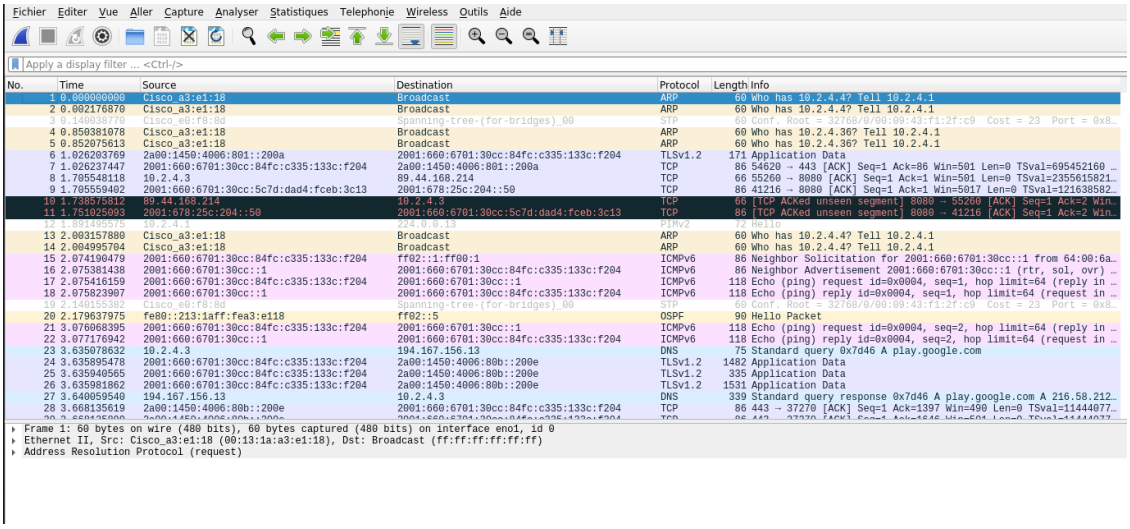
Ping6

March 11, 2022

1 Etude d'un ICMP Echo Request en IPv6

Ce fichier d'entraînement va vous permettre de vous familiariser avec Scapy et à l'écriture d'un script Python, à travers l'étude d'un simple ping en IPv6 entre deux machines du même réseau local.
Le fichier ping6.pcapng correspond à une capture de trames qui intègre, entre autre, un ping en IPv6 entre deux machines.
Ouvrez ce fichier sous wireshark.

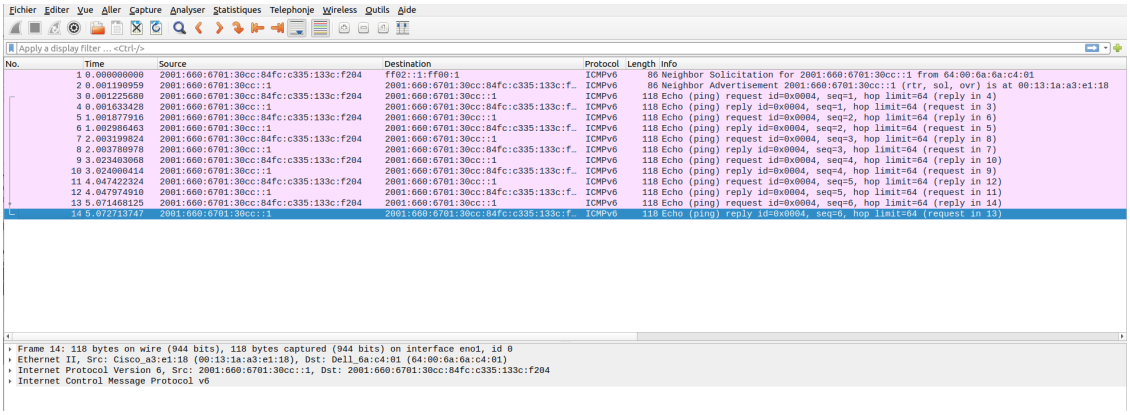


La première des choses à faire est de trouver un filtre d'affichage adapté, vous permettant de ne conserver que les trames relatives à ce ping:

