Logiciel de capture de trames Ethernet : WIRESHARK

1 . Introduction

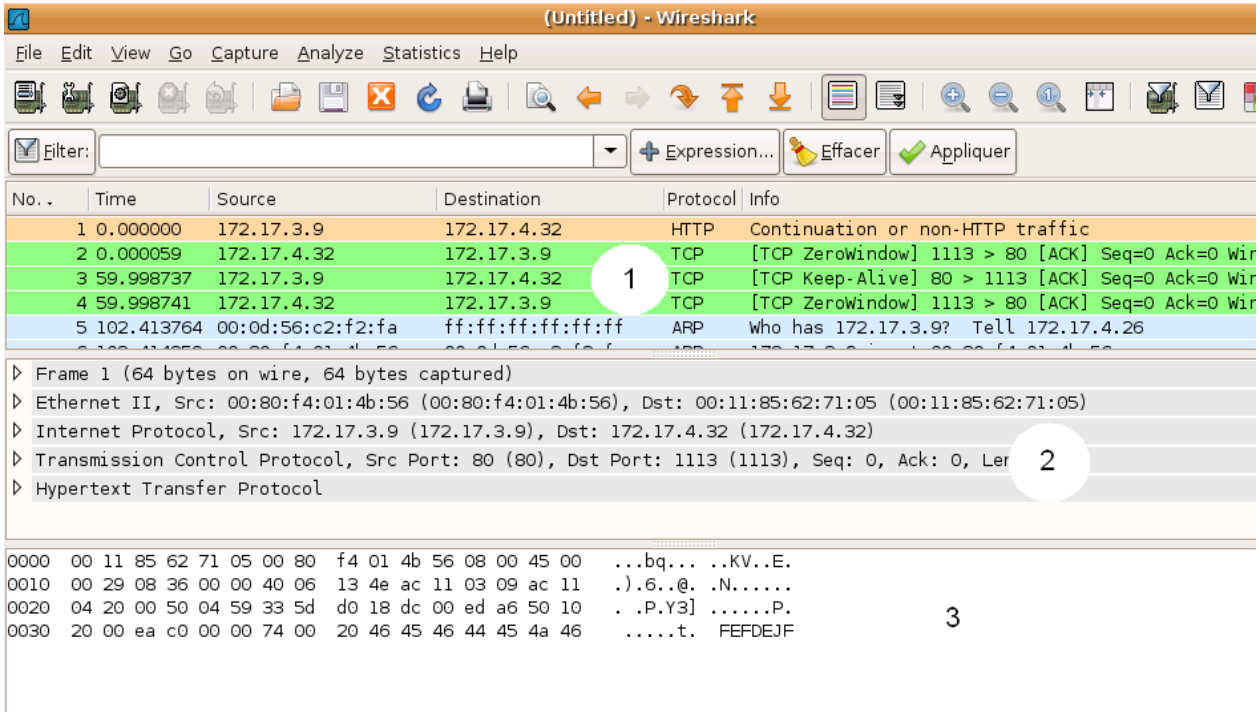
Pour pouvoir analyser finement le trafic réseau, il existe des **logiciels de capture de trames** qui sont des outils qui permettent de récupérer les paquets qui passent physiquement sur un réseau (quelque soit la destination de ces paquets).

Il existe un outil sous licence GNU qui permet de faire cela et qui permet d'interpréter la structure des paquets, cela de façon graphique. Cet outil se nomme **Wireshark**. Avant juin 2006, **Wireshark** répondait au nom d'**Ethereal**.
Wireshark utilise la librairie **Libpcap** et la syntaxe des **filtres** est similaire à celle de la commande Unix **tcpdump**.

Le guide de l'utilisateur est disponible à l'adresse suivante :
http://www.wireshark.org/docs/wsug_html_chunked/.

2 . Interface principale

La fenêtre principale de **Wireshark** comporte trois volets :



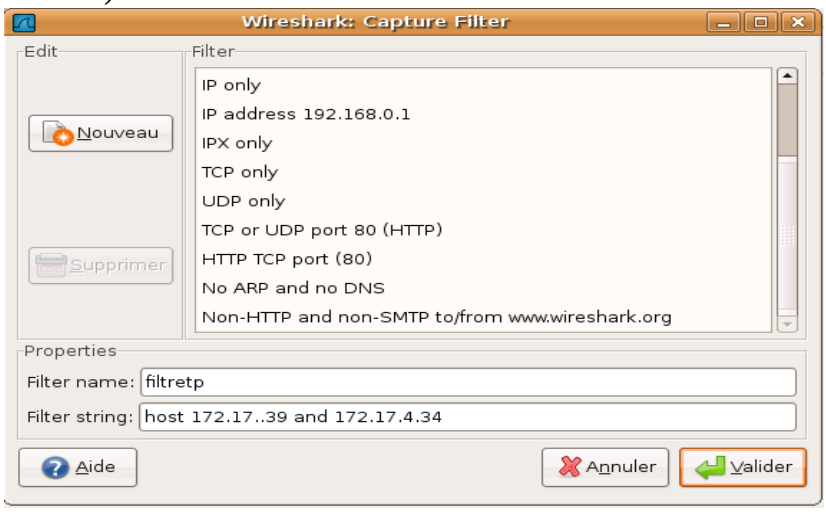
- Le volet **1** permet de recenser **l'ensemble des paquets capturées**. Sont spécifiés **l'émetteur** de la trame, le **destinataire** de la trame et le **protocole réseau** mis en œuvre;
- Le volet **2** permet de visualiser la **pile des protocoles** employés dans le **paquet** sélectionné dans le premier volet;
- Le volet **3** permet de visualiser **l'ensemble du paquet capturée** au format **hexadécimal** et la traduction **ASCII** correspondante.

