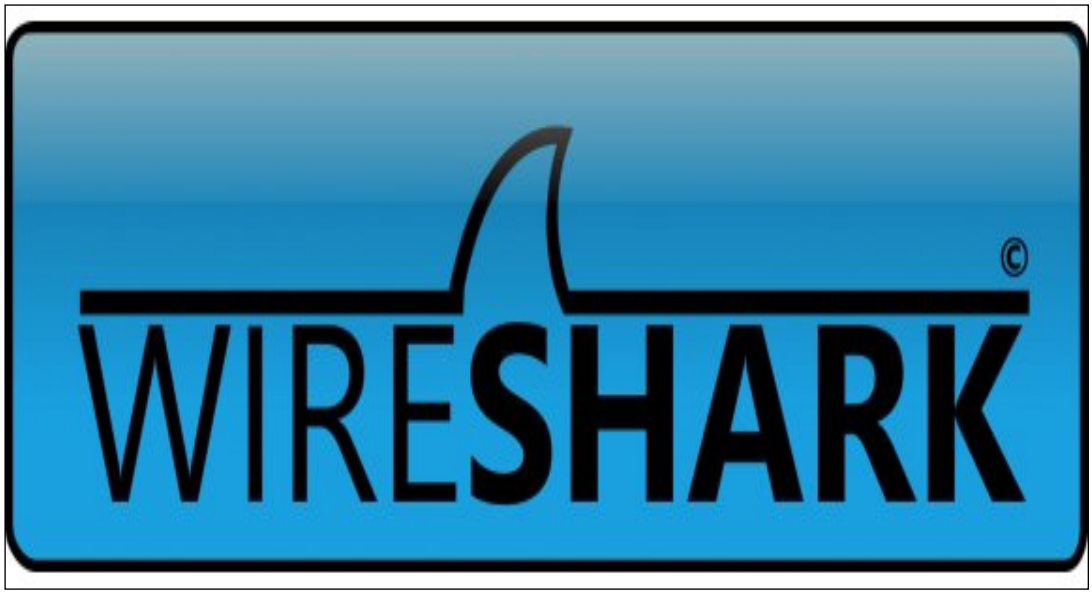
Les Protocoles de Transport

M1204 - Services sur Réseaux

Patrice Gommery - Décembre 2019



Dans la continuité du TD sur la capture de trames, nous allons encore une fois aborder la notion de ports et de protocoles de transport indispensables au bon fonctionnement des applications réseaux.

IMPORTANT : La première partie de ce TP est plutôt théorique mais sa lecture est OBLIGATOIRE pour bien comprendre les exercices suivants.

CONSIGNES GENERALES :

Dans tout l'exercice,
remplacez **VMID** par l'identifiant de votre machine virtuelle,
remplacez **ID** par la seconde partie de l'identifiant

Rappel l'IP de votre machine est **172.16.VM.ID**
Le **serveur FTP** de la salle se trouve à l'adresse : **172.16.100.1**,
Sur ce serveur, vous disposez d'un compte **mmis1** avec le mot de passe : **PASSWORD**

RAPPELS : ACCEDER A VOTRE MACHINE VIRTUELLE :

A l'aide d'un terminal, accéder à votre serveur en SSH. Syntaxe : **`ssh root@172.16.VM.ID`**

Votre serveur a été réinitialisé , le mot de passe de **root** est donc **PASSWORD**.

EN CAS DE PROBLEME AVEC LE TERMINAL SSH
(après un reboot par exemple)
A l'aide d'un navigateur web, accéder à la console de supervision **PROMOX** à l'adresse :
`https://172.16.0.250:8006`

PREAMBULE : INSTALLATION DES OUTILS

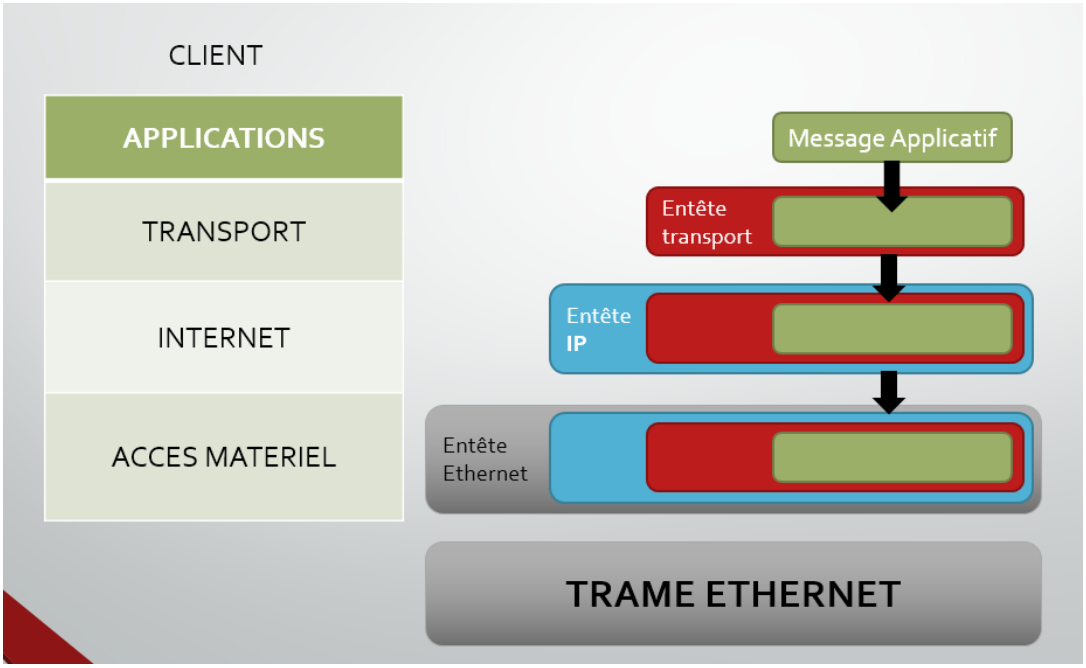
VERIFICATIONS :

Ouvrez un navigateur sur votre poste de travail avec l'URL : **www.domVMID.net**
Vous devriez voir la page apache ou celles des 6 thés perdus.
Si ce n'est pas le cas, apache2 n'est pas installé ou mal démarré , corrigez le problème.

