

Wireshark — Audit et performance



- Comprendre l'analyse des flux réseau
- Savoir filtrer et analyser l'activité du réseau
- Savoir produire des rapports
- Savoir utiliser Wireshark pour diagnostiquer des problèmes de performance du réseau

Chapitre 01 — Rappels des fondamentaux

- Méthodes de communications (unicast, multicast, broadcast)
- Les topologies et le contrôle d'accès — Modèle OSI
- Format d'une trame Ethernet
- Le protocole ARP
- Protocoles de couche 2 (802.3, 802.1p, 802.1q, 802.1ad)
- Format d'un paquet IP
- Adresses particulières (loopback)
- Adresses de multicast
- Protocole ICMP

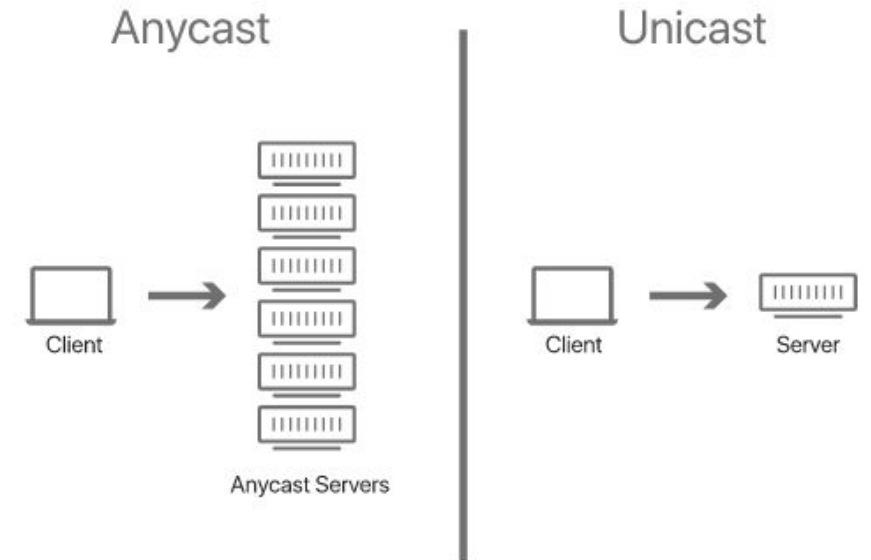
Méthodes de communication

Unicast : communication d'un ordinateur à un autre bien identifié

Anycast : une même adresse IP pour plusieurs serveurs (ex : DNS Anycast → le serveur le plus proche répond)

Multicast : données envoyées à un groupe d'ordinateurs inscrits (ex : streaming live)

Broadcast : paquet envoyé à tous les ordinateurs d'un réseau local



Topologies réseau

Deux modes de propagation :

Mode de diffusion (bus, anneau)

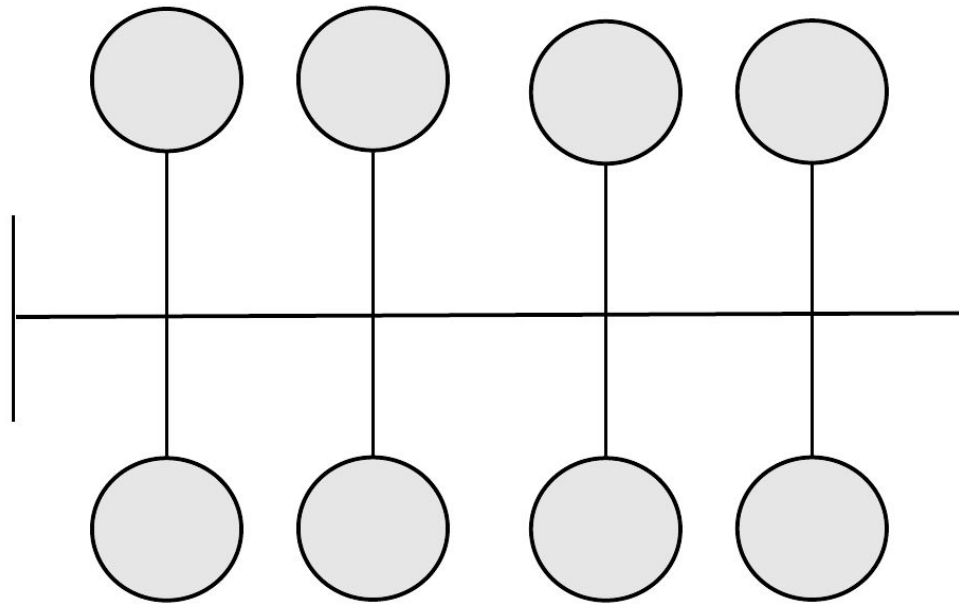
- Un seul support de transmission partagé
- Toute machine voit le message et filtre selon l'adresse destination

Mode point à point (étoile, maillé)

- Support physique reliant une paire d'unités
- Communication via un nœud intermédiaire

Le réseau en bus

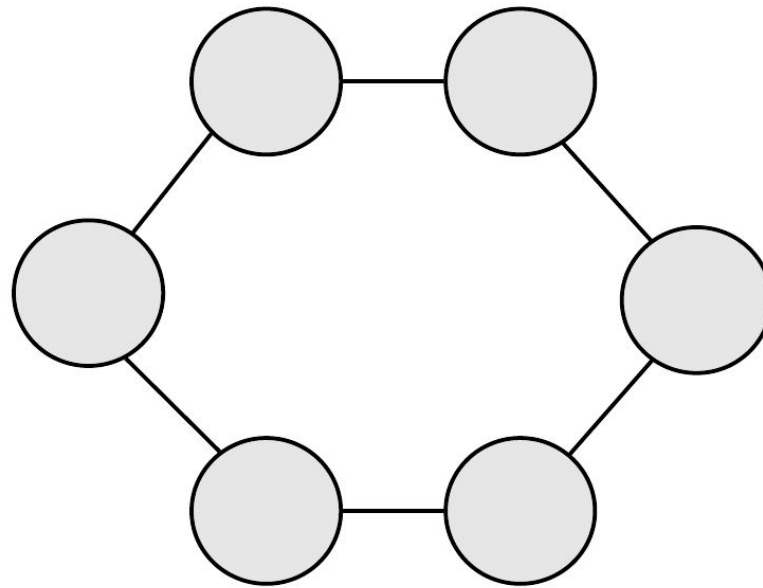
- Un seul canal partagé par tous les points
- ⚠ Une rupture de connexion empêche toute communication



Topologie en bus

Le réseau en anneau

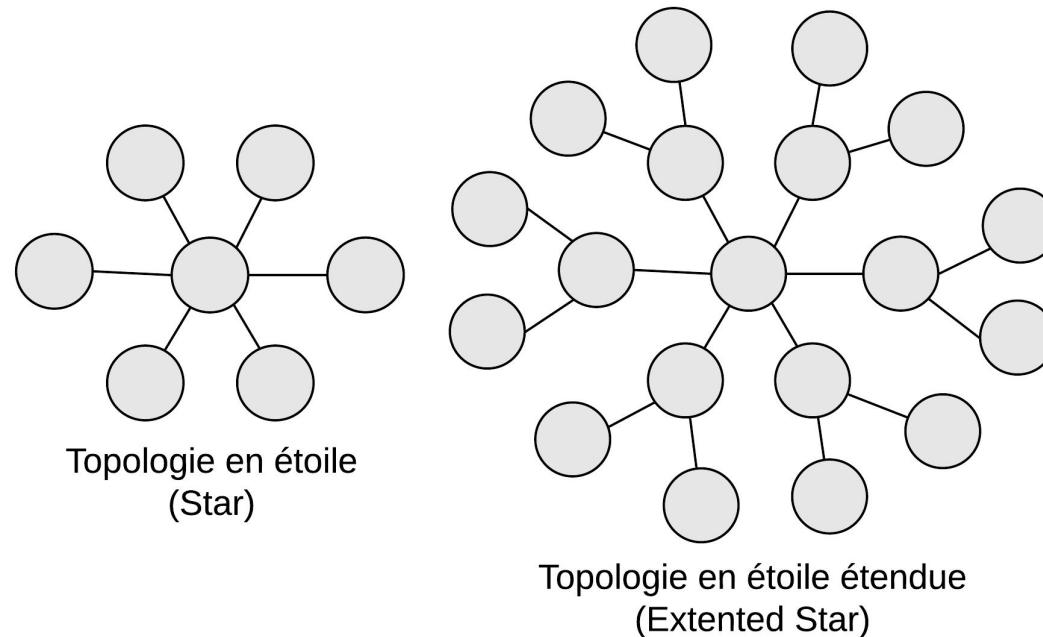
- La rupture d'une liaison n'empêche pas la communication (chemin alternatif)



Topologie en anneau

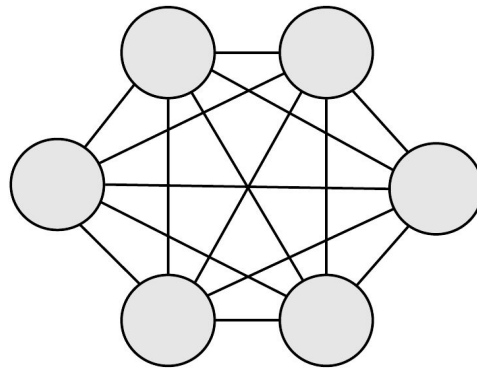
Le réseau en étoile

- Le point de concentration est un **point unique de défaillance**
- Dans une topologie étendue, une rupture isole une partie des nœuds

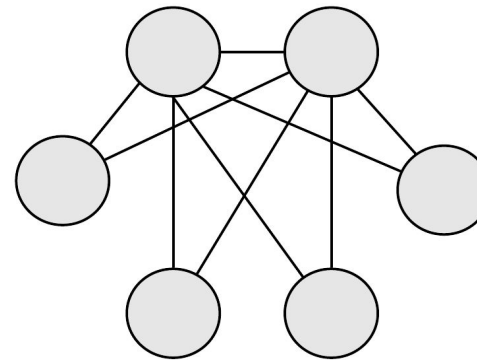


Le réseau maillé

- Tous les nœuds interconnectés entre eux
- Résistant aux ruptures multiples — redondance maximale
- ⚠ Chaque nœud ajouté augmente exponentiellement le nombre de liaisons



Topologie totalement
maillée (Full Mesh)



Topologie Partiellement
maillée (Partial Mesh)

Méthodes d'accès aux réseaux

La méthode d'accès définit **comment la carte réseau accède au réseau** (qui parle, quand, combien de temps).