- l'échange initial du protocole "Neighbor Discovery" permettant de récupérer l'adresse MAC du destinataire

- les échanges ICMP Echo Request / ICMP Echo Reply

Avec le bon filtre, vous devriez avoir à l'écran quelque chose qui ressemble à cela:



Vous allez maintenant utiliser la fonctionalité de wireshark vous permettant d'enregistrer un nouveau fichier qui ne comportera QUE LES TRAMES AFFICHÉES À L'ÉCRAN.
Vous allez ainsi créer le fichier **ping6-display.pcapng**

Fermez votre fichier actuel et ouvrez ce nouveau fichier sous Wireshark pour vérifier que votre manipulation a bien fonctionné.

Notez ici l'adresse IPv6 de la machine qui initie le "ping":

Notez ici l'adresse IPv6 de la machine cible:

Nous allons maintenant nous intéresser à la première trame : "Neighbor Solicitation". Cette manipulation correspond à ce que nous avons déjà fait en TD. Cliquez sur cette trame (dans la fenêtre supérieure) et visualisez les champs intéressants (dans la fenêtre du milieu). Notez les valeurs ci-dessous:

- **En-tête Ethernet**
- Adresse MAC Source:
- Adresse MAC Destination:
- **En-tête IPv6**
- Adresse IP Source:
- Adresse IP Destination:
- **En-tête du paquet ICMP/ND**
- Adresse IP Cible:
- Adresse MAC:

L'objectif de cet exercice sera d'écrire un programme Python/Scapy permettant de lire le fichier pcap, de trouver la trame concernée, puis d'extraire automatiquement les champs intéressants et d'imprimer la même chose que ce que vous avez fait manuellement ci-dessus.
Nous allons procéder par étape.

1.1 Lecture d'un fichier .pcap ou .pcapng

Commençons par importer le fichier avec la fonction python "rdpcap" (pour "read pcap"), qui permet de placer toutes les trames contenues dans le fichier, dans un dictionnaire:

[26]:
```
from scapy.all import *
#L'ensemble des paquets de la capture sont chargés dans la variable "trames"
```

```
trames=rdpcap("Wireshark/ping6-display.pcapng")
print("Voici les trames capturées :\n")
trames.show()
```

```
Voici les trames capturées :

0000 Ether / IPv6 / ICMPv6ND_NS / ICMPv6 Neighbor Discovery Option - Source
Link-Layer Address 64:00:6a:6a:c4:01
0001 Ether / IPv6 / ICMPv6ND_NA / ICMPv6 Neighbor Discovery Option - Destination
Link-Layer Address 00:13:1a:a3:e1:18
0002 Ether / IPv6 / ICMPv6 Echo Request (id: 0x4 seq: 0x1)
0003 Ether / IPv6 / ICMPv6 Echo Reply (id: 0x4 seq: 0x1)
0004 Ether / IPv6 / ICMPv6 Echo Request (id: 0x4 seq: 0x2)
0005 Ether / IPv6 / ICMPv6 Echo Reply (id: 0x4 seq: 0x2)
0006 Ether / IPv6 / ICMPv6 Echo Request (id: 0x4 seq: 0x3)
0007 Ether / IPv6 / ICMPv6 Echo Reply (id: 0x4 seq: 0x3)
0008 Ether / IPv6 / ICMPv6 Echo Request (id: 0x4 seq: 0x4)
0009 Ether / IPv6 / ICMPv6 Echo Reply (id: 0x4 seq: 0x4)
0010 Ether / IPv6 / ICMPv6 Echo Request (id: 0x4 seq: 0x5)
0011 Ether / IPv6 / ICMPv6 Echo Reply (id: 0x4 seq: 0x5)
0012 Ether / IPv6 / ICMPv6 Echo Request (id: 0x4 seq: 0x6)
0013 Ether / IPv6 / ICMPv6 Echo Reply (id: 0x4 seq: 0x6)
```

A ce niveau, il vous est grandement conseillé de vous familiariser avec l'utilisation des "dictionnaires" en Python. Vous pouvez trouver des informations sur ce sujet sur le document de référence "Python3" qui vous a été fourni en version papier, sinon, internet en regorge !

