Fabriquer des paquets réseaux avec Scapy

Remarques préalables

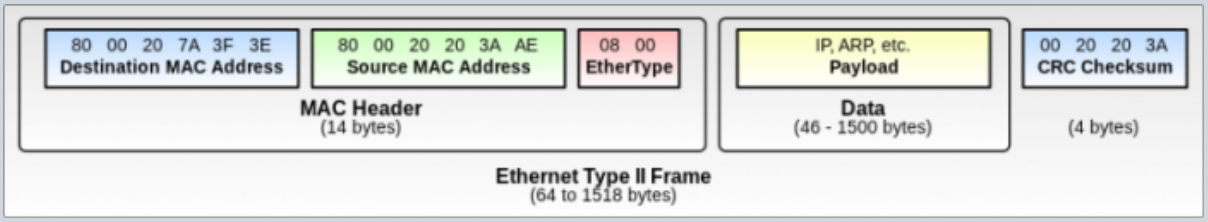
- il n'est pas nécessaire de renseigner tous les champs. Ce sont alors les valeurs par défaut qui sont utilisées ;
- on encapsule simplement les couches réseaux (du modèle OSI), des plus basses aux plus élevées avec l'opérateur **/** ;
- la résolution DNS est automatique automatique.

Créer une trame simple Ethernet simple

- créer une trame en mémoire et l'afficher

```
>>> trame = Ether()
>>> trame.show()
###[ Ethernet ]###
WARNING: Mac address to reach destination not found. Using broadcast.
  dst= ff:ff:ff:ff:ff:ff
  src= 00:00:00:00:00:00
  type= 0x9000
>>>
```

- Une trame éthernet est crée en instanciant la **classe Ether()**.
- la **méthode show()** de la classe affiche les informations de la trame. Comme aucun paramètre n'est fourni, ce sont les valeurs par défaut qui sont utilisées pour les attributs **dst**, **src** et **type**.



- renseigner les attributs de la trame en ajoutant l'adresse MAC du destinataire : <code python>

  trame.dsp = 'ac:84:c9:db:fb:c0'trame.show()

  >>> trame.dst='ac:84:c9:db:fb:c0'

  trame.show()

  #

  ##[Ethernet]### dst= ac:84:c9:db:fb:c0 src= 00:00:00:00:00:00 type= 0x9000

  Il est bien sur possible de préciser cette adresse MAC à la création de la trame : <code python>

  trame = Ether(dst='ac:84:c9:db:fb:c0')

  <

  /code>

- **Envoi** de la **trame Ethernet** sur le réseau. Utilisation de la fonction **sendp()** : <code python>

```
sendp(trame)
```

.

Sent 1 packets.

```
```

<

/code>

> Le point **“.”** représente un envoi.

> La trame Ethernet a été envoyée mais :
>
> - c'est une **coquille vide**
> - car cette trame ne contient aucune donnée.
>
> Il faut maintenant **encapsuler** des données d'un protocole des **couches supérieures** dans cette trame vide.

La commande **sendp()** permet d'envoyer un paquet créé (forgé) à partir du **niveau 2** (couche Ethernet).

La commande **send()** permet d'envoyer un paquet créé (forgé) à partir du **niveau 3** (couche IP). Les informations du niveau 2 (la couche ethernet) sont alors automatiquement renseigné par scapy.

Créer une trame Ethernet contenant un paquet ICMP

La **commande ping** qui utilise le **protocole ICMP** permet :

- d'envoyer un paquet ICMP **echo-request** à un hôte distant,
- et à indiquer si un paquet ICMP **echo-reply** a été renvoyé.

- création d'un paquet ICMP **echo-request** :

```
>>> ping = ICMP()
>>> ping.show()
###[ ICMP ]###
  type= echo-request
  code= 0
  chksum= None
  id= 0x0
  seq= 0x0
>>>
```

> Par défaut, **l'instanciation** de la **classe ICMP()** met le type du ping à **echo-request**.

> Page **Wikipedia** sur le protocole ICMP :
>
> - https://fr.wikipedia.org/wiki/Internet_Control_Message_Protocol

Quelques précisions sur le fonctionnement du protocole ICMP :

- ICMP se situe au même niveau que le protocole IP,
- mais qu'il soit à un niveau équivalent au protocole IP, un paquet ICMP doit néanmoins être encapsulé dans un datagramme IP.

