

CM 3 : Couche Liaison

D'après le cours de Bruno Martin et les slides du livre "Computer Networking: A Top Down Approach, 6th edition, Jim Kurose, Keith Ross, Addison-Wesley, March 2012"

Ramon APARICIO-PARDO

Ramon.Aparicio-Pardo@unice.fr

PLAN CM 3

- 1. COUCHE DE LIAISON**
- 2. FONCTIONS DE LA COUCHE LIAISON**
- 3. DETECTION ET CORRECTION D'ERREURS**
- 4. CONTRÔLE D'ACCÈS AU SUPPORT (MAC)**
- 5. ETHERNET**
- 6. DU HUB AU SWITCH**
- 7. ARP/RARP**

Vue d'ensemble de la couche liaison

❖ La couche liaison :

- définit procédures exploitation du lien de communication
- permet l'envoi de trames en séquence
- détecte et corrige les erreurs du support physique
- contrôle l'accès au canal partagé (sous-couche MAC)

❖ Ethernet [Standard 802.3] :

- Principal protocole de la couche liaison
- Plusieurs standards selon le moyen physique (Wifi, câble, fibre optique)
- Adresses matérielles (MAC)
- Implémenté dans l'interface de réseau (*network interface card, NIC*)

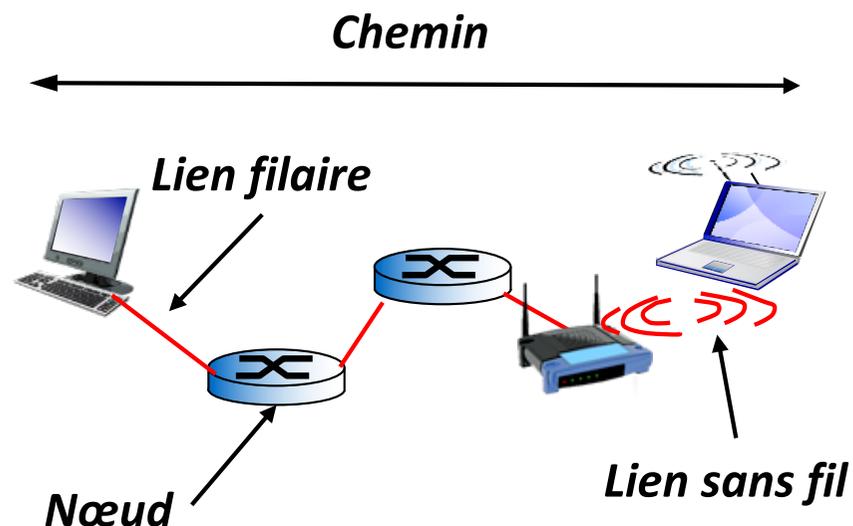
Terminologie de la couche liaison

❖ **Nœuds** :

- hôtes et routeurs

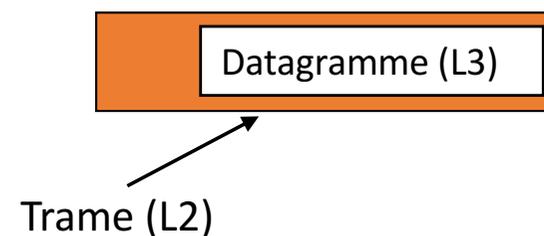
❖ **Liens** :

- Canaux de communication qui connectent 2 nœuds adjacents le long du *chemin* de communication
- *liens filaires*
- *liens sans fil*



❖ **Trame (frame)** : Paquet de la couche 2-liaison :

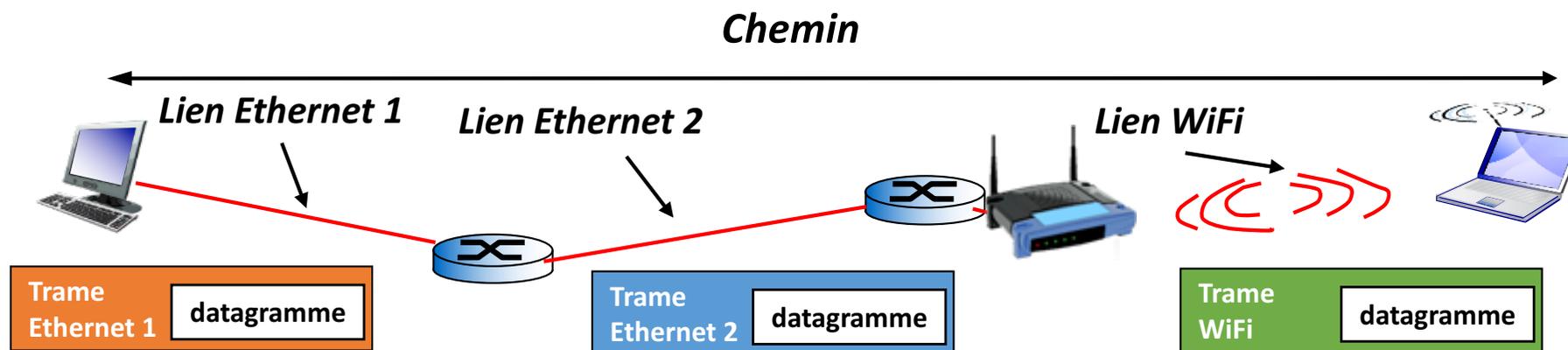
- encapsule un *datagramme* (paquet de la couche 3-réseau)



Terminologie de la couche liaison

❖ Objectif couche *liaison*:

- La couche de liaison a la **responsabilité** du *transfert* des datagrammes d'un nœud au nœud *physiquement adjacent* sur un lien
- datagramme transféré par *différents protocoles de liaison* sur *différentes liaisons* :
 - p. ex., *Ethernet (802.3)* sur la première liaison, *802.11 (WLAN - WiFi)* sur la dernière liaison
- chaque protocole de liaison fournit des services différents
 - p. ex., *transfert fiable* sur le lien



Fonctions de la couche liaison

- ❖ **Mise en trames des datagrammes (*framing*)**
 - Encapsuler un datagramme dans une trame, en ajoutant un en-tête et une queue de bande
- ❖ **Contrôle d'accès au support (*Media Access Control ou MAC*)**
 - Comment accéder au canal si support partagé par plusieurs nœuds
- ❖ **Adressage "MAC" dit *physique ou materiel***
 - Adresses MAC utilisées dans les en-têtes de trame pour identifier la source et la destination
 - Différent de l'adressage IP dit *logique*
- ❖ **Transfert fiable entre les nœuds adjacents (*reliable data transfert*)**
 - Garanti que tous les paquets sont reçus correctement: si un paquet n'est pas reçu par le récepteur, il est retransmit
 - ***Souvent utilisé pour*** des liaisons avec des taux d'erreur élevés (***liaison sans fil***) afin de corriger une erreur localement, sur la liaison sur laquelle l'erreur se produit.
 - ***Rarement utilisé sur*** les liaisons à faible erreur de bit (***fibre, paire torsadée***). Il ne vaut pas la peine, on fera confiance à des autres solutions :
 - Code de détection et correction d'erreurs → à exécuter dans le récepteur
 - S'il faut retransmettre, on le fera *bout à bout* (transfert fiable sur tout le chemin) → on verra cela dans la couche transport

Fonctions de la couche liaison

❖ Contrôle de flux (*flow control*) :

- Régler le débit de transmission entre les nœuds émetteur et récepteur adjacents pour empêcher un émetteur rapide d'accabler un récepteur lent.

