

Techniques et outils de piratage

Rappel : TCP/IP

Comprendre les attaques pour mieux se défendre

Mohamed Mejri
Mohamed.Mejri@ift.ulaval.ca

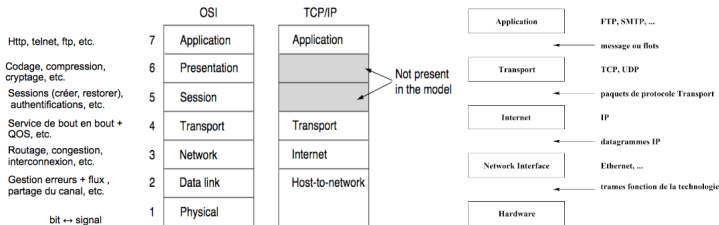
September 6, 2016

Plan

- 1 Introduction
- 2 Couche Internet
- 3 Couche transport
- 4 Wireshark

Modèle en couches

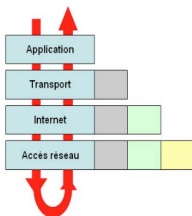
- Pour faire parler des machines, plusieurs fonctionnalités sont nécessaires. Elles se structurent en couches
- Chaque couche offre certains services à la couche supérieure et utilise les services de la couche inférieure



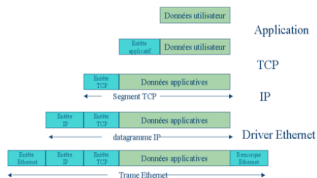
source : livre réseaux (Andrew Tanenbaum)

Modèle en couches

- ➔ Encapsulation : à chaque couche un en-tête est ajouté
- ➔ Différents noms aux unités de données manipulées par les couches (message, segment, datagramme, trame, bit)
- ➔ Différents équipements pour différents niveaux (bit-répéteur, trame-pont, datagramme-routeur, message-passerelle, proxy)



Source : <http://www.abpservices.fr/communication/encapsulation.jpg>



Source : <http://liris.cnrs.fr/alain.mille/enseignements>

Couche Internet

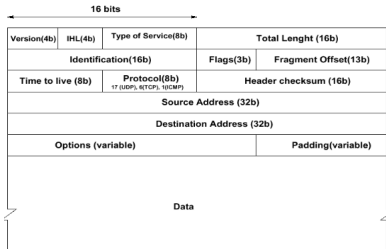
- ➔ Contient plusieurs protocoles : IP, ARP, RAP, ICMP, IGMP
- ➔ IP est capable de fonctionner au-dessus de tous :
 - une connexion physique/liaison : SLIP, PPP, etc.
 - un réseau local : Ethernet, Token-Ring, etc.
 - un autre réseau : ATM, X25, etc.
 - raisons : il n'a besoin que d'un service d'émission/ réception sans aucune garantie.

Pile de protocoles TCP/IP

7 - Application	NFS	rcp	name_server	Motif, Openlook	ping
6 - Présentation	XDR	rlogin	SMTP(mail)	X11	
5- Session	RPC	daytime	telnet ftp, tftp, bootp...		
6 - Transport	UDP		TCP		ICMP
3 - Réseau	ARP	RARP	IP		
2 - Liaison	Ethernet, Token-ring, 802.X, ...		RFC 877, SLIP, PPP, AAL5		
1 - Physique			Réseau public: encaps. X25, RNIS		

source : <http://www.dil.univ-mrs.fr/~jfp/master/m03/cours/os>

Encapsulation



- ➡ **Version** : indique la version IP (IPv4, IPv6) du datagramme
- ➡ **Internet Header Length (IHL)** : longueur de l'entête en nombre de mots de 32 bits (au moins 5 mots)

Encapsulation

```

⊞ Frame 9 (62 bytes on wire, 62 bytes captured)
⊞ Ethernet II, Src: 00:a0:d1:20:e5:b7, Dst: 00:00:0c:07:ac:0a
    Destination: 00:00:0c:07:ac:0a (132.203.120.1)
    Source: 00:a0:d1:20:e5:b7 (132.203.120.179)
    Type: IP (0x0800)
⊞ Internet Protocol, Src Addr: 132.203.120.179 (132.203.120.179), Dst Addr: 132.203.26.3 (132.203.26.3)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 48
    Identification: 0xadea (44522)
    Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
    Header checksum: 0xb090 (correct)
    Source: 132.203.120.179 (132.203.120.179)
    Destination: 132.203.26.3 (132.203.26.3)
⊞ Transmission Control Protocol, Src Port: 3221 (3221), Dst Port: http (80), Seq: 0, Ack: 0, Len: 0

```

```

0000  00 00 0c 07 ac 0a 00 a0 d1 20 e5 b7 08 00 45 00  .....E.
0010  00 30 ad ea 40 00 80 06 b0 90 84 cb 78 b3 84 cb  ..0.@...x...
0020  1a 03 0c 95 00 50 90 46 80 f5 00 00 00 00 70 02  ...P.F.....p.
0030  40 00 89 79 00 00 02 04 04 ec 01 01 04 02  @..y....

