Fonctionnement de base des réseaux

Adressage
Routage
Noms de domaine
Configuration automatique

Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

Fonctionnement des réseaux

- **Propos**
  - Rappeler les élements de base du fonctionnement des réseaux *TCP/IP*
    - Déjà abordés dans le module *RX*
  - Étudier leur mise en œuvre pratique
    - Adressage statique
    - Routage
  - Étudier les services élémentaires
    - Attribution/résolution de noms de domaine : *DNS*
    - Configuration automatique : *DHCP*

La couche liens de *TCP/IP*

- **Le protocole** *MAC* **(***Medium Access Control***)**
  - Chaque dispositif dispose d'une adresse *MAC*
    - Unique, fixée en dur par le fabricant
    - Code fabriquant (3 octets), numéro de carte (3 octets) (*Organizationally Unique Identifier*, *IEEE*, fichier `OUI.txt`)
    - Généralement exprimée en hexa (ex : `00:11:2F:0B:A6:73`)
  - Un *medium* commun reliant tous les dispositifs
    - Interlocuteurs identifiés par leurs adresses
    - Tous joignables directement

La couche liens de *TCP/IP*

- **Émission d'une trame**
  - Adresse *MAC* destinataire fournie
  - Inscription de l'adresse *MAC* de la carte comme source (ou forcée par une *raw-socket*)
  - Écriture sur le *medium* commun
- **Réception d'une trame**
  - Lecture sur le *medium* commun
  - Vérification de l'adresse *MAC* destination
    - Adresse de la carte qui écoute ?
    - Adresse de diffusion (`FF:FF:FF:FF:FF:FF`) ?
    - Ignorée sinon (sauf si mode *promiscuous*)

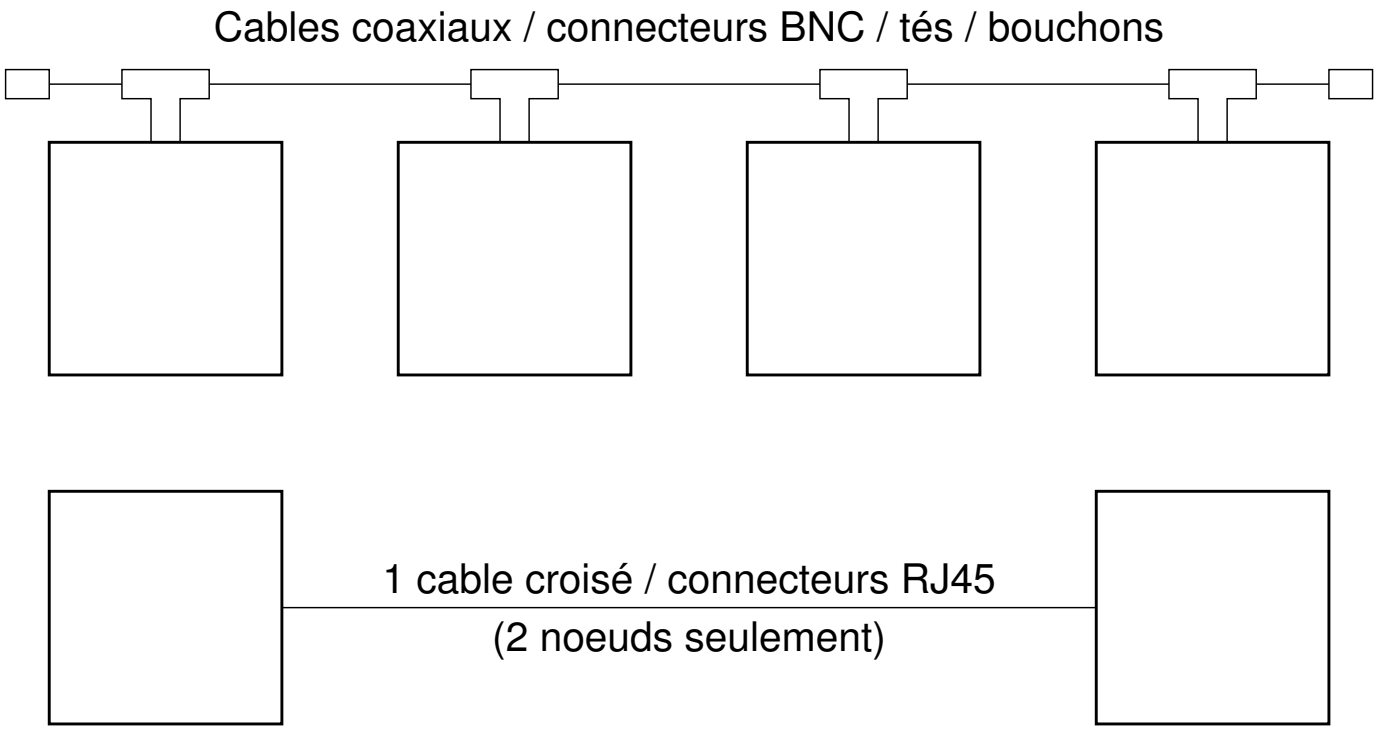
La couche liens de *TCP/IP*

▷ Échanges de trames

- ◇ Les machines d'un même *brin* peuvent communiquer
- ◇ Allonger les *brins* avec des répéteurs
  - ○ Simple remise en forme du signal
- ◇ Relier plusieurs *brins* par un concentrateur (*hub*)
  - ○ Ce qui arrive sur un *port* est répété sur tous les autres
  - ○ Le trafic monopolise tous les *brins* !
- ◇ Relier plusieurs *brins* par un commutateur (*switch*)
  - ○ Il maintient une table (adresse $MAC/port$)
  - ○ Adresse source $\rightarrow$ mise à jour de la table
  - ○ Adresse destination $\rightarrow$ choix du *port* d'émission
  - ○ Cloisonnement : pas de trafic sur tous les *brins*

La couche liens de *TCP/IP*

▷ **Liaison par brin unique (ex : *Ethernet*)**



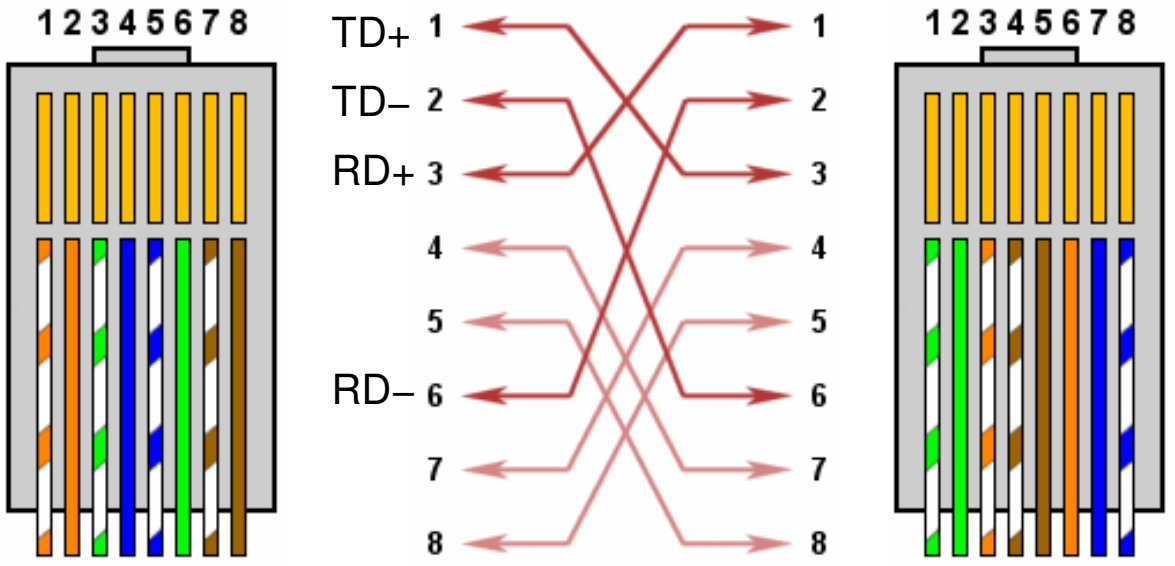
La couche liens de *TCP/IP*

Connectique BNC (cable coaxial)

Connectique RJ45 (paires torsadées)

La couche liens de *TCP/IP*

▷ Cablage Ethernet/RJ45

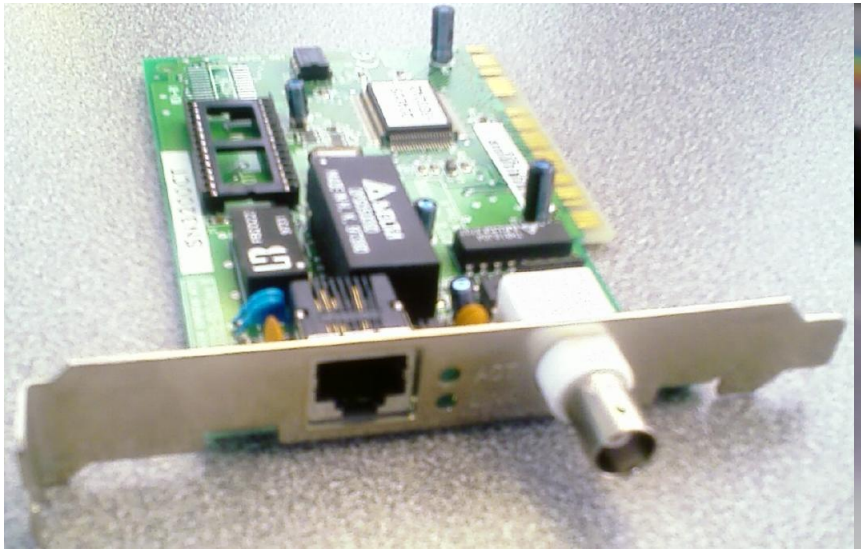


Cablage droit
(aux 2 extrémités)

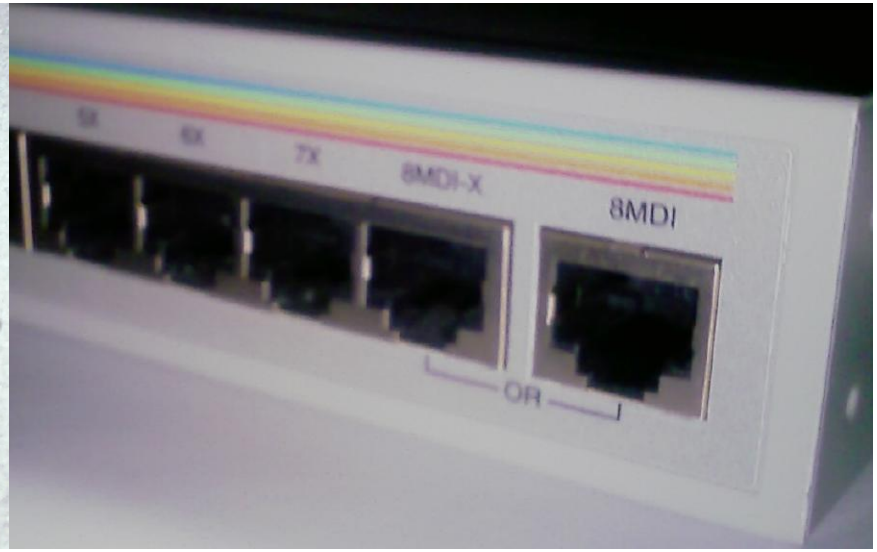
Cablage croisé
(à 1 extrémité)