❖ Détection d'erreur :

- Erreurs causées par l'atténuation du signal, le bruit.
- Le récepteur détecte la présence d'erreurs:
 - signale l'expéditeur pour la retransmission ou jette la trame erronée

❖ Correction des erreurs:

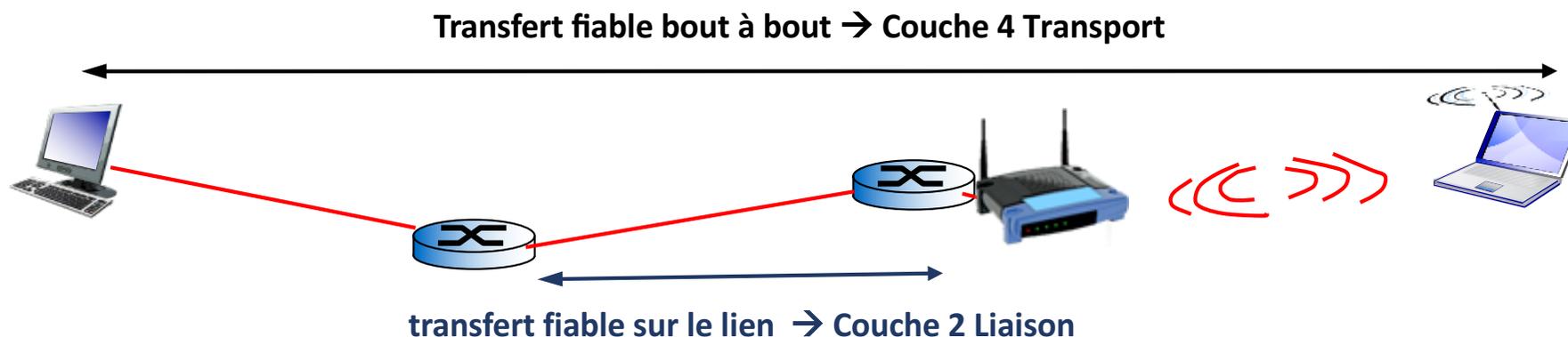
- Le récepteur identifie et corrige les erreurs de bits sans recourir à la retransmission

❖ *Semi-duplex / full-duplex*

- en *semi-duplex*, les nœuds situés aux deux extrémités du lien peuvent transmettre, mais pas en même temps
- en *duplex*, même en même temps

Fonctions de la couche liaison

- ❖ Historiquement, il y a eu beaucoup de protocoles de liaison qu'implémentaient à des degrés différents cette liste de fonctions.
- ❖ Actuellement, où le protocole l'Ethernet (IEEE 802.3) est le protocole dominant, fonctionnes telles que le *transfert fiable* ou le *contrôle de flux* sont réalisées par la *couche de transport* au niveau de tout le chemin (*bout à bout*),
 - c.-à-d. pas entre des deux nœuds adjacentes sinon entre les deux systèmes d'extrémité du chemin.
- ❖ On verra toutes les autres dans les suivants slides: *détection/correction d'erreur, contrôle d'accès au support, framing, adressage "MAC"*



Détection et correction d'erreurs

- ❖ Longtemps la détection et la correction d'erreurs relevaient de la couche de lien (niveau 2)
- ❖ En effet, la qualité des lignes physiques était très insuffisante pour obtenir des taux d'erreurs acceptables dans les trames de niveau 2
- ❖ La situation actuelle a très sensiblement changé pour deux raisons essentielles :
 - La *fiabilité* des lignes s'est beaucoup *accrue*. Le taux d'erreur est $< 10^{-9}$ et souvent moins encore. Ceci a été rendu possible par :
 - des techniques de codages plus efficaces
 - de nouveaux supports physiques filaires : fibre optique, ...
 - La *nature des applications* (*multimédia*, ...). Certaines ne tolèrent pas les pertes de temps associées aux reprises sur erreurs. La *perte* de quelques bits *ne change rien* quant à la qualité que l'oeil ou l'oreille peuvent percevoir. Des délais seraient, eux, beaucoup plus détectables

Détection et correction d'erreurs

- ❖ La détection et la correction restent toutefois indispensables :
 - sur des supports de mauvaise qualité (des ondes radioélectriques, ...)
 - pour des applications ne tolérant pas la moindre erreur (transfert de fichier, ...)
- ❖ Pour la détection/correction, deux grandes possibilités existent :
 1. Toujours envoyer *avec* des bits redondantes pour *détecter et corriger*
 2. Toujours envoyer *avec* des bits redondantes pour *détecter*, après retransmettre si erreur détectée
- ❖ Quel système choisir ?
 - la *détection/correction* exige un *accroissement* d'environ 50% de l'information transportée. Ainsi, pour envoyer 1.000 bits utiles en toute sécurité, il faut envoyer 1.500 bits
 - la *détection seule* se contente d'une augmentation de 16 à 32 bits. Ce n'est qu'en cas d'erreurs avérées, que la retransmission intégrale de l'information doit être faite

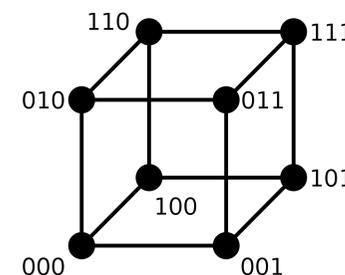
Détection et correction d'erreurs

- ❖ Le *point neutre* pour des trains de bits compris entre 1.000 et 10.000 bits est donc d'environ un taux d'erreurs de 10^{-4} :
 - si le taux d'erreurs est plus faible, une détection est utilisée
 - si le taux d'erreurs est plus élevé, la correction est plus intéressante
- ❖ Or, les supports ont quasiment tous un taux d'erreurs $< 10^{-4}$ → **La détection avec retransmission est donc majoritairement utilisée**
- ❖ La détection / correction d'erreurs suppose :
 - l'utilisation d'algorithmes complexes (temps de traitement non négligeable)
 - une augmentation importante de l'information à transporter (redondance)
- ❖ Tout d'abord, le système trivial :
 - bien que n'offrant que peu de garantie, une méthode consiste à envoyer 3 fois la même information et à choisir la plus probable :
 - envoi 3 bits à 1, réception 2 bits à 1 et 1 bit à 0
 - $P(\text{bonne valeur}=1) = 2/3$, $P(\text{bonne valeur}=0) = 1/3$
 - **On choisit 1 comme bonne valeur correcte**

Détection et correction d'erreurs

❖ Notion de code de *Hamming* :

- Pour pouvoir corriger les erreurs, il faut pouvoir distinguer les différents caractères émis, même s'ils sont erronés
- supposons un mini alphabet de 4 caractères : *00, 01, 10, 11*
- si une erreur se produit, le caractère est 'transformé' en un autre caractère (valide) de ce même alphabet et l'erreur passe inaperçue :
 - envoi de 00 et réception de 10. Le caractère reçu bien qu'erroné fait partie de l'alphabet
- Il nous faut donc *ajouter* de l'information :
 - 00 → **00000**
 - 01 → **01111**
 - 10 → **10110**
 - 11 → **11001**
- Si **une erreur** se produit, on compare la donnée transmise avec les caractères valides de l'alphabet. On en déduit le bon en faisant le rapprochement avec celui qui ressemble le plus :
 - Si 10000 reçu, le plus proche est **00000** (distance +1) car les autres possibilités **10110** et **11001** (distance +2) sont plus éloignées.