Créez un utilisateur nommé **usertest** (mot de passe **123**)
Ouvrez Filezilla et connectez-vous à votre serveur en FTP avec le compte **usertest**
Si la connexion de fonctionne pas, vérifiez que **proftpd** est bien installé et démarré.
Corrigez le problème en installant et configurant le paquet nécessaire.

PARTIE 1 : LES PROTOCOLES DE TRANSPORT

Pour comprendre le rôle des protocoles de transport, reprenons le modèle en couche TCP/IP et le fonctionnement d'une application réseau.



EN PARTANT DU HAUT VERS LE BAS : PRINCIPE DE L'ENCAPSULATION

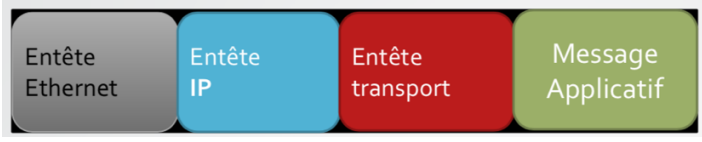
1) L'application génère sa requête (Exemple le navigateur envoi un GET index.html à un serveur web)
Elle utilise le **protocole applicatif** qui la caractérise (Exemple http pour les pages web)

2) L'application confie son **message applicatif** à son **protocole de transport** préféré
(Exemple pour http, le message sera confié à TCP. Voir tableau plus bas)

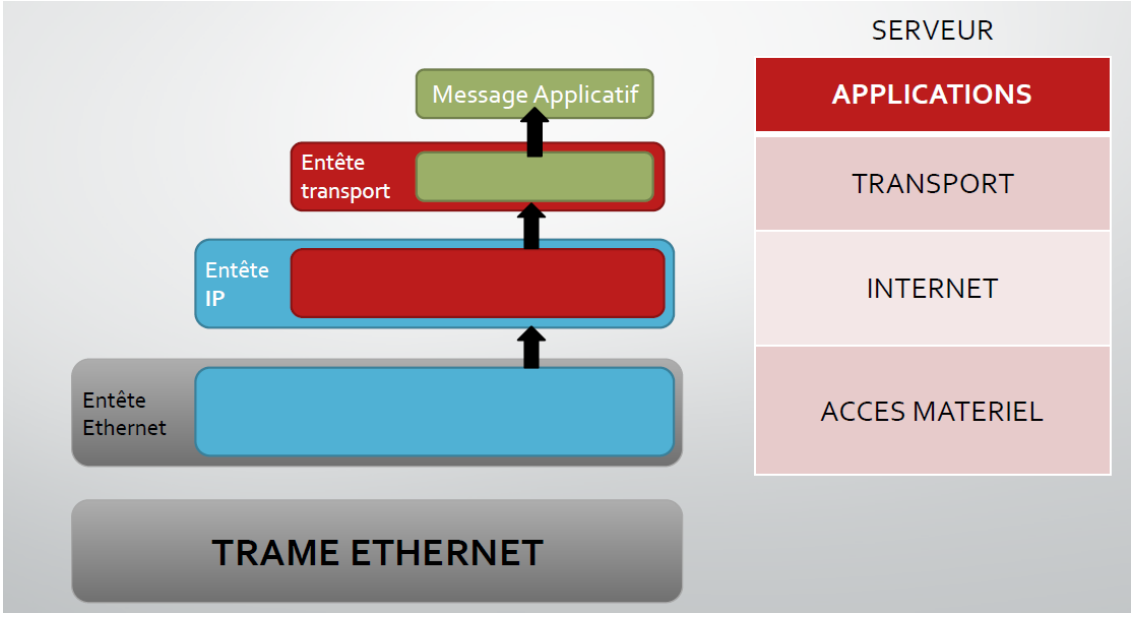
3) Le protocole de transport ajoute son **en-tête** devant le message applicatif pour constituer un **segment** .
Cet **en-tête** contient notamment le **n° de port** de l'application serveur à qui est destiné le segment , ainsi que le n° de port attribué à la session de l'application cliente.
(Exemple pour http, le port serveur sera par défaut 80. Le port client est indépendant du protocole applicatif, il est incrémenté à chaque session de communication en partant de 1024)

4) Le protocole de transport confie le segment au **protocole IP** qui ajoute son propre en-tête pour former un paquet. Cet en-tête contient l'identifiant du protocole de transport (TCP ou UDP) ainsi
5) Le protocole IP confie le **paquet** à la couche matériel (Exemple Ethernet) qui ajoutera son propre en-tête pour constituer une trame. Dans le cas d'Ethernet ou d'une connexion Wifi, l'en-tête contiendra l'**adresse MAC** du poste émetteur, ainsi que celle de la machine de destination . Notez que ces adresses MAC peuvent être celles des routeurs situés physiquement entre le client et le serveur.

6) Finalement , c'est cette **trame** qui est émise sur le réseau.



Une fois la trame arrivée sur la machine de destination , on effectue l'opération inverse (en partant du bas dans le schéma), chacun des protocoles identifient le suivant en relisant l'en-tête de la trame, du paquet ou du segment qui lui est destiné. L'application récupère finalement le message applicatif envoyé.



