Analyse de trames Ethernet

1. Principes généraux

Une **trame réseau** dépend :

- de la **couche OSI concernée**
- du **protocole encapsulé** (Ethernet, ARP, IPv4, IPv6, TCP, UDP, ICMP…)
- de l'**encapsulation** des couches supérieures (une trame contient plusieurs couches successives)

Chaque protocole se caractérise par :

- des **champs spécifiques**
- des **tailles de champs différentes**
- des **règles d'interprétation propres**

2. Méthode d'analyse d'une trame

Toutes les analyses de trame suivent **exactement la même logique** :

1. Lire une suite d'octets
2. Extraire des champs
3. Interpréter ces champs
4. Exploiter les résultats

3. Analyse séquentielle d'une trame

Étape 1 – Identifier la trame Ethernet

Extraire :

- MAC destination
- MAC source
- Type Ethernet
    - `0x0800` → IPv4
    - `0x0806` → ARP

Étape 2 – Décoder le protocole réseau

Cas IPv4 (`0x0800`)

- Version
- IHL (taille de l'en-tête)
- Longueur totale
- TTL
- Protocole encapsulé

- IP source
- IP destination

Cas ARP (`0x0806`)

- Opcode (requête ou réponse)
- MAC source / destination
- IP source / destination

Étape 3 – Décoder le protocole de transport

Selon le champ **Protocole IPv4** :

- **TCP (6)**
    - Port source / destination
    - Numéro de séquence
    - Numéro d'acquittement
    - Flags
- **ICMP (1)**
    - Type
    - Code
    - Checksum
    - Exemples :
        - Type 8 : Echo Request
        - Type 0 : Echo Reply

Étape 4 – Tableau d'analyse

Champ	Valeur
MAC destination	
MAC source	
Type Ethernet	
IPv4 Version	
IHL (en-tête IPv4)	
Longueur totale	
TTL	
Protocole IPv4	
IP source	

Champ	Valeur
IP destination	
Port source TCP	
Port destination TCP	
Numéro de séquence	
Numéro d'acknowledgment	
Flags TCP / ICMP Type-Code	

Étape 5 – Vérification avec le script Python

- Le script est un **outil de contrôle**
- Il **n'explique pas la trame**
- Il permet de :
    - vérifier vos résultats
    - corriger vos erreurs

> **Le travail doit être fait manuellement avant l'exécution du script**

---

4. Procédure de travail

- Analyse sur **papier** (format standardisé)
- Vérification avec le **script Python fourni**
- Le script :
    - n'est **pas à analyser**
    - est utilisé comme **outil de contrôle**, au même titre qu'une calculatrice

5. Format des trames

- Octets écrits en **hexadécimal**
- 1 octet = **2 caractères hexadécimaux**
- Octets séparés par des **espaces**
- Ordre réel de la trame (gauche → droite)

6. Formats de trames

6.1 Ethernet + IPv4

Exemple

```
ff ff ff ff ff ff
00 11 22 33 44 55
08 00
45 00 00 3c 1c 46 40 00 40 06 a6 ec
c0 a8 01 01
c0 a8 01 02
```

En-tête Ethernet (14 octets)

Champ	Taille
MAC destination	6
MAC source	6
Type	2

En-tête IPv4 (20 octets minimum)

Champ	Taille
Version + IHL	1
TOS	1
Longueur totale	2
Identification	2
Flags + Fragment Offset	2
TTL	1
Protocole	1
Checksum	2
IP source	4
IP destination	4

Flags + Fragment Offset

- Flags : 3 bits
- Fragment Offset : 13 bits
- Total : 16 bits (2 octets)

6.2 Format ARP

Exemple

```
ff ff ff ff ff ff
00 11 22 33 44 55
08 06
00 01 08 00 06 04 00 01
00 11 22 33 44 55
c0 a8 01 01
00 00 00 00 00 00
c0 a8 01 02
```

