TP4 : Couche liaison

Gilles Menez - UCA - DS4H - Informatique

RMQ :

Quand on vous donne des codes Python ce n'est pas QUE pour les utiliser.
C'est aussi et surtout pour les comprendre !

---

1 La mission . . .

Notre agent 008, en mission pour l'UCA, a réussi à capturer des informations sur un câble Ethernet qui pourraient aider à l'obtention d'une bonne note.

La seule chose certaine est qu'il s'agissait d'un câble Ethernet.

Nous avons besoin de savoir à quoi correspond cet échange d'information :

- ➢ Que contient cette information ?
- ➢ Comment est structurée l'information ?
- ➢ Quels sont les interlocuteurs ?
- ➢ Pourquoi dialoguent t-ils ?
- ➢ . . .

Et de façon générale, toutes les informations et les explications qui permettront de choisir la meilleure note.

- ➢ **Attention toutefois** à ne PAS SE PERDRE DANS DES DETAILS INUTILES !

2 Les approches

Si vous acceptez votre mission, vous avez deux façons de l'aborder :

2.1 Approche "sans"

Il y a plein d'outils développés par d'autres et permettant d'analyser des trames réseaux.

Je vous propose `https://hpd.gasmi.net/` qui a l'avantage d'être simple et de travailler sur du texte.

sinon il y aussi `https://www.wireshark.org/` qui est "la" référence de ce type d'outils . . . mais plus complexe.

Ces outils facilitent l'analyse **MAIS** vous devez comprendre comment ils procèdent . . . **sinon vous passez à coté de ce que je vous demande !** Ceci nous amène à l'approche "avec codage".

2.2 Approche ”avec”

Vous devrez réaliser un programme Python, exécutable sans avoir à être super-utilisateur, et qui permettra d'analyser les différentes informations sous forme synthétique et pédagogique.

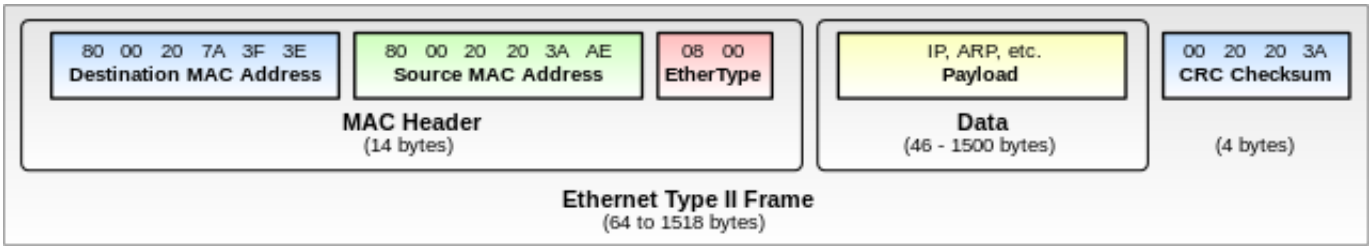
Plus bas dans le sujet, notre expert Qiou, a développé un début de solution qui pourrait peut être vous aider ?

3 Analyse de trames Ethernet . . .

3.1 Organisation de la trame Ethernet

Une trame Ethernet a ”toujours” la même forme/organisation : `https://fr.wikipedia.org/wiki/Ethernet`

- ✔ un entête (MAC header)
- ✔ puis les données / la charge (e.g. Data/Payload)



Lors de la capture, on n'a pas gardé la somme de contrôle (CRC checksum) dans le fichier.

- Ce n'était pas d'une grande utilité puisque si on l'a écrite c'est que la trame était correcte !

1. Le MAC header de la trame contient un ensemble d'informations qui permettent aux entités Ethernet des différentes machines sur le réseau de dialoguer.