Trois catégories principales :

| Méthode | Topologie |
|------------------|------------------------|
| CSMA/CD | Bus, étoile (Ethernet) |
| CSMA/CA | Wi-Fi |
| Passage du jeton | Anneau (Token Ring) |

CSMA/CD

Carrier Sense Multiple Access with Collision Detection

Deux mécanismes fondamentaux :

1. **Détection de porteuse** : écoute du canal avant transmission (s'assure qu'il n'est pas occupé)
2. **Détection de collisions** : écoute continue pendant la transmission (détecte les transmissions parallèles)

La méthode du passage du jeton

- Propre aux réseaux en anneau
- Les collisions sont **impossibles** — stations ne peuvent pas émettre simultanément
- Un **jeton** (paquet spécial) circule de station en station
- Seule la station détenant le jeton peut émettre

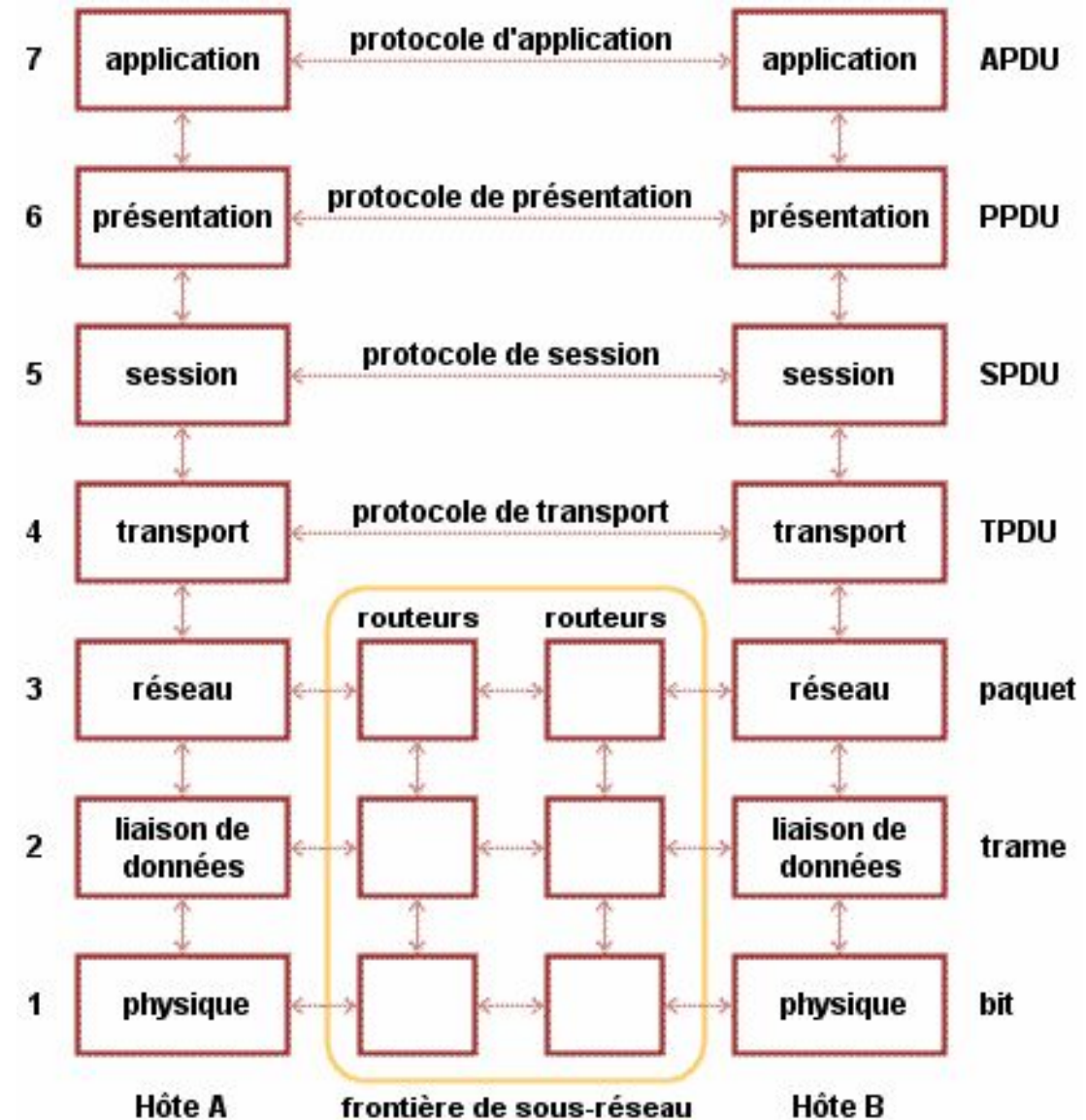
Avantage : pas de collisions → pas de délais de retransmission

Inconvénient : délais d'attente pour obtenir le jeton

Modèle OSI

Sept couches :

1. **Physique** — transmission des bits
2. **Liaison** — gestion du réseau local (MAC, ARP) — trame
3. **Réseau** — routage IP — paquet
4. **Transport** — communication entre processus (TCP, UDP) — message
5. **Session** — gestion des connexions/déconnexions
6. **Présentation** — conversion, chiffrement, compression
7. **Application** — services réseau (FTP,



Couches OSI — Basses (matérielles)

Couche Physique (1)

Transmission des bits, encodage, synchronisation

Couche Liaison (2)

Transmission entre deux ordinateurs via un lien — gère MAC, ARP

Unité : **trame** (quelques centaines à milliers d'octets)

Couche Réseau (3)

Adressage et routage IP — interconnexion des réseaux

Unité : **paquet**

Couches OSI — Hautes (logicielles)

Couche Transport (4) — TCP, UDP — communication entre processus

Unité : **message**

Couche Session (5) — connexions, déconnexions, synchronisation

Couche Présentation (6) — syntaxe/sémantique des données, chiffrement, compression

Couche Application (7) — services réseaux pour l'utilisateur (transfert de fichiers, messagerie...)

L'histoire d'Ethernet

- **Années 1970** : Ethernet conçu par Robert Metcalfe (Xerox PARC)
- **1979** : Metcalfe quitte Xerox pour promouvoir l'Ethernet
- **Coalition DEC + Intel + Xerox** pour en faire un standard

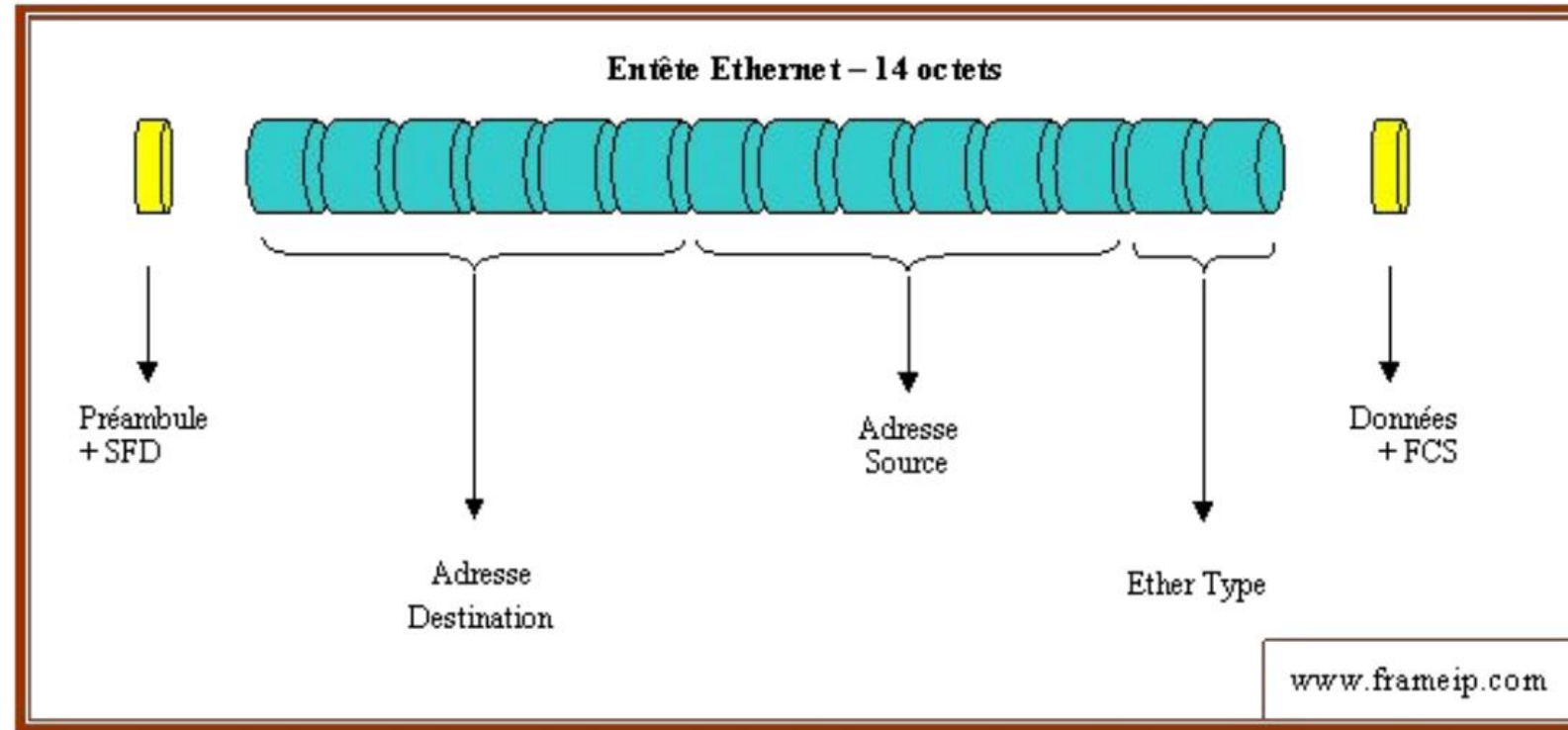
Pourquoi Ethernet a dominé :