---

7. Scripts Python (outil de vérification)

Fonctionnalités

- Lecture d'une trame hexadécimale
- Détection automatique :
  - Ethernet
  - IPv4 ou ARP
- Affichage des champs **comme sur la feuille papier**

Utilisation

```
python analyse_trame.py trame.txt
```

`analyse_trame.py`

```
#!/usr/bin/env python3

import sys


def lire_trame(fichier):
    with open(fichier, "r") as f:
        octets = []
        for ligne in f:
            ligne = ligne.strip()
            if ligne:
                octets.extend(ligne.split())
        return bytes(int(o, 16) for o in octets)


def mac_adresse(octets):
    return ":".join(f"{b:02x}" for b in octets)
```

```
def ip_adresse(octets):
    return ".".join(str(b) for b in octets)


def analyser_ethernet(trame):
    print("=== TRAME ETHERNET ===")
    dest = trame[0:6]
    src = trame[6:12]
    eth_type = trame[12:14]

    print(f"MAC destination : {mac_adresse(dest)}")
    print(f"MAC source      : {mac_adresse(src)}")
    print(f"Type            : 0x{eth_type.hex()}")

    return eth_type, trame[14:]


def analyser_ipv4(trame):
    print("\n=== EN-TÊTE IPv4 ===")
    version_ihl = trame[0]
    version = version_ihl >> 4
    ihl = (version_ihl & 0x0F) * 4
    total_length = int.from_bytes(trame[2:4], "big")
    # Les champs multi-octets sont codés en "big endian" (octet de poids
fort en premier).
    ttl = trame[8]
    protocole = trame[9]
    src_ip = trame[12:16]
    dst_ip = trame[16:20]

    print(f"Version         : {version}")
    print(f"IHL             : {ihl} octets")
    print(f"Longueur totale : {total_length}")
    print(f"TTL             : {ttl}")
    print(f"Protocole       : {protocole}")
    print(f"IP source       : {ip_adresse(src_ip)}")
    print(f"IP destination  : {ip_adresse(dst_ip)}")


def analyser_arp(trame):
    print("\n=== TRAME ARP ===")
    opcode = int.from_bytes(trame[6:8], "big")
    src_mac = trame[8:14]
    src_ip = trame[14:18]
    dst_mac = trame[18:24]
    dst_ip = trame[24:28]

    if opcode == 1:
        print("Opcode          : 1 (requête)")
    elif opcode == 2:
        print("Opcode          : 2 (réponse)")
    else:
        print(f"Opcode          : {opcode}")
    print(f"MAC source      : {mac_adresse(src_mac)}")
```

```
    print(f"IP source       : {ip_adresse(src_ip)}")
    print(f"MAC destination : {mac_adresse(dst_mac)}")
    print(f"IP destination  : {ip_adresse(dst_ip)}")


def main():
    if len(sys.argv) != 2:
        print("Usage : python analyse_trame.py trame.txt")
        sys.exit(1)

    trame = lire_trame(sys.argv[1])

    eth_type, payload = analyser_ethernet(trame)

    if eth_type == b"\x08\x00":
        analyser_ipv4(payload)
    elif eth_type == b"\x08\x06":
        analyser_arp(payload)
    else:
        print("\nType Ethernet non pris en charge")


if __name__ == "__main__":
    main()
```

