

Travaux Pratiques - RT

Construction de trames avec Scapy

NOMs : GUETAT Florian SECK Pape Abdoulaye

Objectifs:

- Comprendre l'encapsulation
- Découverte de l'outil Scapy
- Construire des trames, les envoyer et analyser

Matériel et logiciel:

2 PC Linux Debian et Commutateur Ethernet

Logiciel Wireshark, Scapy

Questions :

1. Cherchez de la documentation sur l'outil scapy, son rôle, trouvez les commandes permettant de créer des trames/paquets, celles permettant d'afficher leur contenu et celles permettant de gérer l'envoi/réception des éléments générés.

Scapy est un programme développé en Python, il permet de forger, de recevoir et d'émettre via un réseau des paquets et/ou des trames de données vers ou depuis une infrastructure informatique et cela pour une multitude de protocoles réseaux différents. Pour créer une trame Ethernet, il faut utiliser Ether(). Pour le paquet ARP, il faut utiliser Ether()/ARP(). Pour le paquet IP, il faut utiliser IP(). Pour le paquet TCP, il faut utiliser TCP(). Pour le paquet ICMP, il faut utiliser ICMP(). Pour le paquet DNS, il faut utiliser DNS(). Pour le paquet DHCP, il faut utiliser DHCP(). Pour le paquet HTTP, il faut utiliser HTTP().

Envoi de paquets : La fonction send() vous permet d'envoyer des paquets préalablement construits vers une destination spécifique.

Capture de paquets : Utilisez la fonction sniff() pour capturer des paquets sur un réseau en spécifiant des filtres tels que les types de protocoles ou des critères spécifiques.

2. Installer scapy sur un de vos postes Debian

apt install scapy

3. Construire une trame Ethernet avec pour adresse MAC source celle de la machine avec Scapy et adresse MAC destination celle de votre 2^{ème} PC.

On lance scapy en tapant scapy dans le terminal. Puis on tape **sendp(trame)**.

Sur le PC avec scapy, on obtient :ape trame=Ether(). Ensuite, on écrit la source avec trame.src='d0:8e:79:15:d3:53'. Puis on écrit l'adresse destination avec trame.dst='d0:8e:79:15:d3:32'. On peut taper trame.show() pour vérifier.

```
>>> trame=Ether()
>>> trame.src='d0:8e:79:15:d3:53'
>>> trame.dst='d0:8e:79:15:d3:32'
>>> trame.show()
###[ Ethernet ]###
dst      = d0:8e:79:15:d3:32
src      = d0:8e:79:15:d3:53
type     = 0x9000

>>> 
```

4. Envoyer cette trame sur le réseau en lançant au préalable une capture Wireshark sur les deux postes. Quel résultat observez-vous ? Commentez.

On tape sendp(trame).

Sur le PC avec scapy, on obtient :

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Dell_86:46:9a	Broadcast	AoE	60	Query Config Information Re
2	2.547871333	Dell_15:d3:53	Dell_15:d3:32	LOOP	14	[Malformed Packet]
3	3.461431450	172.25.255.254	224.0.0.5	OSPF	78	Hello Packet
4	12.036670948	HewlettP_fc:b6:d3	Hangzhou_00:00:07	0x8918	60	Ethernet II
5	13.461375991	172.25.255.254	224.0.0.5	OSPF	78	Hello Packet
6	14.013142280	HewlettP_fc:b6:ff	LLDP_Multicast	LLDP	341	MA/94:3f:c2:fc:b6:d3 IN/Gig
7	17.392195098	HewlettP_0b:b7:b9	Broadcast	ARP	60	Who has 172.25.255.10? Tell
8	22.233134008	HewlettP_40:51:4c	Hangzhou_00:00:07	0x8918	60	Ethernet II
9	23.461528341	172.25.255.254	224.0.0.5	OSPF	78	Hello Packet
10	33.461304346	172.25.255.254	224.0.0.5	OSPF	78	Hello Packet

Sur le PC destination, on obtient :

The screenshot shows the Wireshark interface with a capture on interface *enp0s31f6. The packet list displays a single packet at time 10.315151863, source Dell_15:d3:53, destination Dell_15:d3:32, protocol LOOP, length 60, and info 'Unknown function'. The packet details pane shows 'Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface *enp0s31f6. Configuration Test Protocol (loopback)'. The packet bytes pane shows the MAC address d0:8e:79:15:d3:32 followed by 42 bytes of data.