- Première technologie LAN haut débit grand public
- Protocole décentralisé CSMA/CD (toutes les stations sont égales)
- Ajout/retrait de machines sans perturber le réseau
- Coût bien inférieur aux technologies concurrentes (Token Ring, FDDI, ATM)

Ethernet CSMA/CD

- Les nœuds sont reliés par un **canal à diffusion**
- Toute trame émise est reçue par tous les adaptateurs
- Algorithme **CSMA/CD** : Carrier Sense Multiple Access with Collision Detection

Format d'une trame Ethernet



Champs de la trame Ethernet

| Champ | Taille | Description |
|---------------------|----------------|---|
| Préambule | 7 octets | Synchronisation (10101010) |
| SFD | 1 octet | Délimiteur de début (10101011) |
| Adresse destination | 6 octets | MAC destinataire (FF:FF:FF:FF:FF:FF = broadcast) |
| Adresse source | 6 octets | MAC émetteur |
| Ether Type | 2 octets | Type de protocole (couche 3) |
| Données | 46–1500 octets | Payload (+ padding si < 46 octets) |
| FCS | 4 octets | Contrôle d'erreurs (CRC) |

Adresse MAC

Structure de l'adresse MAC (6 octets) :

| 3 premiers octets | 3 derniers octets |
|--|--------------------------------|
| OUI (Organizationally Unique Identifier) | Identifiant unique de la carte |
| Identifie le constructeur (géré par l'IEEE) | Attribué par le constructeur |

Format : `XX:XX:XX:YY:YY:YY`

FCS — Frame Check Sequence

- **4 octets** — détection d'erreurs dans la trame
- Valeur calculée par un **CRC** (Cyclic Redundancy Code)
- À la réception : recalcul et comparaison des deux résultats
- Sources d'erreurs : variations d'affaiblissement du signal, induction électromagnétique

Définition du protocole ARP

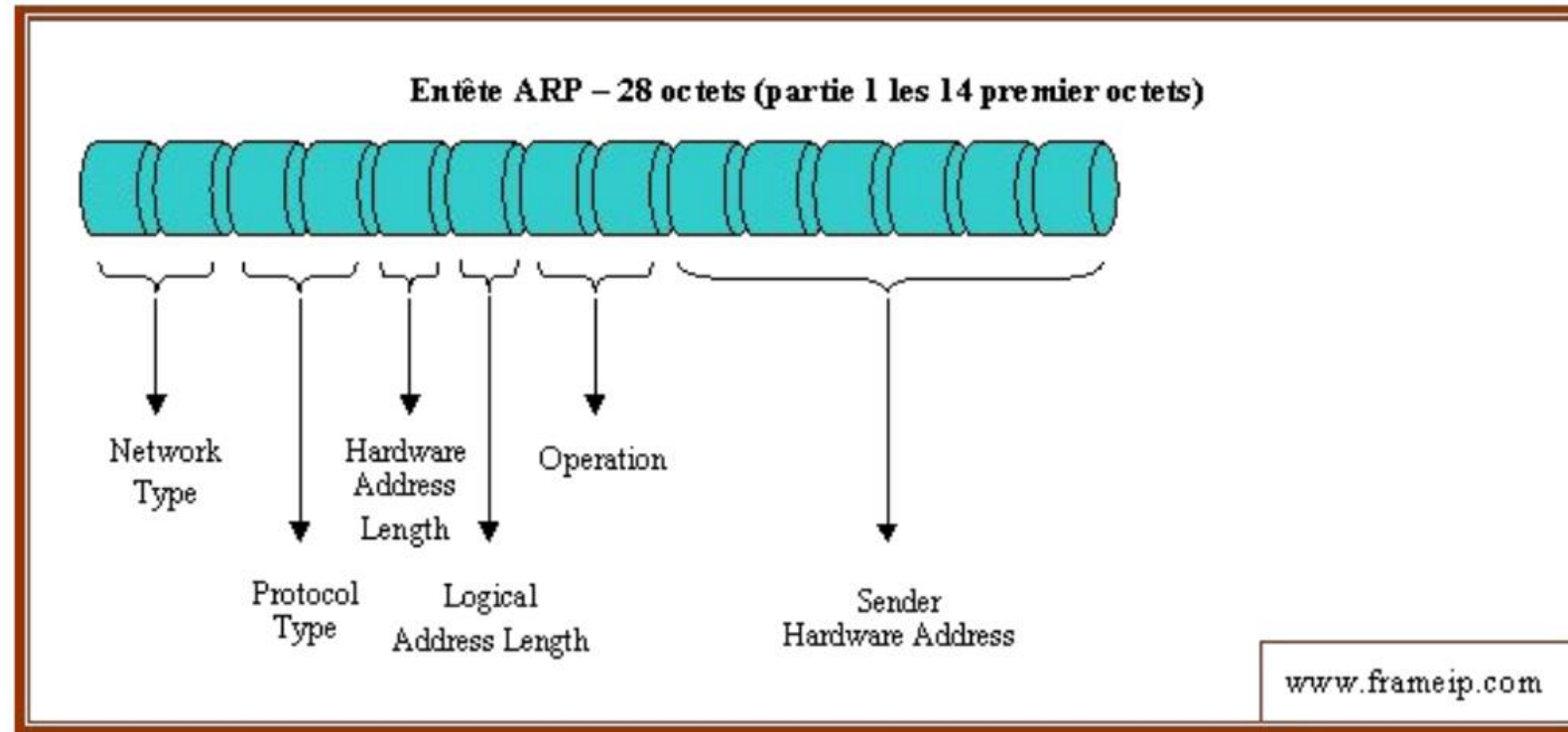
ARP = Address Resolution Protocol — couche Internet (couche 3 OSI)

Objectif : résoudre une adresse physique (MAC) à partir d'une adresse IP

Table ARP : correspondance IP ↔ MAC

- Alimentation **automatique** via ARP
- Alimentation **manuelle** via l'administrateur

Structure de l'entête ARP



Fonctionnement ARP

ARP Request (broadcast)

« Je suis `00:08:54:0B:21:77` . Qui possède l'adresse IP `192.168.0.1` ? »

ARP Reply (unicast)

Le propriétaire répond avec son adresse MAC et met à jour sa propre table ARP

Ressources :

- <https://wiki.wireshark.org/AddressResolutionProtocol>
- <https://www.frameip.com/entete-arp/>

Protocoles de couche 2 — 802.x

| Norme | Description |
|----------------|--|
| 802.3 | Spécifications CSMA/CD — base des LAN filaires |
| 802.1p | QoS/CoS au niveau MAC — priorité des trames (ex: VoIP) |
| 802.1Q | VLAN — ajoute 4 octets dans l'entête (trame "taggée") |
| 802.1ad | QinQ — empile plusieurs tags VLAN (réseaux métropolitains) |

Définition du protocole IP

IP = Internet Protocol — protocole réseau le plus répandu

- Découpe l'information en **paquets**
- Les adresse, transporte indépendamment, et recompose à l'arrivée

Structure de l'entête IP

En-tête IPv4

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|---|---|---|-----------------------|---|---|---|-----------------|---|----|----|----|----|----|----|--------------------------------|----|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Version d'IP | | | | Longueur de l'en-tête | | | | Type de service | | | | | | | | Longueur totale | | | | | | | | | | | | | | | |
| Identification | | | | | | | | | | | | | | | | Indicateur | | Fragment offset | | | | | | | | | | | | | |
| Durée de vie | | | | | | | | Protocole | | | | | | | | Somme de contrôle de l'en-tête | | | | | | | | | | | | | | | |
| Adresse source | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Adresse destination | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Option(s) + remplissage | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Champs de l'entête IP

| Champ | Description |
|---------------------------|---|
| Version (4 bits) | Numéro de version IPv4/IPv6 |
| ToS (8 bits) | Type Of Service — gestion de la QoS (couche 3) |
| TTL | Nombre max de routeurs de transit — valeur recommandée : 64 |
| Protocole (8 bits) | Type de données après l'entête (ICMP, IGMP, TCP, UDP) |