1.2 Exploitation du dictionnaire

Regardons la première trame (donc numérotée "0"):

[4]:
```
trames[0]
```

[4]:
```
<Ether  dst=33:33:ff:00:00:01 src=64:00:6a:6a:c4:01 type=IPv6 |<IPv6  version=6
tc=0 fl=0 plen=32 nh=ICMPv6 hlim=255 src=2001:660:6701:30cc:84fc:c335:133c:f204
dst=ff02::1:ff00:1 |<ICMPv6ND_NS  type=Neighbor Solicitation code=0 cksum=0x1d60
res=0 tgt=2001:660:6701:30cc::1 |<ICMPv6NDOptSrcLLAddr  type=1 len=1
lladdr=64:00:6a:6a:c4:01 |>>>>
```

Nous avons affaire à des dictionnaires imbriqués (ou "encapsulés" :-)). Nous pouvons ainsi afficher les différents niveaux de la trame:

[18]:
```
trames[0][0]
trames[0][Ether]
# Affichent la même chose: la trame intégrale
```

[18]:
```
<Ether  dst=33:33:ff:00:00:01 src=64:00:6a:6a:c4:01 type=IPv6 |<IPv6  version=6
tc=0 fl=0 plen=32 nh=ICMPv6 hlim=255 src=2001:660:6701:30cc:84fc:c335:133c:f204
dst=ff02::1:ff00:1 |<ICMPv6ND_NS  type=Neighbor Solicitation code=0 cksum=0x1d60
```

```
res=0 tgt=2001:660:6701:30cc::1 |<ICMPv6NDOptSrcLLAddr  type=1 len=1
lladdr=64:00:6a:6a:c4:01 |>>>>
```

[23]:
```
trames[0][1]
trames[0][IPv6]
# Affiche la même chose: le paquet IPv6
```

[23]:
```
<IPv6  version=6 tc=224 fl=0 plen=32 nh=ICMPv6 hlim=255
src=2001:660:6701:30cc::1 dst=2001:660:6701:30cc:84fc:c335:133c:f204
|<ICMPv6ND_NA  type=Neighbor Advertisement code=0 cksum=0x11d3 R=1 S=1 O=1
res=0x0 tgt=2001:660:6701:30cc::1 |<ICMPv6NDOptDstLLAddr  type=2 len=1
lladdr=00:13:1a:a3:e1:18 |>>>
```

[47]:
```
trames[0][2]
trames[0][ICMPv6ND_NS]
# Affiche la même chose: le paquet ICMP
```

[47]:
```
<ICMPv6ND_NS  type=Neighbor Solicitation code=0 cksum=0x1d60 res=0
tgt=2001:660:6701:30cc::1 |<ICMPv6NDOptSrcLLAddr  type=1 len=1
lladdr=64:00:6a:6a:c4:01 |>>
```

[36]:
```
trames[0][3]
trames[0][ICMPv6NDOptSrcLLAddr]
# Affiche la même chose: la sous-partie du paquet ICMP, contenant l'adresse MAC␣
 ↪source
```

[36]:
```
<ICMPv6NDOptSrcLLAddr  type=1 len=1 lladdr=64:00:6a:6a:c4:01 |>
```

C'est magique ...
Avec ce principe, il est très simple d'afficher un champ de l'en-tête. Par exemple, pour afficher le champ "lladdr" on peut faire ainsi:

[28]:
```
trames[0][ICMPv6NDOptSrcLLAddr].lladdr
```

[28]:
```
'64:00:6a:6a:c4:01'
```

Mais on peut faire également ainsi:

[29]:
```
trames[0][Ether].lladdr
trames[0][IPv6].lladdr
trames[0][ICMPv6NDOptSrcLLAddr]
# Affichent également toutes les trois la même chose
```