La trame Ethernet est juste envoyé du PC source au PC destination sans suite.

5. Rappeler le principe de l'encapsulation. Comment faire avec Scapy pour créer un paquet IP encapsulé dans une trame Ethernet.

L'encapsulation est un concept en programmation et en réseaux où des données sont enveloppées et encapsulées dans une structure plus large pour les transporter ou les manipuler plus facilement. En termes de réseaux, cela signifie souvent encapsuler des données dans des couches supérieures pour les transmettre sur un réseau.

La commande a utilisée est :

eth_pkt=Ether(src='d0:8e:79:15:d3:53',dst='d0:8e:79:15:d3:32')

ip_pkt = IP(src='172.25.0.75', dst='172.25.0.74')

pkt = eth_pkt/ip_pkt

pkt.show()

Cela créera le paquet IP encapsulé dans une trame Ethernet avec les adresses MAC et IP spécifiées, et la commande pkt.show() affichera les détails de ce paquet.

```
>>> eth_pkt=Ether(src='d0:8e:79:15:d3:53',dst='d0:8e:79:15:d3:32')
>>> ip_pkt = IP(src='172.25.0.75', dst='172.25.0.74')
>>> pkt = eth_pkt/ip_pkt
>>> pkt.show()
###[ Ethernet ]###
  dst      = d0:8e:79:15:d3:32
  src      = d0:8e:79:15:d3:53
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = None
  tos      = 0x0
  len      = None
  id       = 1
  flags    =
  frag     = 0
  ttl      = 64
  proto    = hopopt
  checksum = None
  src     = 172.25.0.75
  dst     = 172.25.0.74
  \options \

>>> █
```

5. Nous souhaitons envoyer une requête ping vers la 2^{ème} machine. Construire la trame avec les éléments nécessaires.

```
>>> frame=Ether(src='d0:8e:79:15:d3:53',dst='d0:8e:79:15:d3:32')
>>> ip_pkt=IP(src='172.25.0.75', dst='172.25.0.74')
>>> icmp_pkt=ICMP()
>>> pkt=frame/ip_pkt/icmp_pkt
>>> pkt.show()
###[ Ethernet ]###
  dst      = d0:8e:79:15:d3:32
  src      = d0:8e:79:15:d3:53
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = None
  tos      = 0x0
  len      = None
  id       = 1
  flags    =
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = None
  src     = 172.25.0.75
  dst     = 172.25.0.74
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  checksum = None
  id       = 0x0
  seq      = 0x0
  unused   = ''
```

commande 6. Envoyer la trame sur le réseau et observez le résultat sur les 2 postes avec Wireshark.

On tape sendp(pkt).

Sur le PC avec scapy, on obtient request et reply car c'est un ping :

No.	Time	Source	Destination	Protocol	Length	Info
33	15.667729594	172.25.0.75	172.25.0.74	ICMP	42	Echo (ping) request id=0x0
36	15.668887134	172.25.0.74	172.25.0.75	ICMP	60	Echo (ping) reply id=0x0

Sur le PC destination, on obtient echo car pour lui, il reçoit un message et envoie un message de réponse :

***enp0s31f6**

Fichier Editer Vue Aller Capture Analyser Statistiques Telephonie Wireless Outils Aide

icmp

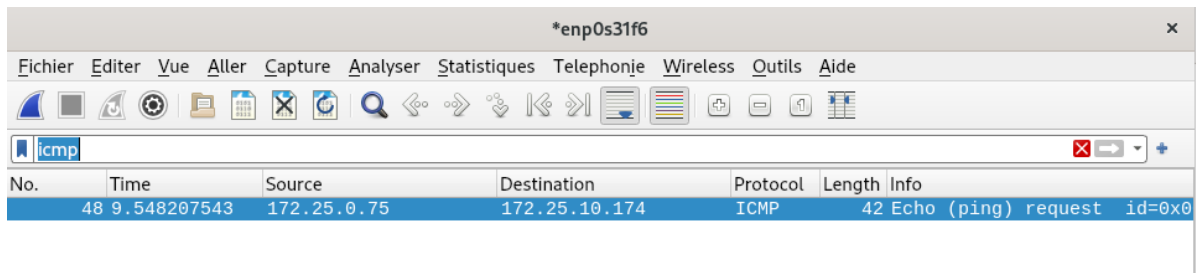
No.	Time	Source	Destination	Protocol	Length	Info
19	25.668496075	172.25.0.75	172.25.0.74	ICMP	60	Echo (ping) request id=0x0
22	25.669026821	172.25.0.74	172.25.0.75	ICMP	42	Echo (ping) reply id=0x0