Détection et correction d'erreurs

❖ Notion de code de *Hamming* :

- Si **deux erreurs** se produisent sur le même caractère, il devient impossible, dans le contexte ci-dessus, de récupérer la valeur exacte.
- Exemple : Emission de *10110* et réception de **10101**. L'estimation du caractère correct n'est plus possible. Il y a deux options (l'une est la bonne) à la distance la plus petite(2):
 - **10110** → la bonne à distance 2
 - **11001** → mais celle-ci est encore un caractère valide à distance 2
- Soit $d(x,y)$ la distance entre deux caractères x et y de même longueur définie comme le nombre de bits (positions) différents, on définit la distance de *Hamming* comme étant :

$$d_H = \inf d(x,y)$$

où la borne inférieure s'applique à l'ensemble des caractères valides de l'alphabet

- c.-à-d., la distance la plus petite entre deux *caractères valides de l'alphabet*

Détection et correction d'erreurs

❖ Notion de code de *Hamming* :

- dans l'exemple ci-dessus, le calcul de d_H donne 3
- pour pouvoir corriger *une seule erreur*, il faut que les différents caractères valides du même alphabet satisfasse à $d_H = 3$, de sorte que, en cas d'erreur, la distance entre le caractère correct et le caractère erroné soit de 1
 - *Encore n'importe quel autre caractère valide est à distance 2 par définition*
- Pour corriger *2 erreurs à la fois*, la d_H doit valoir 5 :
 - le nouvel alphabet vaut alors :
 - $00 \rightarrow 00000000$
 - $01 \rightarrow 01111011$
 - $10 \rightarrow 10110101$
 - $11 \rightarrow 11001110$
- En recevant le caractère *10001010*, on en déduit que le caractère correct est *11001110* puisque c'est le seul caractère de notre alphabet à avoir une distance de *Hamming* de 2 avec le caractère erroné reçu :
 - $d(10001010, 11001110) = 2$
 - $d(10001010, x) > 2$ si $x \neq 11001110$

Détection et correction d'erreurs

- ❖ **Bits de parité**, un autre méthode que l'on peut déterminer à partir d'un caractère (souvent un octet)
 - le bit de parité est un bit supplémentaire ajouté au caractère « protégé »
 - il est calculé en sorte que la *somme des éléments binaires (nombre de bits à 1) modulo 2* soit égale à 0 ou à 1
 - Dans le cas **paire**, le bit de **parité** est 1 si le nombre de bits à 1 est **impaire**.
 - Dans le cas **impaire**, le bit de **parité** est 1 si le nombre de bits à 1 est **paire**.
- ❖ Exemple : soit le caractère 10011001, on pose un *parité paire*
 - il faut donc ajouter un bit valant 0 : 10011001 **0**
 - si une erreur se produit lors de la transmission :
 - 1**0**111001 **0**
 - alors, le nombre de bits à 1 n'étant pas pair, on détecte qu'une erreur s'est produite
 - problème :
 - Il faut ajouter un bit tous les 8 bits (12,5 % de charge)
 - deux erreurs sur le même octet ne sont pas détectables :
 - **0**1011001 0 passera inaperçu !!!!

Détection et correction d'erreurs

- ❖ **Cyclic Redundancy Checksum:** Une autre méthode, plus efficace, qui repose sur une division de polynômes :
 - les deux parties (émetteur, récepteur) se mettent d'accord sur un polynôme (ex. de degré 16 : $X^{16}+X^8+X^7+1$) : le *générateur*
 - à partir des éléments binaires de la trame notés a_i , $i = 0 \rightarrow M-1$, M étant le nombre de bits de la trame, on constitue le polynôme de degré $M - 1$:
 - $P(x) = a_0 + a_1x + \dots + a_{M-1}x^{M-1}$
- ❖ **Fonctionnement :**
 - l'émetteur divise le polynôme issu de sa trame, par le générateur
 - le reste de cette division est un polynôme dont le degré vaut au max. 15 :
 - $R(x) = r_0+r_1x+\dots+r_{15}x^{15}$
 - les valeurs binaires r_0, r_1, \dots, r_{15} sont placées par l'émetteur dans la partie de contrôle de la trame
 - à l'arrivée, le récepteur effectue le même travail
 - il compare le reste qu'il a calculé avec celui se trouvant dans la partie de détection
 - si les deux restes sont identiques, alors la transmission s'est bien passée

Détection et correction d'erreurs

❖ *Cyclic Redundancy Checksum*

❖ Cette méthode détecte quasiment toutes les erreurs, mais :

- si une erreur se glisse dans la partie de détection, on conclut à une erreur sur la partie donnée même si elle est correcte

❖ Exemple de division polynomiale :

- soit à transmettre 1101011011 → ceci donne $P(x) = X^9 + X^8 + X^6 + X^4 + X^3 + X^1 + X^0$
- soit le générateur 10011 → ceci donne $G(x) = X^4 + X^1 + X^0$
- le *résultat* de la division vaut → ceci donne $D(x) = X^9 + X^8 + X^3 + X^1 : 1100001010$
- le *reste* vaut alors : → ceci donne $R(x) = X^3 + X^2 + X^1, 1110$

❖ La zone de détection d'erreurs est communément appelée :

- *Cyclic Redundancy Checksum*

❖ Un autre nom courant est :

- *Frame Check Sequence*

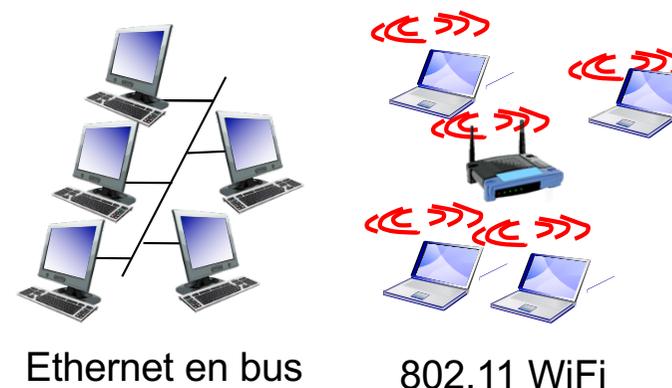
❖ largement utilisé dans la pratique (Ethernet, WiFi 802.11)

Contrôle d'accès au support (MAC)

❖ Deux types de "liens":

- Point à Point (*point-to-point*):
 - Le support physique est partagé uniquement par les deux nœuds directement connectés par le lien
 - Pas de collisions
 - p. ex. point à point entre *commutateur (switch)* Ethernet et un hôte

- Diffusion (*broadcast*)
 - Le support physique est partagé par plusieurs nœuds
 - Collision possibles
 - Ethernet à l'ancienne (avec des nœuds *connectés en bus en partageant le même câble*)
 - Réseau local sans fil WiFi 802.11 (les nœuds *partagent le spectre radioélectrique, typiquement la même fréquence*)



Contrôle d'accès au support (MAC)

❖ Canal de diffusion partagé unique

- Deux ou plusieurs transmissions simultanées par nœuds: brouillage
 - collision si un nœud reçoit deux signaux ou plus en même temps en provenance des deux autres machines.