[29]:
```
'64:00:6a:6a:c4:01'
```

Ceci est possible dans ce cas, parce-que le champ n'existe qu'une seule fois dans toute la trame.
Regardons maintenant cet autre exemple avec le champ "dst" qui est à plusieurs niveaux dans la trame:

[31]:
```
trames[0].dst
trames[0][Ether].dst
# Affichent la même chose: le premier champ "dst" trouvé dans la trame
```

[31]:
```
'33:33:ff:00:00:01'
```

[33]:
```
trames[0][IPv6].dst
# Affiche aussi le premier champ "dst" trouvé ... mais ce n'est plus le même !
```

[33]:
```
'ff02::1:ff00:1'
```

C'est donc le premier champ trouvé qui est affiché, ce qui est juste logique …

Il devrait être maintenant un jeu d'enfant pour vous, d'écrire le programme Python qui affiche le texte demandé ! N'oubliez pas de le déposer (bien commenté) sur votre PortFolio et surtout DE COMMENTER A QUOI CORRESPOND CHACUNE DES VALEURS AFFICHÉES (Ce ne devrait pas être compliqué, c'est pareil qu'en TD !)

1.3 Automatisation du processus

L'objectif de cette dernière phase est maintenant d'écrire un programme qui pourra être lancé sur le fichier pcapng d'origine (le fichier contenant toutes les trames, pas le fichier filtré manuellement), afin d'en extraire automatiquement les trames correspondant à un "Neighbor Solicitation".
Il suffit pour cela de trouver le moyen de reconnaître ce type de trame.
C'est assez simple, il suffit de rechercher les paquets ICMPv6 de type "Neighbor Solicitation".

- Un paquet ICMPv6 se détecte grâce au champ "Next Header" (NH) de l'en-tête IPv6 qui prend la valeur 58 ("ICMPv6") (Rappel: le champ "Next Header" en IPv6 est l'équivalent du champ "protocol" en IPv4).
- Un paquet "Neighbor Solicitation" en IPv6 se détecte grâce à son champ "type" qui doit être égal à 135 ("Neighbor Solicitation" ou "NS")

[27]:
```
from scapy.all import *

#L'ensemble des trames capturées dans le fichier pcap sont chargées dans la␣
 ↪variable "trames"
trames=rdpcap("Wireshark/ping6-display.pcapng")

for trame in trames: # on fait une boucle pour les traiter une par une ...
    if(trame[0][1].version)==6: # on ne prend que les paquets en IPv6
        if (trame[0][1].nh)==58: # on ne prend que les paquets dont le␣
 ↪NextHeader = ICMPv6
            if (trame[0][2].type)==135: # on ne prend que les paquets de type␣
 ↪ICMP = 135 (NS)
                print(f"Ethernet: MAC Source      : {trame[0][0].src}")
                print(f"Ethernet: MAC Destination : {trame[0][0].dst}")
                print(f"IPv6 : IP Source          : {trame[0][1].src}")
                print(f"IPv6 : IP Destination     : {trame[0][1].dst}")
```

```
                print(f"ICMPv6 : IP Target        : {trame[0][2].tgt}")
                print(f"ICMPv6 : MAC Requested    : {trame[0][3].lladdr}")
```

```
Ethernet: MAC Source      : 64:00:6a:6a:c4:01
Ethernet: MAC Destination : 33:33:ff:00:00:01
IPv6 : IP Source          : 2001:660:6701:30cc:84fc:c335:133c:f204
IPv6 : IP Destination     : ff02::1:ff00:1
ICMPv6 : IP Target        : 2001:660:6701:30cc::1
ICMPv6 : MAC Requested    : 64:00:6a:6a:c4:01
```

Le but est maintenant d'appliquer ce script sur le fichier pcapng global, non filtré, c'est à dire comportant toutes les trame enregistrées. Le script ci-dessous est exactement le même que le précédent, seul le nom du fichier pcapng chargé est différent. En exécutant le script, vous vous apercevrez vite qu'il génère des erreurs.