```

Encapsulation

➡ **Type Of Service (TOS)** : types de services souhaités

0	1	2	3	4	5	6	7
priorité	D	T	R	C	inutilisé		

- Priorité : varie de 0 à 7
 - ★ 0 : priorité normale (valeur par défaut)
 - ★ 7 : priorité maximale (pour la supervision du réseau)
- D=1 : minimiser le délai d'acheminement
- T=1 : maximiser le débit de transmission
- R=1 : assurer une plus grande fiabilité
- C=1 : minimiser le coût de transmission

➡ **Total length** :

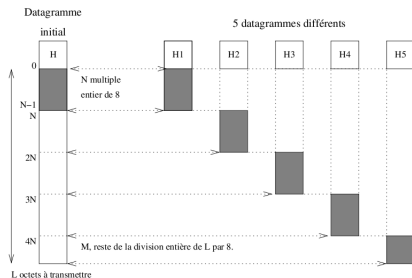
- longueur (en octets) du datagramme IP incluant l'entête
- l'espace réservé pour ce champ est 2 octets: la longueur maximale est 65535

Encapsulation

➔ **Flags:** 

- DF (Dont Fragment)=1 : ne pas fragmenter ce datagramme
- MF (More Fragment)=1 : le datagramme contient d'autres fragments

➔ **Fragment offset** : distance en unités de 64 bits depuis le premier bit du datagramme



	H1	H2	H3	H4	H5
IDENTIFICATION	I	I	I	I	I
FLAG	MF	MF	MF	MF	0
OFFSET	0	$(N)/8$	$(2 \times N)/8$	$(3 \times N)/8$	$(4 \times N)/8$
TOTAL LENGTH	$H + N$	$H + N$	$H + N$	$H + N$	$H + M$
HEADER CHECKSUM	C_1	C_2	C_3	C_4	C_5

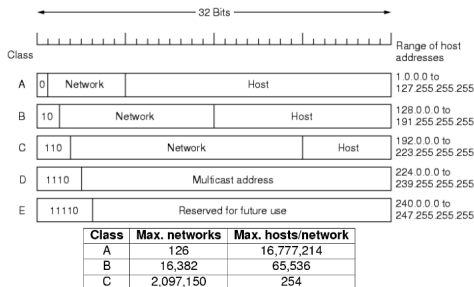
Remarque : **MTU** (Maximum Transport Unit) = **1500** (Ethernet v2), **7981** (WLAN 802.11)

Encapsulation

- ➔ **Checksum** : code détecteur d'erreurs (ne s'applique qu'à l'entête)
- ➔ **Time to live** : contient la durée de vie restante (nombre de routeurs à traverser)
 - initialisé à N (souvent 32 ou 64) par la station émettrice
 - décrémenté de 1 par chaque routeur qui le reçoit
 - quand il devient nul, le datagramme sera détruit et le routeur envoie un message ICMP à l'expéditeur
- ➔ **Protocol** : identification du protocole client (17 : UDP, 6 : TCP, 1 : ICMP). Permet de diriger (démultiplexage) les données vers les protocoles adéquats
- ➔ **Options** : sécurité, enregistrement de la route, horaire, routage strict, etc.
- ➔ **Padding** : permet à l'entête de toujours occuper un nombre entier de mots de 32 bits

Adressage

- ➔ Adresse IP : 4 octets (32 bits)
 - 4,2 milliards d'adresses possibles.
 - en "décimale pointée" : A.B.C.D (e.g. 132.203.120.10)
 - deux parties :
 - identifiant du réseau (network id) : assigné par une autorité (ICANN)
 - identifiant de la machine (host id) : assigné par l'administrateur du réseau



Adressage

➡ Adresses privées :

Préfixe	Plage IP	Nombre d'adresses
10.0.0.0/8	10.0.0.0 – 10.255.255.255	$2^{24} = 16\,777\,216$
172.16.0.0/12	172.16.0.0 – 172.31.255.255	$2^{20} = 1\,048\,576$
192.168.0.0/16	192.168.0.0 – 192.168.255.255	$2^{16} = 65\,536$

- Réservées par l'ICANN
- Ne sont pas uniques (ne sont pas "routables")
- Affectées à des machines qui n'ont pas besoin d'être accessibles de l'extérieur, des machines qui partagent la même adresse physique (via un NAT), etc.
- Un concept similaire existe en IPv6 (fc00::/7)

Adressage

➡ Sous réseaux :

- découpage logique d'un réseau en sous-réseaux
- décidé par l'administrateur du réseau
- prend une partie de l'adresse machine et il la considère comme une adresse sous-réseau
- permet une meilleure structuration du réseau + un routage interne plus rapide

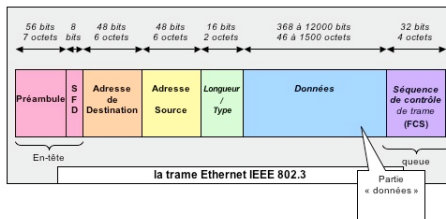


- découpage invisible de l'extérieur : routage interne
- tous les équipements internes utilisent la notion de "masque"
 - ★ Bits de l'@ machine sont à zéro et le reste à 1: (e.g 255.255.255.0 ou /24)
 - ★ $\text{@IP} \& \text{masque} = \text{@réseau} + \text{@sous-réseau}$
 - ★ $\text{@IP} \& \overline{\text{masque}} = \text{@machine}$

ARP

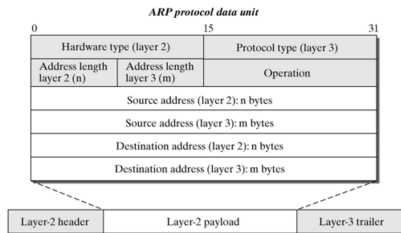
➡ ARP : Address Resolution Protocol

- la couche liaison ne comprend pas les @IP (fonctionne avec des @MAC)



source : <http://www.didascalie.net/>

ARP : Trouver l'@MAC à partir de l'@IP

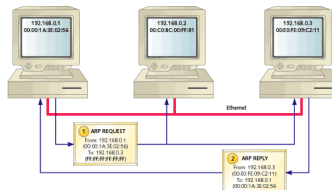


- Hardware type : type du matériel (Ethernet 10Mb : 1)
- Protocol type : protocole de la couche supérieure (IPv4: 0x0800)
- Hardware length : longueur en octets de l'@ MAC (Ethernet : 6)
- Protocol length : longueur en octets de l'adresse logique (IPv4 : 4)
- Operation : nature de l'opération (1: ARP request, 2 : ARP reply, 3 : RARP request, 4 : RARP reply)

Avec IPv6, le protocole NDP (Neighbor Discovery Protocol) remplace de ARP.