Fragmentation IP

- Chaque interface a une taille max de trame : **MTU** (Maximum Transmission Unit)
- Si le paquet > MTU → **fragmentation**
- MTU Ethernet standard : **1500 octets**
- Taille max IPv4 : 65 535 octets (codée sur 16 bits)

Adresses particulières

Loopback

- Interface réservée aux communications locales (Adresse : 127.0.0.1)
- Permet à l'hôte de s'envoyer des paquets à lui-même

Multicast (Classe D)

- Plage : 224.0.0.0 → 239.255.255.255
- 224.0.0.1 à 224.0.0.255 : utilisation locale
- Protocole d'adhésion : **IGMP**

Broadcast

- Ex : 192.168.1.255 pour un réseau en /24
- Reçu par toutes les machines du réseau

Protocole ICMP

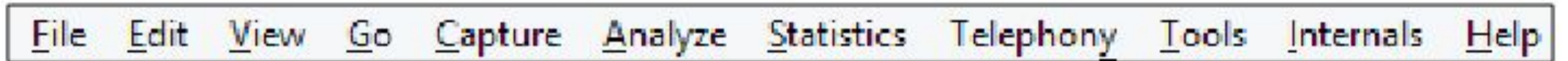
ICMP = Internet Control Message Protocol

- Gère les **informations d'erreur** du protocole IP
- Ne corrige pas les erreurs — **les signale** à l'émetteur
- Implémenté par toutes les piles IP (routeurs, stations)

Chapitre 02 — L'écran de Wireshark

Menu

La barre de menus contient toutes les fonctionnalités de Wireshark.



Barre d'outils principale

- Raccourcis vers les fonctions les plus utilisées
- Certaines options sont **grisées** selon le contexte (capture active ou non)






Barre d'outils de filtrage

Deux types de filtres :

| Type | Description |
|---------------------------|---|
| Filtre de capture | Définit ce qui est capturé — nécessite un redémarrage |
| Filtre d'affichage | Définit ce qui est affiché après capture — modifiable à chaud |

Indicateurs visuels :

-  Rouge : expression invalide
-  Vert : expression correcte
-  Jaune : expression peut ne pas fonctionner comme attendu

Filter:

▼

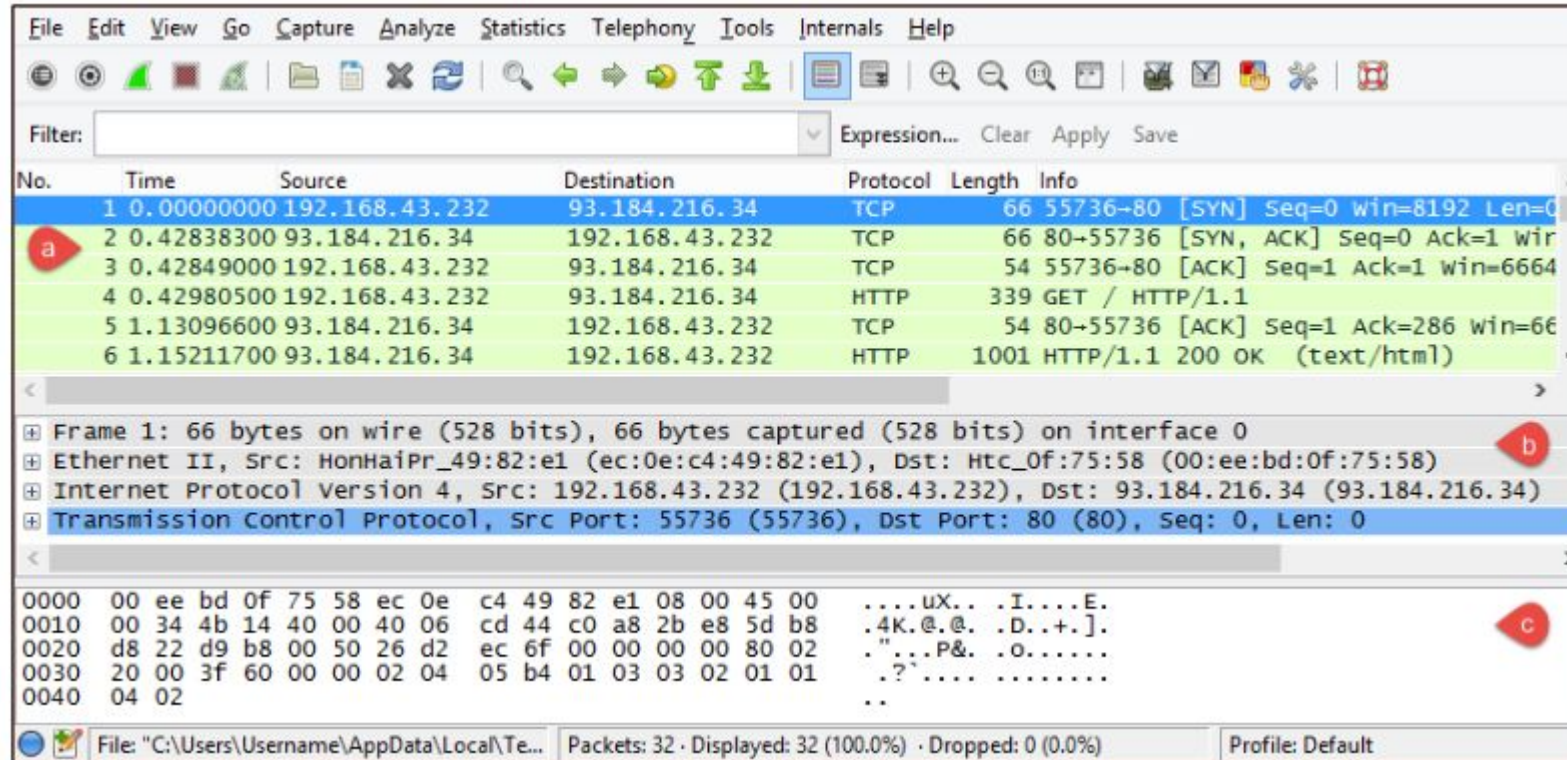
Expression...

Clear

Apply

Save

Zone d'affichage des paquets





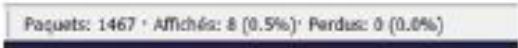
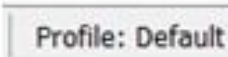
A — Liste des paquets : paquets capturés avec détails de base

B — Détail du paquet : analyse granulaire du paquet sélectionné (ports, flags...)

C — Vue hexadécimale : octets du paquet en format hex

Barre d'état

En dessous de la vue hexadécimale

| Icône | Signification |
|---|--|
|  | Information expert : suivant les erreurs rencontrées ou non, la couleur de cet icône change |
|  | Propriétés du fichier de capture : temps de capture, débit moyen.... |
|  | Le nombre de paquets total, le nombre de paquets affichés ou bien si Wireshark a drop des paquets avec les champs perdus |
|  | Nom du profil utilisé actuellement (on peut changer de profil suivant nos besoins) |

Chapitre 03 — Les tâches d'analyse avec Wireshark

Capture des communications en "clear text"

Wireshark capture tout le trafic réseau — les protocoles **non chiffrés** sont lisibles en clair.

Exemples :

- **FTP** — port 21 — transfert de fichiers en clair (identifiants visibles)
- **HTTP** — port 80 — navigation Web en clair (GET et POST)

FTP — File Transfer Protocol

- Port TCP par défaut : **21**
- Identifiants utilisateur **visibles en clair**

| | | | | | |
|----|-------------|----------------|----------------|-----|---|
| 5 | 0.001510000 | 192.168.20.129 | 192.168.20.200 | TCP | 49944→21 [ACK] Seq=1 Ack=28 win=29696 L |
| 6 | 3.285827000 | 192.168.20.129 | 192.168.20.200 | FTP | Request: USER anonymous |
| 7 | 3.286395000 | 192.168.20.200 | 192.168.20.129 | FTP | Response: 331 Anonymous access allowed, |
| 8 | 3.286570000 | 192.168.20.129 | 192.168.20.200 | TCP | 49944→21 [ACK] Seq=17 Ack=100 win=29696 |
| 9 | 5.610442000 | 192.168.20.129 | 192.168.20.200 | FTP | Request: PASS anonymous |
| 10 | 5.611472000 | 192.168.20.200 | 192.168.20.129 | FTP | Response: 230 Anonymous user logged in. |