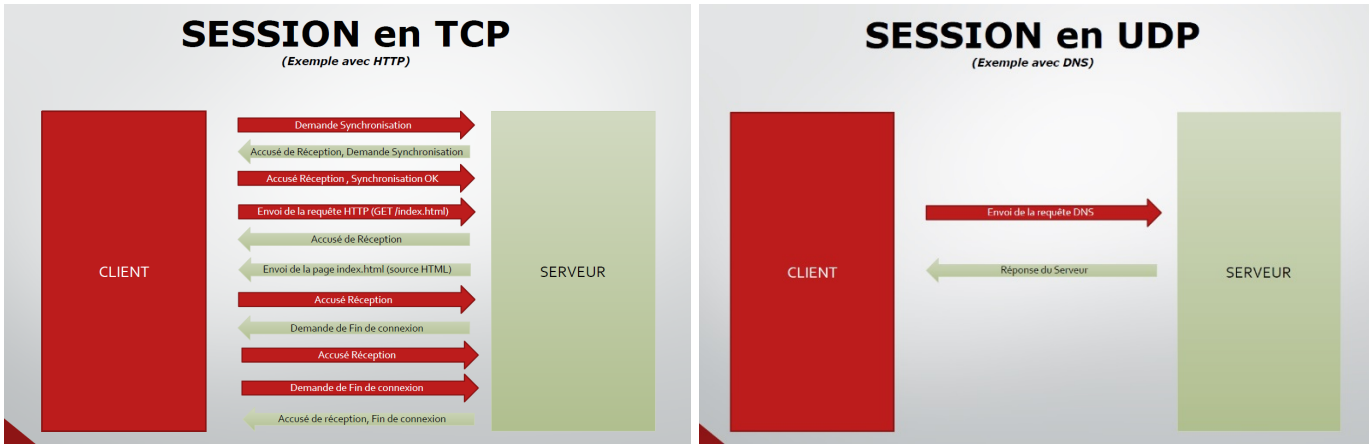
A chaque échange entre le client et le serveur (ou le serveur et le client) on répète les mêmes opérations . Le nombre de trames échangées peut bien sûr varier en fonction de la taille des données échangées entre les deux applications, mais aussi en fonction du protocole de transport utilisé. Voyons les différences fondamentales entre TCP et UDP , les deux protocoles de transport associés à IP.

	UDP	**TCP**
Etablissement d'une connexion	Non	Oui
Fin de connexion	Non	Oui
Accusés de réception	Non	Oui
Segmentation des données	Non	Oui
Taille maximum des données	64 Ko	Segmentation
Erreurs de transmission gérées par le protocole de transport	Non	Oui
Erreurs de transmission gérées par la couche application	Oui	Optionnel

Comme vous pouvez le constater, TCP semble beaucoup plus complet qu'UDP, et c'est le cas.

TCP est en effet un protocole fonctionnant en **mode connecté**, ce qui implique qu'avant d'échanger les données applicatives, il doit déjà établir une connexion fiable entre les deux machines qui communiquent. Dans le même esprit, il s'assurera que l'échange de données c'est bien terminé avant de mettre fin à la session de communication. De plus entre l'ouverture et la fermeture de la session de communication, il utilisera des **accusés de réception** pour vérifier que toutes les données envoyées ont bien été reçues. Cela en fait donc un protocole très **fiable** qui décharge totalement l'application de la gestion des erreurs de communication. Pour finir, sa gestion des segments **ne limite pas la taille des données transportées**.

Alors pourquoi un applicatif préférait utiliser UDP plutôt que TCP ?
Tout simplement parce que cette gestion fiable du transport des données à un coût en nombre de trames générées. Regardez le schéma suivant :



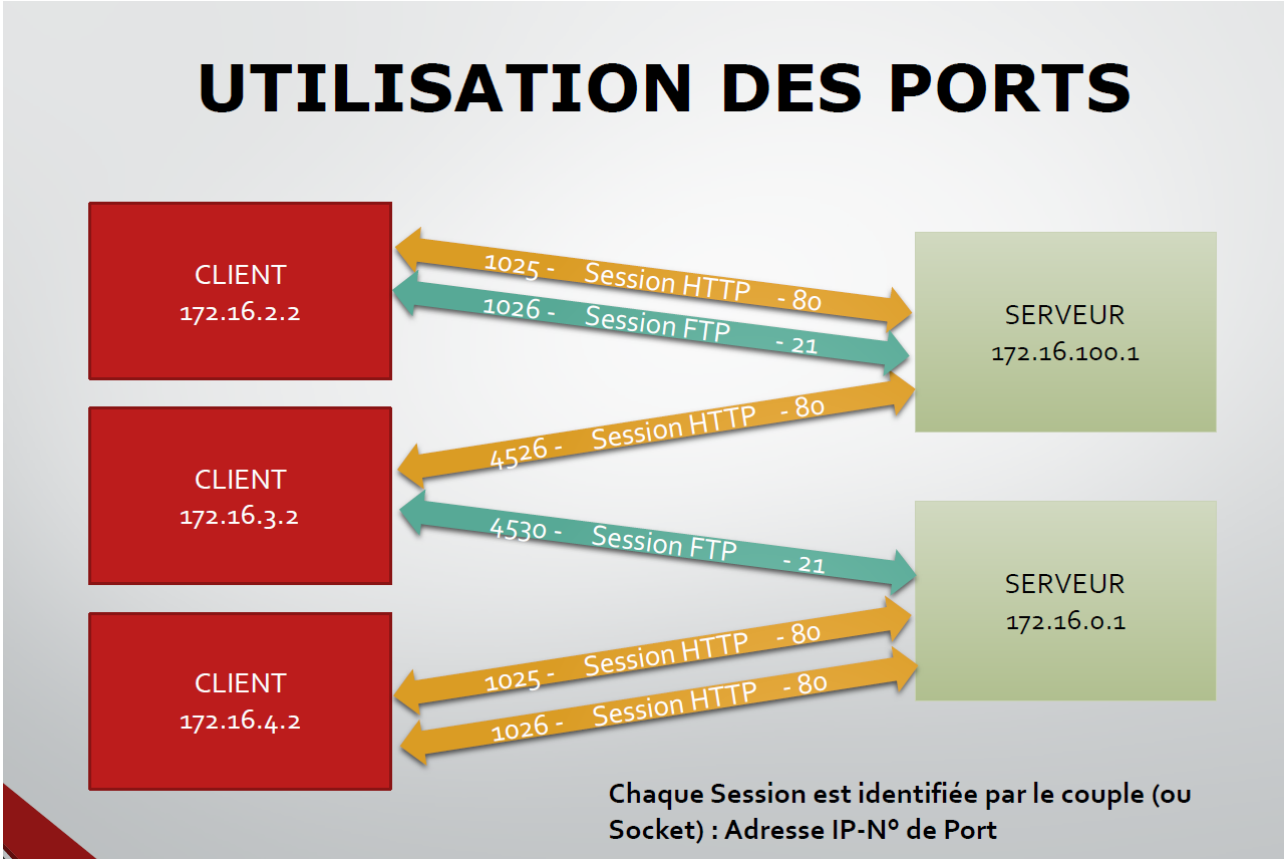
A gauche : Une session TCP. Exemple le chargement d'une page simple web en http.
A droite : Une session UDP. Exemple une requête DNS et sa réponse.