ARP

➡ ARP : principe



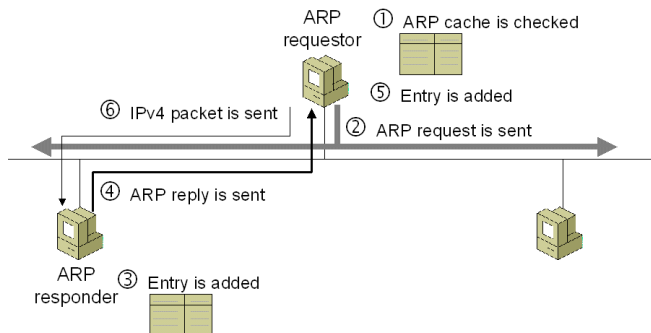
source : <http://www.netrino.com/images/articles/>

➡ Commande arp :

- `arp -a` : affiche le contenu du cache ARP.
- `arp -a @ip` : affichage le cache associé à une seule @ip dans le cas où il y en a plusieurs.
- `arp -s @ip @MAC` : ajout manuel une entrée statique permanente dans le cache (réduire le trafic et le risque).

ARP

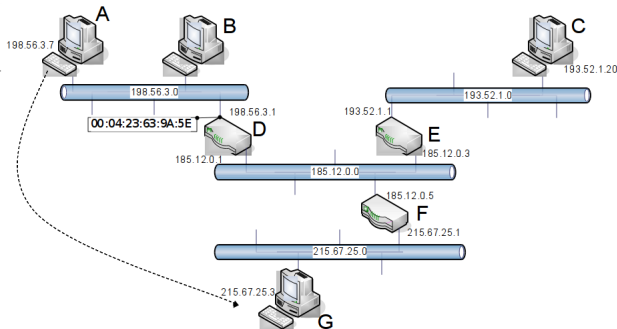
- ➡ Principe du cache : (une information y reste généralement 5 min ou moins)



source : <http://technet.microsoft.com/en-us/library>

ARP

- ➡ ARP n'est pas "routable"



- ➡ Quand la destination n'est pas dans le même réseau, on demande l'@MAC de la prochaine machine intermédiaire responsable d'acheminer le paquet
- ➡ Les @IP sources et destinations ne changent pas
- ➡ Les @MAC sources et destinations changent à chaque saut

ARP

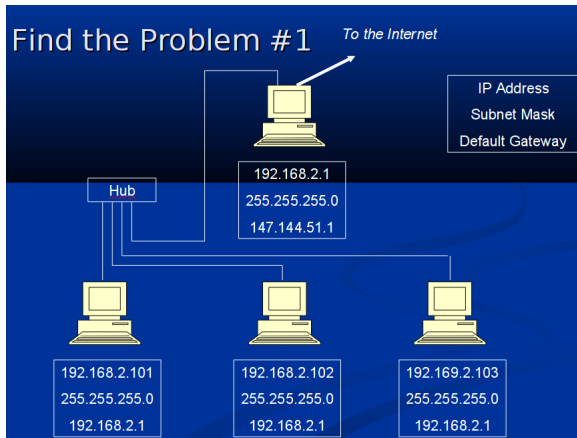
- ➡ Vérifier si la source et la destination sont dans le même réseau :

$\text{@réseau} = \text{@machine} \& \text{masque}$

- ➡ Si oui, envoyer une requête ARP demandant l'@MAC de la destination, puis envoyer le paquet et terminer
- ➡ Sinon demander l'@MAC de la passerelle et lui envoyer le paquet : chaque machine doit connaître l'@ IP de la passerelle (par défaut c'est la première @ du réseau)
- ➡ La passerelle vérifie, via le masque, si la destination fait partie d'un de ses réseaux
- ➡ Si oui elle demande l'@MAC de la destination, après quoi elle lui envoie le paquet et le processus se termine
- ➡ Sinon, elle regarde, dans sa table de routage, le prochain nœud vers la destination et demande son @MAC
- ➡ Ainsi de suite : de nœud en nœud jusqu'à la destination

Configuration

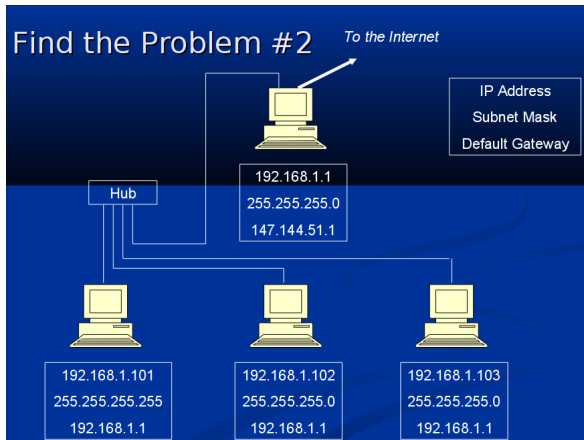
- ➡ Exercice 1 : trouver le problème de cette configuration



source : <http://samsclass.info>

Configuration

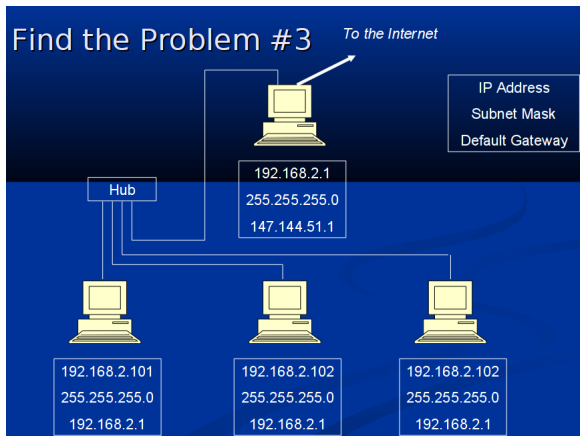
- ➡ Exercice 2 : trouver le problème de cette configuration