[133]:
```
from scapy.all import *

#L'ensemble des trames capturées dans le fichier pcap sont chargées dans la␣
 ↪variable "trames"
trames=rdpcap("Wireshark/ping6-total.pcapng")

for trame in trames: # on fait une boucle pour les traiter une par une ...
    if(trame[0][1].version)==6: # on ne prend que les paquets en IPv6
        if (trame[0][1].nh)==58: # on ne prend que les paquets dont le␣
 ↪NextHeader = ICMPv6
            if (trame[0][2].type)==135: # on ne prend que les paquets de type␣
 ↪ICMP = 135 (NS)
                print(f"Ethernet: MAC Source      : {trame[0][0].src}")
                print(f"Ethernet: MAC Destination : {trame[0][0].dst}")
                print(f"IPv6 : IP Source          : {trame[0][1].src}")
                print(f"IPv6 : IP Destination     : {trame[0][1].dst}")
                print(f"ICMPv6 : IP Target        : {trame[0][2].tgt}")
                print(f"ICMPv6 : MAC Requested    : {trame[0][3].lladdr}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in __getattr__(self, attr)
    427         try:
--> 428             fld, v = self.getfield_and_val(attr)
    429         except ValueError:

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in getfield_and_val(self, attr)
    422             return self.get_field(attr), self.default_fields[attr]
--> 423         raise ValueError
    424
```

```
ValueError:

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in __getattr__(self, attr)
    427         try:
--> 428             fld, v = self.getfield_and_val(attr)
    429         except ValueError:

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in getfield_and_val(self, attr)
    422             return self.get_field(attr), self.default_fields[attr]
--> 423         raise ValueError
    424

ValueError:

During handling of the above exception, another exception occurred:

AttributeError                            Traceback (most recent call last)
<ipython-input-133-d47d0249434a> in <module>
      5
      6 for trame in trames: # on fait une boucle pour les traiter une par une
----> 7     if(trame[0][1].version)==6: # on ne prend que les paquets en IPv6
      8         if (trame[0][1].nh)==58: # on ne prend que les paquets dont le␣
 ↪NextHeader = ICMPv6
      9             if (trame[0][2].type)==135: # on ne prend que les paquets d␣
 ↪type ICMP = 135 (NS)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in __getattr__(self, attr)
    428             fld, v = self.getfield_and_val(attr)
    429         except ValueError:
--> 430             return self.payload.__getattr__(attr)
    431         if fld is not None:
    432             return fld.i2h(self, v)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in __getattr__(self, attr)
    428             fld, v = self.getfield_and_val(attr)
    429         except ValueError:
--> 430             return self.payload.__getattr__(attr)
    431         if fld is not None:
    432             return fld.i2h(self, v)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in __getattr__(self, attr)
    426         # type: (str) -> Any
    427         try:
--> 428             fld, v = self.getfield_and_val(attr)
    429         except ValueError:
    430             return self.payload.__getattr__(attr)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/
 ↪scapy/packet.py in getfield_and_val(self, attr)
   1826     def getfield_and_val(self, attr):
   1827         # type: (str) -> NoReturn
-> 1828         raise AttributeError(attr)
   1829
   1830     def setfieldval(self, attr, val):

AttributeError: version
```

Sans chercher à décortiquer toute la trace, la dernière ligne nous apprend que l'erreur vient de l'attribut "version" qui n'a pas été trouvé (voir la dernière ligne de la trace, juste ci-dessus).
Ce problème vient du fait qu'on essaye de tester un champ d'un en-tête IP sur une trame qui n'en dispose pas…
He oui, en IPv4, comme on l'a vu en TD, il existe des trames (par exemple ARP) qui n'utilisent pas IP, donc qui n'ont pas d'en-tête IP, donc pas de champ "version".

Ce sera un problème récurrent pour analyser un fichier pcapng … il y aura toujours des trames avec certains en-tête et pas d'autres. Tester des champs inexistants va toujours générer des erreurs. Pour cette raison, si vous ne savez pas déjà le faire, il va vous falloir apprendre à utiliser les **exceptions**, grâce aux instructions "try" et "except". Pour apprendre, cherchez de la documentation sur internet.