On voit clairement la gestion de la session par TCP avec les trames de synchronisation et les trames de fin de session, ainsi que les accusés de réception. En UDP aucun contrôle et donc aucune garantie pour l'application que ses données aient été correctement transmises. Le choix est donc simple : **Fiabilité ou Rapidité ?** Comme vous pouvez voir dans le tableau ci-dessous, tous les protocoles ayant besoin de transmettre de manière fiable des données utilisent en majorité TCP, alors que ceux qui ont plutôt besoin d'une réponse rapide à une requête et émettent des messages courts utilisent UDP.

Protocole	Usage	Transport	N°Port
FTP	Transfert de fichiers	TCP	21
SSH	Terminal distant sécurisé	TCP	22
TELNET	Terminal distant non sécurisé	TCP	23
SMTP	Envoi des Mails	TCP	25
DNS	Résolution des noms de domaines	UDP	53
DHCP	Attribution Dynamique de la configuration IP	UDP	67
TFTP	Transfert de fichiers trivial (non fiabilisé)	UDP	69
HTTP	Transport des pages Web	TCP	80
POP	Réception des Mails (Lecture BAL)	TCP	110
NNTP	Synchronisation Serveur de Temps	TCP	119
IMAP	Réception des Mails (Synchronisation)	TCP	143
SNMP	Supervision de réseau	UDP	161
HTTPS	Transport des pages avec chiffrement	TCP	443
FTPS	Transfert de fichiers (Sécurisé par SSL)	TCP	990
MYSQL	Moteur de Base de données	TCP	3306
RDP	Bureau à distance Windows	TCP	3389

Chaque Protocole applicatif possède un n° de port conventionnel. Vous remarquerez qu'à part les deux derniers qui sont particuliers (car issus d'un éditeur de logiciels) , tous **les numéros de ports sont inférieurs à 1024**. Tous **ces numéros désignent exclusivement le port coté Serveur** et restent fixés quel que soit le nombre de sessions de communication ouvertes sur la machine.

Qu'en est-il du port côté client ? Comme vous pourrez le voir dans la suite de ce TP, il évolue en fonction du nombre de sessions ouvertes sur la machine, indépendamment de l'application ou du protocole de transport utilisé.

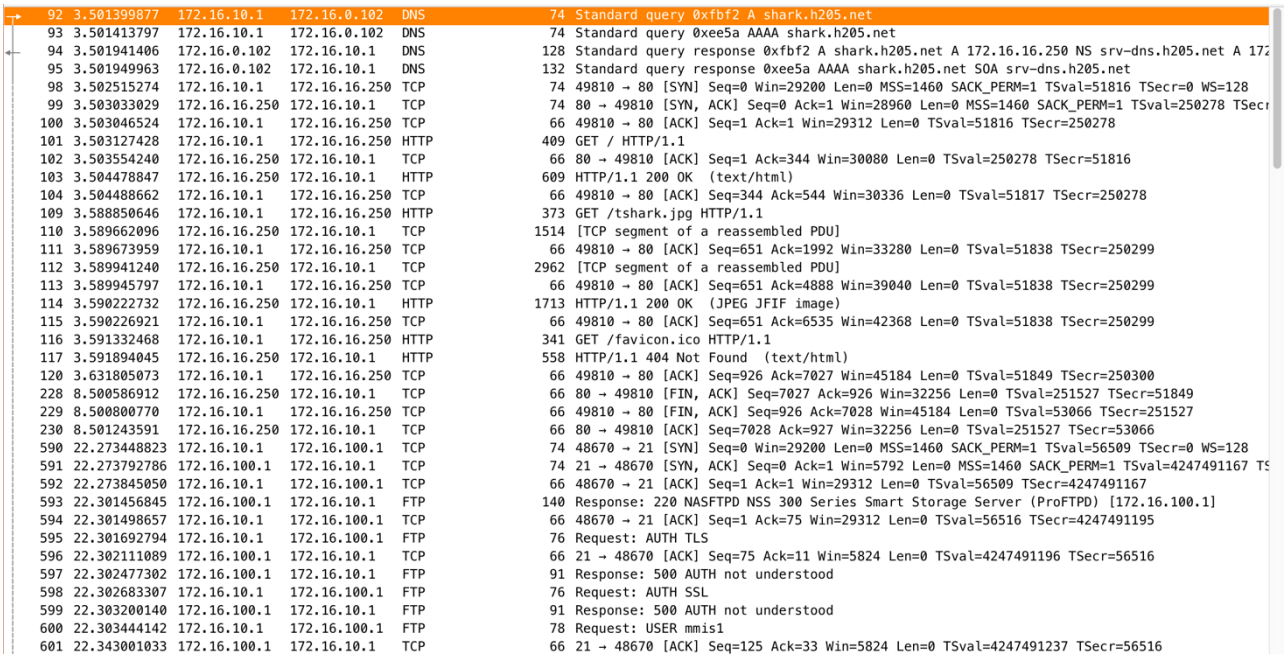


Observez le schéma ci-dessus. On voit le client 172.16.2.2 qui ouvre une première session en FTP en utilisant le port 1025, puis une seconde en http en utilisant le port 1026. S'il ouvre une nouvelle page , il utilisera le port 1027 et ainsi de suite. Ce n'est pas le changement de protocole qui fait changer le n° de port, c'est bien l'ouverture d'une nouvelle session de communication comme le montre le schéma avec le client 172.16.4.2.