❖ Protocole d'accès au support

- Algorithme distribué qui détermine comment les nœuds partagent le canal, c'est-à-dire détermine quand le nœud peut transmettre
- Communication sur comment partager le canal doit utiliser le canal lui-même !
 - pas de *canal hors bande* pour la coordination

❖ Le protocole d'accès au support idéale

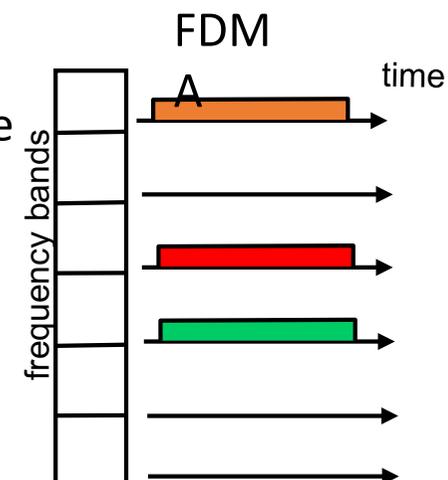
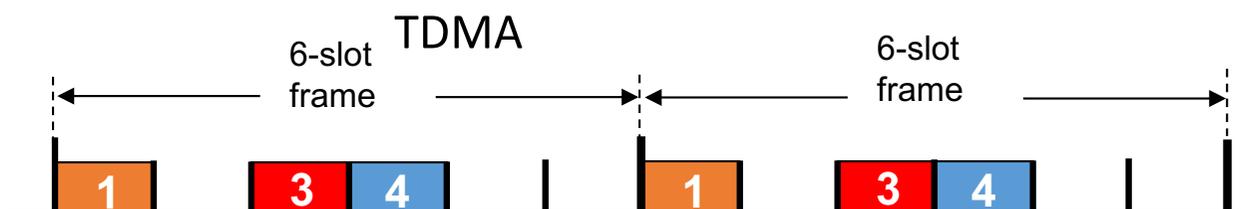
- **Données:** un canal de diffusion à débit R bps
- **Desiderata :**
 1. lorsqu'un nœud veut émettre, il peut envoyer au débit R .
 2. quand M nœuds veulent émettre, chacun peut envoyer au débit moyen R / M
 3. totalement décentralisé : aucun nœud spécial pour coordonner les transmissions, pas de synchronisation des horloges, des slots
 4. simple

Contrôle d'accès au support (MAC)

❖ Deux alternatives pour satisfaire nos desiderata

1. Soit partitionner le canal

- diviser le canal en N « morceaux » plus petits :
 - Slots temporels: **TDMA**: *time division multiple access*
 - Bandes des fréquences: **FDMA**: *frequency division multiple access*
- attribuer un morceau à un nœud pour une utilisation exclusive
- si nœud inactif, « morceaux » inutilisé → *gâchis* de ressources
- partage le canal efficacement et équitablement à **forte charge**
- **inefficace à faible charge** : retard dans l'accès au canal, bande passante $1 / N$ allouée même s'il n'y a qu'un seul nœud actif!



Contrôle d'accès au support (MAC)

❖ Deux alternatives pour satisfaire nos desiderata

2. Soit accéder de manière aléatoire au canal

- canal non divisé: n'importe qui peut accéder n'importe quand
- **efficace à faible charge** : un seul nœud peut utiliser pleinement le canal
- **charge élevée** : surcharge due aux collisions
- quand le nœud a un paquet à envoyer
 - transmettre à plein débit du canal R bps
 - pas de coordination a priori entre les nœuds
- deux ou plusieurs nœuds émetteurs → « collision »
- le protocole MAC à accès aléatoire spécifie :
 - comment détecter les collisions
 - comment se récupérer des collisions (par exemple, via des retransmissions différées)
- exemples de protocoles MAC à accès aléatoire :
 - Pure ALOHA
 - *Slotted* ALOHA
 - CSMA, CSMA / CD, CSMA / CA

Contrôle d'accès au support (MAC)

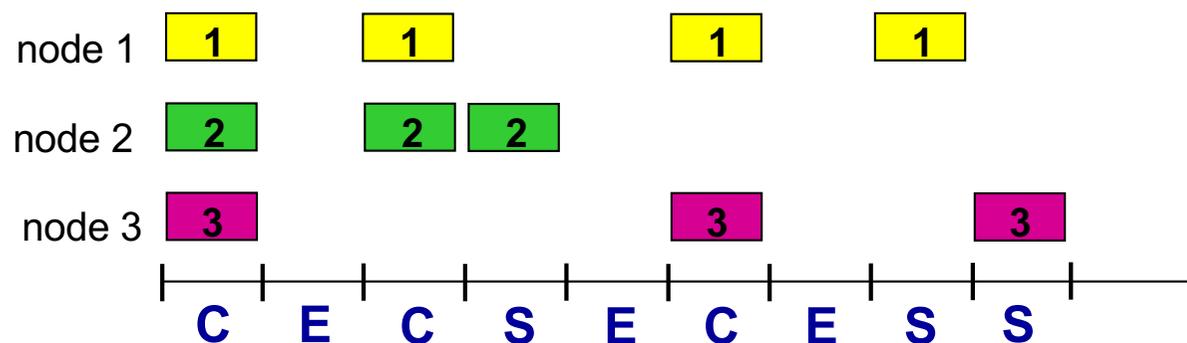
❖ Slotted ALOHA

▪ Hypothèses :

- tous les trames de la même taille
- temps divisé en *slots* de taille égale (temps nécessaire pour transmettre une trame)
- les nœuds commencent à ne transmettre qu'au début d'un *slot*
- les nœuds sont synchronisés
- si 2 nœuds ou plus transmettent dans un *slot*, tous les nœuds détectent une collision

▪ Opération :

- quand le nœud obtient une nouvelle trame, transmet dans le prochain *slot*
 - si aucune collision: le nœud peut envoyer une nouvelle trame dans le prochain *slot*
 - si collision: le nœud retransmet la trame dans chaque slot ultérieur avec probabilité p jusqu'au succès



Contrôle d'accès au support (MAC)