3 . La capture de trames

Les 2 **étapes** permettant la **capture** de **trames** sont les suivantes :

3.1 : Définition d'un filtre de capture

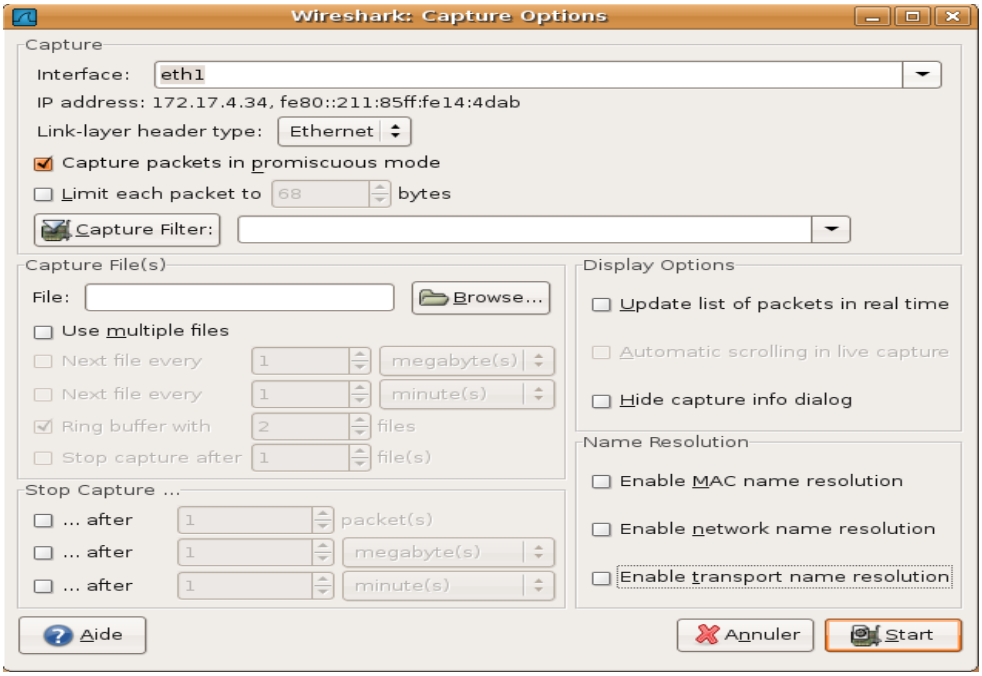
La définition d'un **filtre de capture** (*Menu Capture – Capture Filters*) permet de **cibler** les **trames** à **acquérir** en spécifiant les **protocoles voulus**, les **adresses désirées**.
La **fenêtre** permettant d'établir les **filtres de captures** (*la syntaxe des filtres est identique à celle de **tcpdump** cf. annexe **2***) est la suivante :



Le **nom du filtre** et sa **syntaxe** sont à **spécifier** en premier. Le fait de **cliquer** sur **Nouveau** permet la **sauvegarde** du **filtre** défini.
Si par exemple, on veut observer **uniquement** les **trames échangées** entre les **postes 172.16.2.28** et **172.16.2.9**, le **filtre** à appliquer est **host 172.16.2.28 and host 172.16.2.9**.

3.2 : Définition des options de capture

Une fois le filtre établi, cliquer sur **Capture Options** (*menu Capture*). La fenêtre suivante apparaît :



Plusieurs critères peuvent être spécifiés :

- **Interface** : *permet de sélectionner l'interface physique (carte réseau, …) à partir de laquelle la capture va être effectuée;*
- **Capture filter** : *permet d'établir un filtre de capture (syntaxe tcpdump) ou d'appliquer un filtre sauvegardé (voir précédemment);*
- **Enable MAC name resolution** : *permet de spécifier (si sélectionné) que les adresses Ethernet n'apparaîtront pas sous la forme nn-nn-nn-nn-nn-nn mais avec le nom de l'interface MAC (nom de carte réseau par exemple).*

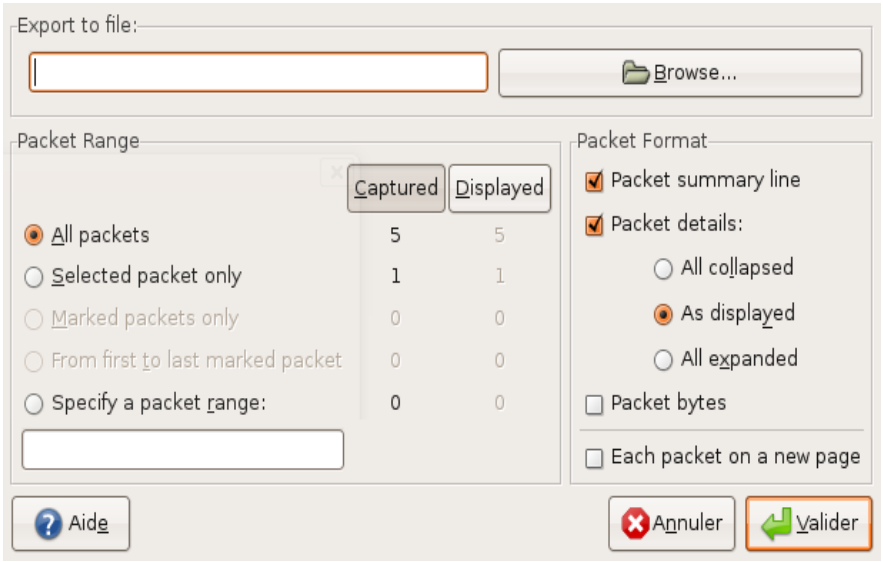
Décocher les options du groupe « **Name Resolution** » et cliquer sur **Start** pour lancer la **capture**.