HTTP — HyperText Transfer Protocol

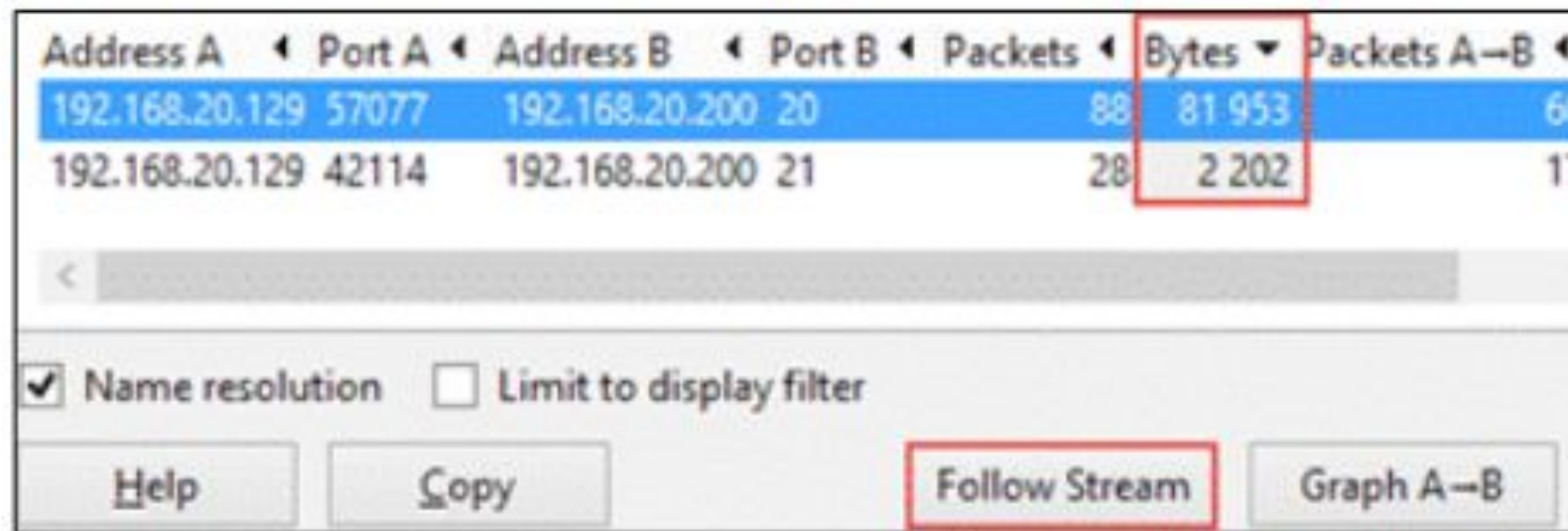
- Port TCP par défaut : **80**
- Trafic en **texte brut** — ni GET ni POST ne sont sécurisés
- Requêtes et réponses lisibles directement

| | |
|----------------|--|
| TCP Ports: | 80,3128,3132,5985,8080,8088,11371,1900,2869,2710 |
| SSL/TLS Ports: | 443 |

Réassemblage du flux de données

Pour reconstituer les fichiers transférés en clair :

1. **Statistiques > Conversations TCP** — trier par octets transférés
2. Clic droit → **Follow Stream** sur la conversation cible



| Address A | Port A | Address B | Port B | Packets | Bytes | Packets A→B |
|----------------|--------|----------------|--------|---------|--------|-------------|
| 192.168.20.129 | 57077 | 192.168.20.200 | 20 | 88 | 81 953 | 60 |
| 192.168.20.129 | 42114 | 192.168.20.200 | 21 | 28 | 2 202 | 17 |

☒ Name resolution ☐ Limit to display filter

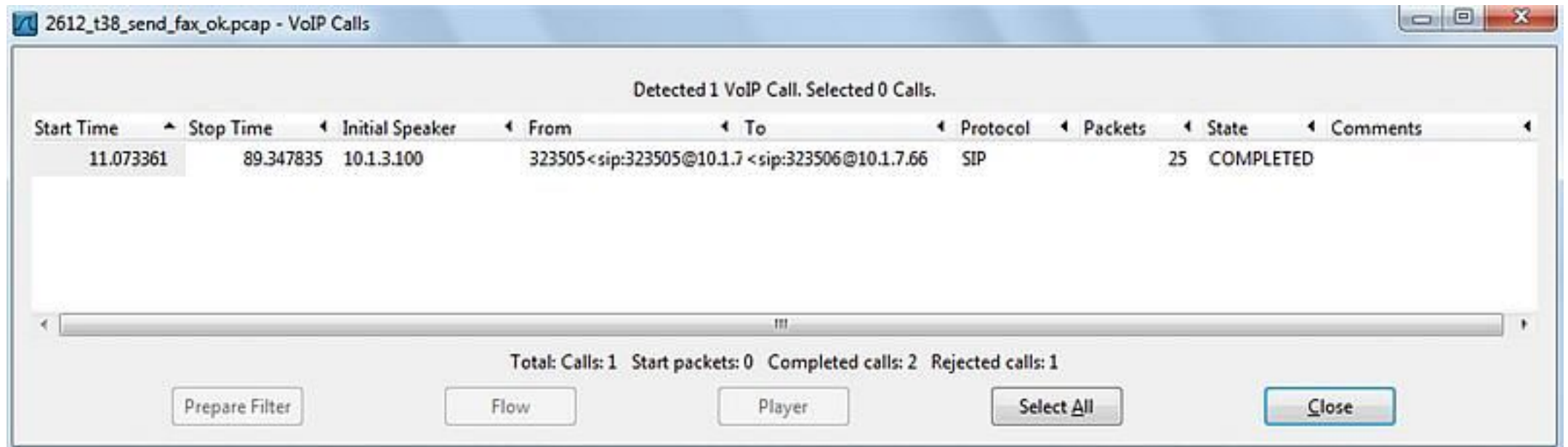
[Help](#) [Copy](#) [Follow Stream](#) [Graph A→B](#)

3. Identifier la signature du fichier (ex: JFIF → JPG)
4. **Enregistrer sous** dans le bon format

Visualiser et écouter des communications VoIP

- Le protocole **RTP** transporte audio et vidéo
- Wireshark peut **suivre, réassembler et rejouer** le flux

Menu : Statistiques (ou Téléphonie) > **Appels VoIP**

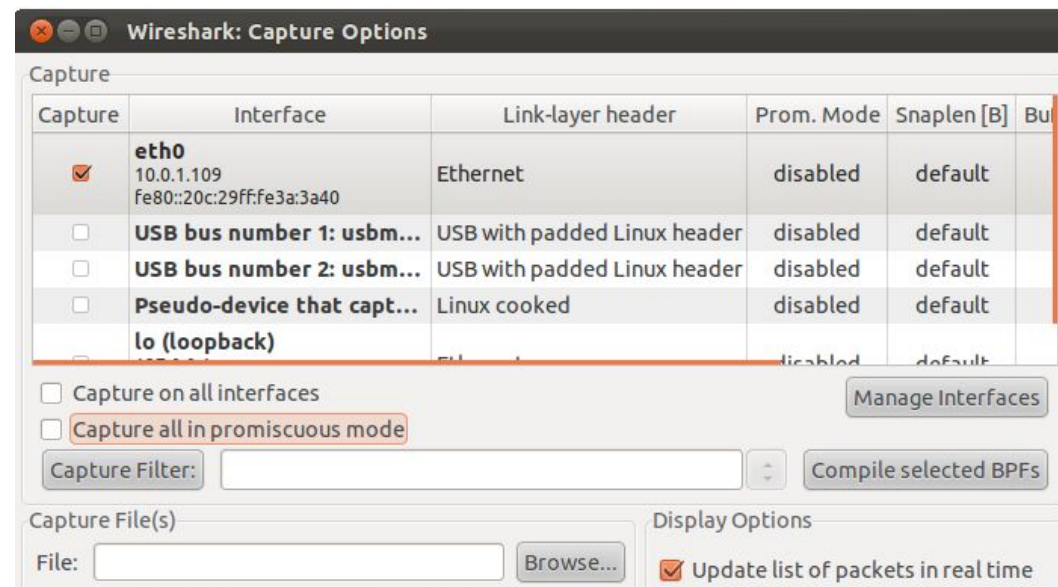


Capturer du trafic inattendu — Mode Promiscuous

Par défaut : une carte réseau ignore les paquets qui ne lui sont pas destinés.

Mode Promiscuous : la carte traite **tous** les paquets du réseau.

→ Permet de capturer le trafic des autres machines.



Filtrer le trafic d'une machine

Filtres de capture (avant la capture) :

```
# Niveau 2 – par adresse MAC  
ether src XX:XX:XX:XX:XX:XX  
  
# Niveau 3 – par adresse IP  
host X.X.X.X
```

Filtres d'affichage (après la capture) :

```
eth.src == XX:XX:XX:XX:XX:XX  
ip.src == 192.168.13.186
```

Filtrer le trafic d'un réseau

Filtre de capture :

```
net 192.168.0.0/24
```

Filtre d'affichage :

```
ip.addr == 192.168.0.0/24
```

Chapitre 04 — Les tâches de dépannage avec Wireshark

Identifier des délais anormaux

Deux types de retards :

| Type | Impact |
|-------------------------|---------------------------------------|
| Retards normaux | N'affectent pas l'utilisateur |
| Retards anormaux | Font sonner le téléphone du support 📞 |

Retards à ignorer :

- Avant un paquet TCP RST (l'utilisateur est déjà passé à autre chose)
- Avant les requêtes DNS déclenchées par l'utilisateur
- Avant TCP FIN (fin normale de connexion)
- Liés aux Keep-Alive applicatifs

Retards qui comptent

| Retard | Cause probable |
|---------------------------------|--|
| Entre SYN → SYN/ACK | RTT élevé entre les hôtes |
| Entre SYN/ACK → ACK | RTT élevé côté client |
| Entre requête → réponse serveur | Serveur lent (traitement, ressources, attaque ?) |
| Dans un flux de données | Expéditeur occupé, buffer saturé |