Pour bien illustrer ce concept, ouvrons **wireshark** sur votre poste de travail comme vous l'avez fait pendant le dernier TD (reprenez le TD si nécessaire) :

1) Lancez une capture
2) puis ouvrez votre navigateur à l'adresse : **shark.h205.net**
3) Ouvrez ensuite **Filezilla** et connectez-vous au serveur ftp de la salle (172.16.100.1) avec l'utilisateur **mmis1 (**mot de passe : **PASSWORD)**
4) Arrêtez la capture

Observons maintenant le résultat :



```
 92 3.501399877  172.16.10.1   172.16.0.102  DNS              74 Standard query 0xfbf2 A shark.h205.net
 93 3.501413797  172.16.10.1   172.16.0.102  DNS              74 Standard query 0xee5a AAAA shark.h205.net
 94 3.501941406  172.16.0.102  172.16.10.1   DNS             128 Standard query response 0xfbf2 A shark.h205.net A 172.16.16.250 NS srv-dns.h205.net A 172
 95 3.501949963  172.16.0.102  172.16.10.1   DNS             132 Standard query response 0xee5a AAAA shark.h205.net SOA srv-dns.h205.net
 98 3.502515274  172.16.10.1   172.16.16.250 TCP              74 49810 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=51816 TSecr=0 WS=128
 99 3.503033029  172.16.16.250 172.16.10.1   TCP              74 80 → 49810 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=250278 TSecr
100 3.503046524  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=51816 TSecr=250278
101 3.503127428  172.16.10.1   172.16.16.250 HTTP            409 GET / HTTP/1.1
102 3.503554240  172.16.16.250 172.16.10.1   TCP              66 80 → 49810 [ACK] Seq=1 Ack=344 Win=30080 Len=0 TSval=250278 TSecr=51816
103 3.504478847  172.16.16.250 172.16.10.1   HTTP             609 HTTP/1.1 200 OK  (text/html)
104 3.504488662  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [ACK] Seq=344 Ack=544 Win=30336 Len=0 TSval=51817 TSecr=250278
109 3.588850646  172.16.10.1   172.16.16.250 HTTP             373 GET /tshark.jpg HTTP/1.1
110 3.589662096  172.16.16.250 172.16.10.1   TCP             1514 [TCP segment of a reassembled PDU]
111 3.589673959  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [ACK] Seq=651 Ack=1992 Win=33280 Len=0 TSval=51838 TSecr=250299
112 3.589941240  172.16.16.250 172.16.10.1   TCP             2962 [TCP segment of a reassembled PDU]
113 3.589945797  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [ACK] Seq=651 Ack=4888 Win=39040 Len=0 TSval=51838 TSecr=250299
114 3.590222732  172.16.16.250 172.16.10.1   HTTP            1713 HTTP/1.1 200 OK  (JPEG JFIF image)
115 3.590226921  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [ACK] Seq=651 Ack=6535 Win=42368 Len=0 TSval=51838 TSecr=250299
116 3.591332468  172.16.10.1   172.16.16.250 HTTP             341 GET /favicon.ico HTTP/1.1
117 3.591894045  172.16.16.250 172.16.10.1   HTTP             558 HTTP/1.1 404 Not Found  (text/html)
120 3.631805073  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [ACK] Seq=926 Ack=7027 Win=45184 Len=0 TSval=51849 TSecr=250300
228 8.500586912  172.16.16.250 172.16.10.1   TCP              66 80 → 49810 [FIN, ACK] Seq=7027 Ack=926 Win=32256 Len=0 TSval=251527 TSecr=51849
229 8.500800770  172.16.10.1   172.16.16.250 TCP              66 49810 → 80 [FIN, ACK] Seq=926 Ack=7028 Win=45184 Len=0 TSval=53066 TSecr=251527
230 8.501243591  172.16.16.250 172.16.10.1   TCP              66 80 → 49810 [ACK] Seq=7028 Ack=927 Win=32256 Len=0 TSval=251527 TSecr=53066
590 22.273448823 172.16.10.1   172.16.100.1  TCP              74 48670 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=56509 TSecr=0 WS=128
591 22.273792786 172.16.100.1  172.16.10.1   TCP              74 21 → 48670 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=4247491167 TS
592 22.273845050 172.16.10.1   172.16.100.1  TCP              66 48670 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=56509 TSecr=4247491167
593 22.301456845 172.16.100.1  172.16.10.1   FTP             140 Response: 220 NASFTPD NSS 300 Series Smart Storage Server (ProFTPD) [172.16.100.1]
594 22.301498657 172.16.10.1   172.16.100.1  TCP              66 48670 → 21 [ACK] Seq=1 Ack=75 Win=29312 Len=0 TSval=56516 TSecr=4247491195
595 22.301692794 172.16.10.1   172.16.100.1  FTP              76 Request: AUTH TLS
596 22.302111089 172.16.100.1  172.16.10.1   TCP              66 21 → 48670 [ACK] Seq=75 Ack=11 Win=5824 Len=0 TSval=4247491196 TSecr=56516
597 22.302477302 172.16.100.1  172.16.10.1   FTP              91 Response: 500 AUTH not understood
598 22.302683307 172.16.10.1   172.16.100.1  FTP              76 Request: AUTH SSL
599 22.303200140 172.16.100.1  172.16.10.1   FTP              91 Response: 500 AUTH not understood
600 22.303444142 172.16.10.1   172.16.100.1  FTP              78 Request: USER mmis1
601 22.343001033 172.16.100.1  172.16.10.1   TCP              66 21 → 48670 [ACK] Seq=125 Ack=33 Win=5824 Len=0 TSval=4247491237 TSecr=56516
```