Frame 19: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface *enp0s31f6
Ethernet II, Src: Dell_15:d3:53 (d0:8e:79:15:d3:53), Dst: 08:00:00:00:00:00
Internet Protocol Version 4, Src: 172.25.0.75, Dst: 172.25.0.74
Internet Control Message Protocol

0000 d0 8e 79 15 d3 32 d0 8e 79 15 d3 53 08
0010 00 1c 00 01 00 00 40 01 22 19 ac 19 00
0020 00 4a 08 00 f7 ff 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00

7. Modifier l'en-tête IP en mettant une adresse IP destination inexistante sur le réseau (sans changer l'adresse MAC). Renvoyer la trame, observez et expliquer le résultat obtenu sur Wireshark sur les 2 machines.

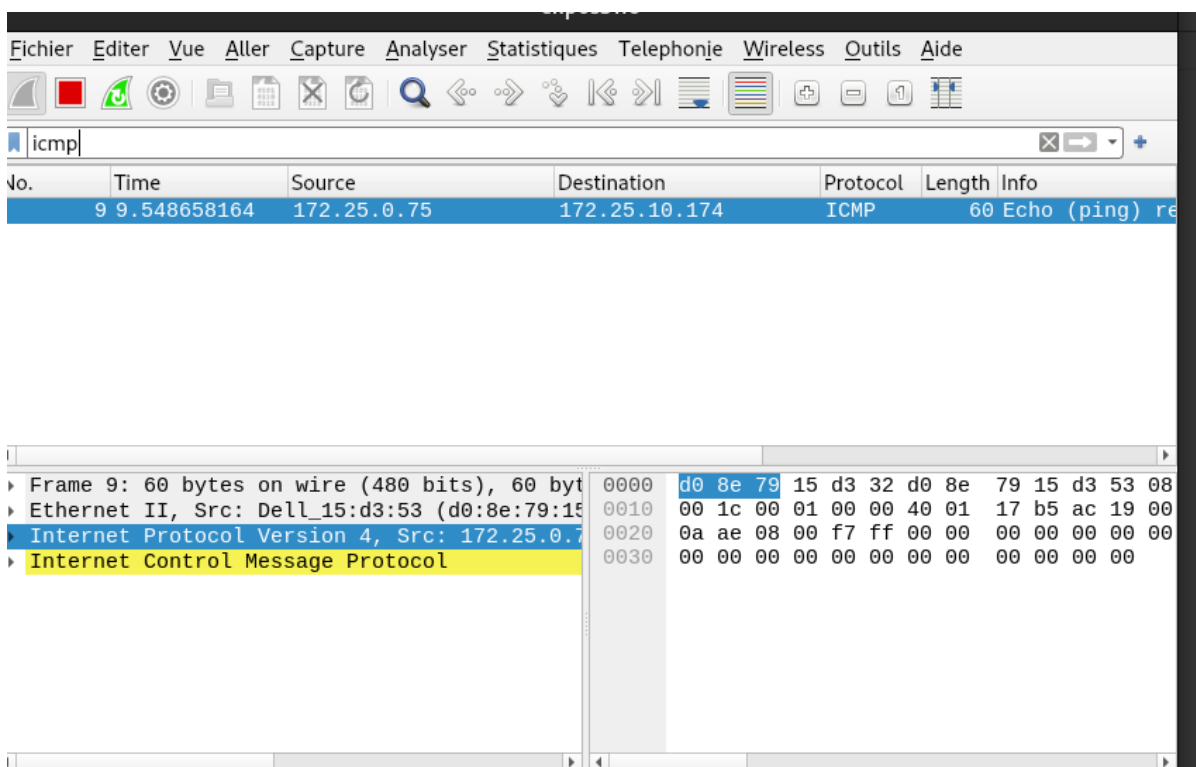
En modifiant avec une adresse Ip destination inexistante, la trame par bien du PC source mais ne revient jamais :



Wireshark capture on interface *enp0s31f6. The packet list shows a single ICMP Echo (ping) request.