La couche liens de *TCP/IP*

▷ Connecteurs des cartes/hubs/switches Ethernet



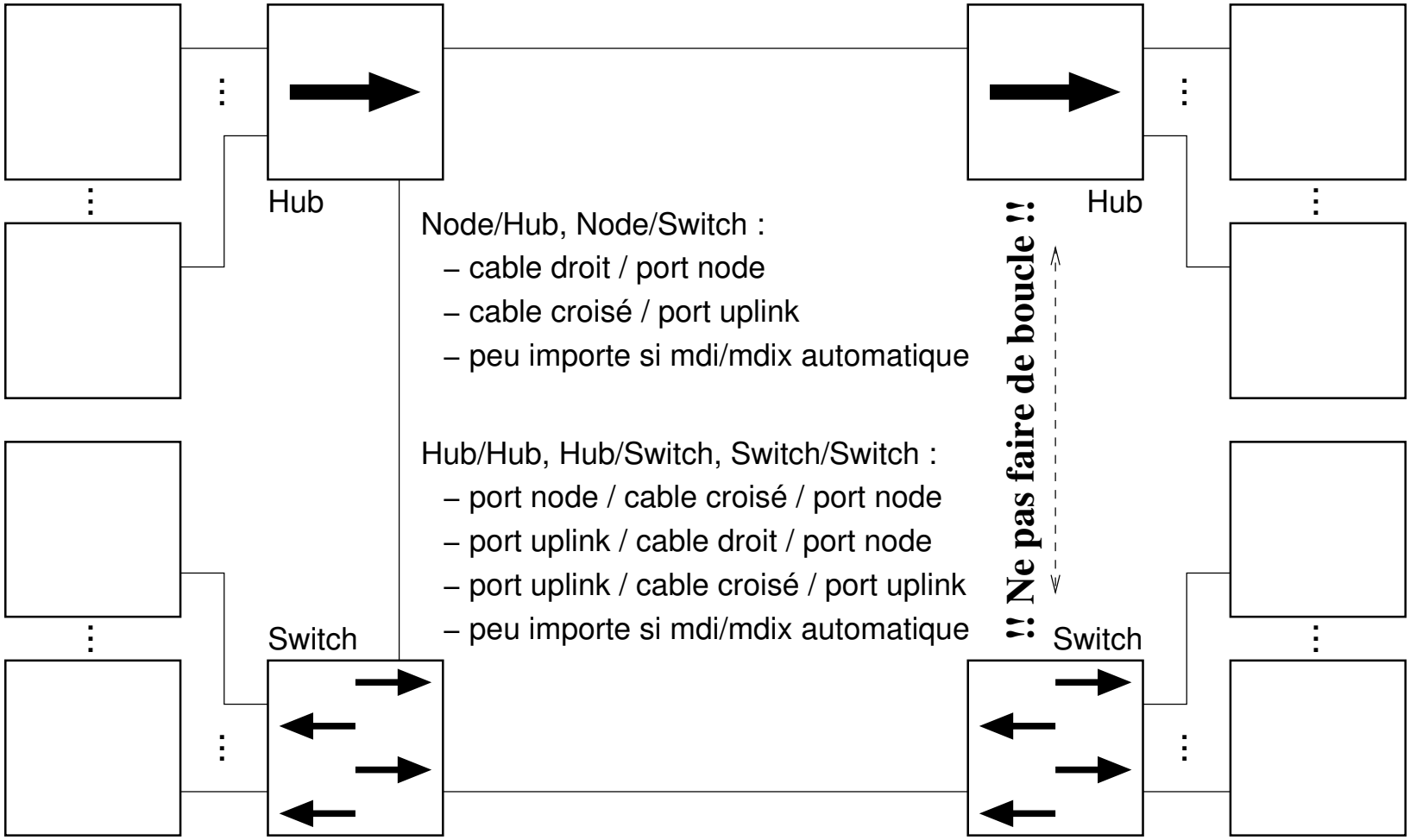
Carte réseau Ethernet
1 connecteur RJ45 + 1 connecteur BNC



Hub/Switch Ethernet à 8 ports
port 8 node (mdix) ou uplink (mdi)

La couche liens de *TCP/IP*

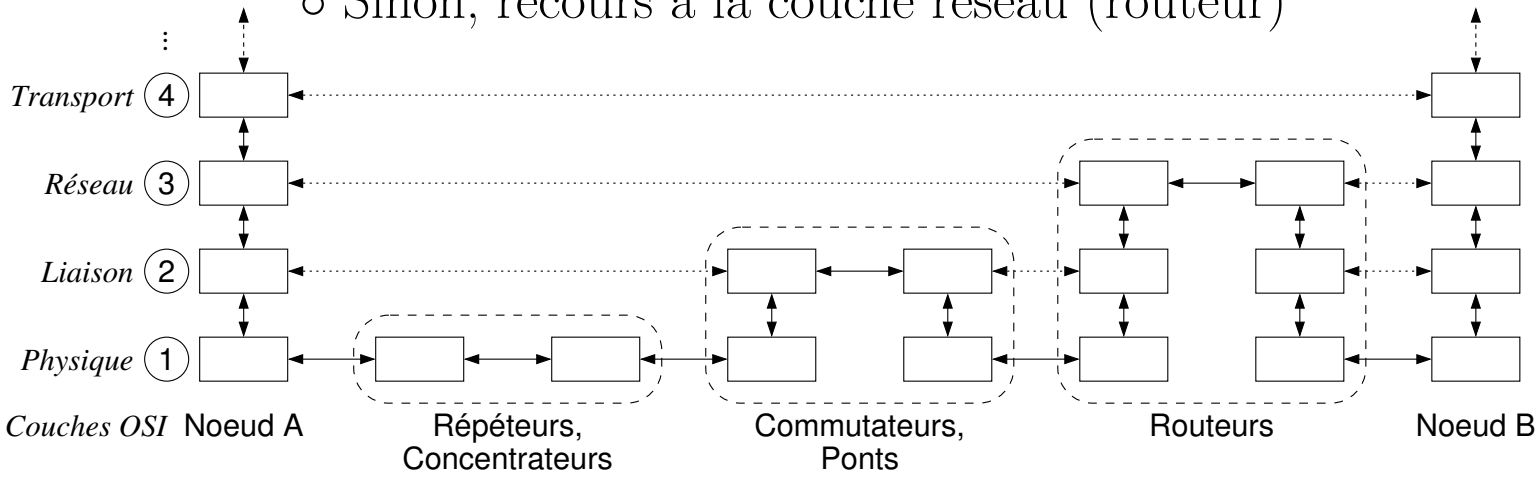
▷ Liaison par plusieurs brins (ex : *Ethernet/RJ45*)



La couche liens de *TCP/IP*

▷ Échanges de trames entre réseaux hétérogènes

- ◇ Utilisation d'un *pont* (*bridge*)
  - ○ Machine disposant de plusieurs types d'interfaces
- ◇ Les protocoles doivent être compatibles
  - ○ L'information déterminante est l'adresse *MAC*
  - ○ Exemple : *Ethernet*, *Token Ring* et *WiFi*
  - ○ Sinon, recours à la couche réseau (routeur)



La couche réseau de *TCP/IP*

- **Apports du protocole *IP* (*Internet Protocol*)**
  - Fragmentation et réassemblage des datagrammes en paquets
    - Respect des *MTU* (*Maximum Transmission Unit*) des couches liens
  - Abstraction de l'identification des nœuds par adresse *IP*
    - Attribution plus souple que les adresses *MAC*
    - Permet une segmentation logique des réseaux (tous les nœuds n'interagissent pas directement !)
  - Routage des paquets
    - Atteindre un nœud distant, malgré la segmentation
    - Pas de connaissance *a priori* du chemin
    - Repose sur la structure logique choisie

La couche réseau de *TCP/IP*

▷ L'adresse *IP*

- ◇ Attribuée à une interface réseau d'un nœud
  → plusieurs adresses *IP* si plusieurs interfaces
- ◇ 4 octets, notation décimale pointée (*IPv4*) (ex : `192.168.20.236`)
- ◇ Associée à un masque de sous-réseau (ex : `255.255.240.0`)
- ◇ (*IP*`&`masque) → adresse de réseau (ex : `192.168.16.0`)
- ◇ (*IP*`|~`masque) → adresse de diffusion (ex : `192.168.31.255`)
- ◇ Désignation usuelle d'un réseau :
  → adresse/largeur de masque (ex : `192.168.16.0/20`)

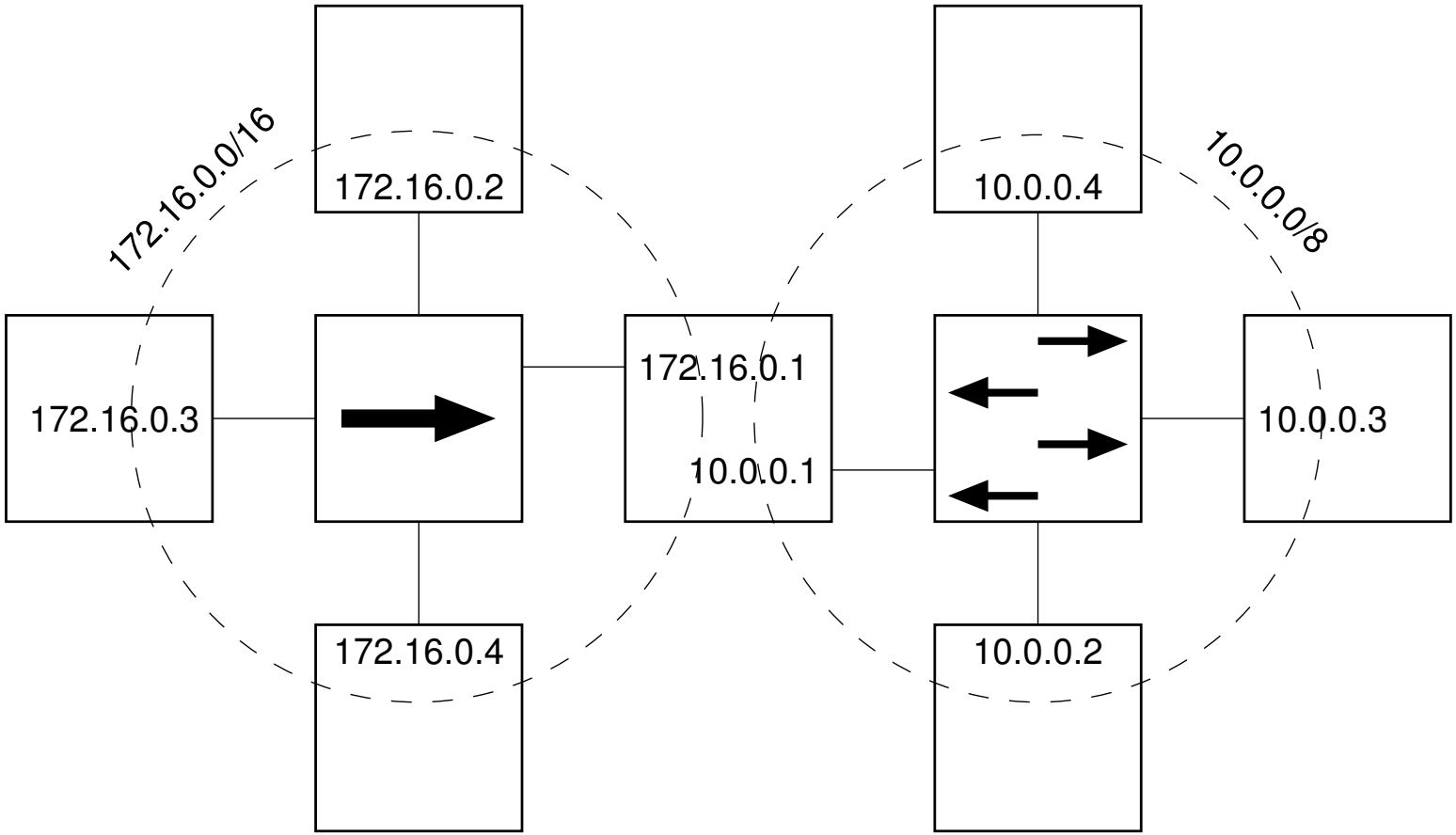
Adresse IP	*192* 11000000	*168* 10101000	*20* 00010100	*236* 11101100
Masque de sous–réseau	*255* 11111111	*255* 11111111	*240* 11110000	*0* 00000000
Adresse de réseau	*192* 11000000	*168* 10101000	*16* 00010000	*0* 00000000
Adresse de diffusion	*192* 11000000	*168* 10101000	*31* 00011111	*255* 11111111

La couche réseau de *TCP/IP*

- **Notion de domaine de diffusion**
  - Un ensemble de nœuds reliés par *hub/switch*
    - Interaction directe au niveau liens (MAC)
    - Tous atteints par une trame diffusée (`FF:FF:FF:FF:FF:FF`)
  - Tous dans le même sous-réseau (adresse+masque)
    - L'adresse IP de chaque nœud est dans ce sous-réseau
    - Interaction directe entre les nœuds du sous-réseau
      (paquet IP destiné à un nœud $\rightarrow$ trame à destination de ce nœud)
    - Tous atteints par un paquet IP diffusé
      (adresse IP de diffusion du sous-réseau $\rightarrow$ trame diffusée)