"Lent" est relatif — connaissez vos temps normaux pour détecter les anomalies

Comment réduire le RTT

- Réduire le nombre de noms d'hôtes uniques (moins de résolutions DNS)
- Minimiser les redirections HTTP/S
- Supprimer les liens brisés (404/410)
- Combiner les scripts et feuilles de style en moins de fichiers
- Activer le **cache navigateur**
- **CDN** : rapprocher le contenu de l'utilisateur

Détecter les retards dans les conversations TCP

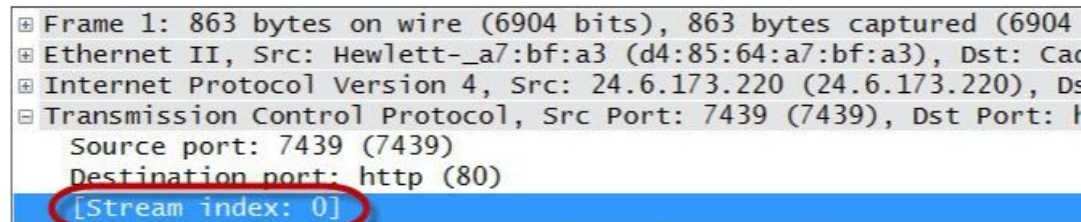
Flux TCP = une conversation TCP distincte (numérotée par Wireshark)

Activer dans les préférences TCP :

"Calculate conversation timestamps"

Deux champs temporels supplémentaires :

- `tcp.time_relative` — Temps depuis la 1ère trame du flux
- `tcp.time_delta` — Temps depuis la trame précédente du flux ← **le plus utile**

A screenshot of the Wireshark packet details pane for Frame 1. The pane shows the following layers: Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The TCP layer details are expanded, showing 'Source port: 7439 (7439)' and 'Destination port: http (80)'. The 'Stream index: 0' field is highlighted with a red circle and a blue background.

```
⊞ Frame 1: 863 bytes on wire (6904 bits), 863 bytes captured (6904
⊞ Ethernet II, Src: Hewlett-_a7:bf:a3 (d4:85:64:a7:bf:a3), Dst: Cac
⊞ Internet Protocol Version 4, Src: 24.6.173.220 (24.6.173.220), Ds
⊞ Transmission Control Protocol, Src Port: 7439 (7439), Dst Port: h
  Source port: 7439 (7439)
  Destination port: http (80)
  [Stream index: 0]
```

Obtenir des statistiques de conversation TCP

1. Ouvrir `tr-tcp.pcapng`
2. **Statistiques > Conversations > onglet TCP**
3. Trier par colonne **Octets** (double-clic) — ordre décroissant
4. Clic droit sur l'entrée du haut → **Appliquer comme filtre > A ↔ B**

Colonnes utiles à ajouter

Index de flux TCP

Clic droit sur [Stream index: 0] → **Apply as Column**

TCP Delta Time

Clic droit sur "Time since previous frame in this TCP" → **Apply as Column**

Filtre pour détecter les retards (hors FIN/RST) :

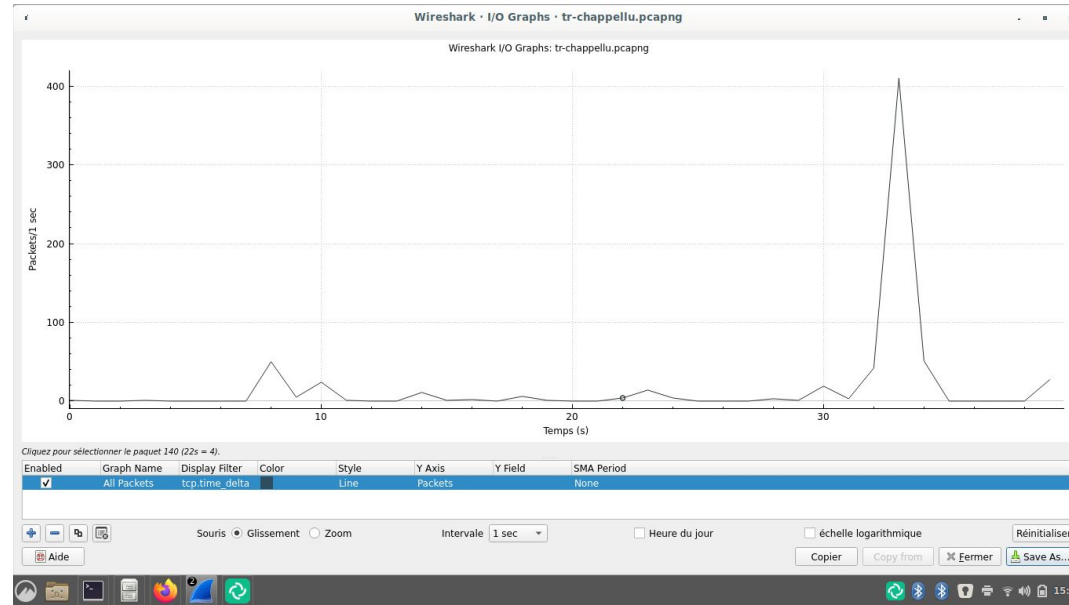
```
tcp.time_delta > 1 && tcp.flags.fin==0 && tcp.flags.reset==0
```

Sans les requêtes HTTP GET :

```
tcp.time_delta > 1 && tcp.flags.fin==0 && tcp.flags.reset==0 && !http.request.method=="GET"
```

Graph TCP Delays — I/O Graph

Statistiques > I/O Graph



Modifier :

- **Y field** → `tcp.time_delta`
- **Y axis** → `MAX(Y field)`

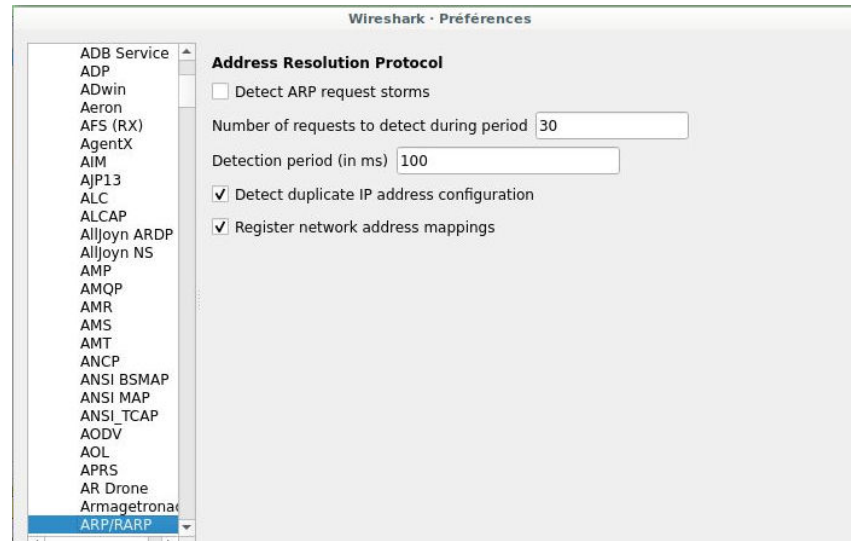
Détecter des adresses IP dupliquées

- Préférences du dissecteur **ARP/RARP** : détection d'IP dupliquées (activée par défaut)

Filtre d'affichage :

```
arp.duplicate-address-detected
```

Tester sur le fichier : `arp-poison.pcapng`



Identifier des problèmes de buffer saturés — Zero Window

Zero Window : l'application destinataire est saturée → réduit la fenêtre TCP à 0.

Causes possibles : trop de processus, CPU lent, buffer applicatif trop petit

Simulation :

```
wget -O /dev/null --limit-rate=1K http://test-debit.free.fr/16384.rnd
```

1. Serveur envoie **TCP Windows Full**
2. Client répond avec **Zero Window**
3. Transfert suspendu jusqu'au **Windows Update**