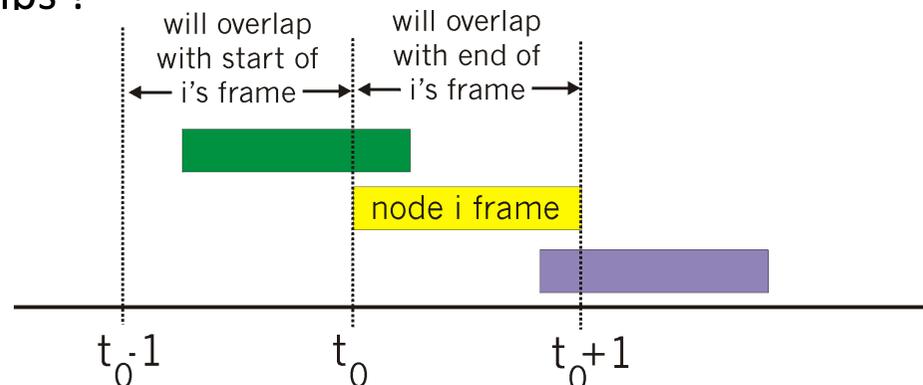
❖ *Slotted* ALOHA

- **Avantages :**
 - un seul nœud actif peut transmettre en continu au débit maximal du canal
 - très décentralisé: seuls les *slots* dans les nœuds doivent être synchronisés
 - simple
- **Inconvénients:**
 - collisions, gaspillage de *slots*, *slots* inactifs
 - synchronisation d'horloge nécessaire
 - la "**zone de silence**" imposée aux autres pour être sûre que "la trame passe" est **au moins *la durée de l'émission d'une trame (un slot) t***
- **Performance**
 - au mieux: canal utilisé pour transmissions sans collisions 37% de temps !

Contrôle d'accès au support (MAC)

❖ Pure (*unslotted*) ALOHA

- plus simple, pas de synchronisation
- quand la trame arrive en premier, elle est transmis immédiatement, pas de slots
- la probabilité de collision augmente : la trame envoyée à t_0 entre en collision avec d'autres trames envoyées dans l'intervalle $[t_0-1, t_0 + 1]$
- donc, il faut imposer une "**zone de silence**" aux autres pour être sure que "la trame passe" d'au moins deux fois la durée de l'émission d'une trame
 - **C'est le double qu'avant**
- Performance divisé par deux: au mieux: canal utilisé pour transmissions sans collisions 18% de temps !



Contrôle d'accès au support (MAC)

❖ CSMA (*carrier sense multiple access*)

- ***écouter avant d'émettre :***
 - si le canal détecté est inactif: transmettre la trame entière
 - si le canal détecté est occupé, différer la transmission
- Analogie humaine: n'interrompez pas les autres!
- Comme pour ALOHA, il n'y a pas de contrôle centralisé pour accéder au canal, ***MAIS, maintenant il y de la politesse !***
- Des collisions peuvent encore se produire: le délai de propagation signifie que deux nœuds ne peuvent pas entendre la transmission l'un de l'autre.
- **Collision:** perte totale de temps de transmission de paquets
 - la distance et le délai de propagation jouent un rôle dans la détermination de la probabilité de collision

Contrôle d'accès au support (MAC)

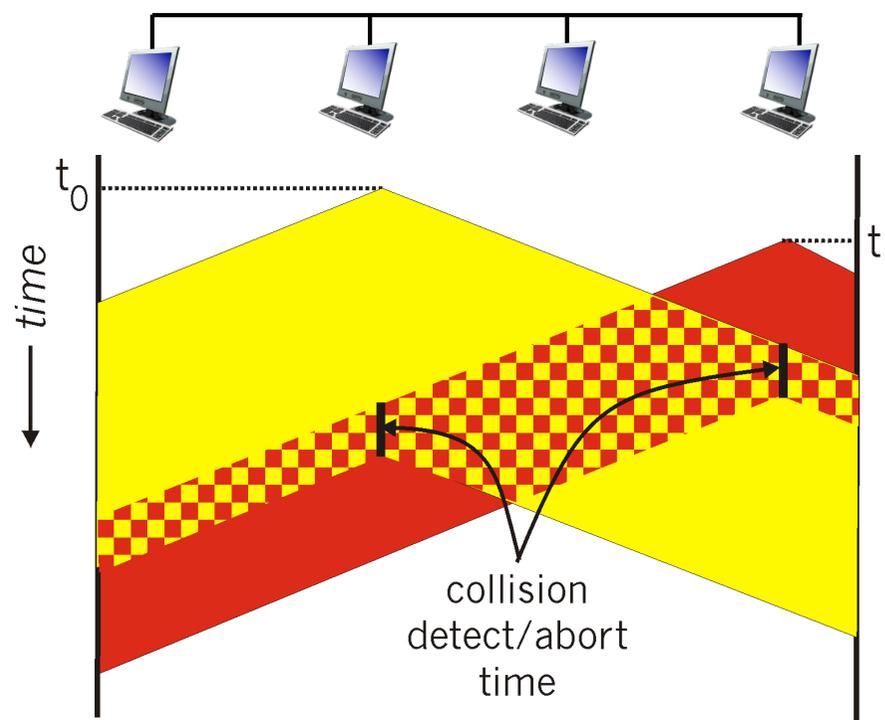
❖ CSMA avec CD (*collision detection*)

- Afin de réduire plus encore, la bande passante gâchée par les collisions, il lui fut ajouté une autre caractéristique : « *écouter pendant transmission* ».
 - On devient plus poli !!!
 - collisions détectées dans un délai plus court qu'avant: on n'attend pas $T(\text{slotted ALOHA})$ ou $2T(\text{ALOHA})$
 - transmissions en collision interrompues, réduisant ainsi le gaspillage de canaux
- Détection de collision:
 - facile dans les réseaux locaux câblés: mesurez l'intensité du signal, comparez les signaux transmis et reçus
 - difficile dans les LAN sans fil (WiFi): la force du signal reçu est dépassée par la force de la transmission locale → Ici CSMA/CA (Congestion Avoidance), on essaie d'éviter absolument des collisions parce que on ne peut pas les détecter !!!
- Performance peut monter jusqu'à 100% en dépendant du délai de propagation et la durée de transmission d'une trame

Contrôle d'accès au support (MAC)

❖ Carrier Sense Multiple Access / Collision Detection (CSMA/CD) :