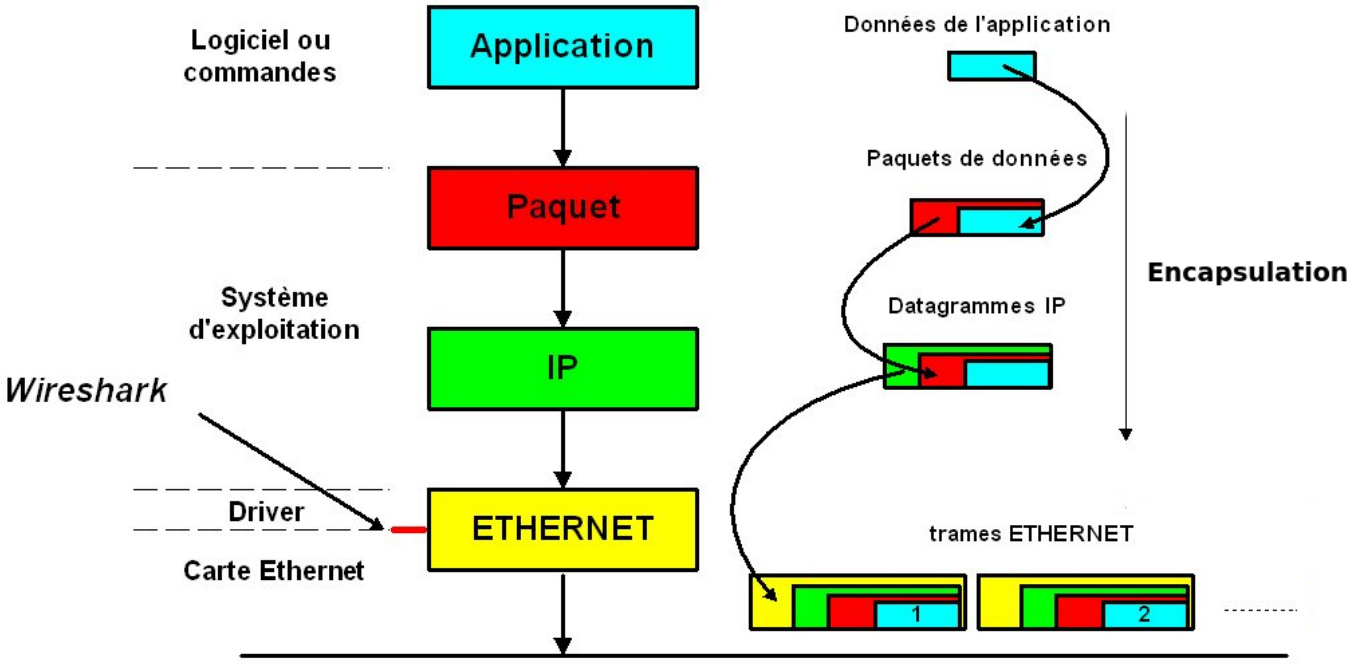
   Ces informations font partie du protocole Ethernet et il faut **savoir retrouver ces informations et les analyser pour poursuivre l'analyse de la partie Data**.

   ➢ Par exemple, **c'est grâce à l'EtherType que l'on sait la syntaxe de ce que contient la partie Data** et donc comment aborder l'analyse de cette partie Data.

   TOUTES LES INFORMATIONS ne sont pas d'égales importances pour votre mission mais, compte tenu du temps alloué, il faudra trouver les informations vraiment pertinentes.

2. La partie ”Data” contient les paquets échangés par les protocoles de niveaux supérieurs (IP, ARP, . . .) qui se servent d'Ethernet comme d'un transporteur de charges (Payload).

   ➢ On appelle cela `l'encapsulation` .



Dans la partie ”Data” (en jaune sur la trame Ethernet), on trouvera un header et les datas du protocole de niveau supérieur.

➢ Dans le header du protocole supérieur, il y aura **certainement** une information similaire à Ethertype qui **permettra de comprendre** ce qu'il transporte lui aussi.