source : <http://samsclass.info>

Configuration

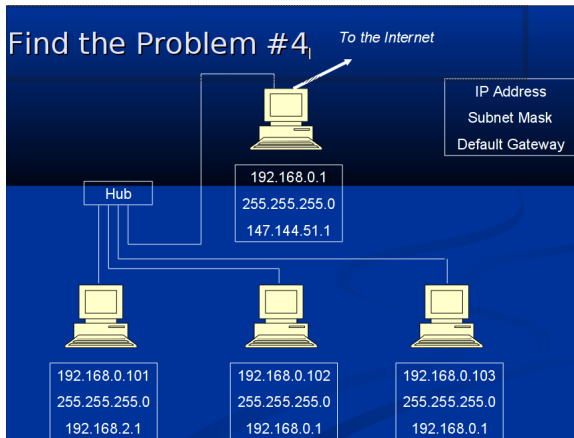
- ➡ Exercice 3 : trouver le problème de cette configuration



source : <http://samsclass.info>

Configuration

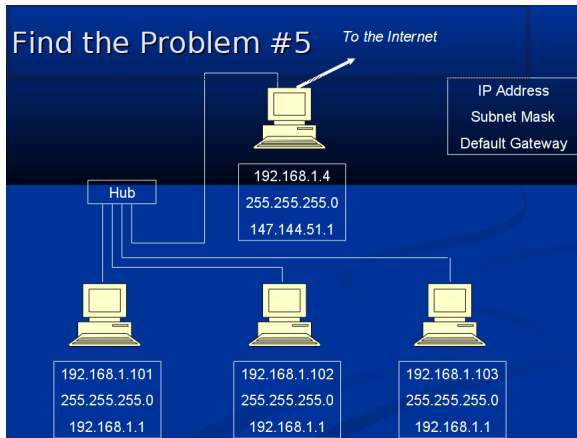
- ➡ Exercice 4 : trouver le problème de cette configuration



source : <http://samsclass.info>

Configuration

- ➡ Exercice 5 : trouver le problème de cette configuration

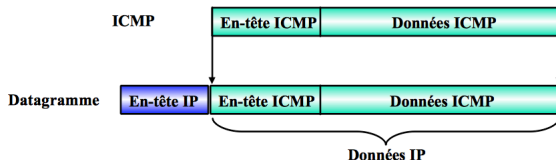


source : <http://samsclass.info>

ICMP

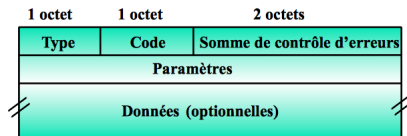
➡ ICMP (Internet Control Message Protocol)

- permet de diagnostiquer le réseau et de communiquer les problèmes de transmission
- il appartient à la couche réseau, mais il est encapsulé dans un paquet IP



ICMP

➡ Format du message



- **type** : spécifie le type
- **code** : donne des informations supplémentaires sur le type
- **somme de contrôle d'erreurs** : calculé sur tout le ICMP
- **paramètres** : utilisés souvent pour indiquer l'identificateur et le numéro de séquence du ICMP (utile lorsqu'on attend une réponse)
- **données** : utilisées pour envoyer des informations additionnelles

ICMP

➡ Exemples

- **destination inaccessible** : ce type de ICMP envoyé lorsqu'un routeur ne peut délivrer un datagramme

1 octet	1 octet	2 octets
Type=3	Code=0-12	Somme de contrôle d'erreurs
Paramètres (inutilisé: doit contenir des zéros)		
// Données = Entête IP + 64 bits de données du datagramme IP qui a généré le problème //		

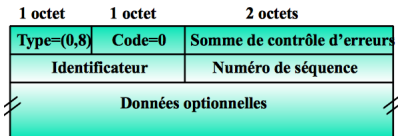
- ★ **code 0** : réseau inaccessible
- ★ **code 2** : port inaccessible
- ★ **code 4** : fragmentation nécessaire et bit DF positionné

64 bits de données \Rightarrow port source + port destination + numéro de séquence TCP

ICMP

➡ Exemples

- **message d'écho** : permet de vérifier l'état d'activité d'un ordinateur destinataire



- ★ **type 0** : réponse à un écho
- ★ **type 8** : demande d'écho
- ★ "Identificateur" et "Numéro de séquence" permettent à l'expéditeur d'associer les réponses reçues avec ses propres demandes
- ★ Normalement, un ordinateur qui reçoit une demande d'écho retourne une réponse à l'expéditeur

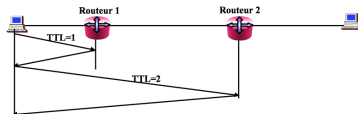
ICMP

➔ Exemples

- **temps expiré (type 11, sans code)** : envoyé lorsqu'un datagramme est détruit parce que son TTL est devenu zéro
- **ralentissement de la source (type 4, sans code)** : demande de réduction de vitesse de transmission
- **redirection (type 5, code multiple)** : demander à une machine d'envoyer ses paquets par une autre voie

➔ Quelques commandes

- **ping** : envoie un message écho, attend la réponse, mesure le temps d'aller-retour
- **tracert** : se base sur des messages ICMP (type 11) pour trouver un chemin entre une source et une destination



ICMP

➔ **Exceptions** : pas de ICMP retourné :

