended).
  - Check with :

    ```
    python --version
    ```

    Or

    ```
    python3 --version
    ```

- Install a package manager such as `pip`, which is generally included with Python.

Step 1: Basic installation with pip
Run the following command in a terminal or command prompt:

```
pip install scapy
```

For Python 3, use :

```
pip3 install scapy
```

This command installs Scapy and its basic dependencies.

Step 2: Installation of additional dependencies
Some functions (such as sniffing or sending packets) require system tools. Here's how to install these dependencies depending on your operating system.

1. **Linux (Ubuntu/Debian):**
   - Install the required system tools:

     ```
     sudo apt update
     sudo apt install python3-pip tcpdump libpcap-dev
     ```

   - Install scapy

     ```
     pip3 install scapy
     ```

2. **macOS**
   - Install `tcpdump` and `libpcap` via `brew` (if `brew` is installed) :

     ```
     brew install tcpdump libpcap
     ```

   - Install scapy

     ```
     pip3 install scapy
     ```

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.

3. **Windows**
   - Install Npcap, a network sniffing tool:
     - Download and install Npcap from the official website: `https://npcap.com/`.
   - Install scapy

     ```
     pip install scapy
     ```

4. **Checking the installation**

   To check that Scapy is correctly installed, open a Python interpreter and run :

   ```python
   from scapy.all import *
   print("Scapy version:", conf.version)
   ```

Step 3: Creating a simple Ethernet frame

Use the following python code to create an Ethernet frame:

```python
from scapy.all import *

# Create an Ethernet frame
eth_frame = Ether(dst="ff:ff:ff:ff:ff:ff", src="00:11:22:33:44:55", type=0x0806)

# Display the frame details
print("Ethernet frame:")
eth_frame.show()
```

Explanation :

- **dst :** Destination MAC address (here, broadcast with ff:ff:ff:ff:ff:ff).
- **src :** Source MAC address.
- **type :** Protocol type (0x0806 for ARP).

Step 4: Building an ARP request

```python
from scapy.all import *

# Create an ARP request
arp_request = ARP(pdst="192.168.1.1", psrc="192.168.1.100", op=1)

# Display the ARP request
print("ARP request:")
arp_request.show()
```

Explanation:

- **pdst:** target IP (address from which the MAC address is requested).
- **psrc:** source IP (your machine).
- **op=1:** Specifies that this is an ARP request (2 for an ARP response).

You can also configure the other fields in the ARP PDU, such as `hwtype`, `ptype`, `hwlen`, `plen`, `hwsrc`, `hwdst`

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.

Step 5: Combining Ethernet and ARP

```python
from scapy.all import *

# Create an Ethernet frame with an ARP request
eth_arp_packet = Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(pdst="192.168.1.1")

# Display the combined packet
print("Ethernet + ARP packet:")
eth_arp_packet.show()
```

Explanation:

- Combines an Ethernet frame and an ARP request.
- The / symbol is used to encapsulate the ARP protocol in the Ethernet frame.

Step 6: Send an ARP request on the network

```python
from scapy.all import *

# Create an Ethernet + ARP request
eth_arp_packet = Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(pdst="192.168.1.1")

# Send the packet and wait for a response
response = srp1(eth_arp_packet, timeout=2, verbose=False)

# Display the response if received
if response:
    print("Received response:")
    response.show()
else:
    print("No response received.")
```

Explanation:

- Sends an ARP packet on the local network and waits for a response.
- The `srp1()` function captures a single response for the packet sent.