Pour **envoyer** un paquet **ICMP**, il faut :

- **encapsuler** le paquet ICMP dans un **datagramme IP**,
- **encapsuler** à son tour le datagramme IP dans une **trame Ethernet**.

Avec Scapy, **l'encapsulation** entre protocoles se réalise avec l'opérateur **/ (slash)**.

- **Création d'une trame Ethernet** encapsulant un paquet **ICMP** destiné à être envoyé à l'adresse 192.168.1.1 (Box Internet) : <code python>

```
trame = Ether() / IP(dst='192.168.1.1') / ICMP()trame.show()
```

#

##[Ethernet]### dst= ac:84:c9:db:fb:c0 src= 00:15:5d:01:c6:02 type= 0x800 ###[IP]### version= 4 ihl= None tos= 0x0 len= None id= 1 flags= frag= 0 ttl= 64 proto= icmp chksum= None src= 192.168.1.159 dst= 192.168.1.1 \options\ ###[ICMP]### type= echo-request code= 0 chksum= None id= 0x0 seq= 0x0

<

/code>

Seule l'**adresse IP** du destinataire a été **renseignée**. Cependant Scapy a complété **automatiquement** les autres champs :

- pour les informations de la couche 2 Ethernet :
  - les **adresses MAC** source et destination : attributs **dst** et src
  - le **type** de trame Ethernet : attribut **type**
- pour la couche 3 IP :
  - les adresses IP source et destination
  - les autres informations de la couche 3.

- envoi de la trame Ethernet : <code python>

```
sendp(trame)
```

.

Sent 1 packets.

<

/code>

Le paquet est envoyé mais **aucune réponse n'est reçue**. Pour cela il faut utiliser les fonctions suivantes qui permettent **d'envoyer** la trame et de **recevoir** la réponse :

- **srp()** qui renvoie **deux** objets :

- le premier contient les **paquets émis** et leurs **réponses associée**s,
- l'autre contient les **paquets sans réponse**.

- **srp1()** fonction **plus simple** car ne renvoie **renvoie** qu'un seul objet, **la première réponse**.

```
>>> rep, non_rep = srp(trame)
Begin emission:
..Finished to send 1 packets.
.*
Received 4 packets, got 1 answers, remaining 0 packets
>>> rep
<Results: TCP:0 UDP:0 ICMP:1 Other:0>
>>> non_rep
<Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>
>>>
```

Avec Scapy :

- un point **“.”** représente un envoi,
- une étoile ***“ représente une réponse. </WRAP> Il y a une réponse et zéro échec. Ce plus la réponse est un paquet ICMP. Pour visualiser le contenu de la réponse, il suffit de regarder le contenu de la variable rep : <code python> »> rep.show() 0000 Ether / IP / ICMP 192.168.1.159 > 192.168.1.1 echo-request 0 ==⇒ Ether / IP / ICMP 192.168.1.1 > 192.168.1.159 echo-reply 0 / Padding »> </code> La variable rep contient une liste de couples de paquets : * le paquet envoyé, * et le paquet reçu (la réponse). Ici il n'y a qu'un seul couple de paquets puisqu'il n'y a qu'un seul envoi et une seule réponse. La variable rep est une liste et est manipulable comme une liste en python. Le résultat est un couple (tuple à deux valeurs) : * on affiche le paquet émis (le ICMP echo-request) avec rep[0][0].show(), * et on affiche le paquet reçu en réponse avec rep[0][1].show(). <code python> »> rep[0] (<Ether type=0x800 |<IP frag=0 proto=icmp dst=192.168.1.1 |<ICMP |»>, <Ether dst=00:15:5d:01:c6:02 src=ac:84:c9:db:fb:c0 type=0x800 |<IP version=4 ihl=5 tos=0x0 len=28 id=43315 flags= frag=0 ttl=64 proto=icmp chksum=0x4dbd src=192.168.1.1 dst=192.168.1.159 options=[] |<ICMP type=echo-reply code=0 chksum=0xffff id=0x0 seq=0x0 |<Padding load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |»») »> rep[0][0].show() ###[Ethernet]### dst= ac:84:c9:db:fb:c0 src= 00:15:5d:01:c6:02 type= 0x800 ###[IP]### version= 4 ihl= None tos= 0x0 len= None id= 1 flags= frag= 0 ttl= 64 proto= icmp chksum= None src= 192.168.1.159 dst= 192.168.1.1 \options\ ###[ICMP]### type= echo-request code= 0 chksum= None id= 0x0 seq= 0x0 »> rep[0][1].show() ###[Ethernet]### dst= 00:15:5d:01:c6:02 src= ac:84:c9:db:fb:c0 type= 0x800 ###[IP]### version= 4 ihl= 5 tos= 0x0 len= 28 id= 43315 flags= frag= 0 ttl= 64 proto= icmp chksum= 0x4dbd src= 192.168.1.1 dst= 192.168.1.159 \options\ ###[ICMP]### type= echo-reply code= 0 chksum= 0xffff id= 0x0 seq= 0x0 ###[Padding]### load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' »> </code> * pour ne visualiser que le paquet ICMP : <code python> »> rep[0][1][ICMP].show() ###[ICMP]### type= echo-reply code= 0 chksum= 0xffff id= 0x0 seq= 0x0 ###[Padding]### load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' »> </code>***