La couche réseau de *TCP/IP*

▷ Exemple de domaines de diffusion (1/2)



La couche réseau de *TCP/IP*

▷ Exemple de domaines de diffusion (2/2)

- ◇ Point-à-point (moyennant la connaissance des adresses MAC)
  - ○ `172.16.`X`.`X $\rightarrow$ `172.16.`X`.`X
  - ○ `10.`X`.`X`.`X $\rightarrow$ `10.`X`.`X`.`X
- ◇ Diffusions (utilisation de l'adresse MAC de diffusion)
  - ○ `172.16.`X`.`X $\rightarrow$ `172.16.255.255`
  - ○ `10.`X`.`X`.`X $\rightarrow$ `10.255.255.255`
- ◇ Le reste nécessite du routage
  - ○ Les trames sont cloisonnées dans leur domaine de diffusion !
  - ○ Aucune communication implicite `172.16.`X`.`X $\leftrightarrow$ `10.`X`.`X`.`X

La couche réseau de *TCP/IP*

- **Choix des adresses *IP* et du masque de sous-réseau**
  - On n'attribue pas d'adresse à un objet matérialisant le sous-réseau !
    - Il ne s'agit que de connectique (cables, *hubs*, *switches*)
  - On attribue les adresses/masque aux nœuds du domaine de diffusion
    - C'est la cohérence de cette démarche qui détermine le sous-réseau
  - Attribuer le même masque pour chaque nœud
  - Le calcul *IP*&masque doit donner la même valeur pour chaque nœud
    - C'est là qu'apparaît l'adresse du sous-réseau
  - Idem pour l'adresse de diffusion : *IP*|~masque
  - Très peu de chances de communiquer en cas d'attribution arbitraire
    - Se renseigner avant l'introduction d'un nouveau nœud

La couche réseau de *TCP/IP*

▷ **Adresses disponibles dans un sous-réseau**

- ◇ Sous-réseau avec un masque de largeur `N`
  - ○ Les `N` premiers bits des adresses sont fixes
  - ○ Les `32-N` derniers bits des adresses sont variables
- ◇ Dans la partie variable, deux combinaisons de bits sont réservées
  - ○ Tous ces bits à `0` : adresse du sous-réseau
  - ○ Tous ces bits à `1` : adresse de diffusion du réseau
- ◇ On peut donc attribuer $2^{32-N} - 2$ adresses
- ◇ ex : soit le sous-réseau `172.16.0.0/16`
  - ○ Adresse de diffusion : `172.16.255.255`
  - ○ 65534 adresses attribuables : de `172.16.0.1` à `172.16.255.254`
- ◇ ex : et pour `192.168.1.16/30` ? (à compléter)

La couche réseau de *TCP/IP*

▷ Configuration d'une interface réseau

```
# ifconfig eth0 0.0.0.0 down
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:15:C5:C1:BD:D2
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:18

# ifconfig eth0 172.16.1.7 netmask 255.255.255.0
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:15:C5:C1:BD:D2
          inet addr:172.16.1.7  Bcast:172.16.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:18
```

La couche réseau de *TCP/IP*

▷ **Les classes d'adresses** *IP*

- ◇ Classe A : premier bit à `0` (`0.X.X.X` à `127.X.X.X`)
  - ○ Masque de sous-réseau implicite : `255.0.0.0`
  - ○ Au maximum 16777214 adresses *IP* distinctes
- ◇ Classe B : deux premiers bits à `10` (`128.0.X.X` à `191.255.X.X`)
  - ○ Masque de sous-réseau implicite : `255.255.0.0`
  - ○ Au maximum 65534 adresses *IP* distinctes
- ◇ Classe C : trois premiers bits à `110` (`192.0.0.X` à `239.255.255.X`)
  - ○ Masque de sous-réseau implicite : `255.255.255.0`
  - ○ Au maximum 254 adresses *IP* distinctes
- ◇ Classe D : quatre premiers bits à `1110` : diffusion restreinte (*multicast*)
- ◇ Classe E : cinq premiers bits à `11110` : usage réservé
- ◇ nb : masques implicites des classes A, B et C par simple convention

La couche réseau de *TCP/IP*

▷ Quelques adresses particulières

- ◇ `0.0.0.0` : normalement inutilisée
  - ○ Sert d'adresse source lors de l'auto-configuration ($DHCP/BOOTP$)
  - ○ Aucune adresse n'est encore attribuée au nœud ...
- ◇ `255.255.255.255` : adresse générique de diffusion
  - ○ Sert de destination lors de l'auto-configuration ($DHCP/BOOTP$)
  - ○ L'adresse de diffusion du sous-réseau n'est pas encore connue ...
  - ○ Utilisable toutefois dans d'autres circonstances
    → Ambigu si plusieurs interfaces → à éviter
    → Préférer l'adresse de diffusion d'un sous-réseau spécifique
- ◇ `127.0.0.0/8` : interface de rebouclage
  - ○ `127.0.0.1` ≡ `localhost`
  - ○ Associée au périphérique virtuel *loopback*

La couche réseau de *TCP/IP*

- **Relation adresse** $IP \rightarrow$ **adresse** MAC
  - Pertinent uniquement **à l'intérieur du domaine de diffusion** !!!
  - Protocole ARP (*Address Resolution Protocol*)
    - Au dessous d'IP mais étroitement lié, proto.$Ethernet$=`0x0806`
  - Les nœuds d'un réseau ont tous une table de résolution ARP
    - Paires (adresse IP, adresse MAC)
    - Mise à jour dynamique + *"oubli"* périodique
    - Configuration statique à la demande

La couche réseau de *TCP/IP*

- **Recherche de l'adresse MAC de destination**
  - Paquet IP diffusé ? $\rightarrow$ destination $MAC =$ `FF:FF:FF:FF:FF:FF`
  - Destination IP dans la table ARP ? $\rightarrow$ destination MAC connue
  - Envoi d'une requête ARP pour la destination IP
    - Source $MAC =$ adresse de l'interface d'émission
    - Destination $MAC =$ `FF:FF:FF:FF:FF:FF`
  - Réponse du nœud concerné $\rightarrow$ mise à jour de la table ARP et envoi
  - Si pas de réponse après un délai et plusieurs tentatives $\rightarrow$ Échec !

La couche réseau de *TCP/IP*

▷ **Algorithme de mise à jour de la table *ARP***

◇ Algorithme décrit dans la *RFC-0826*

○ Réception requête → insertion dans la table
(évite une nouvelle requête très probable dans l'autre direction)

○ Réception réponse → insertion inconditionnelle dans la table !
(alors qu'on n'a pas fait de requête !)

○ Souvent, en pratique, une variante un peu plus robuste décrite ici

◇ Trame *ARP* reçue : `[operation, srcMac, srcIp, dstMac, dstIp]`

◇ Adresses de l'interface réseau : `[localMac, localIp]`

```
- if [srcIp] is already in the ARP cache
  | - update this entry with [srcMac]
- if [dstIp] is [localIp] and [operation] is a request
  | - insert (if not done) a new entry for [srcMac, srcIp] in the ARP cache
  | - send an ARP [reply, localMac, localIp, srcMac, srcIp]
  |   through the receiving interface
```

La couche réseau de *TCP/IP*

▷ Exemple : mise en évidence du trafic *ARP*

```
                                $ ping -n 172.16.1.8
                                PING 172.16.1.8 (172.16.1.8) 56(84) bytes of data.
                                64 bytes from 172.16.1.8: icmp_seq=1 ttl=64 time=0.183 ms
                                64 bytes from 172.16.1.8: icmp_seq=2 ttl=64 time=0.192 ms
# tcpdump -i eth1 -en
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
15:47:50.271651 00:15:c5:cd:db:d4 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
  length 60: arp who-has 172.16.1.8 tell 172.16.1.2
15:47:50.271676 00:04:75:da:69:c1 > 00:15:c5:cd:db:d4, ethertype ARP (0x0806),
  length 42: arp reply 172.16.1.8 is-at 00:04:75:da:69:c1
15:47:50.271776 00:15:c5:cd:db:d4 > 00:04:75:da:69:c1, ethertype IPv4 (0x0800),
  length 98: 172.16.1.2 > 172.16.1.8: ICMP echo request, id 20487, seq 1, length 64
15:47:50.271837 00:04:75:da:69:c1 > 00:15:c5:cd:db:d4, ethertype IPv4 (0x0800),
  length 98: 172.16.1.8 > 172.16.1.2: ICMP echo reply, id 20487, seq 1, length 64
15:47:51.271634 00:15:c5:cd:db:d4 > 00:04:75:da:69:c1, ethertype IPv4 (0x0800),
  length 98: 172.16.1.2 > 172.16.1.8: ICMP echo request, id 20487, seq 2, length 64
15:47:51.271704 00:04:75:da:69:c1 > 00:15:c5:cd:db:d4, ethertype IPv4 (0x0800),
  length 98: 172.16.1.8 > 172.16.1.2: ICMP echo reply, id 20487, seq 2, length 64
```

La couche réseau de *TCP/IP*

▷ Bilan intermédiaire

- ◇ Permet d'abstraire l'identification des nœuds
  - ○ Adresses *IP* attribuables librement
  - ○ À condition de respecter quelques précautions (masque, adresse de sous-réseau, adresse de diffusion)
- ◇ Permet de communiquer au sein d'un domaine de diffusion
  - ○ En *point-à-point* et par diffusion
  - ○ Repose directement sur les adresses *MAC* de la couche liens (recours indispensable au protocole *ARP*)
- ◇ Il reste à voir comment aller au delà du domaine de diffusion
  - ○ Notion de routage
  - ○ Utilisation de nœuds intermédiaires : les routeurs

La couche réseau de *TCP/IP*

- **Les routeurs**
  - Nœud ayant plusieurs interfaces réseau
  - Chacune d'elles est configurée dans un sous-réseau distinct
  - Le nœud peut passer les paquets d'une interface à une autre
    - Uniquement les paquets qui ne le concernent pas
    - Selon ce qu'indique sa table de routage
    - Ceci doit être autorisé (ce n'est pas implicite)
  - Les paquets *IP* passent ainsi dans un autre domaine de diffusion
    - Le procédé peut être répété de sous-réseau en sous-réseau
    - C'est ce qui permet de joindre un nœud sur *Internet*

La couche réseau de *TCP/IP*

- **La table de routage**
  - Chaque nœud en possède une (même rudimentaire)
  - Indique comment atteindre des adresses/masques
  - Les entrées sont classées
    - De la plus spécifique : l'adresse d'un nœud (masque de 32 bits)
    - À la plus générale : la route par défaut (masque de 0 bit)
  - À chaque entrée (adresse/masque) correspond :
    - Une interface si elle est dans le sous-réseau désigné (implicite)
    - L'adresse *IP* d'une passerelle (routeur) sinon (explicite)
      $\rightarrow$ doit pouvoir être atteinte directement !