Un autre détail: l'en-tête ETHERNET peut revêtir deux aspects différents, selon la norme d'ethernet utilisée (ETHERNET II ou 802.3). Seules les trames ETHERNET II disposent d'un champ "type" qui contient le numéro du protocole de niveau supérieur avec une valeur toujours supérieure à 1500 (par exemple 34525 (0x86DD) pour IPv6 ou 2048 (0x0800) pour IPv4). Le même champ en 802.3 indique la longueur de la trame et a une valeur toujours inférieure ou égale à 1500 (0x05DC). Dans l'exemple suivant, le script suivant essaye (try) de lire le champ "type" des trames ETHERNET II et gère l'erreur produite (except) si ce champ n'existe pas.

[ ]:
```
from scapy.all import *

#L'ensemble des trames capturées dans le fichier pcap sont chargées dans la␣
 ↪variable "trames"
trames=rdpcap("Wireshark/ping6-total.pcapng")
for trame in trames: # on fait une boucle pour les traiter une par une ...
    try:
        trame[0][0].type # les trames 802.3 n'ont pas de champ type et génèrent␣
 ↪une exception qu'il faut gérer
    except:
```

```
        continue
    if(trame[0][0].type==34525): # on prend les trames contenant de l'IPv6
        if(trame[0][1].version)==6: # on ne prend que les paquets en IPv6, mais␣
↪ce test est maintenant redondant !
            if (trame[0][1].nh)==58: # on ne prend que les paquets dont le␣
↪NextHeader = ICMPv6
                if (trame[0][2].type)==135: # on ne prend que les paquets de␣
↪type ICMP = 135 (NS)
                    print(f"Ethernet: MAC Source      : {trame[0][0].src}")
                    print(f"Ethernet: MAC Destination : {trame[0][0].dst}")
                    print(f"IPv6 : IP Source          : {trame[0][1].src}")
                    print(f"IPv6 : IP Destination     : {trame[0][1].dst}")
                    print(f"ICMPv6 : IP Target        : {trame[0][2].tgt}")
                    print(f"ICMPv6 : MAC Requested    : {trame[0][3].lladdr}")
```

1.4 "Sniffer" directement le réseau

Plutôt que d'enregistrer du trafic avec Wireshark, il est beaucoup plus intéressant de sniffer le trafic d'un réseau et de l'analyser en temps réel.
La fonction "sniff" proposée par Scapy dispose d'un filtre équivalent au filtre d'affichage proposé par Wireshark. C'est très pratique.
Vous aurez donc le choix de faire faire un premier travail de sélection de trame à la fonction "sniff", ce qui devrait vous éviter de devoir par la suite gérer de multiples exceptions pour faire face à toutes les situations. Par exemple, si vous demandez à "sniff" de ne remonter que les trames contenant un en-tête IPv6, vous pourrez directement rechercher les champs spécifiques, sans vérifier si ils existent …

Nous allons donc capturer du trafic réseau en utilisant la fonction "sniff" dont les paramètres les plus essentiels sont les suivants:

- "filter": c'est justement ici que vous allez spécifier le filtre de capture, ex: "ip6 proto 58" permet de ne filtrer que les paquets contenant un en-tête IPv6 et donc le champ "Next Header" est égal à 58 (protocole ICMPv6).
- "prn": ce paramètre contient le champ de la fonction à appeler à chaque fois qu'une trame est capturée.
- "iface": spécifie le nom de la carte réseau sur laquelle on va écouter
- "count": désigne le nombre de trame à capturer. Une valeur de 0 signifie qu'il n'y a pas de limite, mais c'est DECONSEILLÉ dans un fichier jupyter notebook, car c'est toujours problématique d'interrompre un programme !