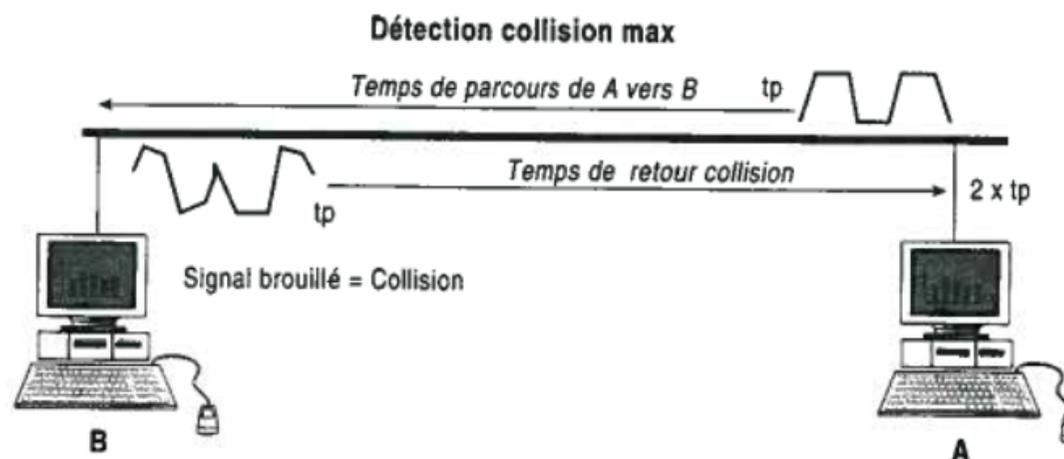
- Chaque hôte est à l'écoute et doivent attendre que le media soit libre («silence media») avant de pouvoir émettre.
- Quand le canal est libre, un hôte peut émettre.
- Quand 2 hôtes tentent d'émettre simultanément, une **collision** se produit. Dans cette situation, les 2 hôtes perçoivent la collision et envoient un signal de collision qui empêche l'émission pendant une durée aléatoire. tiré dans un intervalle déterminé par **l'algorithme de Back-off**:
 - après la m -ième collision, hôte choisit K au hasard parmi $\{0,1,2,\dots, 2^{m-1}\}$, et il attend K fois 512 bits (64 bytes)
 - intervalle de Back-off plus long avec plus de collisions



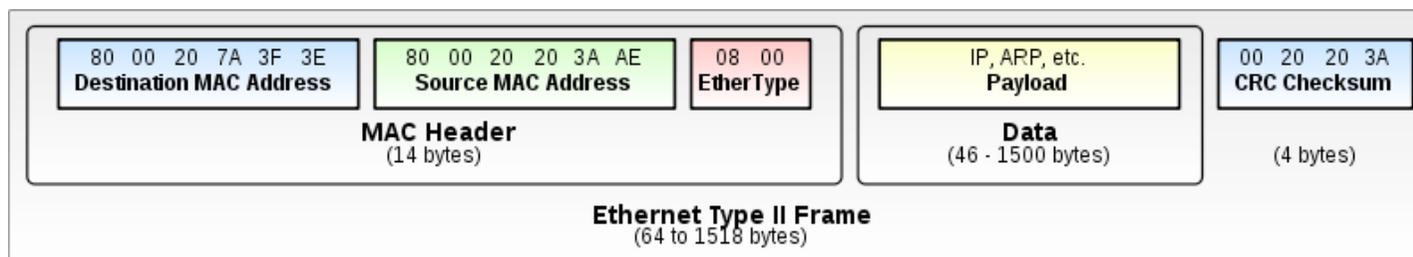
Contrôle d'accès au support (MAC)

❖ *Carrier Sense Multiple Access / Collision Detection (CSMA/CD)* :

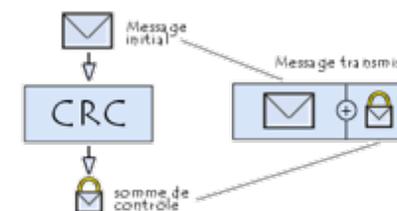
- La station A émet au temps T_A . Supposons que B émette juste avant que le signal de A lui parvienne. Le signal brouillé revient vers A.
- Pour que A « voit » la brouillage, il faut qu'il émette encore quand le signal de retour l'atteindra :
 - Soit *2 fois le temps t_p de propagation (Aller/Retour) = 34 \mu s* pour une distance max. de **2.5 km**
 - Avec une marge de sécurité, cela donne $51,2 \mu s \rightarrow$ soit *une trame minimal de 64 octets à 10Mb/s* afin de pouvoir détecter des collisions



Format de trame Ethernet

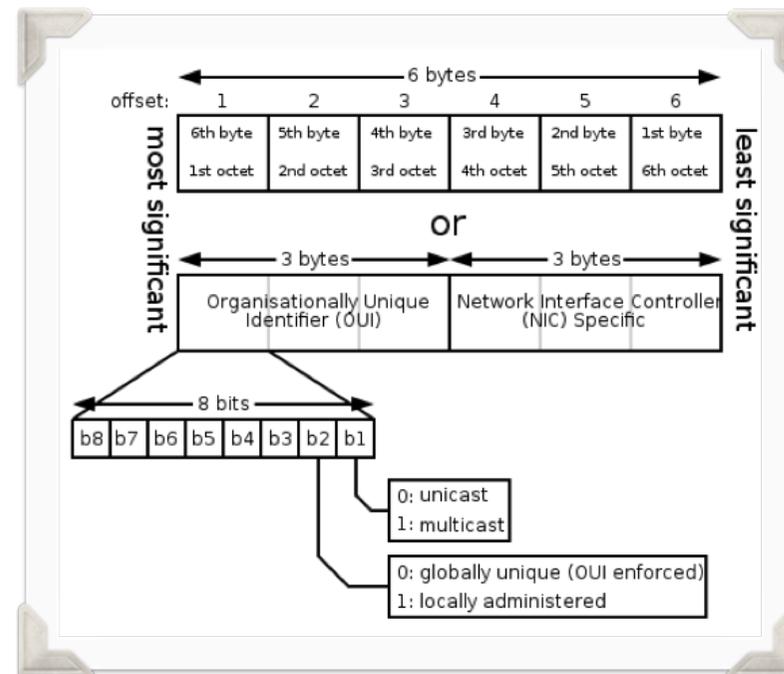


- ❖ Chaque *paquet couche réseaux* est encapsulé dans une *trame Ethernet* (ou fragmenté au besoin) :
 - **Préambule** : 7 octets avec patron 10101010 suivi d'un avec 10101011 = *start of frame delimiter*
 - **Adresses destination, Adresse source** sur 2x6 octets
 - **Type paquet couche réseaux** : 0x0800 IPv4, 0x86DD ipv6, 0x0806 arp, 0x8035 rarp
 - **CRC, Cyclic redundancy check** : puissant code de détection d'erreurs
 - CRC32 de polynôme générateur : $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$
 - On calcule le reste de la division Euclidienne entre les données et le CRC32.
- ❖ Remplissage PAD : trame 64 octets min (hors préambule)
- ❖ Taille max: 1518 octets



Adresses MAC

- ❖ Adresse sur 6 octets en hexadécimal (48 bits), p. ex.: 00:22:41:36:00:41
- ❖ Octet → hexadécimal: (0x5E = 0101 1110)
- ❖ Brûlés dans NIC (identifiant physique de l'interface)
- ❖ Multicast: adresses dest. débute par 1
- ❖ Broadcast: adresses dest. avec tous bits à 1