La couche réseau de *TCP/IP*

▷ Suite de l'exemple de la page 15

◇ Table de routage initiale d'un nœud de `10.0.0.0/8`

```
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

◇ Table de routage initiale d'un nœud de `172.16.0.0/16`

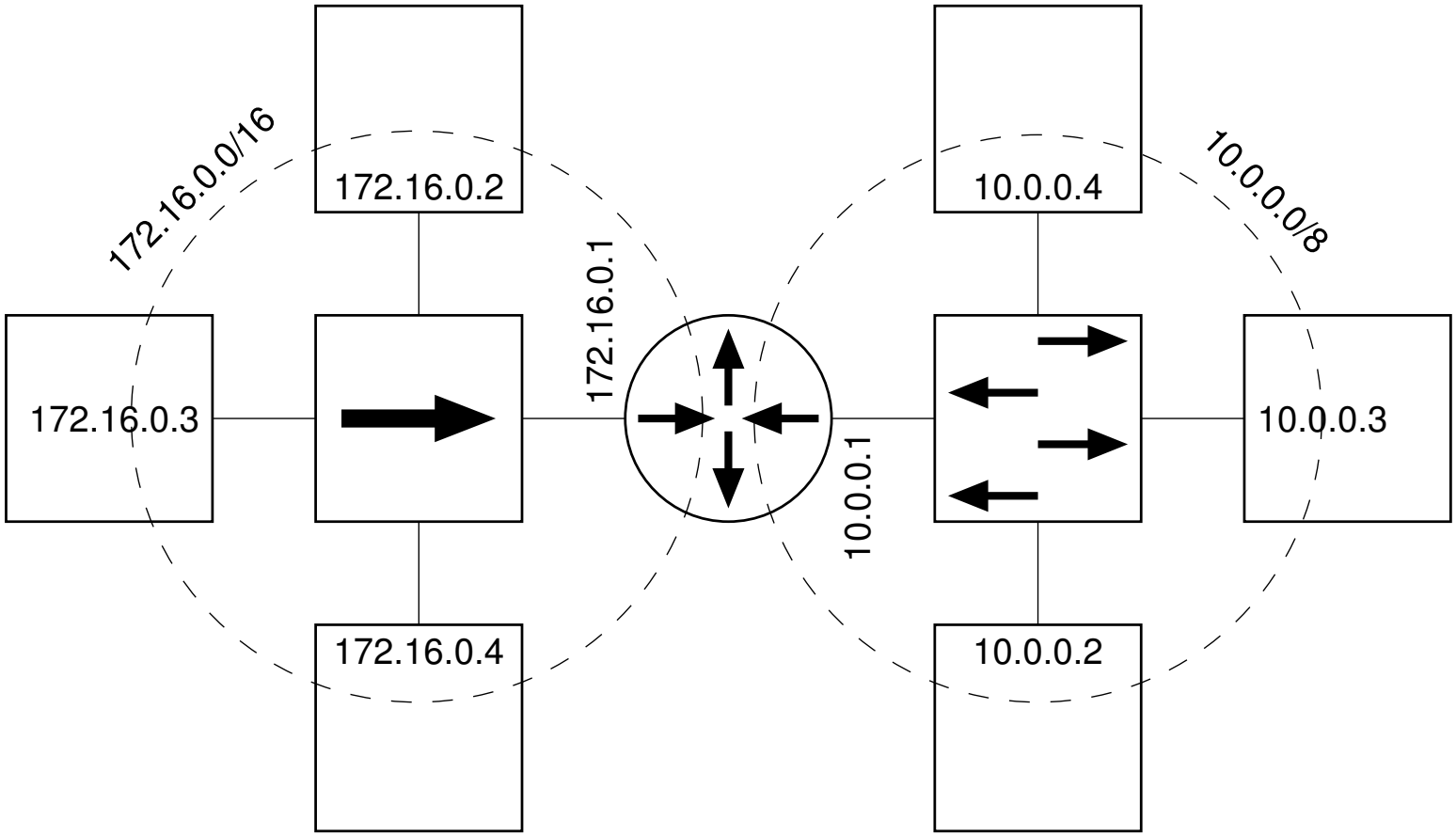
```
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

◇ Table de routage initiale du nœud intermédiaire (routeur)

```
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth1
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

La couche réseau de *TCP/IP*

▷ Routage entre domaines de diffusion (1/5)



La couche réseau de *TCP/IP*

▷ Routage entre domaines de diffusion (2/5)

◇ Table de routage complétée d'un nœud de `10.0.0.0/8`

```
# route add -net 172.16.0.0/16 gw 10.0.0.1
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.0.0      10.0.0.1        255.255.0.0     UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

◇ Table de routage complétée d'un nœud de `172.16.0.0/16`

```
# route add -net 10.0.0.0/8 gw 172.16.0.1
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
10.0.0.0        172.16.0.1      255.0.0.0       UG    0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

◇ Maintenant, une communication `172.16.`*X.X* $\leftrightarrow$ `10.`*X.X.X* est possible

○ Le routeur doit l'autoriser (`sysctl -w net.ipv4.ip_forward=1`)

La couche réseau de *TCP/IP*

▷ Routage entre domaines de diffusion (3/5)

◇ `ping` de `10.0.0.3` vers `172.16.0.3` vu depuis `10.0.0.1`

```
# tcpdump -i eth0 -en
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:35:49.336407 de:ad:be:ef:00:31 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
  length 60: arp who-has 10.0.0.1 tell 10.0.0.3
19:35:49.337488 de:ad:be:ef:00:22 > de:ad:be:ef:00:31, ethertype ARP (0x0806),
  length 42: arp reply 10.0.0.1 is-at de:ad:be:ef:00:22
19:35:49.454389 de:ad:be:ef:00:31 > de:ad:be:ef:00:22, ethertype IPv4 (0x0800),
  length 98: 10.0.0.3 > 172.16.0.3: ICMP echo request, id 28676, seq 1, length 64
19:35:49.618514 de:ad:be:ef:00:22 > de:ad:be:ef:00:31, ethertype IPv4 (0x0800),
  length 98: 172.16.0.3 > 10.0.0.3: ICMP echo reply, id 28676, seq 1, length 64
19:35:50.335239 de:ad:be:ef:00:31 > de:ad:be:ef:00:22, ethertype IPv4 (0x0800),
  length 98: 10.0.0.3 > 172.16.0.3: ICMP echo request, id 28676, seq 2, length 64
19:35:50.375435 de:ad:be:ef:00:22 > de:ad:be:ef:00:31, ethertype IPv4 (0x0800),
  length 98: 172.16.0.3 > 10.0.0.3: ICMP echo reply, id 28676, seq 2, length 64
```

La couche réseau de *TCP/IP*

▷ Routage entre domaines de diffusion (4/5)

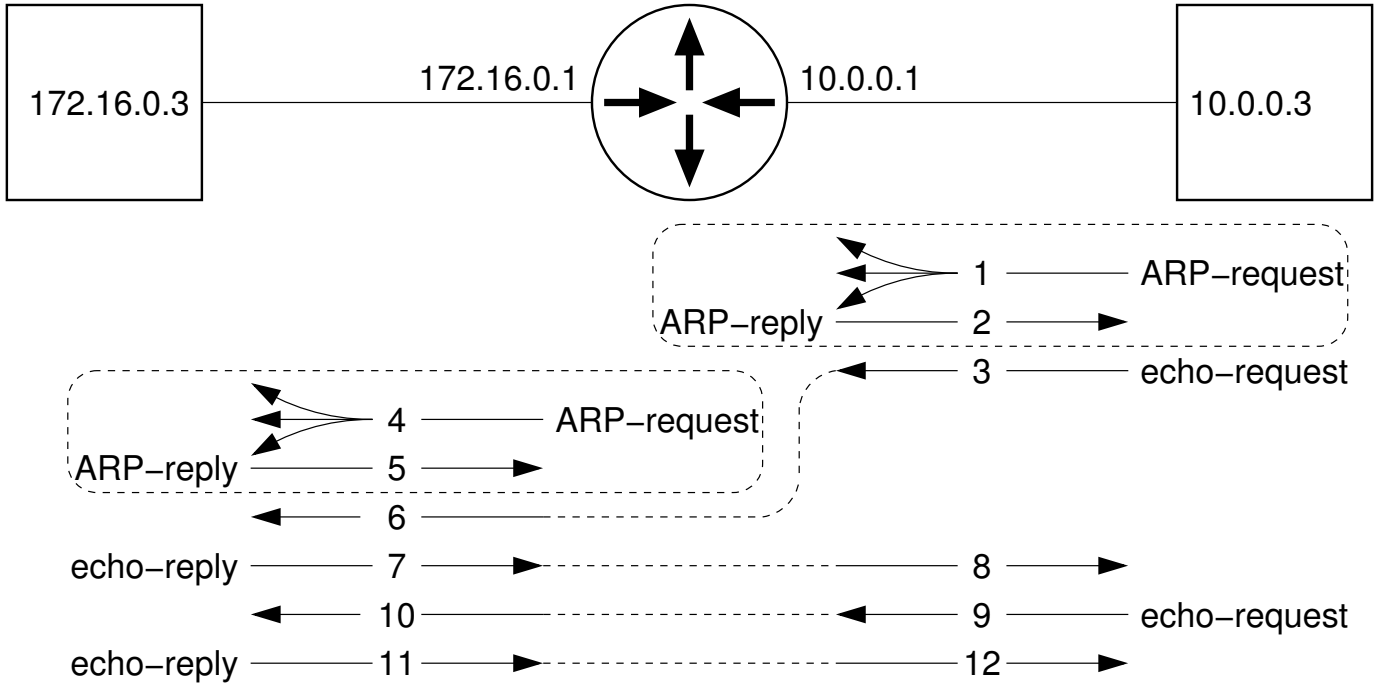
◇ `ping` de `10.0.0.3` vers `172.16.0.3` vu depuis `172.16.0.1`

```
# tcpdump -i eth1 -en
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
19:35:49.458253 de:ad:be:ef:00:21 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806),
  length 42: arp who-has 172.16.0.3 tell 172.16.0.1
19:35:49.538363 de:ad:be:ef:00:11 > de:ad:be:ef:00:21, ethertype ARP (0x0806),
  length 60: arp reply 172.16.0.3 is-at de:ad:be:ef:00:11
19:35:49.538548 de:ad:be:ef:00:21 > de:ad:be:ef:00:11, ethertype IPv4 (0x0800),
  length 98: 10.0.0.3 > 172.16.0.3: ICMP echo request, id 28676, seq 1, length 64
19:35:49.618348 de:ad:be:ef:00:11 > de:ad:be:ef:00:21, ethertype IPv4 (0x0800),
  length 98: 172.16.0.3 > 10.0.0.3: ICMP echo reply, id 28676, seq 1, length 64
19:35:50.335450 de:ad:be:ef:00:21 > de:ad:be:ef:00:11, ethertype IPv4 (0x0800),
  length 98: 10.0.0.3 > 172.16.0.3: ICMP echo request, id 28676, seq 2, length 64
19:35:50.375222 de:ad:be:ef:00:11 > de:ad:be:ef:00:21, ethertype IPv4 (0x0800),
  length 98: 172.16.0.3 > 10.0.0.3: ICMP echo reply, id 28676, seq 2, length 64
```

La couche réseau de *TCP/IP*

▷ Routage entre domaines de diffusion (5/5)

- ◇ Mise en évidence des requêtes ARP dans le routage
- ◇ Destination MAC à rechercher ?
  - ○ Consultation de la table de routage : cible directe ou routeur



La couche réseau de *TCP/IP*

- *CIDR* (*Classless InterDomain Routing*)
  - Préciser explicitement un masque de sous-réseaux
  - Les classes A, B, C proposent un découpage *"grossier"*
    - Peut conduire à un *"gaspillage"* d'adresses
      (ex : passer d'une classe C à B alors qu'un bit supplémentaire suffirait)
      (ex : plusieurs classes C alors qu'on pourrait en subdiviser une seule)
  - *Supernetting* : agréger les entrées des tables de routage
    ex : `192.168.20.0` et `192.168.21.0` → `192.168.0.0/16`
  - *Subnetting* : découper en sous-réseaux internes (filtrage, administration)
    ex : `172.16.0.0` → `172.16.10.0/24` et `172.16.20.0/24`
    (découper une classe B en 256 sous-réseaux de 254 nœuds)
  - Nécessite de visualiser les masques en binaire
  - C'est ce qui est utilisé maintenant au niveau d'*Internet*
    IANA (monde) → *RIR* (regions/continents) → *FAI* ...

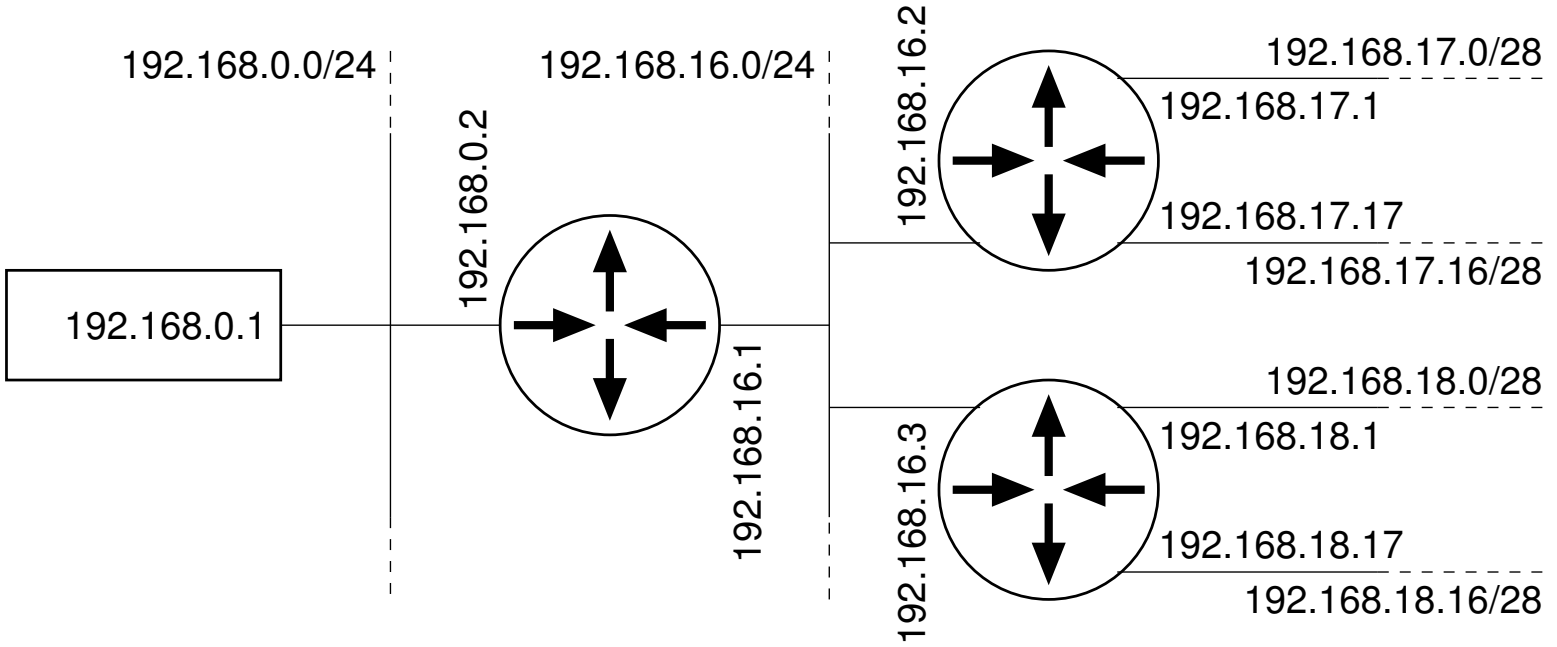
La couche réseau de *TCP/IP*

▷ **Exemple d'utilisation de** *CIDR*

◇ Les classes C `192.168.17.0` et `192.168.18.0` sont subdivisées en `/28`

○ Petits sous-réseaux de moins de 14 nœuds

◇ Le routage de `192.168.0.0` vers les autres sous-réseaux peut être agrégé



La couche réseau de *TCP/IP*

▷ Exemple d'utilisation de *CIDR*

◇ Table de routage dans `192.168.0.1` sans *supernetting*