IMPORTANT: Vous remarquerez une différence de taille entre le code ci-dessous et le code des exemples précédents. Lorsqu'on fait référence à une trame capturée, on utilisait jusqu'à maintenant un tableau à deux dimensions:

ex: **trame[0][1].src**
ce qui signifiait "trame n°0 (la première de la liste), en-tête n°1 (en-tête IP), champ"src" Maintenant, on ne travaillera plus sur un **ENSEMBLE** de trames (dont il fallait préciser laquelle, comme trame [0] dans l'exemple précédent …), mais sur **UNE SEULE À LA FOIS:** celle qui est passée à la fonction par le paramètre "trame". Aussi, allons nous travailler de la même façon, mais sur un

tableau à une seule dimension:

ex: **trame[1].src**
signifiera en-tête n°1, champ “src”, de la trame en cours de traitement.

ATTENTION: sniffer un réseau nécessite de placer la carte réseau dans un mode spécial appelé “PROMICIOUS”, ce qui signifie que toutes les trames vont être remontées à votre programme, y compris les trames qui ne vous sont pas destinées.

MAIS pour se positionner dans ce mode, il est nécessaire d’exécuter le programme avec les droits de “root”, sinon vous obtiendrez une erreur.
OR pour exécuter un programme en tant que root sous “jupyter notebook” … il est nécessaire de lancer le serveur depuis une fenêtre dans laquelle vous aurez fait au préalable un “sudo -s”, en utilisant la commande:
`jupyter notebook --allow-root`

N’oubliez pas de fermer toutes vos fenêtres jupyter sous votre navigateur, de relancer votre serveur dans ce mode, puis de réouvrir les fenêtres sous votre navigateur.
Si l’exécution est toujours refusée, vérifiez que votre fichier .ipynb a bien les droits 444 (rw-rw-rw-)

Le programme ci-dessous capture tous les paquets ICMPv6, et affiche certains champs en fonction du type.