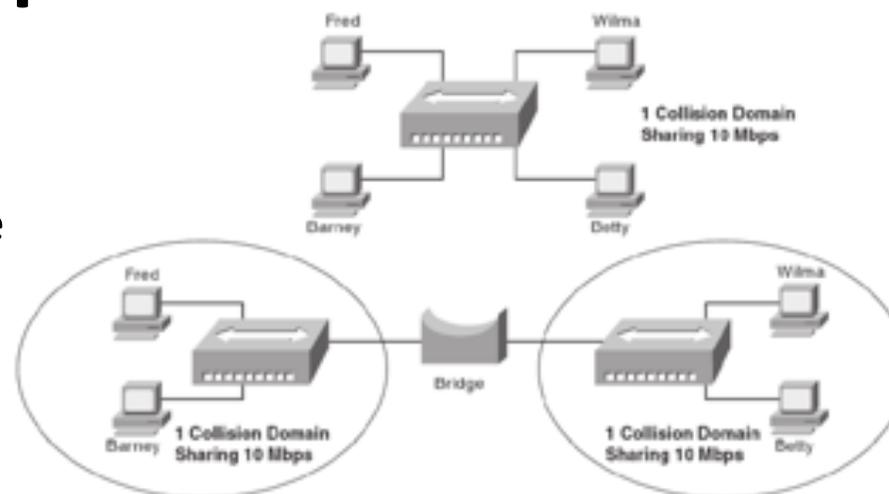
<http://standards.ieee.org/regauth/oui/index.shtml>

- ❖ Utilisés « localement » pour amener une trame d'une interface à une autre interface physique connecté (dans le même sous-réseau IP)
- ❖ Fabricant achète une partie de la plage de l'adressage MAC (pour assurer l'unicité), p. ex. Apple 00:22:41:xx:yy:zz

Types de réseaux couche liaison selon dispositif

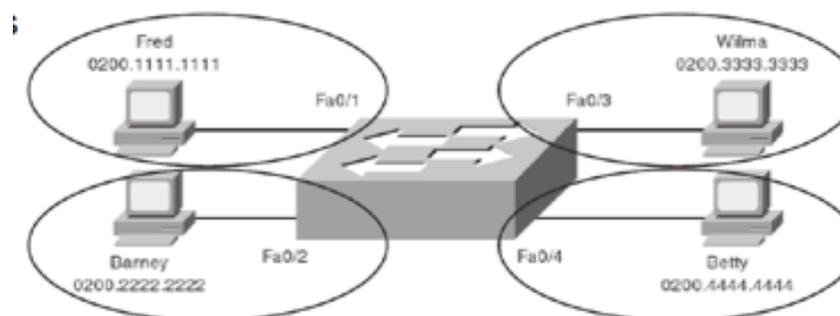
❖ Réseau concentré / ponté

- Avec hubs / ponts (*bridges*)
- *Hub* : ne préserve pas la bande passante
- *Bridge*: préserve la bande passante



❖ Réseau commuté

- Avec commutateurs (*switches*)
- *Switch* : préserve la bande passante
- Et, *en plus*, trames sont *commutées*



Fonctionnement hub

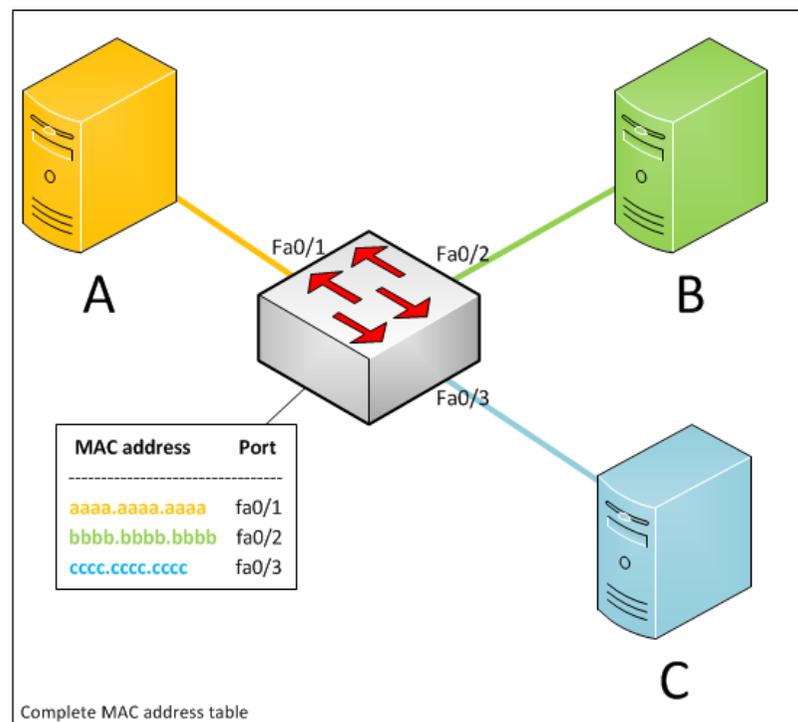
- ❖ Quand un paquet est reçu, il est propagé sur toutes les interfaces sauf celle de l'émetteur.
- ❖ Hub ne délimite ni les *domaines de collision* ni les *domaines de broadcast* :
 - **Domaine de collision** est une région du réseau au sein de laquelle les hôtes partagent l'accès au media.
 - **Domaine de broadcast** : aire logique du réseau où tout ordinateur connecté peut directement transmettre à tous les autres ordinateurs du même réseau, sans devoir passer par un routeur.
- ❖ Permet à la carte réseau d'un hôte d'accepter tous les paquets qu'elle reçoit, même s'ils ne lui sont pas destinés (mode de promiscuité).
- ❖ Détection du mode de promiscuité:
 - augmentation charge de l'hôte qui traite tous les paquets et
 - augmente la latence du réseau
 - détection par `detectpromisc`

Du hub au switch

- ❖ Réseau avec beaucoup d'hôtes, problèmes de performance avec hubs:
 - **disponibilité:** partage de la bande passante; un hôte peut monopoliser tout le trafic (gros transfert)
 - **latence:** (temps nécessaire à un paquet pour atteindre sa destination). *Avec des hubs on attend une opportunité de transmission pour ne pas causer de collision (RAPPELER MAC).* La latence croît en fonction du nombre d'hôtes du réseau.
 - **défaillance:** hub plus sensible aux pannes ou aux mauvaises configuration de la vitesse de transmission
- ❖ Un *switch* (commutateur) résout ces problèmes en délimitant les *domaines de collision*
 - Chaque hôte connecté dispose de toute la bande passante.
 - Un paquet qui arrive sur un port du switch n'est retransmis que sur le port auquel le destinataire est connecté
 - Signaux de collision non retransmis par les switches