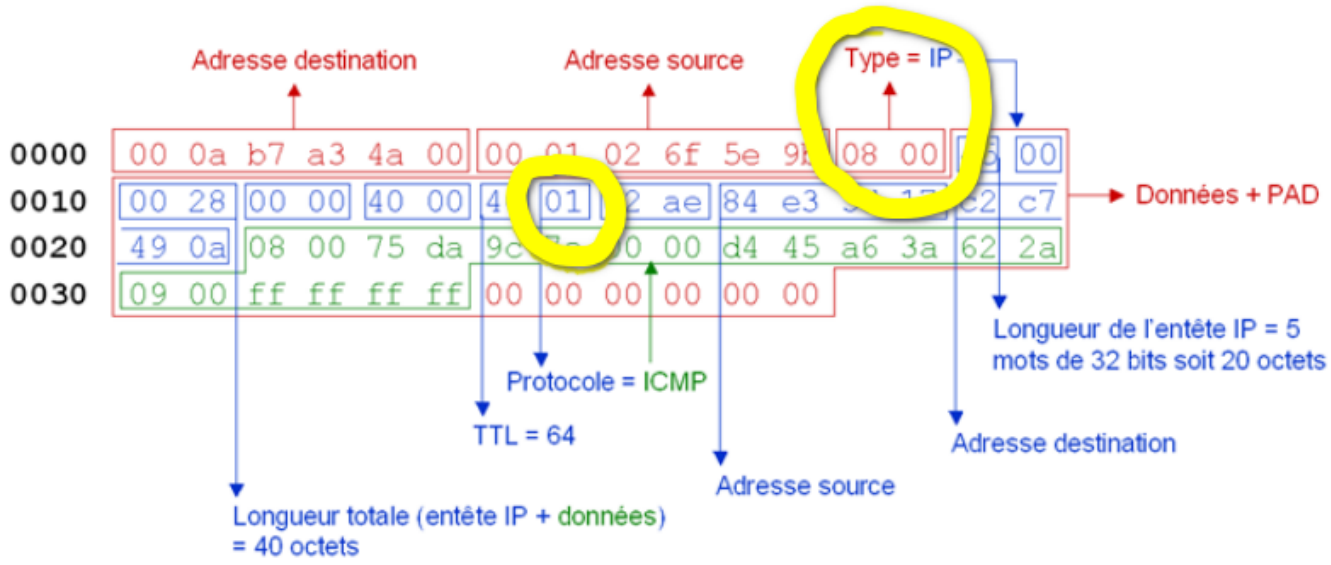
A ce stade, j'espère que vous avez compris la démarche à suivre pour ”explorer” le contenu d'une trame dans toute sa profondeur.

Dans l'exemple qui suit (mais qui n'est pas celui que vous rencontrez dans le fichier XXX.txt) on a coloré différemment les headers de chaque protocole et on a entouré en jaune les informations qui permettent le cheminement :

- Dans le header d'Ethernet en rouge, le **champ "Type" ou "Ethertype"** vaut 0800.

  Ceci indique que la data/charge d'Ethernet vient d'un protocole IP et du coup on sait comment elle est structurée et comment ce qui suit doit être analysé !

- Dans le header d'IP en bleu, le **champ "protocole"** vaut 01 ce qui indique que la charge de IP vient du protocole ICMP.



Vous remarquez que pour "avancer" dans la trame, vous n'avez pas besoin de comprendre TOUS les champs. Seulement ceux qui permettent de déduire la structure de la charge.

4 Implémentation

L'information dont on dispose, c'est à dire le fichier de texte récupéré par 008, contient probablement plusieurs trames échangées entre les participants.

4.1 Lire ces trames en Python

Le premier problème est d'accéder à ces trames. C'est un problème parce que les trames sont réparties sur plusieurs lignes de texte dans le fichier.