4 . Analyse et sauvegarde des trames

Une fois les captures effectuées, il est possible de faire le travail d'analyse.

Wireshark offre de nombreuses possibilités d'analyse de haut-niveau. Ces fonctionnalités sont accessibles par le menu [Statistiques] (Summary, Protocol Hierarchy, ...).

Vous pouvez enregistrer les informations capturés dans le format "Wireshark/tcpdump/ - libcap" pour une analyse ultérieure avec Wireshark : [File] – [Save As...]. Les informations peuvent être exportées dans un format texte, incluant une mise en forme simple qui correspond à peu près à l'arborescence affiché dans la fenêtre "Packet Details".



5 . Les filtres

5.1 : Introduction

Dans un réseau Ethernet sous IP, les informations circulent sous forme de **datagrammes**, c'est-à-dire de paquets encapsulant les données à transmettre.

Il y a deux sortes de filtres. Les filtres à la **capture** et les filtres à l'**affichage**. Ces filtres n'ont pas la même syntaxe. La syntaxe des filtres à la capture est la même que les filtres utilisés pour la commande **tcpdump**.

Quand aux filtres à l'affichage, la syntaxe est une syntaxe propriétaire à **Wireshark**. La section présente donne des exemples pour ces deux types de filtres.

5.2 : Filtres de capture

Ne seront gardés que les paquets pour lesquels le filtre est vrai. Les filtres se décomposent en 3 parties :

- le **protocole** qui peut être **arp**, **ether**, **fddi**, **icmp**, **ip**, **ip6**, **link**, **ppp**, **radio**, r**arp**, **slip**, **tcp**, **tr**, **udp** ou **wlan** ;
- la **direction** qui peut être **src** (source) ou **dst** (destination) ;
- un **champ** qui peut être **host**, **net** ou **port** suivi d'une valeur.

Les opérateurs **and** (ou **&&**), **or** (ou ||) et **not** (ou **!**) peuvent être utilisés pour combiner des filtres.

Voici quelques exemples de filtres de capture :

Filtre	Fonction
host 172.16.0.1 and tcp	ne conserve que les paquets TCP à destination ou en provenance de la machine 172.16.0.1
udp port 53	ne conserve que les paquets UDP en provenance ou en destination du port 53
udp port 53 and dst host 172.16.0.1	ne conserve que les paquets UDP en provenance ou en destination du port 53 à destination de la machine 172.16.0.1
tcp dst port 80 and dst host 172.16.0.1 and src net 172.16.0.0 mask 255.255.255.0	ne conserve que les paquets TCP en destination de la machine 172.16.0.1 sur le port 80 et en provenance des machines du réseau 172.16.0/24

En **annexe 1**, vous trouverez tous les types de filtres de **capture** compatibles **TCPDUMP**.

5.3 : Filtres d'affichage

Les filtres d'affichage sont un peu plus fin que ceux de la capture. Seuls les paquets pour lesquels l'expression du filtre est vraie seront gardés. Les expressions sont basées sur les champs disponibles dans un paquet. Le simple ajout d'un champ veut dire que l'on garde le paquet si ce champ est disponible.

Maintenant, on peut aussi utiliser les opérateurs ==, **!**=, >, <, >= et <= pour comparer les champs avec des valeurs. Les expressions ainsi fabriquées peuvent être combinées avec les opérateurs **&&** (pour un et logique), || (pour un ou logique), ^^ (pour le ou exclusif) et **!** pour la négation.

L'usage des parenthèses est possible.

Voici quelques exemples de champs disponibles :

Champ	Type	Signification
ip.addr	adresse IPv4	adresse IP source ou destination
ip.dst	adresse IPv4	adresse IP destination
ip.flags.df	booléen	Drapeau IP, ne pas fragmenter
ip.flags.mf	booléen	Drapeau IP, fragments à venir
ip.ttl	entier non signé sur 8 bits	Time to live
http.request	booléen	requête HTTP
http.response	booléen	réponse HTTP
icmp.code	entier non signé sur 8 bits	numéro du code d'une commande ICMP
icmp.type	entier non signé sur 8 bits	numéro du type d'une commande ICMP

Voici quelques exemples de filtres d'affichage :

Filtre	Signification
ip.addr == 172.16.0.100	tous les paquets IP en provenance ou à destination de la machine 172.16.0.100
(ip.addr >= 172.16.0.100) && (ip.addr <= 172.16.0.123)	tous les paquets IP en provenance ou à destination des machines comprises entre l'adresse IP 172.16.0.100 et l'adresse IP 172.16.0.123 (comprises)

En **annexe 1**, vous trouverez tous les types de filtres d'**affichage** utilisables avec **Wireshark**, en **annexe 2**, les types de filtres de **capture** utilisables avec **Wireshark** et en **annexe 3**, le détail de la trame **Ethernet**.