Effectivement, dans un premier temps la suite de trames capturées est longue, très longue.
Heureusement, wireshark vous permet d'appliquer des filtres (voir en fin de ce pdf)

Ici, un filtre sur l'adresse du poste ou a été réalisée la capture.
Observons maintenant le détail de la première trame :

```
 92 3.501399877  172.16.10.1   172.16.0.102  DNS              74 Standard query 0xfbf2 A shark.h205.net
```

```
▶ Frame 92: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▶ Ethernet II, Src: Dell_2c:94:ac (18:66:da:2c:94:ac), Dst: fe:f1:39:45:07:01 (fe:f1:39:45:07:01)
▶ Internet Protocol Version 4, Src: 172.16.10.1, Dst: 172.16.0.102
▶ User Datagram Protocol, Src Port: 57127, Dst Port: 53
▶ Domain Name System (query)
```

Nous voyons ici, les informations des différentes couches qui ont constitué la trame :
(en partant du bas)

APPLICATIONS	Domaine Name System (DNS)	Requête DNS (query)
TRANSPORT	User Datagram Protocol (UDP)	Port Destination : 53 (Serveur DNS) Port Source : 57127 (Poste Client)
INTERNET	Internet Protocol Version 4 (IPv4)	IP source : 172.16.10.1 (Poste Client) IP Destination : 172.16.0.102 (Serveur DNS)
ACCES MATERIEL	Ethernet II	MAC source : 18:66:da:2c:94:ac (Poste) MAC destination : fe:f1:39:45:07:01 (Serveur)

Bien entendu, les valeurs Poste Client et Serveur dépendent du sens de communication.
Dans cette trame c'est bien le poste qui est la source, dans les trames suivantes il deviendra la destination (Normal, c'est le serveur qui lui répond). Ne confondez donc pas source/destination avec client/serveur, ce sont 2 concepts différents.

Dans un premier temps, ces informations nous suffisent pour comprendre à quoi sert cette trame. On voit bien que c'est un poste qui émet une requête DNS vers un serveur DNS (53) pour connaître l'IP d'une machine . Mais quelle machine ? Pour le savoir, il suffit de développer les informations du protocole applicatif (Domain Name System) :

```
▼ Queries
  ▼ shark.h205.net: type A, class IN
      Name: shark.h205.net
      [Name Length: 14]
      [Label Count: 3]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
```

On voit clairement, ici que l'adresse IP recherchée (Type A) est celle de la machine **shark.h205.net** (le nom qui a été saisi dans le navigateur)

Regardons, la réponse du serveur (trame 94 dans l'exemple) :

```
▼ Answers
  ▶ shark.h205.net: type A, class IN, addr 172.16.16.250
▼ Authoritative nameservers
  ▶ h205.net: type NS, class IN, ns srv-dns.h205.net
▼ Additional records
  ▶ srv-dns.h205.net: type A, class IN, addr 172.16.0.102
```

La réponse est bien là : **172.16.16.250**. On peut aussi voir que c'est la machine srv-dns.h205.net à l'adresse **172.16.0.102** qui a autorité sur le domaine **h205.net** et qui a donc répondu.

Comme expliqué dans la partie théorique, seulement 2 trames sont nécessaires pour résoudre un nom en IP. En effet, le protocole DNS s'appuie sur le protocole de transport UDP.

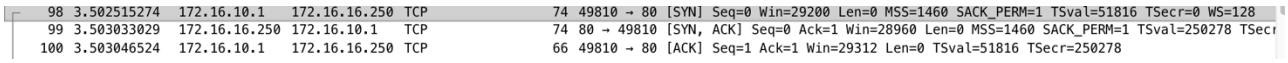
REMARQUE : (En vérité, vous observez 4 trames, car il y a aussi la requête et la réponse pour l'adresse en IPv6, type : AAAA)

Voyons, maintenant ce qu'il en est avec un protocole utilisant TCP comme transport.