```
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.17.0    192.168.0.2     255.255.255.240 UG    0      0        0 eth0
192.168.17.16   192.168.0.2     255.255.255.240 UG    0      0        0 eth0
192.168.18.0    192.168.0.2     255.255.255.240 UG    0      0        0 eth0
192.168.18.16   192.168.0.2     255.255.255.240 UG    0      0        0 eth0
192.168.16.0    192.168.0.2     255.255.255.0   UG    0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

◇ Table de routage dans `192.168.0.1` avec *supernetting*

```
# route add -net 192.168.16.0/20 gw 192.168.0.2
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.16.0    192.168.0.2     255.255.240.0   UG    0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

◇ nb : dans les deux cas `192.168.0.2` doit connaître les passerelles suivantes

La couche réseau de *TCP/IP*

▷ La route par défaut

- ◇ Les sous-réseaux sont généralement hiérarchisés
  - ○ Un routeur permet d'*"entrer"* dans un sous-réseau et d'en *"sortir"*
- ◇ Atteindre un nœud sur ***Internet***
  - ○ Pas besoin de connaître les routes vers tous les sous-réseaux ! (la route par défaut désigne le routeur le plus proche)
  - ○ *"Remonter"* de routeur en routeur (*hop-by-hop*)
  - ○ Jusqu'à un routeur qui connaisse la route pour *"redescendre"*
- ◇ Dernière entrée de la table de routage (`/0`)
- ◇ `route add default gw` ***`adresse_du_routeur`***
  (≡ `route add -net 0.0.0.0/0 gw` ***`adresse_du_routeur`***)

```
# route -n
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.40.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         192.168.40.1    0.0.0.0         UG    0      0        0 eth0
```

La couche réseau de *TCP/IP*

- **La route par défaut et les messages** *ICMP-redirect*
  - Contraignant de donner à chaque nœud les routes vers les sous-réseaux !
    - Par confort, on se contente de leur donner une route par défaut
  - Le routeur doit avoir une table de routage correctement renseignée
  - À la réception d'un paquet qu'il doit router
    - Si l'émetteur est dans le même sous-réseau que le *"bon"* routeur
    - Le routeur envoie d'un message *ICMP-redirect* à l'émetteur
    - Il s'adressera désormais au *"bon"* routeur pour cette destination
  - Ex : en réutilisant les sous-réseaux de la page 36
    - `192.168.16.8` veut atteindre `192.168.17.5`
    - Il envoie le paquet *IP* à son routeur par défaut `192.168.16.1`
    - Qui envoie à son tour le paquet à `192.168.16.2`
    - Et indique à `192.168.16.8` de joindre `192.168.17.5` par `192.168.16.2`

La couche réseau de *TCP/IP*

- **Routage statique**
  - Celui qui est décrit ici, routes définies *"en dur"* par l'administrateur
  - Les messages *ICMP-redirect* apportent un minimum de dynamicité
  - Convient largement à l'administration d'un site local
- **Routage dynamique**
  - Concerne les *"gros"* routeurs pour les liaisons *"longues distances"*
  - Des protocoles permettent aux routeurs de mettre à jour leurs tables (*RIP*, *BGP* ...)
  - Annonce de routes, modifications dynamiques, redondance
  - Calculs de distance pour le choix des routes
- **Routage des paquets diffusés**
  - En théorie rien ne s'y oppose (confrontation aux masques ...)
  - En pratique les routeurs filtrent → limité au domaine de diffusion local

La couche réseau de *TCP/IP*

- **Liaison** *point-à-point* **(*PPP*)**
  - Uniquement une adresse locale et une adresse distante (/32)
    - Utilisé pour l'accès à *Internet*
  - Couche liens sous-jacente minimale (bidirectionnelle, sans MAC ...)
    - Sur un nœud : `pppd pty 'prog1' passive noauth`
    - Sur l'autre : `pppd pty 'prog2' noauth` *`IP_locale:IP_distante`*
    - Les deux programmes communiquent par un moyen quelconque
      $\rightarrow$ modem, liaison série, logiciel, signaux de fumée ...
  - Créer des routes pour les sous-réseaux de chaque extrémité
    - `route add -net` *`reseau_distant`* `gw` *`IP_distante`*
  - Nombreuses options possibles dont l'authentification (FAI)
  - Ex : VPN de fortune à travers *SSH*

```
pppd pty 'ssh -t -e none some.where.net pppd passive noauth' \
     noauth 172.16.100.1:172.16.100.2
```

La couche réseau de *TCP/IP*

▷ Le programme `ping`

- ◇ Tester si une adresse *IP* est joignable
- ◇ Envoie un *ICMP-echo-request* et attend l'*ICMP-echo-reply*
- ◇ `-R` pour noter les adresses de sortie (dans les options *IP*, 9 max.)

```
# ping -n -c2 195.221.233.9
PING 195.221.233.9 (195.221.233.9) 56(84) bytes of data.
64 bytes from 195.221.233.9: icmp_seq=1 ttl=254 time=2.37 ms
64 bytes from 195.221.233.9: icmp_seq=2 ttl=254 time=1.20 ms
--- 195.221.233.9 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 1.201/1.788/2.376/0.589 ms
# ping -n -c2 -R 195.221.233.9
PING 195.221.233.9 (195.221.233.9) 56(124) bytes of data.
64 bytes from 195.221.233.9: icmp_seq=1 ttl=254 time=3.11 ms
RR:     192.168.20.236
        195.221.233.1
        195.221.233.9
        195.221.233.9
        192.168.16.1
        192.168.20.236
64 bytes from 195.221.233.9: icmp_seq=2 ttl=254 time=2.72 ms    (same route)
--- 195.221.233.9 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 2.723/2.920/3.118/0.204 ms
```

La couche réseau de *TCP/IP*

- **Utilisation du champ `TTL` dans le routage**
  - Le paquet *IP* est émis avec une valeur `TTL` initiale (*Time To Live*)
    - Normalement `64` (<`256` dans la pratique)
  - Chaque routeur rencontré décrémente cette valeur avant de relayer
  - Celui qui le passe à `0` détruit le paquet
    - Un message *ICMP-Time-Exceeded* est envoyé à l'émetteur initial
  - Permet d'éviter les boucles dans le routage
    - Erreurs dans le routage (rare !)
    - Phases transitoires dans le routage dynamique

La couche réseau de *TCP/IP*

- **Le programme** `traceroute`
  - Lister les routeurs qui permettent d'atteindre un nœud cible
  - Envoi d'un datagramme *IP* vers la cible avec un `TTL` de 1, 2, 3 ...
    - Routeur qui annule `TTL` → *ICMP-Time-Exceeded* à l'émetteur
    - L'adresse *IP* d'entrée du routeur est dans l'*ICMP* en question
    - Incrémenter `TTL` jusqu'à atteindre la cible
  - `-I` pour envoyer un *ICMP-echo-request*
    (par défaut *UDP*/33434 risque d'être filtré par les *firewalls*)

```
# traceroute -In 192.44.75.206
traceroute to 192.44.75.206 (192.44.75.206), 30 hops max, 38 byte packets
 1  192.168.16.1  1.241 ms  1.252 ms  1.361 ms
 2  192.168.128.20  4.301 ms  3.292 ms  3.326 ms
 3  193.50.69.217  3.272 ms  3.122 ms  3.142 ms
 4  193.48.78.29  3.904 ms  3.735 ms  3.755 ms
 5  193.48.78.18  4.996 ms  4.964 ms  4.993 ms
 6  193.50.69.90  5.672 ms  5.674 ms  5.355 ms
 7  192.44.75.206  6.018 ms  4.592 ms  4.422 ms
#
```

La couche réseau de *TCP/IP*

▷ **Plan des sous-réseaux, contrôle du routage**

- ◇ Les outils de base : `ping -R` et `traceroute`
  - ○ `traceroute` ne reporte que les adresses d'entrée
  - ○ `ping -R` ne reporte que les adresses de sortie (9 enregistrements maxi dont le nœud émetteur)
  - ○ En conjuguant les deux on peut découvrir complètement 8 routeurs (Généralement suffisant en local)
- ◇ Démarche indispensable à la mise au point par l'administrateur
  - ○ Vérifier que les routes sont bien renseignées ...
- ◇ Outils également utiles pour les actions malveillantes !
  - ○ Découvrir la topologie du réseau avant d'entreprendre quoi que ce soit → Filter les messages *ICMP* par précaution

La couche réseau de *TCP/IP*

▷ **Bilan**

- ◇ Permet d'abstraire l'identification des nœuds
  - ○ Adresses *IP* attribuables librement
  - ○ À condition de respecter quelques précautions (masque, adresse de sous-réseau, adresse de diffusion)
- ◇ Permet de communiquer au sein d'un domaine de diffusion
  - ○ En *point-à-point* et par diffusion
  - ○ Limité par les propriétés de la couche liens sous-jacente
- ◇ Possibilité de joindre des nœuds quelconques sur *Internet*
  - ○ En *point-à-point* uniquement
  - ○ En respectant toutefois les principes du routage
  - ○ *Via* des domaines de diffusion connexes ou des liaisons *PPP*
- ◇ La segmentation en sous-réseaux permet d'envisager le filtrage des flux

Les noms de domaine

▷ **Expression du besoin**

- ◇ Désignation facile des nœuds par les humains
  - ○ On retient plus facilement des noms que des adresses *IP*
    → Mécanisme de correspondance nom ↔ adresse
  - ○ C'est un procédé applicatif (*TCP/IP* n'en a pas besoin)
- ◇ Offrir une vue logique décorrélée de la structure des sous-réseaux
  - ○ Des noms *"proches"* peuvent désigner des adresses *"éloignées"*
- ◇ Trois principaux types de renseignements :
  - ○ Les noms/adresses des nœuds locaux (serveurs, postes de travail)
  - ○ Les noms/adresses des nœuds qu'on trouve exposés sur *Internet*
  - ○ Les noms/adresses des nœuds qu'on expose nous-même sur *Internet*

Les noms de domaine

▷ Notion de nom de domaine

- ◇ Séquence de *labels* séparés deux à deux par un point
  - ○ Au maximum 63 caractères par *label* et 255 caractères au total
  - ○ Pas de distinction majuscule/minuscule
  - ○ *Labels* constitués de lettres, chiffres ou tirets
    (premier caractère : lettre, dernier caractère : lettre ou chiffre)
- ◇ La séquence de *labels* illustre la structure hiérarchique des domaines
  - ○ *FQDN* : *Fully Qualified Domain Name*
    → séquence complête de domaines menant de la racine à la cible
  - ○ Un nombre quelconque de sous-domaines peuvent être imbriqués
  - ○ ex : `www.enib.fr`, nœud `www` du sous-domaine `enib` du domaine `fr`

Les noms de domaine

- **Nom de domaine $\leftrightarrow$ adresse** *IP*
  - Fonctions `gethostbyname()`, `gethostbyaddr()` ... de la `libc`
  - Consulter le fichier `/etc/hosts` (`C:\WINDOWS\HOSTS` sous *Window$*)
    - Associer une adresse *IP* à un (ou des) nom de domaine
    - Mise à jour manuelle envisageable pour un parc très réduit ! (dans le domaine local de préférence)
  - Interroger un serveur *DNS* (*Domain Name Service*)
    - Paramètres spécifiés dans le fichier `/etc/resolv.conf`
    - Interroger le serveur désignée sur le port `53`/*UDP*/*TCP* (serveur secondaire, tertiaire ... en cas de panne)
    - Compléter implicitement le nom de domaine si nécessaire (ex : `galet13` $\rightarrow$ `galet13.enib.fr`)
    - Le serveur doit renseigner sur le domaine local et sur les autres

Les noms de domaine

▷ **Exemple de fichier** `/etc/hosts`

◇ Un ou plusieurs noms peuvent être associés à une adresse