- en réponse à un autre message ICMP : sinon on pourrait avoir une tempête ICMP (e.g. on crée un paquet UDP avec des ports sources et destinations non disponibles)
- à une adresse de "broadcast"
- lors de la fragmentation sauf pour le premier paquet (inutile de retourner un ICMP pour chacun des fragments)
- à une adresse source "localhost"

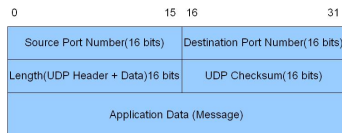
ICMP

➡ Quelques utilisations possible par l'intrus :

- découvrir les machines actives et les ports ouverts
- attaque de type déni de service via des ICMP de type "ralentissement de la source"
- attaque de type déni de service via des ICMP de type redirection (rediriger les paquets vers un "trou noir")
- ping de la mort (1997-1998 : Unix, Lunix, Mac, Windows, etc.) : envoyer des ICMP de longueur totale dépassant le 65 535
- etc.

UDP

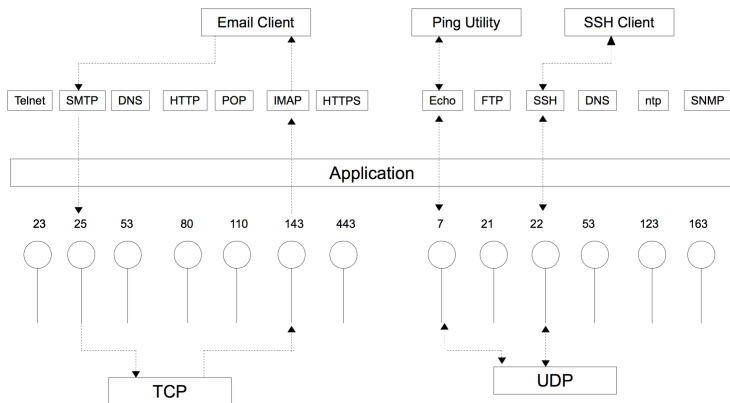
- ➡ Mode sans connexion
- ➡ Aucune garantie de transmission
- ➡ Aucune gestion de l'ordre des paquets
- ➡ Aucun contrôle du trafic
- ➡ Support unicast, broadcast et multicast
- ➡ Protocole très léger (taille de l'entête est fixe = 8 octets; taille de données variable < 65535 octets)
- ➡ Encapsulé dans l'entête IP sous le numéro 17 (0x11)



Les applications qui utilisent UDP doivent résoudre, par elles-mêmes, les problèmes de communication (erreur, flux, séquençement, etc.)

UDP

- Port source, peut-être mis à zéro si on n'attend pas de réponse
- Port destination, s'il n'est pas alloué, on reçoit un ICMP (type 3, code 3)
- L'adresse d'une application sur une machine= (@IP, numéro du protocole de transport, numéro du port)



UDP

- ➔ **Port source/ Port destination** : identifient les applications \implies multiplexage
- ➔ "Well Known Ports" (sont dit **réservées**) [0,1023] (durée de vie infinie!) : 7 Echo, 53 DNS etc. (<http://www.iana.org/assignments/port-numbers>)
- ➔ "Registered Ports" (sont dit **enregistrées**) [1024, 49151] (L'ICANN les associe à des applications de vendeurs suite à leurs demandes, mais peuvent être utilisées)
- ➔ "Dynamic and/or Private Ports" (sont dit **dynamiques ou à usage privé**) [49152,65535] : sont attribués dynamiquement par le système d'exploitation (durée de la connexion) ou choisis par l'application (durée de vie de l'application).

```

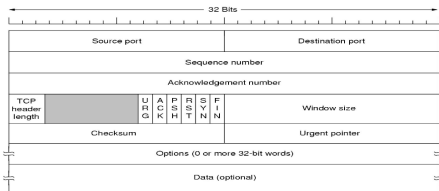
Frame 13 (89 bytes on wire, 89 bytes captured)
Ethernet II, Src: 00:00:01:00:00:00, Dst: fe:ff:20:00:01:00
Internet Protocol, Src Addr: 145.254.160.237 (145.254.160.237), Dst Addr: 145.253.2.203 (145.253.2.203)
User Datagram Protocol, Src Port: 3009 (3009), Dst Port: domain (53)
  Source port: 3009 (3009)
  Destination port: domain (53)
  Length: 55
  Checksum: 0x10af (correct)
Domain Name System (query)

0000  fe ff 20 00 01 00 00 00 01 00 00 00 08 00 45 00  .. ....E.
0010  00 4b 0f 49 00 00 80 11 63 a5 91 fe a0 ed 91 fd  .K.I...C.....
0020  02 cb 00 c1 00 35 00 37 10 a1 00 23 01 00 00 01  ..5.7..#.
0030  00 00 00 00 00 00 07 70 61 67 65 61 64 32 11 67  .....p agead2.g
0040  6f 6f 67 6c 65 73 79 6e 64 69 63 61 74 69 6f 6e  ooglesyn dication
0050  03 63 6f 6d 00 00 01 00 01  ..com....