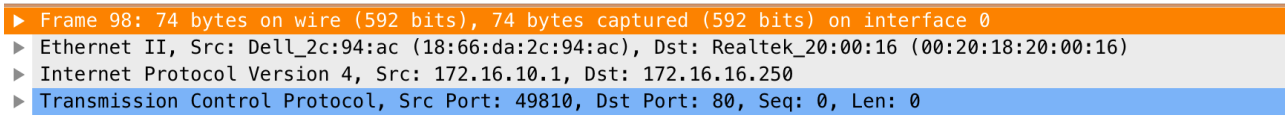
```
 98 3.502515274  172.16.10.1   172.16.16.250 TCP      74 49810 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=51816 TSecr=0 WS=128
 99 3.503033029  172.16.16.250 172.16.10.1   TCP      74 80 → 49810 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=250278 TSec
100 3.503046524  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=51816 TSecr=250278
101 3.503127428  172.16.10.1   172.16.16.250 HTTP    409 GET / HTTP/1.1
102 3.503554240  172.16.16.250 172.16.10.1   TCP      66 80 → 49810 [ACK] Seq=1 Ack=344 Win=30080 Len=0 TSval=250278 TSecr=51816
103 3.504478847  172.16.16.250 172.16.10.1   HTTP    609 HTTP/1.1 200 OK  (text/html)
104 3.504488662  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [ACK] Seq=344 Ack=544 Win=30336 Len=0 TSval=51817 TSecr=250278
109 3.588850646  172.16.10.1   172.16.16.250 HTTP    373 GET /tshark.jpg HTTP/1.1
110 3.589662096  172.16.16.250 172.16.10.1   TCP     1514 [TCP segment of a reassembled PDU]
111 3.589673959  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [ACK] Seq=651 Ack=1992 Win=33280 Len=0 TSval=51838 TSecr=250299
112 3.589941240  172.16.16.250 172.16.10.1   TCP     2962 [TCP segment of a reassembled PDU]
113 3.589945797  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [ACK] Seq=651 Ack=4888 Win=39040 Len=0 TSval=51838 TSecr=250299
114 3.590222732  172.16.16.250 172.16.10.1   HTTP    1713 HTTP/1.1 200 OK  (JPEG JFIF image)
115 3.590226921  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [ACK] Seq=651 Ack=6535 Win=42368 Len=0 TSval=51838 TSecr=250299
116 3.591332468  172.16.10.1   172.16.16.250 HTTP    341 GET /favicon.ico HTTP/1.1
117 3.591894045  172.16.16.250 172.16.10.1   HTTP    558 HTTP/1.1 404 Not Found  (text/html)
120 3.631805073  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [ACK] Seq=926 Ack=7027 Win=45184 Len=0 TSval=51849 TSecr=250300
228 8.500586912  172.16.16.250 172.16.10.1   TCP      66 80 → 49810 [FIN, ACK] Seq=7027 Ack=926 Win=32256 Len=0 TSval=251527 TSecr=51849
229 8.500800770  172.16.10.1   172.16.16.250 TCP      66 49810 → 80 [FIN, ACK] Seq=926 Ack=7028 Win=45184 Len=0 TSval=53066 TSecr=251527
230 8.501243591  172.16.16.250 172.16.10.1   TCP      66 80 → 49810 [ACK] Seq=7028 Ack=927 Win=32256 Len=0 TSval=251527 TSecr=53066
```

Après avoir obtenu l'IP du serveur shark.h205.net grâce à la requête DNS précédente, notre navigateur a donc pu établir sa connexion avec le serveur web et récupérer la page index.html, ainsi que l'image tshark.jpg qu'elle contient. Il a donc pour cela ouvert une session en HTTP qui lui-même s'appuie sur le protocole de transport TCP.

C'est donc bien une session en TCP que nous voyons ici, avec les caractéristiques suivantes :
Au début de la session : 3 trames que l'on peut identifier grâce à leurs indicateurs (partie de droite) : [SYN], [SYN, ACK] et [ACK] . C'est la séquence synchronisation.



Elles ne contiennent aucune donnée applicative, elles ne servent qu'à établir la connexion entre les deux machines. Il suffit de regarder le détail pour s'en rendre compte :

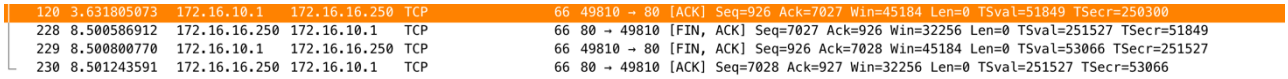


Pas de trace d'http dans les en-têtes. Juste les informations concernant, les adresses MAC, IP et les n° de ports :

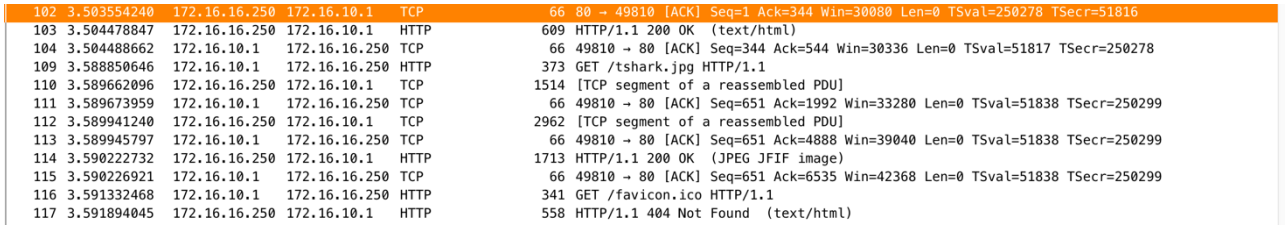
TRANSPORT	Transmission Control Protocol (TCP)	Port Destination : 80 (Serveur HTTP) Port Source : 49810 (Poste Client)
INTERNET	Internet Protocol Version 4 (IPv4)	IP source : 172.16.10.1 (Poste Client) IP Destination : 172.16.16.250 (Serveur HTTP)
ACCES MATERIEL	Ethernet II	MAC source : 18:66:da:2c:94:ac (Poste) MAC destination : 00:20:18:20:00:16 (Serveur)

Remarquez que le port du client : 49810 n'est plus le même que pour la session DNS, pourtant c'est toujours le même poste client. Le port Serveur est bien celui utilisé pour HTTP: 80

En fin de session, nous retrouvons 4 trames qui mettent fin à la session de communication entre les deux machines . La séquence est repérable grâce aux indicateurs : [ACK],[FIN,ACK],[FIN,ACK],[ACK]



Entre les deux, nous avons le dialogue HTTP constitué de requête (GET) du client et de réponse du serveur (HTTP/1.1) entrecoupé de trames TCP qui accusent réception des données ([ACK])



On remarquera que pendant tout cet échange, les n° de ports ne changent pas . 80 pour le serveur, 49810 pour le client.