| No. | Time | Source | Destination | Protocol | Length | Window | Info |
|-----|-------------|---------------|---------------|----------|--------|------------------|--|
| 1 | 0.000000000 | 10.2.0.2 | 93.184.216.34 | TCP | 60 | 1152:40342 → 80 | [SYN] Seq=0 Win=1152 Len=0 MSS=1460 SACK_PERM=1 TSval=2629333636 TSecr=0 |
| 2 | 0.106969293 | 93.184.216.34 | 10.2.0.2 | TCP | 60 | 80 → 1152:40342 | [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2629333636 TSecr=0 |
| 3 | 0.107025590 | 10.2.0.2 | 93.184.216.34 | TCP | 52 | 1152 → 80 | [ACK] Seq=1 Ack=1 Win=1152 Len=0 TSval=2629333636 TSecr=3645 |
| 4 | 0.107192767 | 10.2.0.2 | 93.184.216.34 | HTTP | 176 | 1152 → 80 | GET / HTTP/1.1 |
| 5 | 0.350420754 | 93.184.216.34 | 10.2.0.2 | TCP | 52 | 80 → 1152:40342 | [ACK] Seq=1 Ack=125 Win=65280 Len=0 TSval=3645942832 TSecr=2 |
| 6 | 0.47625590 | 93.184.216.34 | 10.2.0.2 | TCP | 1204 | 255 → 1152:40342 | [TCP Window Full] 80 → 1152:40342 [PSH, ACK] Seq=1 Ack=125 Win=65280 Len=1152 |
| 7 | 0.47625590 | 10.2.0.2 | 93.184.216.34 | TCP | 52 | 0 → 1152:40342 | [TCP ZeroWindow] 40342 → 80 [ACK] Seq=125 Ack=1153 Win=0 Len=0 TSval=2629333636 TSecr=3645 |
| 8 | 0.47625590 | 10.2.0.2 | 93.184.216.34 | TCP | 52 | 1152 → 80 | [TCP Window Update] 40342 → 80 [ACK] Seq=125 Ack=1153 Win=1152 Len=0 TSval=2629333636 TSecr=3645 |
| 9 | 0.867332713 | 93.184.216.34 | 10.2.0.2 | HTTP | 491 | 255 → 1152:40342 | HTTP/1.1 200 OK (text/html) |
| 10 | 0.909783020 | 10.2.0.2 | 93.184.216.34 | TCP | 52 | 713 → 80 | [ACK] Seq=125 Ack=1592 Win=713 Len=0 TSval=2629337138 TSecr=3645 |
| 11 | 1.770751521 | 10.2.0.2 | 93.184.216.34 | TCP | 52 | 713 → 80 | [FIN, ACK] Seq=125 Ack=1592 Win=713 Len=0 TSval=2629337999 TSecr=3645 |
| 12 | 1.993340930 | 93.184.216.34 | 10.2.0.2 | TCP | 52 | 255 → 80 | [ACK] Seq=1592 Ack=126 Win=65280 Len=0 TSval=3645943723 TSecr=3645 |
| 13 | 2.040763320 | 93.184.216.34 | 10.2.0.2 | TCP | 52 | 255 → 80 | [FIN, ACK] Seq=1592 Ack=126 Win=65280 Len=0 TSval=3645943764 TSecr=3645 |
| 14 | 2.040814270 | 10.2.0.2 | 93.184.216.34 | TCP | 52 | 713 → 80 | [ACK] Seq=126 Ack=1593 Win=713 Len=0 TSval=2629338276 TSecr=3645 |
| 15 | 2.198416498 | 93.184.216.34 | 10.2.0.2 | TCP | 40 | 80 → 713 | [RST] Seq=1593 Win=0 Len=0 |

DHCP — Fonctionnement

- Permet aux clients d'obtenir leur configuration IP dynamiquement
- Basé sur **BOOTP**, utilise **UDP**
- Ports : **67** (serveur) / **68** (client)

DHCP Decline : si le serveur propose une IP déjà utilisée sur le réseau

1. Le client effectue un test ARP
2. Une machine répond → IP déjà prise
3. Le client envoie un **DHCP Decline**

Tester sur : `DHCP_decline.pcapng`

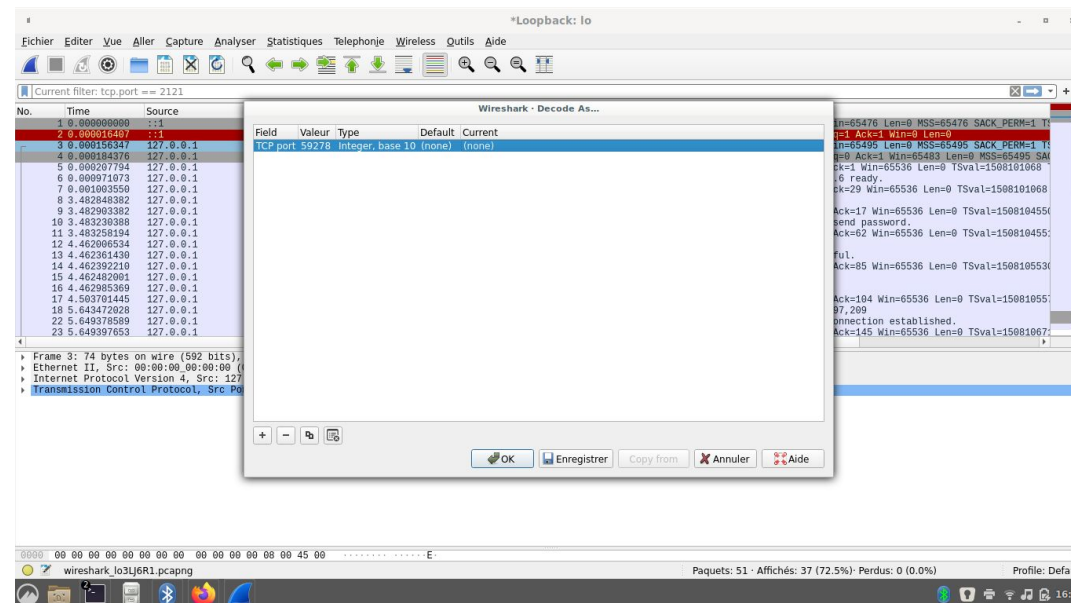
Chapitre 05 — Les tâches d'analyse de sécurité

Applications sur ports non-standard

Wireshark identifie les protocoles **par le numéro de port** (pas d'heuristique).

Pour décoder manuellement une trame :

1. Clic droit sur la trame → **Decode As**
2. Indiquer le protocole dans le champ **Current**



Exercice : ouvrir `unknown.pcap` et identifier le protocole utilisé.

Trafic suspect à surveiller

| Type | Description |
|------------------------|---|
| Scans MAC/IP | Identification des hôtes actifs |
| Scans de ports TCP/UDP | Identification des services |
| Mots de passe en clair | FTP, Telnet (détectables dans Packet Details) |
| Données en clair | HTTP, bases de données sur réseau non isolé |
| Brute force | Tentatives répétées de découverte de mot de passe |
| Paquets malformés | Exploitation de vulnérabilités applicatives |
| DoS | Trafic à très haut débit vers un ou plusieurs hôtes |
| ARP Poisoning | Tentatives d'attaque Man-in-the-Middle |

Scans — Détection de découverte d'hôte

| Technique | Filtre Wireshark | Commande |
|-------------------|--|---|
| Balayage ARP | <code>arp.dst.hw_mac==00:00:00:00:00:00</code> | <code>arp-scan -l</code> |
| Scan protocole IP | <code>icmp.type==3 and icmp.code==2</code> | <code>nmap -s0 <cible></code> |
| Ping ICMP | <code>icmp.type==8 or icmp.type==0</code> | <code>nmap -sn -PE <sous-réseau></code> |
| Ping TCP | <code>tcp.dstport==7</code> | <code>nmap -sn -PS/-PA <sous-réseau></code> |
| Ping UDP | <code>udp.dstport==7</code> | <code>nmap -sn -PU <sous-réseau></code> |

Scan ARP

```
arp.dst.hw_mac==00:00:00:00:00:00
```

***wlp4s0 (arp)**

Fichier Editer Vue Aller Capture Analyser Statistiques Telephonie Wireless Outils Aide

arp.dst.hw_mac==00:00:00:00:00:00

| No. | Time | Source | Destination | Protocol | Length | Window | Info |
|-----|--------------|-------------------|-------------|----------|--------|--------|--|
| 5 | 28.600901393 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.1? Tell 192.168.2.190 |
| 6 | 28.600990593 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.2? Tell 192.168.2.190 |
| 7 | 28.601049204 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.3? Tell 192.168.2.190 |
| 8 | 28.601110844 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.4? Tell 192.168.2.190 |
| 9 | 28.601155021 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.5? Tell 192.168.2.190 |
| 10 | 28.601202382 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.6? Tell 192.168.2.190 |
| 11 | 28.601246725 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.7? Tell 192.168.2.190 |
| 12 | 28.601307972 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.8? Tell 192.168.2.190 |
| 13 | 28.601370430 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.9? Tell 192.168.2.190 |
| 14 | 28.601445026 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.10? Tell 192.168.2.190 |
| 16 | 28.607027210 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.13? Tell 192.168.2.190 |
| 17 | 28.607103450 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.14? Tell 192.168.2.190 |
| 18 | 28.701077777 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.27? Tell 192.168.2.190 |
| 19 | 28.701377208 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.30? Tell 192.168.2.190 |
| 20 | 28.701438678 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.31? Tell 192.168.2.190 |
| 21 | 28.701489414 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.32? Tell 192.168.2.190 |
| 22 | 28.701553647 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.33? Tell 192.168.2.190 |
| 23 | 28.701798429 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.36? Tell 192.168.2.190 |
| 24 | 28.701857809 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.37? Tell 192.168.2.190 |
| 25 | 28.701905573 | IntelCor_5c:cf:bf | Broadcast | ARP | 42 | | Who has 192.168.2.38? Tell 192.168.2.190 |

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface wlp4s0, id 0
Ethernet II, Src: IntelCor_5c:cf:bf (f4:8c:50:5c:cf:bf), Dst: Fortinet_09:00:03 (00:09:0f:09:00:03)
Address Resolution Protocol (request)
Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: IntelCor_5c:cf:bf (f4:8c:50:5c:cf:bf)
Sender IP address: 192.168.2.190
Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
Target IP address: 192.168.2.254