```

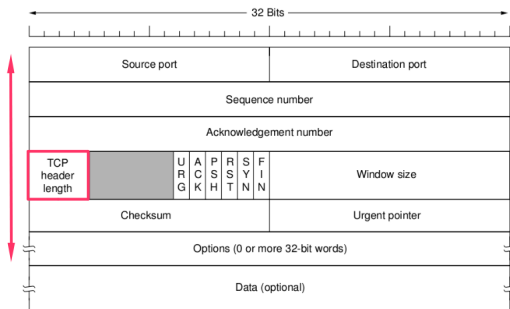
TCP

- ➔ Mode avec connexion : **connexions bidirectionnelles**
- ➔ Connexion identifiée par "@IP + num port" source + destination p. ex. 130.190.5.1-21 et 132.203.120.5-1094
- ➔ Connexion point à point (**unicast**) : pas de "multicast" et de "broadcast"
- ➔ Une fois la connexion établie, les deux applications la voient comme un **circuit virtuel dédié** (les données sont ordonnées).
- ➔ Protocole **fiable**
- ➔ Correspond à environ 90% du trafic Internet
- ➔ Encapsulé dans l'entête IP sous le numéro 6 (0x06)



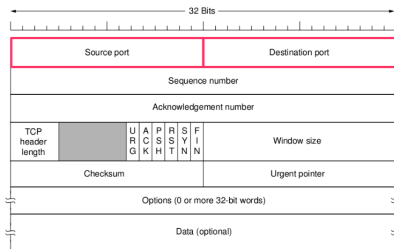
TCP entête

- ➔ **TCP Header Length (4 bits)** : taille de l'en-tête en mots de 32 bits (min=20 octets si pas d'options, max=60 octets)
- ➔ Donne la position de début du champ des données par rapport au début du segment



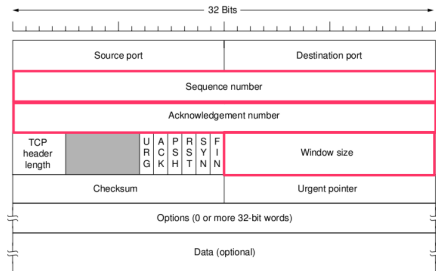
TCP Multiplexage

- ➔ **Port source/ Port destination** : identifient les applications \Rightarrow multiplexage
- ➔ "Well Known Ports" 0..1023 (durée de vie infinie!) : 25 SMTP, 35 Printer, 53 DNS, 80 HTTP, etc. (<http://www.iana.org/assignments/port-numbers>)
- ➔ "Registered Ports" 1024 ... 49151 (L'ICANN les associe à des applications de vendeurs suite à leurs demandes)
- ➔ "Dynamic and/or Private Ports" 49152...65535 : sont attribués dynamiquement par le système d'exploitation (durée de la connexion) ou choisis par l'application (durée de vie de l'application).



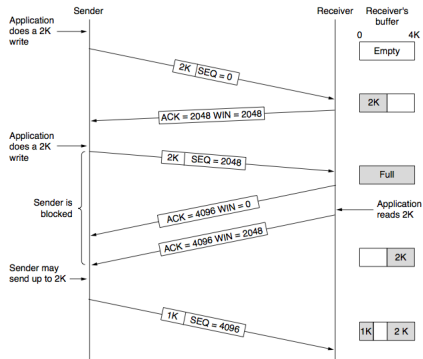
TCP Fenêtre coulissante

- ➡ **Sequence number (32 bits)** : position (mod 2^{32}) du 1er octet des données du segment dans le flux total. La valeur initiale est quelconque.
- ➡ **Acknowledgment number (32 bits)** : le numéro du prochain octet attendu. Acquitte tous les octets précédents. Un segment non acquitté est retransmis après un *timeout*.
- ➡ **Window size (16 bits)** : nombre d'octets pouvant être envoyés par anticipation. $W=0 \implies$ contrôle de flux



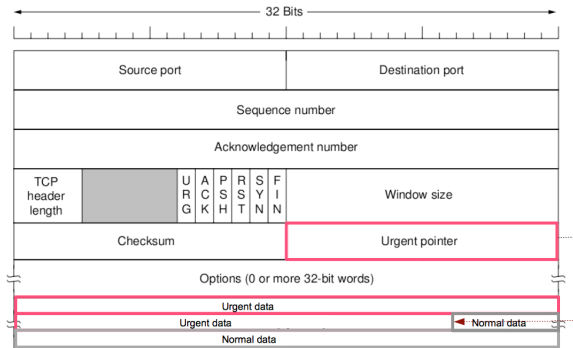
TCP Fenêtre coulissante

➡ Contrôle de flux :



TCP

- ➔ **Urgent pointer (16 bits)** : un "offset" positif qui pointe vers la fin des données urgentes
- ➔ Données envoyées hors du contrôle de flux
- ➔ Permet de transmettre de données sans retard qui doivent être traitées d'une manière urgente (e.g. commande d'interruption du traitement en cours "abort")

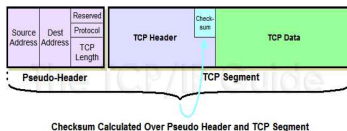


TCP : rôles de segments

- ➡ **URG (1 bit)** : le pointeur de données urgentes est valide. Le récepteur traite les données d'urgence en premier (sans attendre de finir avec les données reçues précédemment).
- ➡ **ACK (1 bit)** : le champ d'accusé de réception est valide
- ➡ **PSH (1 bit)** : ce segment requiert un "push". Transférer les données de ce segment avec les données qui n'ont pas été traitées encore directement à l'application (sans plus tamponner)
- ➡ **RST (1 bit)** : Utiliser pour signaler un problème grave (connexion est libéré immédiatement, données non traitées sont perdues) ou pour refuser une connexion.
- ➡ **SYN (1 bit)** : synchroniser les numéros de séquences pour initialiser une connexion
- ➡ **FIN (1 bit)** : l'émetteur a atteint la fin de son flot de données

TCP

- ➡ **Checksum (16 bits)** : pour vérifier la validité de l'en-tête et des données



source: <http://www.tcpipguide.com>

- ➡ **Options** : parmi les plus importantes, on trouve "Maximum Segment Size" (MSS)
 - permet de négocier la taille maximale des segments envoyés
 - grand segment \Rightarrow surcoût de traitement (dû à la fragmentation)
 - segment de taille adapté \Rightarrow minimise le délai d'acheminement
 - c'est une sorte de MTU