Version étendue

- analyse_trame_V2.py
- Ajout du décodage :
    - ICMP
    - TCP

analyse_trame_V2.py

```
#!/usr/bin/env python3

import sys

def lire_trame(fichier):
    with open(fichier, "r") as f:
        octets = []
        for ligne in f:
            ligne = ligne.strip()
            if ligne:
                octets.extend(ligne.split())
        return bytes(int(o, 16) for o in octets)

def mac_adresse(octets):
```

```
    return ":".join(f"{b:02x}" for b in octets)

def ip_adresse(octets):
    return ".".join(str(b) for b in octets)

def analyser_ethernet(trame):
    print("=== TRAME ETHERNET ===")
    dest = trame[0:6]
    src = trame[6:12]
    eth_type = trame[12:14]

    print(f"MAC destination : {mac_adresse(dest)}")
    print(f"MAC source      : {mac_adresse(src)}")
    print(f"Type            : 0x{eth_type.hex()}")

    return eth_type, trame[14:]

def analyser_ipv4(trame):
    print("\n=== EN-TÊTE IPv4 ===")
    version_ihl = trame[0]
    version = version_ihl >> 4
    ihl = (version_ihl & 0x0F) * 4
    total_length = int.from_bytes(trame[2:4], "big")
    ttl = trame[8]
    protocole = trame[9]
    src_ip = trame[12:16]
    dst_ip = trame[16:20]

    print(f"Version         : {version}")
    print(f"IHL             : {ihl} octets")
    print(f"Longueur totale : {total_length}")
    print(f"TTL             : {ttl}")
    print(f"Protocole       : {protocole}")
    print(f"IP source       : {ip_adresse(src_ip)}")
    print(f"IP destination  : {ip_adresse(dst_ip)}")

    payload = trame[ihl:total_length]
    # Le script suppose que la trame est complète et cohérente.

    # Analyse protocole supérieur
    if protocole == 1:      # ICMP
        analyser_icmp(payload)
    elif protocole == 6:    # TCP
        analyser_tcp(payload)

def analyser_arp(trame):
    print("\n=== TRAME ARP ===")
    opcode = int.from_bytes(trame[6:8], "big")
    src_mac = trame[8:14]
    src_ip = trame[14:18]
    dst_mac = trame[18:24]
    dst_ip = trame[24:28]

    print(f"Opcode          : {opcode}")
```

```
    print(f"MAC source      : {mac_adresse(src_mac)}")
    print(f"IP source       : {ip_adresse(src_ip)}")
    print(f"MAC destination : {mac_adresse(dst_mac)}")
    print(f"IP destination  : {ip_adresse(dst_ip)}")

def analyser_icmp(trame):
    print("\n=== EN-TÊTE ICMP ===")
    type_icmp = trame[0]
    code = trame[1]
    checksum = int.from_bytes(trame[2:4], "big")
    print(f"Type     : {type_icmp}")
    print(f"Code     : {code}")
    print(f"Checksum : {checksum}")

def analyser_tcp(trame):
    print("\n=== EN-TÊTE TCP ===")
    src_port = int.from_bytes(trame[0:2], "big")
    dst_port = int.from_bytes(trame[2:4], "big")
    seq_num = int.from_bytes(trame[4:8], "big")
    ack_num = int.from_bytes(trame[8:12], "big")
    data_offset = (trame[12] >> 4) * 4
    flags = trame[13]
    print(f"Port source      : {src_port}")
    print(f"Port destination : {dst_port}")
    print(f"Numéro de seq.   : {seq_num}")
    print(f"Numéro d'ack     : {ack_num}")
    print(f"Offset           : {data_offset} octets")
    #print(f"Flags             : 0x{flags:02x}")
    flags_str = []
    if flags & 0x02:
        flags_str.append("SYN")
    if flags & 0x10:
        flags_str.append("ACK")
    if flags & 0x01:
        flags_str.append("FIN")

    print(f"Flags            : {', '.join(flags_str)}")

def main():
    if len(sys.argv) != 2:
        print("Usage : python analyse_trame.py trame.txt")
        sys.exit(1)

    trame = lire_trame(sys.argv[1])
    eth_type, payload = analyser_ethernet(trame)

    if eth_type == b"\x08\x00":
        analyser_ipv4(payload)
    elif eth_type == b"\x08\x06":
        analyser_arp(payload)
    else:
        print("\nType Ethernet non pris en charge")
```

```
if __name__ == "__main__":
    main()
```