Remarque : Si vous n'avez pas d'interface graphique, vous pourriez être intéressé par "**TShark**" qui est une version en ligne de commande de **Wireshark**. **TShark** supporte les mêmes fonctionnalités que **Wireshark**.

Par exemple : **#tshark -i eth1 host 192.168.1.10**

Annexe 1 : Les filtres d'affichage

Ethernet

eth.addr
eth.dst
eth.lg
eth.len
eth.src
eth.trailer
eth.type
eth.multicast

IEEE 802.1Q

vlan.cfi
vlan.etype
vlan.id
vlan.len
vlan.priority
vlan.trailer

IPv4

ip.addr
ip.checksum
ip.checksum_bad
ip.checksum_good
ip.checksum_bo
ip.dsfield
ip.dsfield.ce
ip.dsfield.dscp
ip.dsfield.ect
ip.dst
ip.dst_host
ip.flags
ip.flags.df
ip.flags.mf
ip.flags.rb
ip.frag_offset
ip.fragment
ip.fragment.error
ip.fragment.multipletails
ip.fragment.overlap
ip.fragment.overlap.conflict
ip.fragments
ip.hdr_len
ip.host
ip.id
ip.len
ip.proto
ip.reassembled_in
ip.src
ip.src_host
ip.tos
ip.tos.cost
ip.tos.delay
ip.tos.precedence
ip.tos.reliability
ip.tos.throughput
ip.ttl
ip.version

IPv6

ipv6.addr
ipv6.class
ipv6.dst
ipv6.dst_host
ipv6.dst_opt
ipv6.flow
ipv6.fragment
ipv6.fragment.error
ipv6.fragment.more
ipv6.fragment.multipletails
ipv6.fragment.offset
ipv6.fragment.overlap
ipv6.fragment.overlap.conflict
ipv6.fragment.toolongfragment
ipv6.fragments
ipv6.fragment.id
ipv6.hlim
ipv6.hop_opt
ipv6.host
ipv6.mipv6_home_address
ipv6.mipv6_length
ipv6.mipv6_type
ipv6.nxt
ipv6.opt.padi
ipv6.opt.pads
ipv6.plen
ipv6.reassembled_in
ipv6.routing_hdr
ipv6.routing_hdr.addr
ipv6.routing_hdr.left
ipv6.routing_hdr.type
ipv6.src
ipv6.src_host
ipv6.version

ARP

arp.dst.hw_mac
arp.dst.proto_ipv4
arp.hw.size
arp.hw.type
arp.opcode
arp.proto.size
arp.proto.type
arp.src.hw_mac
arp.src.proto_ipv4

TCP

tcp.ack
tcp.checksum
tcp.checksum_bad
tcp.checksum_good
tcp.continuation_to
tcp.dstport
tcp.flags
tcp.flags.ack
tcp.flags.cwr
tcp.flags.ecn
tcp.flags.fin
tcp.flags.push
tcp.flags.reset
tcp.flags.syn
tcp.flags.urg
tcp.hdr_len
tcp.len
tcp.nxtseq
tcp.options
tcp.options.cc
tcp.options.ccecho
tcp.options.ccnew
tcp.options.echo
tcp.options.echo_reply
tcp.options.md5
tcp.options.mss
tcp.options.mss_val
tcp.options.qs
tcp.options.sack
tcp.options.sack_le
tcp.options.sack_pere
tcp.options.sack_re
tcp.options.time_stamp
tcp.options.wscale
tcp.options.wscale_val
tcp.pdu.last_frame
tcp.pdu.size
tcp.pdu.time
tcp.port
tcp.reassembled_in
tcp.segment
tcp.segment.error
tcp.segment.multipletails
tcp.segment.overlap
tcp.segment.overlap.conflict
tcp.segment.toolongfragment
tcp.segments
tcp.seq
tcp.srcport
tcp.time_delta
tcp.time_relative
tcp.urgent_pointer
tcp.window_size

UDP

udp.checksum
udp.checksum_bad
udp.checksum_good
udp.dstport
udp.length
udp.port
udp.srcport

Operators

eq == and && Logical AND
ne != or || Logical OR
gt > xor ^^ Logical XOR
lt < not ! Logical NOT
ge >= [n] [...] Substring operator
le <=

Logic

Frame Relay

fr.becn
fr.chdlctype
fr.control
fr.control.f
fr.control.ftype
fr.control.n_r
fr.control.n_s
fr.control.p
fr.control.s_ftype
fr.control.u_modifier_cmd
fr.control.u_modifier_resp
fr.cr
fr.dc
fr.de
fr.dlci
fr.dlcore_control
fr.ea
fr.fecn
fr.lower_dlci
fr.nlpid
fr.second_dlci
fr.snap.oui
fr.snap.pid
fr.snaptype
fr.third_dlci
fr.upper_dlci