Voilà ce que contient le fichier :

```
0000  90 b1 1c 40 39 8f 00 27 13 53 96 29 08 00 45 00
0010  00 22 74 3e 40 00 40 11 b4 67 86 3b 83 ac 86 3b
0020  82 02 95 d9 00 07 00 0e 12 45 68 65 6c 6c 6f 0a

0000  90 b1 1c 40 39 8f 00 27 13 53 96 29 08 00 45 00
0010  00 22 75 4a 40 00 40 11 b3 5b 86 3b 83 ac 86 3b
0020  82 02 95 d9 00 07 00 0e 12 45 73 79 6d 70 61 0a

0000  90 b1 1c 40 39 8f 00 27 13 53 96 29 08 00 45 00
0010  00 20 76 6b 40 00 40 11 b2 3c 86 3b 83 ac 86 3b
0020  82 02 95 d9 00 07 00 0c 12 43 63 65 74 0a

0000  90 b1 1c 40 39 8f 00 27 13 53 96 29 08 00 45 00
0010  00 20 76 83 40 00 40 11 b2 24 86 3b 83 ac 86 3b
0020  82 02 95 d9 00 07 00 0c 12 43 65 78 6f 0a

0000  90 b1 1c 40 39 8f 00 27 13 53 96 29 08 00 45 00
0010  00 21 7b 94 40 00 40 11 ad 12 86 3b 83 ac 86 3b
0020  82 02 95 d9 00 07 00 0d 12 44 6e 6f 6e 3f 0a
```

Dans ce fichier, il y a donc cinq trames Ethernet.

4.2 Analyse syntaxique du fichier

Pour pouvoir analyser sémantiquement ces trames, il faut en faire une structure de données Python.
Le fichier de texte a une structure syntaxique qui facilite la chose, avec une ligne vide entre chaque trame.

- ➢ On va "assembler" les lignes pour former chaque trame.
- ➢ et on va placer ces trames dans une liste de trames.

C'est la fonction `readtrames()` qui réalise cela.

```
'''
Created on 28  Fev 2022
@author: menez

Syntax analysis of the file of frames
'''

import binascii
#=================================================
def readtrames(filename):
```

```
    """
    Cette fonction fabrique une liste de chaines de caracteres a partir du
    fichier contenant les trames.

    Chaque chaine de la liste rendue est une trame du fichier.

    return : liste des trames contenues dans le fichier
    """
    file = open(filename)
    lestrames = [] # List of frames (= lestrames)
    trame = ""  # Current frame .. string vide
    i = 1
    for line in file : # acces au fichier ligne par ligne
        line = line.rstrip('\n') # on enleve le retour chariot de la ligne
        line = line[6:53]        # on ne garde que les colonnes interessantes
        print ("ligne {} : {}".format(i,line))
        trame = trame + line

        if (len(line) == 0): # Trame separator

            # On enregistre la trame dans lestrames
            trame = trame.replace(' ','') # on enleve les blancs
            lestrames.append(trame) # on ajoute la trame a la liste
            trame = ""       # reset trame
        i = i+1

    # Si a la fin du fichier, il reste une trame à enregister
    if len(trame) != 0 : # Last frame
        trame = trame.replace(' ','') # on enleve les blancs
        lestrames.append(trame) # on ajoute la trame a la liste

    return lestrames

def unhexlify_lestrames(l):
    ''' l : liste de trames '''
    for i,trame in enumerate(l):
        print("{}({})".format(type(trame),len(trame)),end='')
        l[i] = binascii.unhexlify(trame)
        print("   --> {}({}) : {} ...".format(type(trame), len(trame), trame[0:10]))

def analyse_syntaxique(filename) :
    ''' Analyse syntaxique du fichier filename'''

    # Transformation des echanges contenus dans le fichier
    # vers une liste de strings :  une string = une trame
    print("="*60)
    trames = readtrames(filename)

    print("="*60)
    print("\nTrames lues depuis le fichier :\n")
    for i,t in enumerate(trames):
        print("trame #{} : {}".format(i,t))

    # Unhexlify la liste de trames
    unhexlify_lestrames(trames)
```

```
    return trames

#=============================================================
if __name__ == '__main__':

    filename = "XXX.txt"
    #filename = "a_req.txt"
    #filename = "d_req.txt"
    #filename = "hbin_org.txt"
    #filename = "p_req.txt"
    #filename = "pnull_req.txt"
    #filename = "s_req.txt"

    # Analyse syntaxique
    analyse_syntaxique(filename)
```

4.3 Interpréter ces trames

Pour l'instant ces trames sont des chaînes de caractères et chaque caractère représente une valeur hexadécimale (de 0 à f) donc 4 bits.