[18]:
```
from scapy.all import *

ICMPv6_types={ 128 : 'Echo-Request', 129 : 'Echo-Reply', 135 : 'Neighbor␣
 ↪Solicitation', 136 : 'Neighbor Advertisement', 133 : 'Router Solicitaion',␣
 ↪134 : 'Router Advertisement'}

def print_icmpv6 (trame) :
    print(trame.summary())
    type=trame[2].type
    if (type==135 or type==136):
        print(f"TYPE PACKET ICMP          : {ICMPv6_types[type]}")
        print(f"Ethernet: MAC Source      : {trame[0].src}")
        print(f"Ethernet: MAC Destination : {trame[0].dst}")
        print(f"IPv6 : IP Source          : {trame[1].src}")
        print(f"IPv6 : IP Destination     : {trame[1].dst}")
        print(f"ICMPv6 : IP Target        : {trame[2].tgt}")
        print(f"ICMPv6 : MAC Requested    : {trame[3].lladdr}")
        print ("\n")
    else:
        print(f"TYPE PACKET ICMP          : {ICMPv6_types[type]}")
        print(f"Ethernet: MAC Source      : {trame[0].src}")
        print(f"Ethernet: MAC Destination : {trame[0].dst}")
        print(f"IPv6 : IP Source          : {trame[1].src}")
        print(f"IPv6 : IP Destination     : {trame[1].dst}")
        print ("\n")

carte=conf.iface
```

```
print(f"On commence le 'sniffing' sur la carte {carte}:")
print("\n")
sniff(filter="ip6 proto 58", prn=print_icmpv6, store=0, iface=carte, count=6)
```

```
On commence le 'sniffing' sur la carte eno1:


Ether / IPv6 / ICMPv6ND_NS / ICMPv6 Neighbor Discovery Option - Source Link-
Layer Address 64:00:6a:6a:c4:01
TYPE PACKET ICMP          : Neighbor Solicitation
Ethernet: MAC Source      : 64:00:6a:6a:c4:01
Ethernet: MAC Destination : 33:33:ff:fe:de:ca
IPv6 : IP Source          : 2001:660:6701:30cc:1b0:ece0:a977:568c
IPv6 : IP Destination     : ff02::1:fffe:deca
ICMPv6 : IP Target        : 2001:660:6701:30cc:cafe:deca:cafe:deca
ICMPv6 : MAC Requested    : 64:00:6a:6a:c4:01


Ether / IPv6 / ICMPv6ND_NA / ICMPv6 Neighbor Discovery Option - Destination
Link-Layer Address 38:c9:86:0d:9b:5c
TYPE PACKET ICMP          : Neighbor Advertisement
Ethernet: MAC Source      : 38:c9:86:0d:9b:5c
Ethernet: MAC Destination : 64:00:6a:6a:c4:01
IPv6 : IP Source          : 2001:660:6701:30cc:cafe:deca:cafe:deca
IPv6 : IP Destination     : 2001:660:6701:30cc:1b0:ece0:a977:568c
ICMPv6 : IP Target        : 2001:660:6701:30cc:cafe:deca:cafe:deca
ICMPv6 : MAC Requested    : 38:c9:86:0d:9b:5c


Ether / IPv6 / ICMPv6 Echo Request (id: 0x28 seq: 0x1)
TYPE PACKET ICMP          : Echo-Request
Ethernet: MAC Source      : 64:00:6a:6a:c4:01
Ethernet: MAC Destination : 38:c9:86:0d:9b:5c
IPv6 : IP Source          : 2001:660:6701:30cc:1b0:ece0:a977:568c
IPv6 : IP Destination     : 2001:660:6701:30cc:cafe:deca:cafe:deca


Ether / IPv6 / ICMPv6 Echo Reply (id: 0x28 seq: 0x1)
TYPE PACKET ICMP          : Echo-Reply
Ethernet: MAC Source      : 38:c9:86:0d:9b:5c
Ethernet: MAC Destination : 64:00:6a:6a:c4:01
IPv6 : IP Source          : 2001:660:6701:30cc:cafe:deca:cafe:deca
IPv6 : IP Destination     : 2001:660:6701:30cc:1b0:ece0:a977:568c


Ether / IPv6 / ICMPv6 Echo Request (id: 0x28 seq: 0x2)
TYPE PACKET ICMP          : Echo-Request
```

```
Ethernet: MAC Source      : 64:00:6a:6a:c4:01
Ethernet: MAC Destination : 38:c9:86:0d:9b:5c
IPv6 : IP Source          : 2001:660:6701:30cc:1b0:ece0:a977:568c
IPv6 : IP Destination     : 2001:660:6701:30cc:cafe:deca:cafe:deca


Ether / IPv6 / ICMPv6 Echo Reply (id: 0x28 seq: 0x2)
TYPE PACKET ICMP          : Echo-Reply
Ethernet: MAC Source      : 38:c9:86:0d:9b:5c
Ethernet: MAC Destination : 64:00:6a:6a:c4:01
IPv6 : IP Source          : 2001:660:6701:30cc:cafe:deca:cafe:deca
IPv6 : IP Destination     : 2001:660:6701:30cc:1b0:ece0:a977:568c
```

[18]: `<Sniffed: TCP:0 UDP:0 ICMP:0 Other:0>`

A présent c'est à vous !
Dans cette deuxième phase du projet, je vous propose:

- De commenter au mieux la sortie de programme ci-dessus, afin de la rapprocher avec les concepts vus en TD (adresses sources et adresses destinations aux différents niveaux, IP Target et MAC Requested). Quelues explications sur les types ICMPv6 seront aussi les bienvenus.

- De créer un programme équivalent permettant de capturer et aficher les champs essentiels d'un ping en v4, incluant l'échange Req/Rep ARP.

Il vous est conseillé de passer par la phase intermédiaire d'analyse d'un fichier pcapng pour trouver les bons champs des bonnes trames ! Enregistrez un "ping" sur une machine (avec l'échange ARP), travaillez sur ce fichier, après quoi ce sera sans doute beaucoup plus facile.

Dans la prochaine étape, nous commencerons à sniffer automatiquement des choses indiscrètes qui passent sur les réseaux …
Mais avant tout, il faut vous familiariser avec tout cela.

Bon courage et n'oubliez pas de bien commenter vos programmes et de mettre tout votre travail sous Karuta !