Important pour la suite :

- quand il y a une **réponse**, **Scapy** indique que le type est **echo-reply**,
- en fait le champ type contient alors la valeur 0 : **type = 0** ! pour indiquer qu'il y a une réponse
- revoir la page ICMP de Wikipedia à ce sujet : https://fr.wikipedia.org/wiki/Internet_Control_Message_Protocol

```
>>> rep[0][1][ICMP].type
0
>>>
```

Utilisation de la fonction **srp1()** qui ne renvoie qu'un seul objet, la **première réponse** : <code python> »> rep = srp1(trame) Begin emission: .Finished to send 1 packets. ..* Received 4 packets, got 1 answers, remaining 0 packets »> rep.show() ###[Ethernet]### dst= 00:15:5d:01:c6:02 src= ac:84:c9:db:fb:c0 type= 0x800 ###[IP]### version= 4 ihl= 5 tos= 0x0 len= 28 id= 43316 flags= frag= 0 ttl= 64 proto= icmp chksum= 0x4dbc src= 192.168.1.1 dst= 192.168.1.159 \options\ ###[ICMP]### type= echo-reply code= 0 chksum= 0xffff id= 0x0 seq= 0x0 ###[Padding]### load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' »> </code> ====== Créer un datagramme IP contenant un paquet ICMP ====== Si on ne s'**intéresse qu'à la partie IP** des paquets à gérer, on utilise alors les fonctions suivantes sans s'occuper du niveau 2 Ethernet qui est alors **automatiquement** renseigné par Scapy : * **send()** équivalent à **sendp()**, * **sr()** équivalent à **srp()**, * **sr1()** équivalentes à **srp1()**.

<code python> »> paquet = IP(dst='192.168.1.1') / ICMP() »> rep = sr1(paquet) Begin emission: ..Finished to send 1 packets. .* Received 4 packets, got 1 answers, remaining 0 packets »> rep.show() ###[IP]### version= 4 ihl= 5 tos= 0x0 len= 28 id= 43318 flags= frag= 0 ttl= 64 proto= icmp chksum= 0x4dba src= 192.168.1.1 dst= 192.168.1.159 \options\ ###[ICMP]### type= echo-reply code= 0 chksum= 0xffff id= 0x0 seq= 0x0 ###[Padding]### load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' »> </code>

====== Envoi d'un paquet sur un hôte non existant ======

<code python> »> paquet = IP(dst='192.168.1.200') / ICMP() »> rep = sr1(paquet) Begin emission: .....WARNING: Mac address to reach destination not found. Using broadcast. Finished to send 1 packets. ....^C Received 371 packets, got 0 answers, remaining 1 packets »> AttributeError: 'NoneType' object has no attribute 'show' »> rep »> </code>

Arrêt de l'envoi **(CTRL + C)** après quelques secondes et la variable **rep** est vide.

Il est possible de préciser une **limite de temps** en secondes à la fonction sr1(). <code python> »> rep = sr1(paquet, timeout=0.5) Begin emission: .....WARNING: Mac address to reach destination not found. Using broadcast. Finished to send 1 packets. ..... Received 67 packets, got 0 answers, remaining 1 packets »> </code>

Pour **visualiser** les autres paramètres de la **fonction sr1()** utiliser la commande:

```
>>> help(sr1)
```

==== Retour à Python : la bibliothèque Scapy ... ====

- Python : la bibliothèque Scapy pour manipuler les paquets réseau