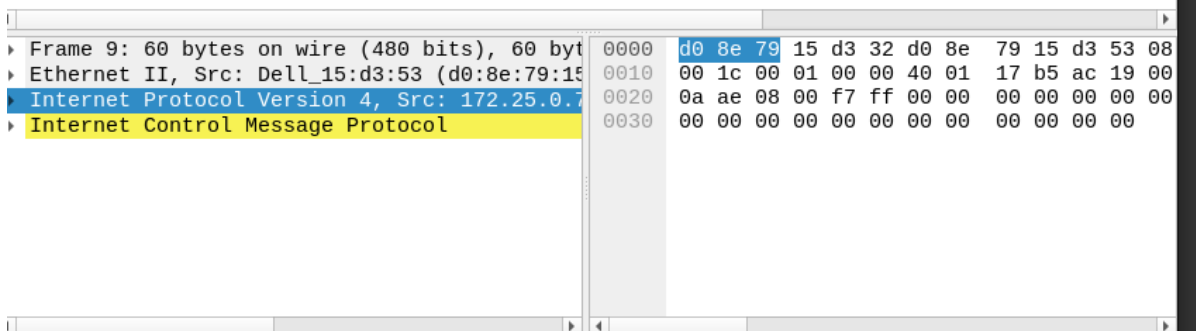
No.	Time	Source	Destination	Protocol	Length	Info
48	9.548207543	172.25.0.75	172.25.10.174	ICMP	42	Echo (ping) request id=0x0

Sur le PC destination, on reçoit la trame du fait de l'adresse MAC mais ne la calcule pas car l'@IP ne correspond pas à la sienne :



Wireshark capture on interface *enp0s31f6. The packet list shows a single ICMP Echo (ping) request.

No.	Time	Source	Destination	Protocol	Length	Info
9	9.548658164	172.25.0.75	172.25.10.174	ICMP	60	Echo (ping) request id=0x0



Wireshark packet details for the selected ICMP Echo (ping) request (Frame 9).

Frame 9: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface *enp0s31f6
Ethernet II, Src: Dell_15:d3:53 (d0:8e:79:15:d3:53), Dst: 172.25.10.174 (08:00:27:00:00:00)
Internet Protocol Version 4, Src: 172.25.0.75, Dst: 172.25.10.174
Internet Control Message Protocol

8. Renvoyer la même trame mais en utilisant la commande scapy qui permet d'attendre la réponse au ping et la conserver dans une variable. Afficher le résultat.

On fait `r=srp(pkt2)` pour avoir la réponse dans `r`.

Puis on fait `r.show` pour montrer la réponse.

```
>>> r = srp(pkt2)
Begin emission:
Finished sending 1 packets.
^C
Received 0 packets, got 0 answers, remaining 1 packets
>>> r.show
-----
AttributeError                                Traceback (most recent call last)
Cell In [47], line 1
----> 1 r.show

AttributeError: 'tuple' object has no attribute 'show'
>>> 
```

Il n'y a pas de réponse car l'adresse ip n'est pas dans le réseau.

9. Il est possible d'utiliser Scapy comme analyseur de réseau (commande sniff). Utiliser cette commande pour capturer des pings envoyés ou reçus par votre machine et afficher le résultat. Même question pour capturer 10 trames de trafic de navigation sur internet (HTTP ou HTTPS).

```
>>> d=sniff(filter="icmp")
^C>>> d.show()
0000 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0001 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0002 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0003 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0004 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0005 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0006 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0007 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0008 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0009 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0010 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0011 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0012 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0013 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0014 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0015 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0016 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0017 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0018 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0019 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0020 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0021 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0022 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0023 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0024 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0025 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0026 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0027 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0028 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-request 0 / Raw
0029 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-reply 0 / Raw
0030 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-request 0 / Raw
0031 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-reply 0 / Raw
0032 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-request 0 / Raw
0033 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-reply 0 / Raw
0034 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-request 0 / Raw
0035 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-reply 0 / Raw
0036 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-request 0 / Raw
0037 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-reply 0 / Raw
0038 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-request 0 / Raw
0039 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-reply 0 / Raw
0040 Ether / IP / ICMP 172.25.0.74 > 172.25.0.75 echo-request 0 / Raw
0041 Ether / IP / ICMP 172.25.0.75 > 172.25.0.74 echo-reply 0 / Raw
>>> █
```