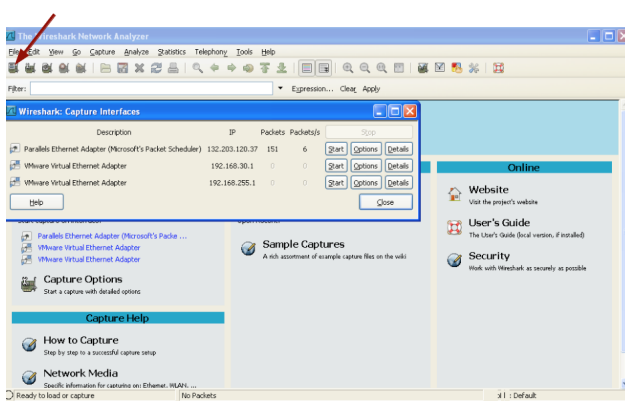
TCP

Établissement d'une connexion (syntaxe de Wireshark)

No. Info	Time	Source	Destination	Protocol	
1	0.000000	145.254.160.237	65.208.228.223	TCP	3372
		> http [SYN] Seq=0 Ack=0 Win=8760 Len=0 MSS=1460			
2	0.911310	65.208.228.223	145.254.160.237	TCP	http
		> 3372 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380			
3	0.000000	145.254.160.237	65.208.228.223	TCP	3372
		> http [ACK] Seq=1 Ack=1 Win=9660 Len=0			

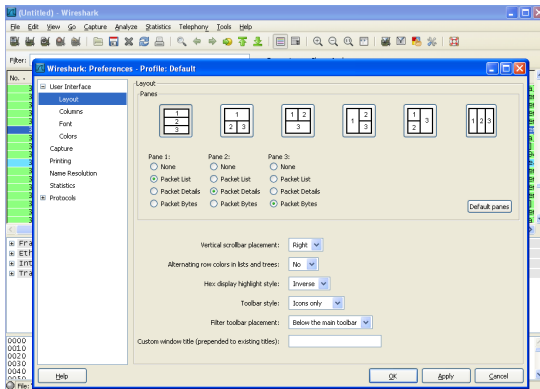
Wireshark

Choisir l'interface + démarrer la capture (menu->capture->Interfaces)



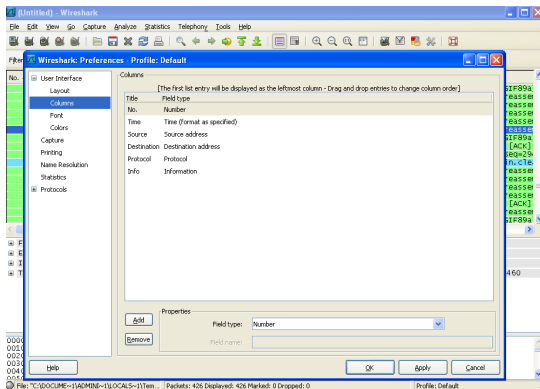
Wireshark

Configuration de l'affichage : offre une grande flexibilité (voir menu->edit->preferences)



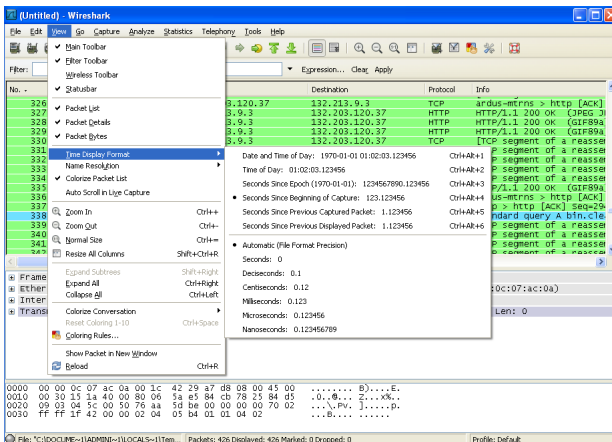
Wireshark

Configuration de l'affichage : choisir l'information à afficher



Wireshark

Temps : le temps affiché peut être modifié selon différents formats



Wireshark

Recherche dans le trafic capté : (menu->edit->find packet)

- ➔ Trouver un paquet à partir d'un filtre, d'un texte, d'une valeur en Hex.
- ➔ Chercher un mot de passe dans un trafic
- ➔ Chercher s'il y a eu échange de fichiers de certains types (jpeg, etc)
- ➔ Etc.

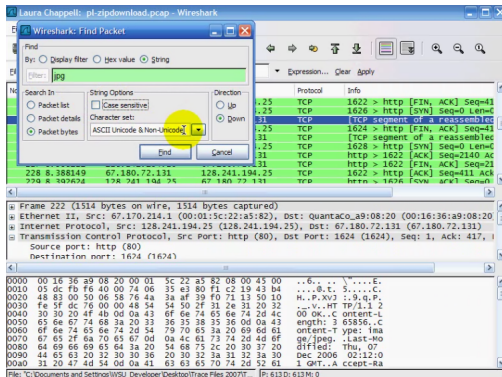
Filtrer le trafic :

- ➔ **Protocoles** : ip, tcp, arp, dns, http, etc.
- ➔ **Adresses** : ip.addr, ip.src, ip.dst, eth.addr, etc.
- ➔ **IP/ICMP** : ip.proto, icmp.code, etc.
- ➔ **TCP** : tcp.flags.syn, tcp.flags.ack, etc.
- ➔ **HTTP** : http.request, http.response, etc.
- ➔ **Opérateurs** : ==, >, <, &&, ||, etc.

```
tcp.flags.syn==1 && ip.addr>10.10.1.1
```


Wireshark

Recherche dans le trafic capté : (menu->edit->find packet)