PPP

ppp.address
ppp.control
ppp.direction
ppp.protocol

MPLS

mpls.bottom
mpls.cw.control
mpls.cw.res
mpls.exp
mpls.label
mpls.oam.defect_location
mpls.oam.defect_type
mpls.oam.frequency
mpls.oam.function_type
mpls.oam.ttsi
mpls.ttl

ICMP

icmp.checksum
icmp.checksum_bad
icmp.code
icmp.ident
icmp.redir_gw
icmp.seq
icmp.type

DTP

dtp.neighbor
dtp.tlv_len
dtp.tlv_type
dtp.version
vtp.neighbor

VTP

vtp.code
vtp.conf_rev_num
vtp.followers
vtp.md
vtp.md5_digest
vtp.md_len
vtp.seq_num
vtp.start_value
vtp.upd_id
vtp.upd_ts
vtp.version
vtp.vlan_info.802_10_index
vtp.vlan_info.isl_vlan_id
vtp.vlan_info.len
vtp.vlan_info.mtu_size
vtp.vlan_info.status.vlan_susp
vtp.vlan_info.tlv_len
vtp.vlan_info.tlv_type
vtp.vlan_info.vlan_name
vtp.vlan_info.vlan_name_len
vtp.vlan_info.vlan_type

ICMPv6

icmpv6.all_comp
icmpv6.checksum
icmpv6.checksum_bad
icmpv6.code
icmpv6.comp
icmpv6.head.ha_addrs
icmpv6.identifier
icmpv6.option
icmpv6.option.cga
icmpv6.option.cga.pad_length
icmpv6.option.length
icmpv6.option.name_type
icmpv6.option.name_type.fqdn
icmpv6.option.name_x501
icmpv6.option.rsa.key_hash
icmpv6.option.type
icmpv6.ra.cur_hop_limit
icmpv6.ra.reachable_time
icmpv6.ra.retrans_timer
icmpv6.ra.router_lifetime
icmpv6.recursive_dns_serv
icmpv6.type

RIP

rip.auth.passwd
rip.auth.type
rip.command
rip.family
rip.ip
rip.metric
rip.netmask
rip.next_hop
rip.route_tag
rip.routing_domain
rip.version

BGP

bgp.aggregator_as
bgp.aggregator_origin
bgp.as_path
bgp.cluster_identifier
bgp.cluster_list
bgp.community_as
bgp.community_value
bgp.local_pref
bgp.mp_nlri_tnl_id
bgp.mp_reach_nlri_ipv4_prefix
bgp.mp_unreach_nlri_ipv4_prefix
bgp.multi_exit_disc
bgp.next_hop
bgp.nlri_prefix
bgp.origin
bgp.originator_id
bgp.type
bgp.withdrawn_prefix

HTTP

http.accept
http.accept_encoding
http.accept_language
http.authbasic
http.authorization
http.cache_control
http.connection
http.content_encoding
http.content_length
http.content_type
http.cookie
http.date
http.host
http.last_modified
http.location
http.notification
http.proxy_authenticate
http.proxy_authorization
http.proxy_connect_host
http.proxy_connect_port
http.referer
http.request
http.request.method
http.request.uri
http.request.version
http.response
http.response.code
http.server
http.set_cookie
http.transfer_encoding
http.user_agent
http.www_authenticate
http.x_forwarded_for

Annexe 2 : Les filtres de capture (TCPDUMP)

Command Line Options			
`-A`	Print frame payload in ASCII	`-q`	Quick output
`-c <count>`	Exit after capturing **count** packets	`-r <file>`	Read packets from **file**
`-D`	List available interfaces	`-s <len>`	Capture up to **len** bytes per packet
`-e`	Print link-level headers in the capture dump	`-S`	Print absolute TCP sequence numbers
`-F <file>`	Use **file** as the filter expression	`-t`	Don't print timestamps
`-G <n>`	Rotate the dump file every **n** seconds	`-v[v[v]]`	Print more verbose output
`-i <iface>`	Specifies the capture interface	`-w <file>`	Write captured packets to **file**
`-K`	Don't verify TCP checksums	`-x`	Print frame payload in hex
`-L`	List data link types for the interface	`-X`	Print frame payload in hex and ASCII
`-n`	Don't convert addresses to names	`-y <type>`	Specify the data link type
`-p`	Don't capture in promiscuous mode	`-Z <user>`	Drop privileges from root to **user**

Capture Filter Primitives	
`[src|dst] host <host>`	Matches a host as the IP source, destination, or either
`ether [src|dst] host <ehost>`	Matches a host as the Ethernet source, destination, or either
`gateway host <host>`	Matches packets which used **host** as a gateway
`[src|dst] net <network>/<len>`	Matches packets to or from an endpoint residing in **network**
`[tcp|udp] [src|dst] port <port>`	Matches TCP or UDP packets sent to/from **port**
`[tcp|udp] [src|dst] portrange <p1>-<p2>`	Matches TCP or UDP packets to/from a port in the given range
`less <length>`	Matches packets less than or equal to **length**
`greater <length>`	Matches packets greater than or equal to **length**
`(ether|ip|ip6) proto <protocol>`	Matches an Ethernet, IPv4, or IPv6 protocol
`(ether|ip) broadcast`	Matches Ethernet or IPv4 broadcasts
`(ether|ip|ip6) multicast`	Matches Ethernet, IPv4, or IPv6 multicasts
`type (mgt|ctl|data) [subtype <subtype>]`	Matches 802.11 frames based on type and optional subtype
`vlan [<vlan>]`	Matches 802.1Q frames, optionally with a VLAN ID of **vlan**
`mpls [<label>]`	Matches MPLS packets, optionally with a label of **label**
`<expr> <relop> <expr>`	Matches packets by an arbitrary expression