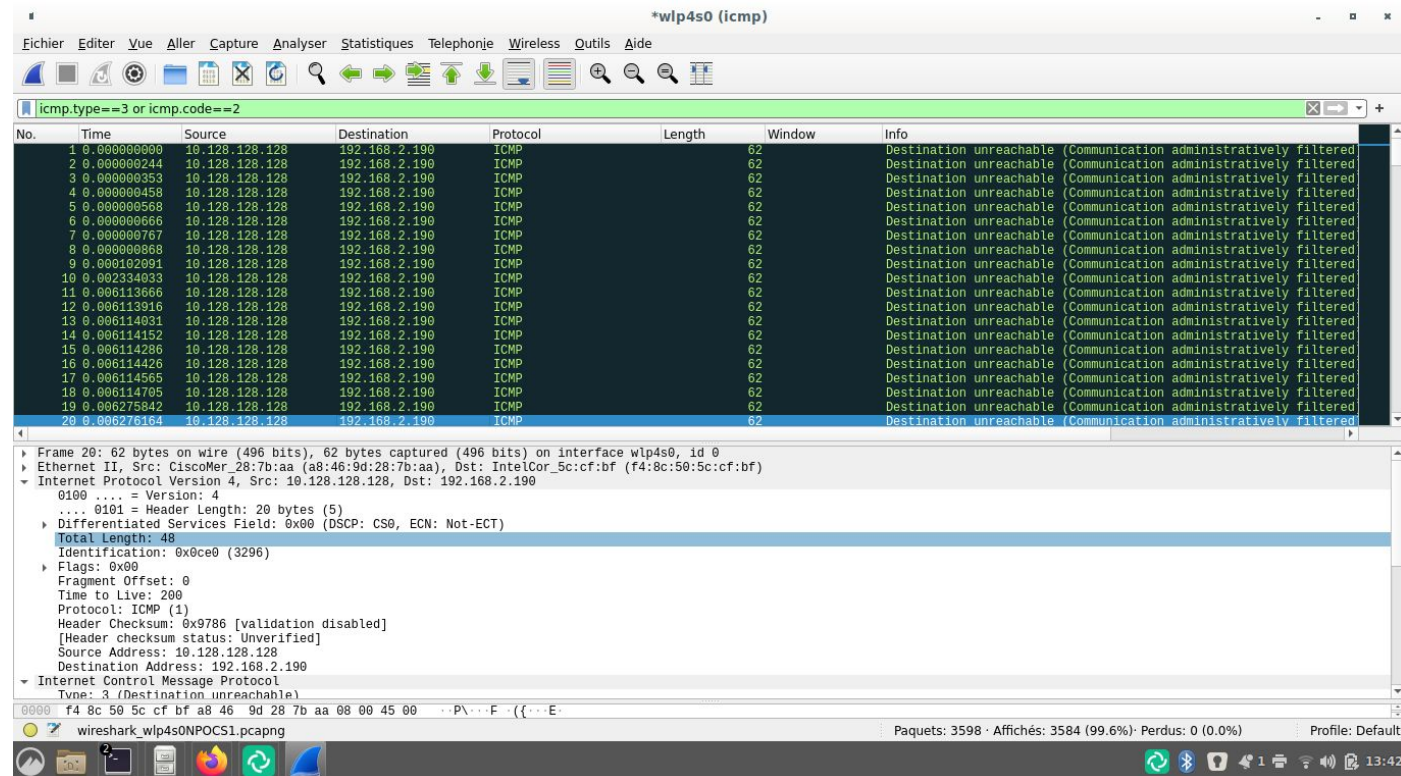
0000 00 09 0f 09 00 03 f4 8c 50 5c cf bf 08 06 00 01 P\.....

wireshark_wlp4s0P7UZR1.pcapng Paquets: 745 · Affichés: 728 (97.7%) · Perdus: 0 (0.0%) Profile: Default

Scan protocole IP

`icmp.type==3 or icmp.code==2`

Commande source : `nmap -s0 192.168.2.0/24`



Scan ping TCP

```
tcp.dstport==7
```

Commande source : `nmap -PS 192.168.2.0/24`

Wireshark capture of a TCP SYN scan on 192.168.2.0/24. The filter is `tcp.dstport == 7 and icmp`. The packet list shows multiple SYN packets from 192.168.2.190 to various destinations in the 192.168.2.0/24 range. The packet details for the selected packet (No. 30547) show a SYN flag set, sequence number 3298790753, and window size 1024.

| No. | Time | Source | Destination | Protocol | Length | Window | Info |
|-------|-------------|---------------|---------------|----------|--------|--------|---|
| 28722 | 5.267643464 | 192.168.2.190 | 192.168.2.35 | TCP | 58 | | 192.168.2.35 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 29974 | 5.333613177 | 192.168.2.190 | 192.168.2.11 | TCP | 58 | | 192.168.2.11 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30007 | 5.334337547 | 192.168.2.190 | 192.168.2.6 | TCP | 58 | | 192.168.2.6 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30088 | 5.337744495 | 192.168.2.190 | 192.168.2.14 | TCP | 58 | | 192.168.2.14 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30099 | 5.337938919 | 192.168.2.190 | 192.168.2.156 | TCP | 58 | | 192.168.2.156 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30100 | 5.337956521 | 192.168.2.190 | 192.168.2.157 | TCP | 58 | | 192.168.2.157 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30184 | 5.342430460 | 192.168.2.190 | 192.168.2.254 | TCP | 58 | | 192.168.2.254 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30200 | 5.342707381 | 192.168.2.190 | 192.168.2.216 | TCP | 58 | | 192.168.2.216 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30240 | 5.345862689 | 192.168.2.190 | 192.168.2.45 | TCP | 58 | | 192.168.2.45 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30246 | 5.345970071 | 192.168.2.190 | 192.168.2.154 | TCP | 58 | | 192.168.2.154 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30369 | 5.353087206 | 192.168.2.190 | 192.168.2.19 | TCP | 58 | | 192.168.2.19 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30407 | 5.353954415 | 192.168.2.190 | 192.168.2.112 | TCP | 58 | | 192.168.2.112 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30510 | 5.360166647 | 192.168.2.190 | 192.168.2.117 | TCP | 58 | | 192.168.2.117 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30511 | 5.360184776 | 192.168.2.190 | 192.168.2.138 | TCP | 58 | | 192.168.2.138 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30523 | 5.360392000 | 192.168.2.190 | 192.168.2.31 | TCP | 58 | | 192.168.2.31 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30546 | 5.360817347 | 192.168.2.190 | 192.168.2.141 | TCP | 58 | | 192.168.2.141 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |
| 30547 | 5.360834725 | 192.168.2.190 | 192.168.2.152 | TCP | 58 | | 192.168.2.152 → 7 [SYN] Seq=0 Win=1024 Len=0 MSS=1460 |

Frame 30547: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface wlp4s0, id 0

Ethernet II, Src: IntelCor_5c:cf:bf (f4:3c:50:5c:cf:bf), Dst: Huawei1e_86:80:ef (a8:7d:12:86:80:ef)

Internet Protocol Version 4, Src: 192.168.2.190, Dst: 192.168.2.152

Transmission Control Protocol, Src Port: 63328, Dst Port: 7, Seq: 0, Len: 0

Source Port: 63328

Destination Port: 7

[Stream index: 15051]

[TCP Segment Len: 0]

Sequence Number: 0 (relative sequence number)

Sequence Number (raw): 3298790753

[Next Sequence Number: 1 (relative sequence number)]

Acknowledgment Number: 0

Acknowledgment number (raw): 0

0110 = Header Length: 24 bytes (6)

Flags: 0x002 (SYN)

Window: 1024

[Calculated window size: 1024]

Checksum: 0xc416 (unverified)

0000 a8 7d 12 86 80 ef f4 3c 50 5c cf bf 08 00 45 00

Ethernet (eth), 14 byte(s)

Paquets: 34604 · Affichés: 17 (0.0%) · Perdus: 0 (0.0%) · Profile: Default

Détection des scans de ports

| Technique | Filtre Wireshark | Commande |
|---------------|--|-------------------------------------|
| TCP SYN scan | <code>tcp.flags.syn==1 and tcp.flags.ack==0 && tcp.window_size<=1024</code> | <code>nmap -sS <cible></code> |
| TCP Connect() | <code>tcp.flags.syn==1 and tcp.flags.ack==0 && tcp.window_size>1024</code> | <code>nmap -sT <cible></code> |
| TCP Null | <code>tcp.flags==0</code> | <code>nmap -sN <cible></code> |
| TCP FIN | <code>tcp.flags==0x001</code> | <code>nmap -sF <cible></code> |
| Scan UDP | <code>icmp.type==3 && icmp.code==3</code> | <code>nmap -sU <cible></code> |

Déchiffrement du trafic SSL/TLS

Trois méthodes disponibles dans Wireshark :

| Méthode | Fonctionne avec Perfect Forward Secrecy ? |
|--|---|
| RSA Keys list (clé privée du serveur) | ✗ Non |
| Pre-Shared Key | ✗ Non |
| (Pre)-Master-Secret log | ✓ Oui |

(Pre)-Master-Secret Log — Configuration

Variable d'environnement à configurer :

```
# Linux
export SSLKEYLOGFILE=/tmp/sslkeyfilelog

# Mac
launchctl setenv SSLKEYLOGFILE ~/Desktop/sslkeyfile.txt

# Windows
set SSLKEYLOGFILE=c:\sslkeyfile.txt
"C:\Program Files (x86)\Mozilla Firefox\firefox.exe"
```

(Pre)-Master-Secret Log — Wireshark

Dans Wireshark :

Préférences > Protocoles > TLS

→ Indiquer le chemin dans **(Pre)-Master-Secret log filename**

Wireshark affichera alors :

- La version chiffrée de la trame
- La version **déchiffrée** dans l'onglet **Decrypted TLS**