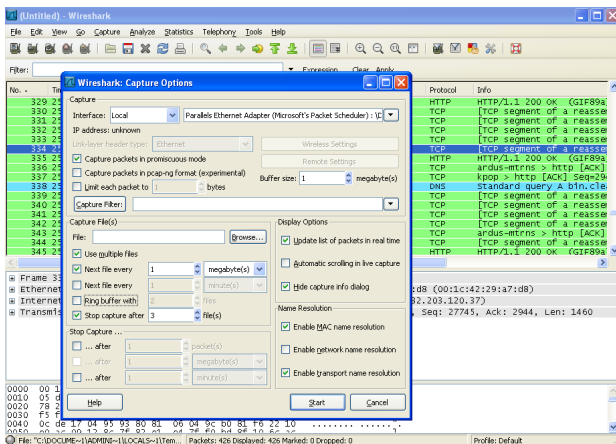
Wireshark

Enregistrement : l'enregistrement du trafic capté consomme énormément d'espace. Wireshark offre différentes options utiles d'enregistrement :

- ➔ Enregistrement limité par le nombre de paquets
- ➔ Enregistrement durant un nombre limité de minutes
- ➔ Enregistrement s'arrêtant quand un fichier atteint une taille donnée
- ➔ Etc.

Wireshark

Enregistrement : (menu-capture-option)



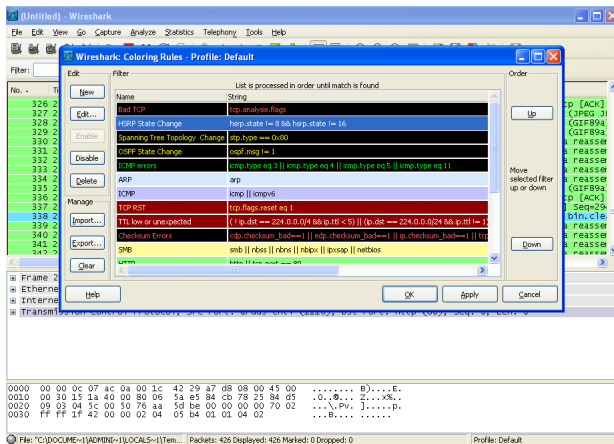
Wireshark

Coloration de paquets : (menu->view->coloring rules)

- ➔ L'ordre des règles est important
- ➔ Les premières règles cherchent généralement les paquets problématiques ou douteux
- ➔ Si plusieurs règles s'appliquent sur un même paquet, c'est la première qui l'emporte

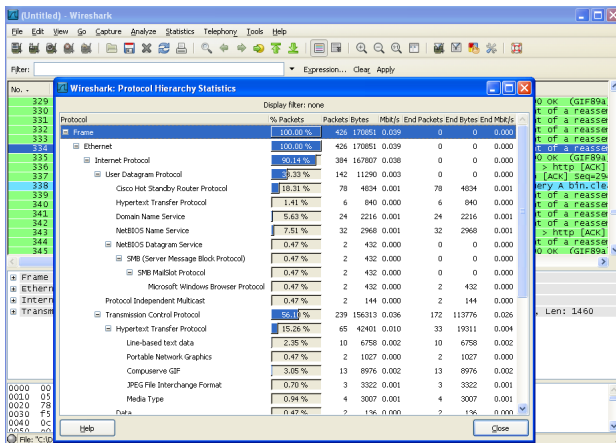
Wireshark

Coloration de paquets : (menu->view->coloring rules)



Wireshark

Statistiques : différentes statistiques importantes sont offertes (menu->Statistics)



Wireshark

Code source : pour comprendre les détails liés à l'interprétation des trames capturées, le code source est disponible et il est assez lisible

- ➡ Chaque type (IP, TCP, etc.) d'unité de donnée est traité par un bout de code appelé "dissectors" qui lui a été développé
- ➡ Les "dissectors" sont disponibles sur :

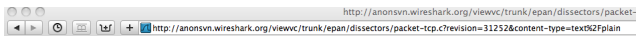
<http://anonsvn.wireshark.org/viewvc/trunk/epan/dissectors>

- ➡ Pour comprendre comment les segments tcp sont interprétés, voir :

<http://anonsvn.wireshark.org/viewvc/trunk/epan/dissectors/packet-tcp.c>

Wireshark

Code source :



```

    }
    return offset;
}

/* if we saw a PDU that extended beyond the end of the segment,
   use this function to remember where the next pdu starts
*/
struct tcp_multisegment_pdu *
pdu_store_sequencenumber_of_next_pdu(packet_info *pinfo, guint32 seq, guint32 nrt pdu, emem_tree_t *multisegment_pdus)
{
    struct tcp_multisegment_pdu *msp;

    msp=se_alloc(sizeof(struct tcp_multisegment_pdu));
    msp->nrt pdu=nrt pdu;
    msp->seq=seq;
    msp->first_frame=pinfo->fd->num;
    msp->last_frame=pinfo->fd->num;
    msp->last_frame_time=pinfo->fd->abs_ts;
    msp->flags=0;
    se_tree_insert32(multisegment_pdus, seq, (void *)msp);
    return msp;
}

/* This is called for SYN+ACK packets and the purpose is to verify that we
   * have seen window scaling in both directions.
   * If we cant find window scaling being set in both directions
   * that means it was present in the SYN but not in the SYN+ACK
   * (or the SYN was missing) and then we disable the window scaling
   * for this tcp session.
   */
static void
verify_tcp_window_scaling(struct tcp_analysis *tcpd)
{
    if (tcpd && ((tcpd->flow1.win_scale==1) || (tcpd->flow2.win_scale==1))){
        tcpd->flow1.win_scale=-1;
        tcpd->flow2.win_scale=-1;
    }
}

/* if we saw a window scaling option, store it for future reference
   */
static void

```