On a mis le sniff de l'icmp car on veut que les pings dans une variable. Ensuite, on ping dans un sens et dans l'autre puis on fait afficher avec d.show().


```

^C>>> p=sniff(filter="tcp")
^C>>> p.show()
0000 Ether / IP / TCP 34.107.243.93:https > 172.25.0.75:48324 PA / Raw
0001 Ether / IP / TCP 172.25.0.75:48324 > 34.107.243.93:https PA / Raw
0002 Ether / IP / TCP 34.107.243.93:https > 172.25.0.75:48324 A
0003 Ether / IP / TCP 193.49.117.19:https > 172.25.0.75:34608 PA / Raw
0004 Ether / IP / TCP 193.49.117.19:https > 172.25.0.75:34608 PA / Raw
0005 Ether / IP / TCP 193.49.117.19:https > 172.25.0.75:34608 FA
0006 Ether / IP / TCP 172.25.0.75:34608 > 193.49.117.19:https A
0007 Ether / IP / TCP 172.25.0.75:34608 > 193.49.117.19:https A
0008 Ether / IP / TCP 172.25.0.75:34608 > 193.49.117.19:https PA / Raw
0009 Ether / IP / TCP 172.25.0.75:34608 > 193.49.117.19:https FPA / Raw
0010 Ether / IP / TCP 193.49.117.19:https > 172.25.0.75:34608 RA
0011 Ether / IP / TCP 193.49.117.19:https > 172.25.0.75:34608 R / Padding
0012 Ether / IP / TCP 172.25.0.75:33394 > 34.120.237.76:https PA / Raw
0013 Ether / IP / TCP 172.25.0.75:33394 > 34.120.237.76:https PA / Raw
0014 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 A
0015 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 A
0016 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 PA / Raw
0017 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 PA / Raw
0018 Ether / IP / TCP 172.25.0.75:33394 > 34.120.237.76:https PA / Raw
0019 Ether / IP / TCP 172.25.0.75:33394 > 34.120.237.76:https A
0020 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 A
0021 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 PA / Raw
0022 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 PA / Raw
0023 Ether / IP / TCP 172.25.0.75:33394 > 34.120.237.76:https PA / Raw
0024 Ether / IP / TCP 172.25.0.75:33394 > 34.120.237.76:https A
0025 Ether / IP / TCP 34.120.237.76:https > 172.25.0.75:33394 A
0026 Ether / IP / TCP 172.25.0.75:33922 > 34.149.97.1:https PA / Raw
0027 Ether / IP / TCP 172.25.0.75:46120 > 34.199.19.32:https PA / Raw
0028 Ether / IP / TCP 34.149.97.1:https > 172.25.0.75:33922 PA / Raw
0029 Ether / IP / TCP 172.25.0.75:33922 > 34.149.97.1:https A
0030 Ether / IP / TCP 34.199.19.32:https > 172.25.0.75:46120 PA / Raw
0031 Ether / IP / TCP 172.25.0.75:46120 > 34.199.19.32:https A
0032 Ether / IP / TCP 172.25.0.75:40708 > 216.58.198.78:https S
0033 Ether / IP / TCP 172.25.0.75:40716 > 216.58.198.78:https S
0034 Ether / IP / TCP 216.58.198.78:https > 172.25.0.75:40708 SA
0035 Ether / IP / TCP 216.58.198.78:https > 172.25.0.75:40716 SA
0036 Ether / IP / TCP 172.25.0.75:40708 > 216.58.198.78:https A
0037 Ether / IP / TCP 172.25.0.75:40716 > 216.58.198.78:https A
0038 Ether / IP / TCP 172.25.0.75:40716 > 216.58.198.78:https PA / Raw
0039 Ether / IP / TCP 172.25.0.75:40708 > 216.58.198.78:https PA / Raw
0040 Ether / IP / TCP 216.58.198.78:https > 172.25.0.75:40716 A
0041 Ether / IP / TCP 216.58.198.78:https > 172.25.0.75:40708 A
0042 Ether / IP / TCP 216.58.198.78:https > 172.25.0.75:40716 PA / Raw

```

On a mis le sniff du tcp car on veut les http et https dans une variable. Ensuite, on va sur internet et on fait afficher [variable].show().