Protocols		
`arp`	`ip6`	`slip`
`ether`	`link`	`tcp`
`fddi`	`ppp`	`tr`
`icmp`	`radio`	`udp`
`ip`	`rarp`	`wlan`

TCP Flags	
`tcp-urg`	`tcp-rst`
`tcp-ack`	`tcp-syn`
`tcp-push`	`tcp-fin`

Modifiers	
! or `not`	
`&&` or `and`	
`||` or `or`	

Examples	
`udp dst port not 53`	All UDP not bound for port 53
`host 10.0.0.1 && host 10.0.0.2`	All packets between these hosts
`tcp dst port 80 or 8080`	All packets to either TCP port

ICMP Types		
`icmp-echoreply`	`icmp-routeradvert`	`icmp-tstampreply`
`icmp-unreach`	`icmp-routersolicit`	`icmp-ireq`
`icmp-sourcequench`	`icmp-timxceed`	`icmp-ireqreply`
`icmp-redirect`	`icmp-paramprob`	`icmp-maskreq`
`icmp-echo`	`icmp-tstamp`	`icmp-maskreply`

Annexe 3 : LA TRAME ETHERNET

1 – Format de la trame Ethernet

Préambule	Adresse Physique Destinataire	Adresse Physique Source	Type	Données	CRC
8 octets	*6 octets*	*6 octets*	*2 octets*	*46 à 1500 octets*	*4 octets*

Préambule : 64 bits de synchronisation, alternance de **1** et **0** avec les deux derniers bits à 1 (**SFD**).

Adresse Physique destination (6 octets) : adresse physique (Ethernet) de la station devant recevoir la trame. Les trois premiers octets de cette adresse sont imposés par l'IEEE aux fabricants de contrôleurs, ce qui garantit son unicité. Il y a diffusion si tous les bits sont à 1.

Adresse Physique source (6 octets) : adresse physique (Ethernet) de la station ayant émis la trame.

Type (2 octets) : identifie le protocole de niveau supérieur associé au paquet.

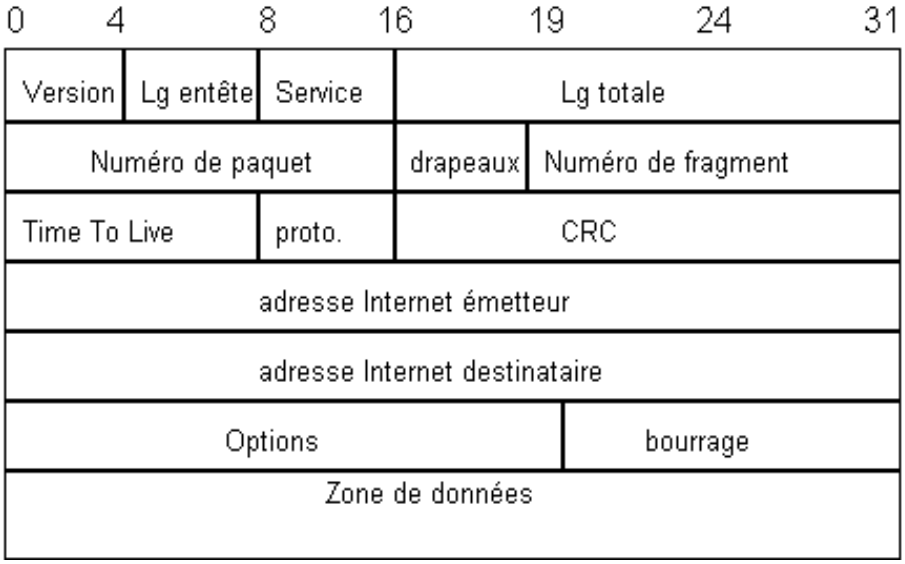
- **0x0800** = Protocole **IP**,
- **0x0806** = Protocole **ARP**,
- **0x8035** = Protocole **RARP**.

Données (4 octets) : les informations à transporter. La trame ARP ou l'entête IP (qui comporte les adresses Internet sources et destinataires) suivi des données propres aux protocoles de niveaux supérieurs (par exemple les données TCP).

CRC (4 octets) : Contrôle de Redondance Cyclique (**C**yclic **R**edundancy **C**heck). C'est une somme de contrôle portant sur tout ce qui précède sauf le préambule.

2 – Format du datagramme IP (Internet Protocol)

Un **datagramme IP** a la structure suivante :



Version (4 bits) : Le champ Version renseigne sur le format de l'en-tête Internet. Ce document décrit le format de la version **4** du protocole.

Longueur d'En-Tête (4 bits) : Le champ Longueur d'En-Tête (LET) code la longueur de l'en-tête Internet, l'unité étant le mots de 32 bits, et de ce fait, marque le début des données. Notez que ce champ ne peut prendre une valeur en dessous de 5 pour être valide.