```
# cat /etc/hosts
127.0.0.1      localhost
192.168.20.236 nowin     nowin.c022.enib.fr
192.168.20.221 winout    winout.c022.enib.fr
```

▷ **Exemple de fichier** `/etc/resolv.conf`

◇ Completer implicitement par `c022.enib.fr` (`enib.fr` si non trouvé)

◇ Interroger `192.168.18.4` (puis `192.168.18.3` si pas de réponse)

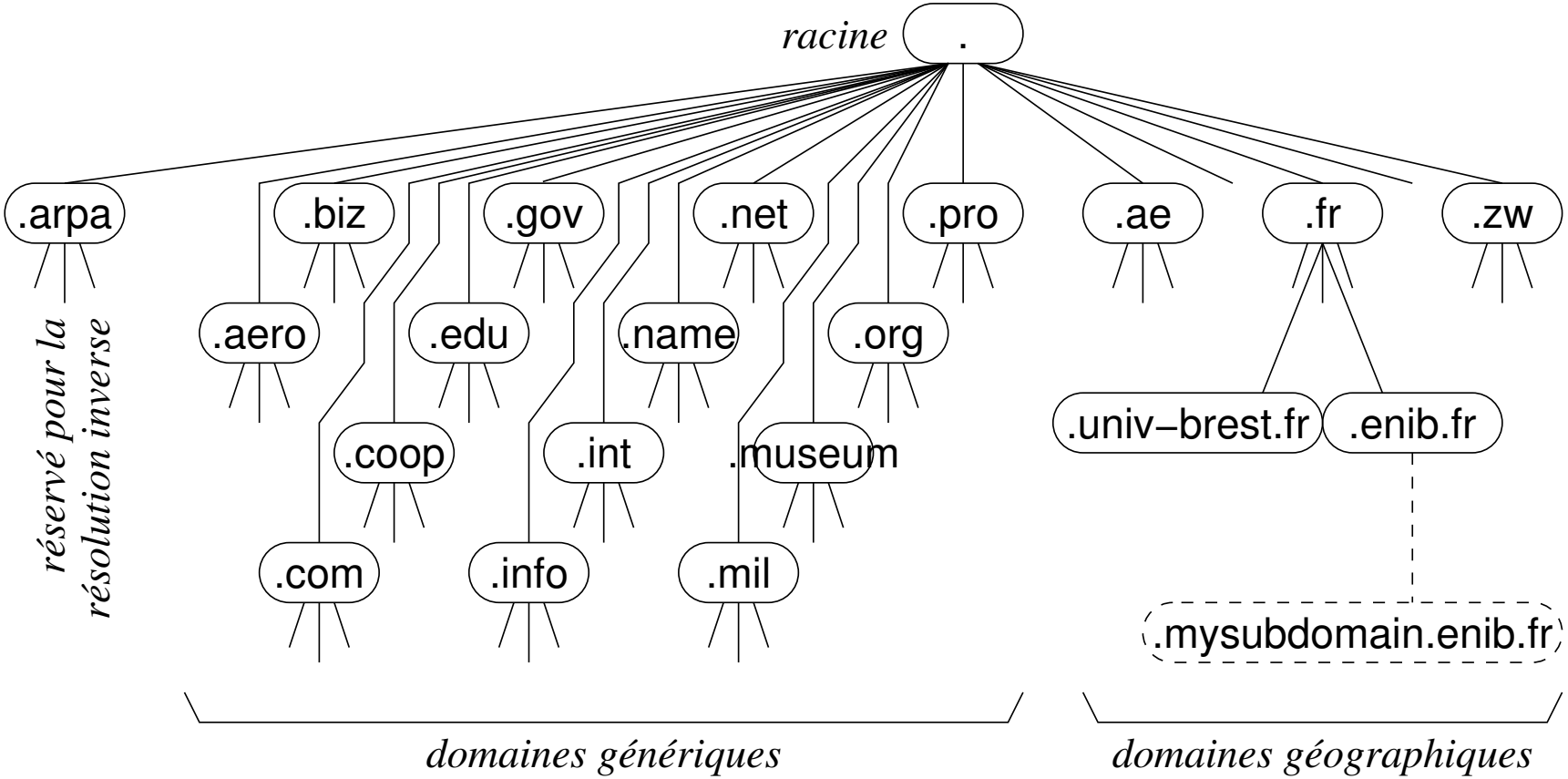
```
# cat /etc/resolv.conf
search c022.enib.fr enib.fr
nameserver 192.168.18.4
nameserver 192.168.18.3
```

Les noms de domaine

- **Le serveur** *DNS*
  - Maintient une liste d'association nom $\leftrightarrow$ adresse *IP*
    - Uniquement pour les nœuds de la zone d'autorité
  - Autres domaines ? Demander à un *DNS racine*
    - ex : `.univ-brest.fr` $\rightarrow$ `.`
  - Le *DNS racine* connaît les *DNS* des zones inférieures, etc
    - ex : `.` $\rightarrow$ `.fr` $\rightarrow$ `.enib.fr` ...
  - Il s'agit d'une base de données distribuée
    - Chaque serveur *DNS* ne maintient que des informations partielles
    - L'arborescence des *DNS* donne accès à l'ensemble
    - Les serveurs ont un cache avec une durée d'expiration (éviter de refaire la recherche complète à chaque fois)

Les noms de domaine

▷ **L'arborescence des serveurs** *DNS*



Les noms de domaine

▷ Configuration d'un serveur *DNS* (*ISC-BIND*)

◇ Fichier `/etc/named.conf` pour donner les options et énumérer les *zones* (De très nombreuses options non vues ici . . .)

```
options {
  version "";                                  # dissimuler la version (securite)
  directory "/var/named";                      # emplacement des fichiers de zone
};
zone "." IN            # acces aux serveurs ‘‘racine’’ (recursion vers internet)
  { type hint; file "named.root"; };
zone "localhost" IN                          # resolution directe de ‘‘localhost’’
  { type master; file "localhost.zone"; };
zone "0.0.127.in-addr.arpa" IN               # resolution inverse de ‘‘localhost’’
  { type master; file "0.0.127.in-addr.arpa.zone"; };
zone "example.net" IN                    # resolution directe dans ‘‘example.net’’
  { type master; file "example.net.zone"; };
zone "233.221.195.in-addr.arpa" IN     # resolution inverse dans ‘‘example.net’’
  { type master; file "233.221.195.in-addr.arpa.zone"; };
```

Les noms de domaine

▷ **Les fichiers de zone (*RFC-1034* & *RFC-1035*)**

- ◇ Décrivent un ensemble d'enregistrements (*RR* : *Ressource Record*)
- ◇ Forme d'un *RR* : ***nom ttl classe type valeur***
  - ○ ***nom*** : objet du *RR* (le même que le *RR* précédent si omis)
  - ○ ***ttl*** : durée de validité (paramètre global `$TTL` si omis)
  - ○ ***classe*** : `IN` pour *Internet*
  - ○ ***type*** : signification du *RR* (`SOA`, `NS`, `MX`, `A`, `CNAME`, `PTR` ...)
  - ○ ***valeur*** : donnée associée à ***nom*** (la forme dépend de ***type***)
- ◇ Les ***ttl*** sont exprimés en secondes (sauf si suffixe `M`, `H`, `D`, `W`)
- ◇ Le symbole `@` représente le nom de la zone (directive `zone` dans `/etc/named.conf`)

Les noms de domaine

▷ Les types d'enregistrements

- ◇ `SOA` (*Start Of Authority*) : annonce le contenu une zone d'autorité
  - ○ Le ***nom*** est généralement `@` (la zone d'autorité)
  - ○ Nom du serveur *DNS*, *e-mail* de l'administrateur (`@` devient `.`)
  - ○ Numéro de version du contenu de la zone
    → Doit évoluer à chaque mise à jour !
    → Généralement la date et un compteur (format ***yyyymmddnn***)
  - ○ Quatre durées non discutées ici (généralement toujours les mêmes)
  - ○ ex : `@ IN SOA ns.ex.net. root.ex.net. ( 2007101501 3H 15M 1W 1D )`
- ◇ `NS` (*Name Server*) : indique un serveur *DNS*
  - ○ Le ***nom*** est généralement le domaine courant ou un sous-domaine
  - ○ La ***valeur*** est un nom de domaine (pas une adresse !)
    (plusieurs `NS` possibles pour le même ***nom*** → redondance, équilibrage)
  - ○ ex : `@ IN NS ns.ex.net.`

Les noms de domaine

▷ Les types d'enregistrements

- ◇ `MX` (*Mail eXchanger*) : indique un serveur de réception d'*e-mail*
  - ○ Le ***nom*** est généralement `@` (la zone d'autorité)
  - ○ La ***valeur*** est un numéro d'ordre et un nom de domaine
  - ○ *e-mail* à `qqun@ex.net` → recherche des *RR* de type `MX` de `ex.net` (préférence des ordres faibles, les suivants en cas de panne)
  - ○ ex : `@ IN MX 10 smtp`
- ◇ `A` (*Address*) : donne une adresse
  - ○ Information principalement recherchée
  - ○ Le ***nom*** est un nom de domaine (généralement un nœud de la zone d'autorité)
  - ○ La ***valeur*** est une adresse
  - ○ ex : `server IN A 195.221.233.3`

Les noms de domaine

▷ Les types d'enregistrements

- ◇ `CNAME` (*Cannonical Name*) : crée un alias
  - ○ La ***valeur*** est un nom de domaine ou un autre alias
  - ○ Le ***nom*** désigne alors la même chose que la ***valeur***
  - ○ ex : `www IN CNAME server`
- ◇ `PTR` (*Pointer*) : résolution inverse (adresse $\rightarrow$ *FQDN*)
  - ○ Le ***nom*** est une adresse en ordre inverse (la partie manquant à `@`)
  - ○ La ***valeur*** est un nom de domaine
  - ○ ex : `3 IN PTR server.ex.net.`
    (`195.221.233.3` si la zone est `233.221.195.in-addr.arpa`)
- ◇ Il existe d'autres ***type***s de *RR* non traités ici ...

Les noms de domaine

▷ Les zones d'autorité : description du domaine local

- ◇ Généralement décrites par deux fichiers de zone de type `master`
  - ○ Le nom de domaine pour la résolution directe (ex : `example.net`)
    → les RR sont principalement des `A` et des `CNAME` voire des `NS`
  - ○ La partie fixe de l'adresse inversée pour la résolution inverse
    (ex : `233.221.195.in-addr.arpa` pour `195.221.233.0/24`)
    → les RR sont principalement des `PTR`
  - ○ Ils **doivent** commencer par un RR de type `SOA` concernant `@`
    → le numéro de série doit augmenter à chaque mise à jour !
  - ○ Ils **doivent** contenir un RR de type `NS` concernant `@`
    (redondant avec celui de la *`valeur`* du `SOA` ?!?!)
- ◇ Interrogations utiles pour :
  - ○ Les nœuds locaux : descriptions des nœuds environnants
  - ○ Les nœuds distants : descriptions des nœuds exposées sur *Internet*

Les noms de domaine

▷ **Exemple : définition d'une zone d'autorité (directe)**

◇ Type `master` dans `/etc/named.conf`

◇ `@` représente `example.net`

◇ Noms de domaine sans point terminal → implicitement complétés par `@`

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
zone "example.net" IN { type master; file "example.net.zone"; };

# cat /var/named/example.net.zone
$TTL  1D
@         IN SOA    ns root ( 2007101501 3H 15M 1W 1D )
@         IN NS     ns
@         IN MX     10 smtp
desktop   IN A      195.221.233.1
laptop    IN A      195.221.233.2
server    IN A      195.221.233.3
ns        IN CNAME  server
www       IN CNAME  server
smtp      IN CNAME  server
```