- Remarquons que puisque c'est une chaîne de caractères, un caractère représentant 4 bits occupe en mémoire 1 octet donc 8 bits.

Pour ceux qui ont fait du C, c'est un peu le même problème que celui résolut par la fonction atoi().

➢ Si vous avez une chaîne de caractères représentant un entier, par exemple "123". Vous avez 3 octets contenant des codes ASCII mais pour obtenir la valeur entière il va falloir convertir ces 3 octets en une valeur entière.

Rester avec une représentation de type chaîne de caractères pourrait compliquer l'interprétation de ces trames car la granularité des informations formant les entêtes des protocoles n'est pas forcément celle d'une lettre (4 bits) et leur découpage peut ne pas être un multiple de 4 bits.

On va donc utiliser quelques fonctions du module `binascii` pour convertir cette chaine de caractère en des objets "bytes-like" (tableau d'octets) qui permettront une interprétation plus fine que la notion de caractère.

C'est la fonction `unhexlify_lestrames()` qui réalise cela.

Vous remarquez, si vous lancez le programme, que l'on passe bien d'un tableau de caractères à un tableau d'octets et que deux lettres sont désormais stockées dans un seul octet.

Les thèmes abordés ici peuvent être approfondis :

➢ Manipulations de caractères

➢ http://stackoverflow.com/questions/227459/ascii-value-of-a-character-in-python

➢ binascii

http://stackoverflow.com/questions/32461630/convert-from-mac-address-to-hex-string-and-vice-versa-both-python-2-and-3

➢ Format strings : pack et unpack

https://docs.python.org/fr/3/library/struct.html

4.4 Analyse sémantique d'une trame

Vous disposez désormais d'une liste de trames sous forme de tableaux d'octets et vous allez devoir interpréter la sémantique de trames Ethernet échangées entre des machines.

Le code qui suit vous donne une base pour démarrer ce travail : Le fichier est sur le site Web !

```
import binascii
import struct # pour unpack
import syntaxe

def analyse_semantique(trame):
    """
    Analyse une trame Ethernet :  cf https://fr.wikipedia.org/wiki/Ethernet
    Input : trame est un tableau d'octets
    """
    print("-"*60)
    print ("\nTrame Ethernet en cours d'analyse ({}): \n{} ... ".format( type(trame), trame[0:10]))

    # Analyse du header ETHERNET
    eth_header = trame[0:14]
    #print(eth_header)
    eth_mac_dest, eth_mac_src, eth_type = struct.unpack('!6s6sH' , eth_header) #  https://docs.python.org/fr/3/library

    print ('Destination MAC : {}'.format(eth_mac_dest.hex(":"))) # .hex() ssi Python > 3.8
    print ('Source MAC \t: {}'.format(binascii.hexlify(eth_mac_src)))

    #https://stackoverflow.com/questions/4959741/python-print-mac-address-out-of-6-byte-string

    ############################################################
    # TODO : Maintenant que vous avez compris il faut analyser la suite de la trame  !!!

#================================================================
if __name__ == '__main__':
    filename = "XXX.txt"
    # Analyse syntaxique
    lestrames = syntaxe.analyse_syntaxique(filename)
    # Analyse sémantique
    for trame in lestrames:
        analyse_semantique(trame)
```

Ce programme travaille par défaut sur le fichier `XXX.txt` qui contient cinq trames et qui est fourni.

4.5 TODO :

J'espère de vous une finalisation/amélioration du code qui vous permettra de valider la syntaxe de la trame **jusque dans les protocoles encapsulés**.

Vous en déduirez la sémantique de ces trames et conclurez en expliquant **qui communique et pourquoi** ?
On espère grâce à votre programme Python voir s'afficher :

- ✔ Adresses physiques des machines
- ✔ Adresses logiques (IP)
- ✔ Nom des protocoles utilisés
- ✔ Information contenue dans les charges des trames du protocole le plus haut.