Si l'on regarde le détail des trames HTTP contenant un GET (Sens Poste->Serveur) :

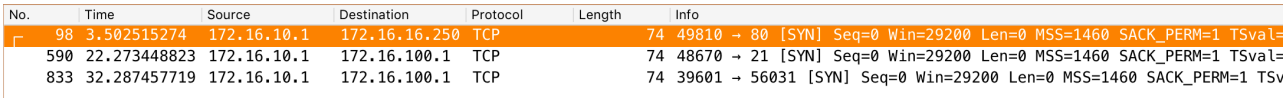
```
▼ Hypertext Transfer Protocol
  ▼ GET / HTTP/1.1\r\n
    ▶ [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.1
    Host: shark.h205.net\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
```

On peut y voir les en-têtes du protocole http (que nous étudierons en S2)
Elles indiquent notamment au serveur , la version de notre navigateur, le système d'exploitation de notre machine, les encodages acceptés en réponse. Autant d'informations qui serviront au serveur pour répondre au client.

```
▼ Hypertext Transfer Protocol
  ▼ HTTP/1.1 200 OK\r\n
    ▶ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
    Date: Fri, 06 Dec 2019 10:07:38 GMT\r\n
    Server: Apache/2.4.25 (Debian)\r\n
    Last-Modified: Fri, 06 Dec 2019 10:05:14 GMT\r\n
    ETag: "105-599062ea05be0-gzip"\r\n
    Accept-Ranges: bytes\r\n
    Vary: Accept-Encoding\r\n
    Content-Encoding: gzip\r\n
  ▶ Content-Length: 206\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/3]
    [Time since request: 0.001351419 seconds]
    [Request in frame: 101]
    [Next request in frame: 109]
    [Next response in frame: 114]
    Content-encoded entity body (gzip): 206 bytes -> 261 bytes
    File Data: 261 bytes
▼ Line-based text data: text/html
    <!DOCTYPE html>\n
    <html>\n
    <head>\n
    <title>PAGE POUR WIRESHARK</title>\n
    <meta charset="utf-8" />\n
```

La réponse du serveur contient un code réponse (**200**: OK, **404** page non trouvée, etc.) , ainsi que d'autres informations comme l'OS utilisé (**debian**) et la version du serveur web (**Apache 2.4**). Bien entendu, elle contient aussi la page demandée en **html** qui sera interprétée et affichée par le navigateur.

ASTUCE : Donc si l'on veut savoir combien de sessions de communication TCP ont été établies entre deux machines, il suffit juste de repérer (ou de filtrer) les trames contenant l'indicateur [SYN]. Toutes les informations de ces trames suffisent pour connaître les adresses des machines (MAC et IP), ainsi que les numéros de port et applicatifs utilisés.



No.	Time	Source	Destination	Protocol	Length	Info
98	3.502515274	172.16.10.1	172.16.16.250	TCP	74	49810 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=
590	22.273448823	172.16.10.1	172.16.100.1	TCP	74	48670 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=
833	32.287457719	172.16.10.1	172.16.100.1	TCP	74	39601 → 56031 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSv

En filtrant correctement les trames, on voit ici que seulement 3 sessions TCP ont été établies à partir du poste 172.16.10.1, vers deux machines différentes : Un serveur HTTP (Port 80) à l'adresse 172.16.16.250, Un serveur FTP (Port 21) à l'adresse 172.16.100.1

Suite à cette capture et en analysant les données, on peut donc compléter assez facilement les tableaux suivants :

Protocole de Transport	Port Client	Port Serveur	Protocole Applicatif
UDP	57127	53	dns
TCP	49810	80	http
TCP	48670	21	ftp
TCP	39601	56031	ftp-data

REMARQUE : le port ftp-data (côté serveur) peut varier lui aussi . Cela dépend du mode de transfert utilisé. Ici FTP fonctionne en mode **passif**, c'est donc le serveur qui détermine les ports utilisés pour le transfert (C'est le mode conseillé par défaut). Il existe un mode **actif**, dans lequel le port ftp-data est déterminé par le client, et dans ce cas on utilise le port 20.

On peut aussi compléter le tableau suivant :

Adresse IP	Adresse MAC	Rôle	Protocole Applicatif
172.16.10.1	18:66:da:2c:94:ac	Client	DNS, HTTP, FTP
172.16.0.102	fe:f1:39:45:07:01	Serveur	DNS
172.16.16.250	00:20:18:20:00:16	Serveur	HTTP
172.16.100.1	54:75:d0:d5:7c:6b	Serveur	FTP

PARTIE 3 : EXERCICE FINAL : A la pêche aux infos.

1) Lancez une capture avec **wireshark**.
2) puis ouvrez votre navigateur à l'adresse : **www.domVMID.net .**
3) Ouvrez ensuite **Filezilla** et connectez-vous avec l'utilisateur **usertest** (mot de passe : **123**)
4) **Arrêtez** la capture et **observez** les n° de ports serveurs et clients en développant les informations des couches transport (TCP ou UDP).

IMPORTANT : Avant de continuer, sauvegardez votre capture dans un fichier nommé captureVMID.txt et uploadez ce fichier sur votre serveur avec le compte usertest

Dans le dossier **/var/www/html** , créez une page **tp09.html** contenant 2 tableaux semblables à ceux de la partie 2.
(les couleurs ne sont pas obligatoires, seulement la forme)

TABLEAU DES PROTOCOLES :

- Protocole de transport utilisé (TCP ou UDP)
- Ports client : Les différents n° de ports utilisés par votre poste pendant la capture
- Ports Serveur : Les différents n° de ports utilisés par votre serveur
- Protocoles utilisés : Le nom des protocoles applicatifs mis en œuvre.

TABLEAU DES MACHINES :

- Adresse IP
- Adresse MAC
- Rôle (Client ou Serveur)
- Protocoles Applicatifs

IMPORTANT : Vous devez répertorier **TOUTES** les sessions mises en œuvre pendant la capture (Page web et Connexion FTP via Filezilla), ainsi que TOUTES les machines, mais seulement celles appartenant à notre réseau (172.16.0.0)

ANNEXE : WIRESHARK

Quelques Filtres

ip.addr == adresse IP	Filtre une IP (source ou destination)
tcp	Transport TCP uniquement
udp	Transport UDP uniquement
tcp.port == n° de port	Filtre sur un port en TCP
udp.port == n° de port	Filtre sur un port en UDP

Quelques Opérateurs :

||	OU
&&	ET
==	Équivalence stricte

!=	Différence stricte