Les noms de domaine

▷ **Exemple : définition d'une zone d'autorité (inverse)**

◇ Type `master` dans `/etc/named.conf`

◇ `@` représente `233.221.195.in-addr.arpa`

◇ *"Noms"* de domaine sans point terminal → implicitement complétés par `@`

○ Il faut donc indiquer des *FQDN* ici (avec le point terminal) !

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
zone "example.net" IN { type master; file "example.net.zone"; };
zone "233.221.195.in-addr.arpa" IN
  { type master; file "233.221.195.in-addr.arpa.zone"; };

# cat /var/named/233.221.195.in-addr.arpa.zone
$TTL  1D
@  IN SOA  ns.example.net. root.example.net. ( 2007101501 3H 15M 1W 1D )
@  IN NS   ns.example.net.
1  IN PTR  desktop.example.net.
2  IN PTR  laptop.example.net.
3  IN PTR  server.example.net.
```

Les noms de domaine

▷ La zone d'autorité de `localhost`

- ◇ Peut être utile au nœud serveur *DNS* lui même (?)
- ◇ Ces fichiers sont généralement présents et utilisés par défaut

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
zone "localhost" IN { type master; file "localhost.zone"; };
zone "0.0.127.in-addr.arpa" IN { type master; file "0.0.127.in-addr.arpa.zone"; };

# cat /var/named/localhost.zone
$TTL  1D
@  IN SOA  @ root ( 2007101501 3H 15M 1W 1D )
@  IN NS   localhost.
@  IN A    127.0.0.1

# cat /var/named/0.0.127.in-addr.arpa.zone
$TTL  1D
@  IN SOA  localhost. root.localhost. ( 2007101501 3H 15M 1W 1D )
@  IN NS   localhost.
1  IN PTR  localhost.
```

Les noms de domaine

▷ Résolutions à l'extérieur de la zone d'autorité

- ◇ Nécessite d'interroger un serveur racine
- ◇ Zone `.` de type `hint` dans `/etc/named.conf` (disponible sur `ftp://ftp.internic.net/domain/named.root`)
- ◇ Plusieurs `NS` pour `.` → utilisation aléatoire (redondance, équilibrage)

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
zone "." IN { type hint; file "named.root"; };

# cat /var/named/localhost.zone
.                    3600000  IN NS  a.root-servers.net.
a.root-servers.net.  3600000  IN A   198.41.0.4

; ... serveurs  b.root-servers.net.  a  l.root-servers.net.  ...

.                    3600000  IN NS  m.root-servers.net.
m.root-servers.net.  3600000  IN A   202.12.27.33
```

Les noms de domaine

▷ Bilan intermédiaire

- ◇ Les zones vues ici peuvent être rassemblées dans un même serveur *DNS* (comme dans l'exemple de la page 53)
- ◇ Il est utile pour les nœuds distants (sur *Internet*)
  - ○ Permet de décrire les nœuds qu'on expose
  - ○ Il doit être connu du serveur *DNS* de la zone d'autorité supérieure
- ◇ Il est utile pour les nœuds locaux
  - ○ Permet de décrire les nœuds environnants
  - ○ Permet de relayer les resolutions vers *Internet* (résolution récursive, mise en cache des réponses)
  - ○ C'est le seul serveur *DNS* que les nœuds locaux ont besoin de connaître
- ◇ ex : `kiwi.enib.fr` → `enib.fr.` → `citron.enib.fr`
- ◇ ex : `kiwi.enib.fr` → `enib.fr.` → `.` → `org.` → `isc.org.` → `www.isc.org`

Les noms de domaine

▷ Subdivision en sous-domaines

- ◇ Une zone d'autorité peut être divisée en sous-domaines
- ◇ Gérés par le même serveur → fichiers de zone supplémentaires
- ◇ Gérés par d'autres serveurs → il faut les référencer (RR de type `NS`)

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
zone "example.net" IN { type master; file "example.net.zone"; };
zone "sub1.example.net" IN { type master; file "sub1.example.net.zone"; };

# cat /var/named/example.net.zone
$TTL  1D
@         IN SOA  ns root ( 2007101501 3H 15M 1W 1D )
@         IN NS   ns
ns        IN A    195.221.233.3
sub2      IN NS   ns.sub2
ns.sub2   IN A    195.221.233.5
; ... autres RR de la zone example.net ...
```

Les noms de domaine

- **Relai des requêtes vers un autre serveur**
  - Raisonnement sur un cas pratique :
    - Un premier serveur pour une zone
    - Un second serveur pour un sous-domaine du premier (usage local)
    - Les *RR* du premier concernent aussi les clients du second
  - Démarche inappropriée : utilisation d'une zone `"."` de type `hint`
    - Le second passe par un serveur racine pour revenir au premier !
    - Tous les nœuds locaux ne sont pas forcément visibles (voir plus loin) (la requête vient de *"l'extérieur"*)
  - Démarche appropriée : option `forwarders` vers le premier serveur
    - Échec du second serveur $\rightarrow$ relai vers le premier serveur *DNS*
    - Tous les nœuds locaux seront visibles (voir plus loin) (la requête vient de *"l'intérieur"*)

Les noms de domaine

- **Relai des requêtes vers un autre serveur**
  - Exemple : `xxx.sub.example.net` veut résoudre `yyy.example.net`
  - Il interroge son serveur le plus proche (autorité sur `sub.example.net`)
  - Ce serveur ne peut résoudre `yyy.example.net`
  - Sa directive `forwarders` relaye vers le serveur immédiatement supérieur (il se comporte en client vis-à-vis de ce dernier)
  - Le serveur supérieur a autorité sur `example.net` et répond pour `yyy`
  - Le serveur intermédiaire met la réponse en cache et transmet à `xxx`

```
# cat /etc/named.conf
options {
  version ""; directory "/var/named";
  forwarders { 195.221.233.3; }; # adresse du serveur de example.net
};
zone "sub.example.net" IN { type master; file "sub.example.net.zone"; };
```

Les noms de domaine

- **Serveurs *DNS* secondaires**
  - Les clients peuvent interroger plusieurs serveurs
    - Plusieurs directives `nameserver` dans `/etc/resolv.conf`
    - Ils sont interrogés dans l'ordre jusqu'à ce que l'un d'eux réponde
  - Un serveur *esclave* se maintient à jour depuis un serveur *maître*
    - Mécanisme de *notification* et de *transfert de zone*
    - Le numéro de version est important pour la mise à jour
    - Il peut alors le remplacer en cas de panne

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
zone "example.net" IN
  { type slave; file "example.net.zone"; masters { 195.221.233.3 }; };
zone "233.221.195.in-addr.arpa" IN
  { type slave; file "233.221.195.in-addr.arpa.zone"; masters { 195.221.233.3 }; };
```

Les noms de domaine

- **Restreindre l'accès aux informations : serveur multi-vues**
  - Comportement différemment selon l'origine d'une requête
  - La première *vue* à laquelle correspond l'adresse source est retenue
  - Chaque vue contient ses propres zones et options
    - Aucune zone ne doit être à l'extérieur d'une *vue*
    - Le bloc `options` reste global (surdéfinitions possibles dans les *vues*)
  - Vue *"interne"* typique :
    - Autoriser toutes les fonctionnalités présentées précédemment
  - Vue *"externe"* typique :
    - Ne décrire que les serveurs exposés sur *Internet*
    - Les *transferts de zone* et les *récursions* sont interdits
      → Ne résoudre que des noms explicitement identifiées (pas de liste)
      → Ne pas servir de *DNS* général à quiconque sur *Internet*

Les noms de domaine

▷ Restreindre l'accès aux informations : serveur multi-vues

```
# cat /etc/named.conf
options { version ""; directory "/var/named"; };
view "internal" {
  match-clients { 195.221.233.0/24; 127.0.0.1; };          # clients locaux uniquement
  zone "." IN { type hint; file "named.root"; };  # recursion vers les serveurs racines
  zone "localhost" IN { type master; file "localhost.zone"; };
  zone "0.0.127.in-addr.arpa" IN
    { type master; file "0.0.127.in-addr.arpa.zone"; };
  zone "example.net" IN                      # fichier de zone complet (tous les noeuds)
    { type master; file "example.net.zone"; };
  zone "233.221.195.in-addr.arpa" IN         # fichier de zone complet (tous les noeuds)
    { type master; file "233.221.195.in-addr.arpa.zone"; };
};
view "external" {                  # pas d'option match-clients --> tous sont acceptes
  allow-recursion { none; }; allow-transfer { none; };
  zone "example.net" IN                       # fichier de zone partiel (noeuds publics)
    { type master; file "example.net.pub-zone"; };
  zone "233.221.195.in-addr.arpa" IN          # fichier de zone partiel (noeuds publics)
    { type master; file "233.221.195.in-addr.arpa.pub-zone"; };
};
```

Les noms de domaine

- **Interroger les serveurs** *DNS* **avec** `dig`
  - Ligne de commande : `dig [@serveur] nom_de_domaine type`
    - `server` est facultatif (serveur par défaut si omis)
    - `type` indique les *RR* attendus (`SOA`, `NS`, `MX`, `A`, `CNAME` . . .)
      (`ANY` : tous, `AXFR` : transfert de zone)
  - Affiche les *RR* dans le format des fichiers de zone
  - Les requêtes sont récursives
    - Si le serveur n'a pas l'autorité il poursuit la résolution
    - Sauf le transfert de zone : démander directement au serveur autoritaire
  - Ex : obtenir tous les *RR* concernant le nom de domaine `example.net`
    → `dig example.net ANY`
  - Ex : lister le contenu de la zone `example.net`
    → `dig @ns.example.net example.net AXFR`

Les noms de domaine

▷ Interroger les serveurs *DNS* avec `dig`

```
# dig slackware.com NS
...
slackware.com.          83948   IN      NS      ns1.cwo.com.
slackware.com.          83948   IN      NS      ns2.cwo.com.
...
# dig @ns1.cwo.com slackware.com AXFR
...
slackware.com.          86400   IN      SOA     ns1.cwo.com. hostmaster.cwo.com.
                                                200401033 43200 3600 604800 86400
slackware.com.          86400   IN      MX      1 mail.slackware.com.
slackware.com.          86400   IN      NS      ns1.cwo.com.
slackware.com.          86400   IN      NS      ns2.cwo.com.
slackware.com.          86400   IN      A       64.57.102.34
...
store.slackware.com.    86400   IN      A       69.50.233.153
www.slackware.com.      86400   IN      CNAME   slackware.com.
...
```

Les noms de domaine

▷ **Interroger les serveurs** *DNS* **avec** `host`

◇ Ligne de commande : `host [-l] nom_de_domaine [serveur]`

◇ Semblable à `dig` mais plus lisible et moins complet