Type de Service (8 bits) : Le "Type de Service" sert à préciser le traitement effectué sur le datagramme pendant sa transmission à travers Internet. Principalement, le choix offert est une négociation entre les trois contraintes suivantes : faible retard, faible taux d'erreur, et haut débit.

```
Bits 0-2 :     Priorité.
Bit    3 : D   0 = Retard standard,        1 = Retard faible.
Bits   4 : T   0 = Débit standard,         1 = Haut débit.
Bits   5 : R   0 = Taux d'erreur standard  1 = Taux d'erreur
                                           faible.
Bit  6-7 :     Réservé.
```

```
          +  0     1     2     3     4     5     6     7  +
          +-----+-----+-----+-----+-----+-----+-----+-----+
          |    PRIORITE     |  D  |  T  |  R  |  0  |  0  |
          +-----+-----+-----+-----+-----+-----+-----+-----+
```

Priorité

```
111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine
```

Longueur Totale (16 bits) : Le champ "Longueur Totale" est la longueur du datagramme entier y compris en-tête et données, mesurée en octets. Ce champ ne permet de coder qu'une longueur de datagramme d'au plus 65535 octets.

Identification (16 bits) : Une valeur d'identification assignée par l'émetteur pour identifier les fragments d'un même datagramme.

Flags (3 bits) : Divers commutateurs de contrôle.

```
Bit 0: réservé, doit être laissé à zéro
Bit 1: (AF) 0 = Fragmentation possible,  1 = Non fractionnable.
Bit 2: (DF) 0 = Dernier fragment, 1 = Fragment intermédiaire.
    0   1   2
  +---+---+---+
  |   | A | D |
  | 0 | F | F |
  +---+---+---+
```

Fragment Offset (Position relative) (13 bits) : Ce champ indique le décalage du premier octet du fragment par rapport au datagramme complet. Cette position relative est mesurée en blocs de 8 octets (64 bits). Le décalage du premier fragment vaut zéro.

Durée de vie (8 bits) : Ce champ permet de limiter le temps pendant lequel un datagramme reste dans le réseau. Si ce champ prend la valeur zéro, le datagramme doit être détruit. Ce champ est modifié pendant le traitement de l'en-tête Internet. La durée de vie est mesurée en secondes.

Protocole (8 bits) : Ce champ indique quel protocole de niveau supérieur est utilisé dans la section données du datagramme IP.

- **0x01** = Protocole **ICMP**,
- **0x02** = Protocole **IGMP**,
- **0x06** = Protocole **TCP,**
- **0x11** = Protocole **UDP**.

Checksum d'en-tête (16 bits) : Un Checksum calculé sur l'en-tête uniquement. Comme certains champs de l'en-tête sont modifiés (ex., durée de vie) pendant leur transit à travers le réseau, ce Checksum doit être recalculé et vérifié en chaque point du réseau où l'en-tête est réinterprétée.
L'algorithme utilisé pour le Checksum est le suivant : On calcule le complément à un sur 16 bits de la somme des compléments à un de tous les octets de l'en-tête pris par paires (mots de 16 bits).

Adresse source (32 bits) : L'adresse IP de la source.

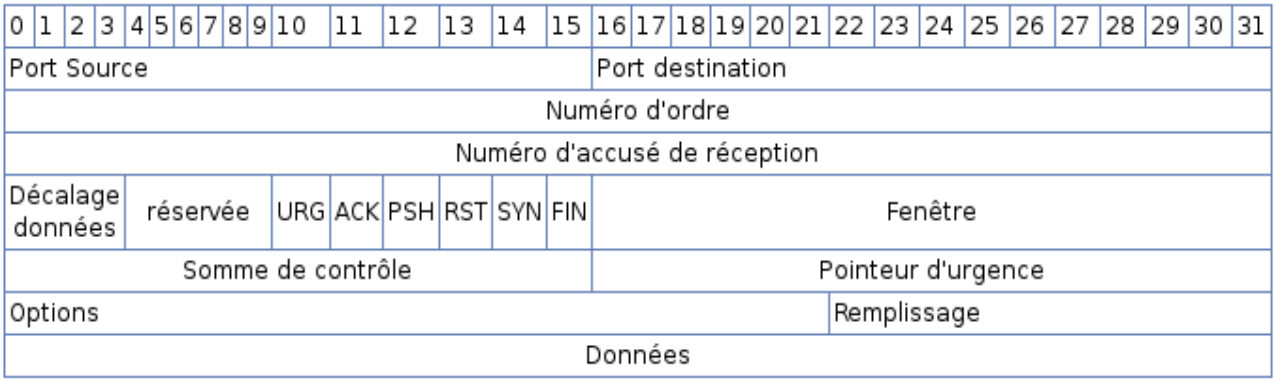
Adresse destination (32 bits) : L'adresse IP du destinataire.

Options (variable) : Les datagrammes peuvent contenir des options. Celles-ci doivent être implémentées par tous les modules IP (hôtes et routeurs). Le caractère "optionnel" concerne leur transmission, et non leur implémentation.

Bourrage (variable) : Le champ de bourrage n'existe que pour assurer à l'en-tête une taille totale multiple de 4 octets. Le bourrage se fait par des octets à zéro.