Step 7: Sniffing Ethernet or ARP frames on the network

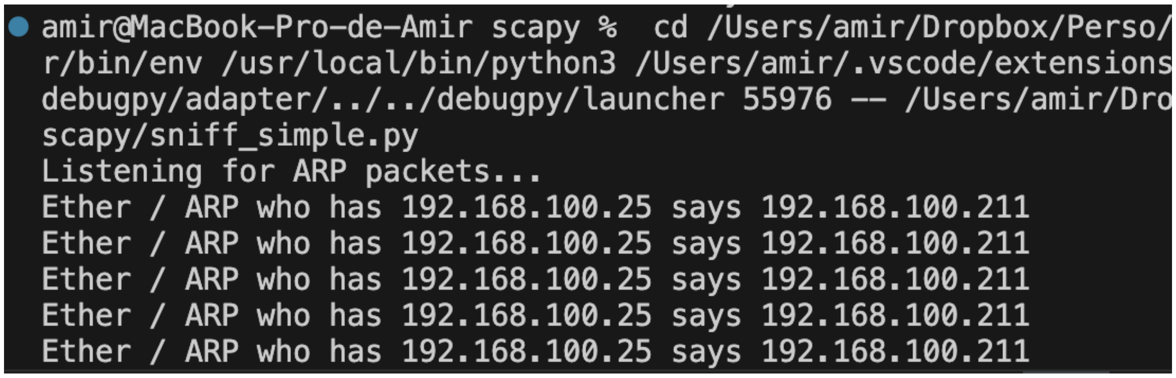
```python
from scapy.all import *

# Sniff packets and display only ARP packets
print("Listening for ARP packets...")
sniff(filter="arp", count=5, prn=lambda pkt: pkt.summary())
```

Explanation:

- The arp filter only captures ARP packets.
- The prn function is used to define an action (here, to display a summary of the packet).
- count=5 stops capture after 5 packets.

You can ping any IP address to display the result (Figure 4).

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.



```
amir@MacBook-Pro-de-Amir scapy %  cd /Users/amir/Dropbox/Perso/
r/bin/env /usr/local/bin/python3 /Users/amir/.vscode/extensions
debugpy/adapter/../../debugpy/launcher 55976 -- /Users/amir/Dro
scapy/sniff_simple.py
Listening for ARP packets...
Ether / ARP who has 192.168.100.25 says 192.168.100.211
Ether / ARP who has 192.168.100.25 says 192.168.100.211
Ether / ARP who has 192.168.100.25 says 192.168.100.211
Ether / ARP who has 192.168.100.25 says 192.168.100.211
Ether / ARP who has 192.168.100.25 says 192.168.100.211
```

Figure 4: ARP sniffing result

Step 8: A complete python program
This script shows how to construct an ARP packet, send it on the local network and analyse the response:

```python
from scapy.all import *



# Create an ARP packet to request the MAC address associated with an IP address
def create_arp_request(target_ip, source_ip):
    arp_request = ARP(pdst=target_ip, psrc=source_ip, op=1)  # op=1 for an ARP request
    ether_layer = Ether(dst="ff:ff:ff:ff:ff:ff")  # Ethernet frame for broadcast
    arp_packet = ether_layer / arp_request
    return arp_packet

# Send the ARP packet and capture the response
def send_and_receive_arp(packet):
    print("Sending ARP request...")
    response, unanswered = srp(packet, timeout=2, verbose=False)
    return response

# Analyze the ARP response
def analyze_response(response):
    if response:
        for sent, received in response:
            print(f"Target IP address: {received.psrc}")
            print(f"Associated MAC address: {received.hwsrc}")
    else:
        print("No response received. The target machine may be unreachable.")

if __name__ == "__main__":
    # Replace these addresses with your own for testing
    target_ip = "192.168.1.1"  # Target IP address
    source_ip = "192.168.1.100"  # Your local IP address

    # Create an ARP packet
    arp_packet = create_arp_request(target_ip, source_ip)
    print("ARP packet created:")
    arp_packet.show()  # Display the details of the packet

    # Send and receive ARP responses
    response = send_and_receive_arp(arp_packet)

    # Analyze the responses
    analyze_response(response)
```

Copyright © A. Bachir, A. Bensalem, A. Djouama, K. Lounis, Y. Sahraoui 2024/2025.