```
# host www.example.net
www.example.net is an alias for server.example.net.
server.example.net has address 195.221.233.3
# host 195.221.233.3
3.233.221.195.in-addr.arpa domain name pointer server.example.net.
# host -l example.net ns.example.net
Using domain server:
Name: ns.example.net
Address: 195.221.233.3#53
Aliases:

example.net name server ns.example.net.
desktop.example.net has address 195.221.233.1
laptop.example.net has address 195.221.233.2
server.example.net has address 195.221.233.3
```

Les noms de domaine

▷ Bilan

- ◇ Nous sommes en mesure de gérer complètement un sous-domaine
- ◇ Il peut être subdivisé en d'autres sous-domaines (usage local ou public)
  - ○ Gérés par le même serveur ou d'autres (avec d'éventuelles redondances)
- ◇ Nos serveurs coopèrent avec les serveurs d'*Internet*
  - ○ Pour résoudre dans des domaines externes
  - ○ Pour répondre à des requêtes externes
- ◇ Une ébauche de démarche sécuritaire est envisageable
  - ○ Proposer plusieurs *vues* sur les zones
    → Ne pas renseigner l'*"extérieur"* sur la configuration *"intérieure"*
- ◇ De nombreux points ne sont pas traités ici
  - ○ Mise à jour dynamique, authentification ...
- ◇ nb : requêtes/réponses sur 53/*UDP*, transferts de zone sur 53/*TCP*

Configuration automatique des nœuds

- **DHCP :** *Dynamic Host Configuration Protocol*
  - Simplifier la configuration des nœuds d'un sous-réseau
    - Ne concerne pas les serveurs (peu nombreux et *"ajustés"* précisément)
    - Concerne les postes de travail (nombreux et d'usage similaire)
  - Attribuer automatiquement une adresse *IP* et un masque
    - Dans des plages d'adresses librement accessibles
    - Ou selon une attribution prédéterminée
  - Renseigner sur l'utilisation du réseau
    - Route par défaut
    - Nom d'hôte, serveur(s) *DNS* et domaine(s) par défaut
    - Autres services ...

Configuration automatique des nœuds

- **Déroulement du dialogue *DHCP* (*UDP* sur les port `67` et `68`)**
  - `DHCP_DISCOVER` : le client sollicite un serveur
    - `0.0.0.0:68` (*MAC*-client) → `255.255.255.255:67` (`FF:FF:FF:FF:FF:FF`)
  - `DHCP_OFFER` : le serveur fait une proposition
    - *IP*-serveur`:67` (*MAC*-serveur) → *IP*-client`:68` (*MAC*-client)
    - L'adresse est réservée quelques temps en attendant la confirmation
  - `DHCP_REQUEST` : le client accepte la proposition
    - `0.0.0.0:68` (MAC-client) → `255.255.255.255:67` (`FF:FF:FF:FF:FF:FF`)
  - `DHCP_ACK` : L'adresse est attribuée pour une certaine durée (*bail*)
    - *IP*-serveur`:67` (*MAC*-serveur) → *IP*-client`:68` (*MAC*-client)
  - Renouvellement du *bail* : `DHCP_REQUEST` et `DHCP_ACK`
    - Le client utilise cette fois les *"bonnes"* adresses (déjà connues)

Configuration automatique des nœuds

- **Le client** *DHCP*
  - Utilisé à la place de `ifconfig` pour configurer une interface réseau
  - Permet de réclamer des options particulières au serveur
    - Adresse *IP* dernièrement utilisée, nom d'hôte, durée du *bail* ...
    - Le serveur n'est pas obligé de respecter ces souhaits
    - Le client n'est pas obligé d'utiliser toutes les options reçues
  - Le client est un service qui tourne en arrière plan
    - Il doit demander à renouveler le *bail* avant son expiration (sinon l'adresse peut être réattribuée à un autre client)
    - Lorsqu'il s'arrête il peut envoyer un message `DHCP_RELEASE` (le serveur pourra immédiatement réattribuer l'adresse)
  - ex : `dhcpcd -H -t 5 eth2`
    (accepter le nom d'hôte reçu, échec après 5 secondes sans réponse)

Configuration automatique des nœuds

- **Le serveur *DHCP* (*ISC-DHCPD*)**
  - Écoute sur une ou plusieurs interfaces réseau
  - Les requêtes sont traitées selon :
    - Des paramètres globaux
    - Des définitions de sous-réseaux (`subnet`)
    - Des spécifications pour des nœuds particuliers (`host`)
  - Informations déterminantes pour le choix :
    - L'interface de réception de la requête
      → Choix d'une adresse dans le même sous-réseau
    - L'adresse *MAC* du client
      → Informations spécifiques à un nœud
  - ex : `dhcpd eth1 eth2`

Configuration automatique des nœuds

▷ Exemple : configuration d'un serveur *DHCP*

```
ddns-update-style none;   # pas de mise a jour du DNS
default-lease-time 86400; # bail par default : 1 jour
max-lease-time 86400;     # meme si le client demande plus

use-host-decl-names true; # les noms des blocs ‘‘host’’ sont des hostname
option domain-name "example.net";
option domain-name-servers 195.221.233.3;

subnet 195.221.233.0 netmask 255.255.255.0 { # sous-reseau a attribuer
  option routers 195.221.233.254;            # route par defaut
  range 195.221.233.100 195.221.233.200;     # plage d'adresses dynamiques
}

# fixer les adresses de quelques noeuds connus
host desktop { hardware ethernet 00:30:65:4E:21:60; fixed-address 195.221.233.1; }
host laptop { hardware ethernet 00:40:45:07:F8:06; fixed-address 195.221.233.2; }
```

Configuration automatique des nœuds

- **Configuration d'un serveur *DHCP* (*ISC-DHCPD*)**
  - Il doit y avoir au moins un bloc `subnet`
    - Les adresses attribuées sont nécessairement dans un tel sous-réseau
    - Généralement un bloc `subnet` pour chaque interface à l'écoute (chacune est dans un sous-réseau différent)
    - Les options définies ici surdéfinissent les options globales (la route par défaut est généralement spécifique au sous-réseau)
    - Il peut y avoir 0, 1 ou plusieurs plages d'adresses dynamiques
  - Un bloc `host` caractérise une adresse *MAC* particulière
    - Surdéfinition des options globales et du *subnet*
    - Possibilité de spécifier une (ou plusieurs) adresse *IP* fixe (doit tomber dans un bloc `subnet`)
    - Possibilité de factoriser les options de plusieurs `host` dans un `group`

Configuration automatique des nœuds

- **Vision simplifiée de l'attribution d'une adresse**
  - La requête arrive sur une interface
  - Adresse IP de l'interface $\rightarrow$ choix du bloc `subnet`
  - Les options de ce bloc masquent les options globales
  - S'il existe un bloc `host` ayant l'adresse MAC du client
    - Les options de ce bloc masquent les options précédentes
    - S'il donne une adresse IP fixe correspondant au bloc `subnet`
      $\rightarrow$ Attribuer cette adresse IP avec les options retenues
  - Si l'adresse IP n'a pas encore été attribuée
    - S'il y a une plage dynamique dans le bloc `subnet`
      $\rightarrow$ Choisir une adresse IP libre et l'attribuer avec les options retenues

Configuration automatique des nœuds

▷ Exemple : adresses fixes différentes dans plusieurs sous-réseaux

```
ddns-update-style none; default-lease-time 86400; max-lease-time 86400;
use-host-decl-names true;
subnet 192.168.10.0 netmask 255.255.255.0 # l'interface eth1 est dans ce sous-reseau
  { option routers 192.168.10.1; range 192.168.10.50 192.168.10.254; }
subnet 192.168.20.0 netmask 255.255.255.0 # l'interface eth2 est dans ce sous-reseau
  { option routers 192.168.20.1; }
host aaa
  { hardware ethernet 00:30:65:4E:21:60; fixed-address 192.168.10.2,192.168.20.2; }
host bbb
  { hardware ethernet 00:40:45:07:F8:06; fixed-address 192.168.20.3; }
# une requete provenant de aaa obtient :
#   - 192.168.10.2 si elle arrive par eth1
#   - 192.168.20.2 si elle arrive par eth2
# une requete provenant de bbb obtient :
#   - une adresse dynamique si elle arrive par eth1
#   - 192.168.20.3 si elle arrive par eth2
# une requete provenant d'un autre noeud obtient :
#   - une adresse dynamique si elle arrive par eth1
#   - aucune reponse si elle arrive par eth2
```

Configuration automatique des nœuds

▷ **Clients et serveur *DHCP* dans des domaines de diffusion distincts**

◇ Les requêtes *DHCP* (diffusion) ne passent pas les routeurs

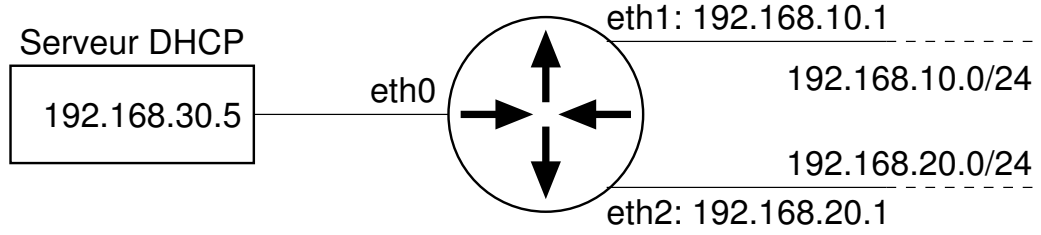
○ Le serveur devrait avoir une interface dans chaque sous-réseau
(Il pourrait s'agir du routeur lui-même, mais ce n'est pas obligatoire)

◇ Le routeur doit exécuter un *agent de relais DHCP*

○ Le serveur ne peut déterminer seul l'origine des requêtes
(une seule interface pour recevoir toutes les requêtes)

○ Ajout de l'adresse *IP* de réception lors du relais des requêtes
→ Le serveur s'en sert alors pour le choix du bloc `subnet`

○ ex : `dhcrelay -i eth0 -i eth1 -i eth2 192.168.30.5`
(ne pas oublier l'interface côté serveur)



Configuration automatique des nœuds

▷ Démarrage de nœuds sans disque

- ◇ Protocole *BOOTP* (*Boot Protocol*) à l'origine de *DHCP*
- ◇ La carte réseau effectue un requête pour obtenir le code de *boot*
- ◇ Le serveur *DHCP*/*BOOTP* donne les informations suivantes
  - ○ `next-server` *`adresse_serveur_tftp`* `;`
  - ○ `filename "`*`fichier_de_boot`*`";`
- ◇ La carte réseau obtient le fichier de *boot* auprès du serveur *TFTP*
  - ○ Il est chargé en mémoire et exécuté
- ◇ Ce code peut exploiter des options *BOOTP* spécifiques
  - ○ ex : `option root-path "`*`adresse:repertoire_racine`*`";`
    (Répertoire racine du système accessible à distance par *NFS*)
- ◇ nb : le système peut ensuite effectuer une nouvelle requête *DHCP*
  (comme s'il avait démarré depuis son propre disque)

Configuration automatique des nœuds

▷ **Bilan**

- ◇ Les postes clients peuvent exploiter le réseau sans configuration
  - ○ Il faut tout de même qu'ils utilisent *DHCP* !
- ◇ La route par défaut fournie est suffisante
  - ○ Le routeur enverra des messages *ICMP-redirect* si nécessaire
- ◇ De nombreuses options peuvent être fournies
  - ○ Usuellement : nom d'hôte, serveur(s) *DNS*, domaine(s) par défaut
  - ○ Services variés : *NIS*, *NTP*, *POP*, *SMTP*, impression ...
  - ○ Les clients les exploitent (ou non) à leur guise
- ◇ Le déplacement d'un nœud ne nécessite pas de reconfiguration (centralisé au niveau du serveur *DHCP*)
- ◇ De nombreuses possibilités non vues ici (expressions logiques, interactions avec le *DNS* ...)