3 – Format du segment TCP (Transfert Control Protocol)

Un segment **TCP** a la structure suivante :



Port Source (16 bits) : Port relatif à l'application en cours sur la machine source;

Port Destination (16 bits) : Port relatif à l'application en cours sur la machine de destination;

Numéro d'ordre (32 bits) : Lorsque le drapeau SYN est à 0, le numéro d'ordre est celui du premier mot du segment en cours. Lorsque SYN est à 1, le numéro d'ordre est égal au numéro d'ordre initial utilisé pour synchroniser les numéros de séquence (ISN);

Numéro d'accusé de réception (32 bits) : Le numéro d'accusé de réception également appelé numéro d'acquittement correspond au numéro (d'ordre) du prochain segment attendu, et non le numéro du dernier segment reçu;

Décalage des données (4 bits) : il permet de repérer le début des données dans le paquet. Le décalage est ici essentiel car le champ d'options est de taille variable

Drapeaux (flags) (6 x 1 bit) :

- **URG** : si ce drapeau est à 1 le paquet doit être traité de façon urgente;
- **ACK** : si ce drapeau est à 1 le paquet est un accusé de réception;
- **PSH** (PUSH) : si ce drapeau est à 1, le paquet fonctionne suivant la méthode PUSH;
- **RST** : si ce drapeau est à 1, la connexion est réinitialisée;
- **SYN** : Le Flag TCP SYN indique une demande d'établissement de connexion;
- **FIN** : si ce drapeau est à 1 la connexion s'interrompt;

Fenêtre (16 bits) : Champ permettant de connaître le nombre d'octets que le récepteur souhaite recevoir sans accusé de réception;

Somme de contrôle (Checksum ou CRC) : La somme de contrôle est réalisée en faisant la somme des champs de données de l'en-tête;

Pointeur d'urgence (16 bits) : Indique le numéro d'ordre à partir duquel l'information devient urgente;

Options (Taille variable) : Des options diverses;

Remplissage: On remplit avec des **0** pour avoir une longueur multiple de 32 bits.

4 – Protocole ICMP (Internet Control Message Protocol)

Le protocole **ICMP** (*Internet Control Message Protocol*) est un protocole qui permet de gérer les informations relatives aux **erreurs** des machines connectées.
Il permet non pas de corriger les erreurs mais de faire part de ces erreurs aux protocoles des couches voisines.

Voici à quoi ressemble un **message ICMP** encapsulé dans un datagramme **IP** :

Type (8 bits)	**Code (8 bits)**	**Checksum (16 bits)**	**Message (taille variable)**

Type du message :

Valeur du type	Signification
0	Réponse à une demande d'écho
3	Destination inaccessible
4	Limitation de production à la source
5	Redirection
8	Demande d'écho
11	Expiration de délai pour un datagramme
12	Problème de paramètre d'un datagramme
13	Demande d'horodatage
14	Réponse à une demande d'horodatage
17	Demande de masque d'adresse
18	Réponse à une demande de masque

Code du message :

Valeur du code	Signification
0	Réseau inaccessible (si type != 0 ou != 8)
1	machine inaccessible
2	Protocole inaccessible
3	Port inaccessible
4	fragmentation nécessaire mais impossible à cause du drapeau DF
5	le routage a échoué
6	réseau inconnu
7	machine inconnue
8	machine non connectée au réseau (inutilisé)
9	communication avec le réseau interdite
10	communication avec la machine interdite
11	réseau inaccessible pour ce service
12	machine inaccessible pour ce service

5 – Protocole ARP (Adress Resolution Protocol)

Address Resolution Protocol (**ARP**, protocole de résolution d'adresse) est un protocole effectuant la traduction d'une **adresse** de protocole **de couche réseau** (typiquement une adresse **IPv4**) en une **adresse ethernet** (typiquement une adresse **MAC**).
Il est nécessaire au fonctionnement d'**IPv4**, mais semble inutile au fonctionnement d'**IPv6**. En IPv6, ARP devient obsolète et est remplacé par Internet Control Message Protocol V6.

Voici à quoi ressemble un **message ARP** encapsulé dans la trame **Ethernet** :

Bits 0 - 7	Bits 8 - 15	Bits 16 - 31
type de matériel		**type de Protocole**
longueur de l'adresse physique	**longueur de l'adresse logique**	**Opération**
adresse physique (MAC) source (6 octets)		
adresse logique (IP) source (4 octets)		
adresse physique (MAC) de destination (6 octets)		
adresse logique (IP) de destination (4 octets)		

Type de matériel (16 bits) :
- **01** - Ethernet (10Mb) [JBP]
- **15** - Frame Relay [AGM]
- **16** - Asynchronous Transmission Mode (ATM) [JXB2]
- **17** - HDLC [JBP]
- **18** - Fibre Channel [Yakov Rekhter]
- **19** - Asynchronous Transmission Mode (ATM) [RFC2225]
- **20** - Serial Line [JBP]
- **21** - Asynchronous Transmission Mode (ATM) [MXB1]

Type de protocole (16 bits) : Type de protocole couche 3 (OSI) qui utilise Arp.
- **0x0800** - IP

Longueur de l'adresse physique (8 bits) : La longueur doit être prise en octets.
- **01** - Token Ring
- **06** - Ethernet

Longueur de l'adresse logique (8 bits) : La longueur doit être prise en octets.
- **04** - IP v4
- **16** - IP v6

Opération (16 bits) :
- **01** - Request (*requête)*
- **02** - Reply (*réponse*)