Fonctionnement switch

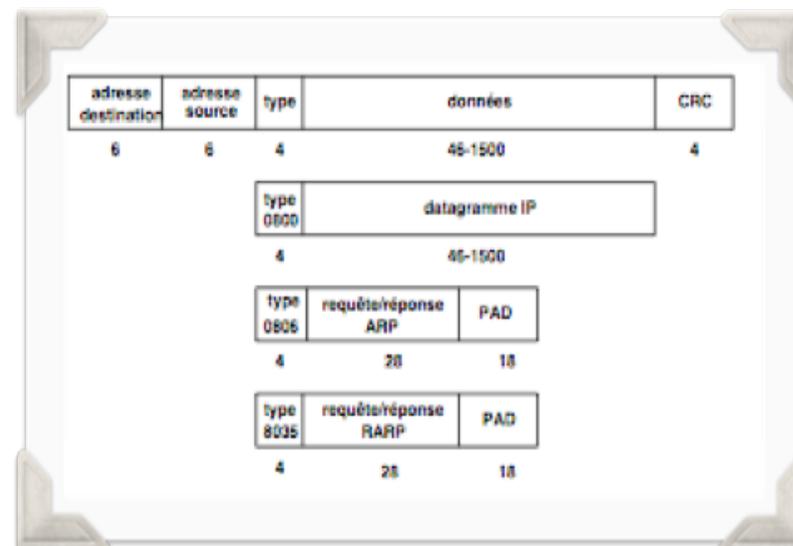
- ❖ Un paquet qui arrive dans le switch est mis dans un buffer. L'adresse MAC de destination du paquet est lue et comparée à la liste des MAC connues rangées dans la **table de lookup**.
- ❖ 3 modes d'acheminement:
 - **cut through**: lecture des 6 octets MAC dest. et routage direct sans traitement vers le port (interface) du destinataire
 - **store and forward**: mise en mémoire et traitement du paquet avant son acheminement (rejeter les paquets mal formés, gérer les messages de collision)
 - **fragment free**: analogue au *cut through* mais lit les 64 premiers octets (longueur min de la trame) avant le routage. (cela limite les erreurs de collision qui arrivent souvent sur les 64 premiers octets)
- ❖ Le mode le plus utilisé est le **store and forward**.



<https://cdynamicprogramming.blogspot.fr/p/mac-learning.html>

Protocole ARP/RARP [RFC 826]

- ❖ *Address Resolution Protocol* (ARP) [RFC 826] sert à trouver l'adresse matérielle correspondant à l'IP de destination
- ❖ La version *reverse* (RARP) sert à trouver l'adresse logique (IP) correspondant à l'adresse physique (MAC)
- ❖ Protocoles qui permettent d'assurer la correspondance entre adresse IP (logique) et adresse physique (MAC)
- ❖ Assure traduction adresses de niveau 3 (IP) en adresses de niveau 2 (MAC) (n° OSI)
- ❖ Chaque paquet (R)ARP est encapsulé dans une trame Ethernet
 - RFC894 précise l'encapsulation IP dans eth.
 - ARP : Ethertype 0x0806 dans la trame ethernet
 - RARP: Ethertype 0x0835 dans la trame ethernet



Protocole ARP/RARP [RFC 826]

- ❖ **Tableau ARP**: chaque nœud IP (hôte, routeur) sur le réseau local maintient un tableau
 - Mappages d'adresses IP / MAC pour certains nœuds de réseau local (au min, son propre mapping):
`<Adresse IP; Adresse Mac; TTL>`
 - TTL (Time To Live): temps après lequel le mappage d'adresses sera oublié (généralement 20 min)
 - Commande `arp -a`

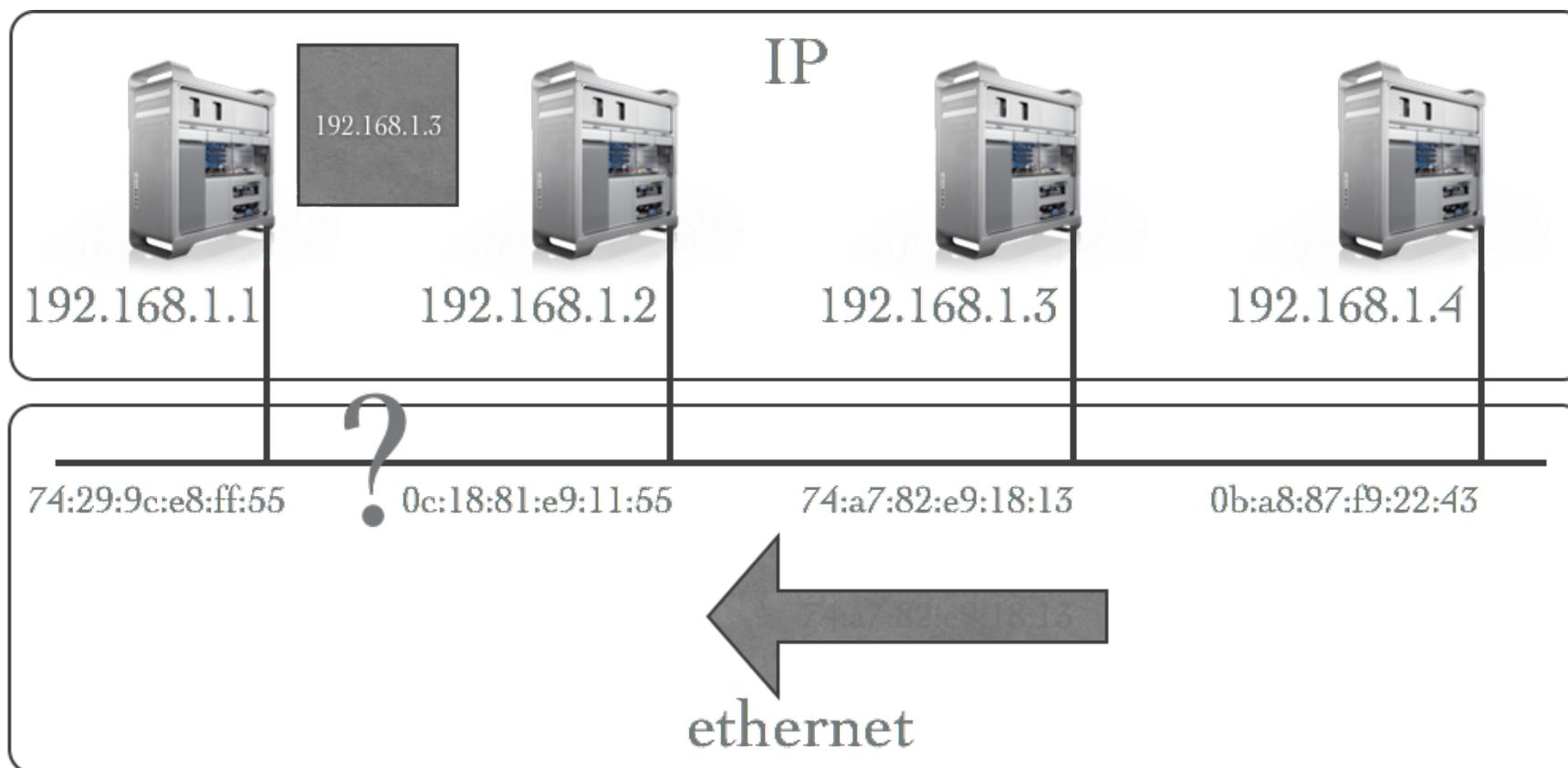
```
Ramons-MacBook-Air:~ raparicio$ arp -a  
? (192.168.1.254) at f4:ca:e5:44:79:d1 on en0 ifscope [ethernet]
```

Protocole ARP/RARP [RFC 826]

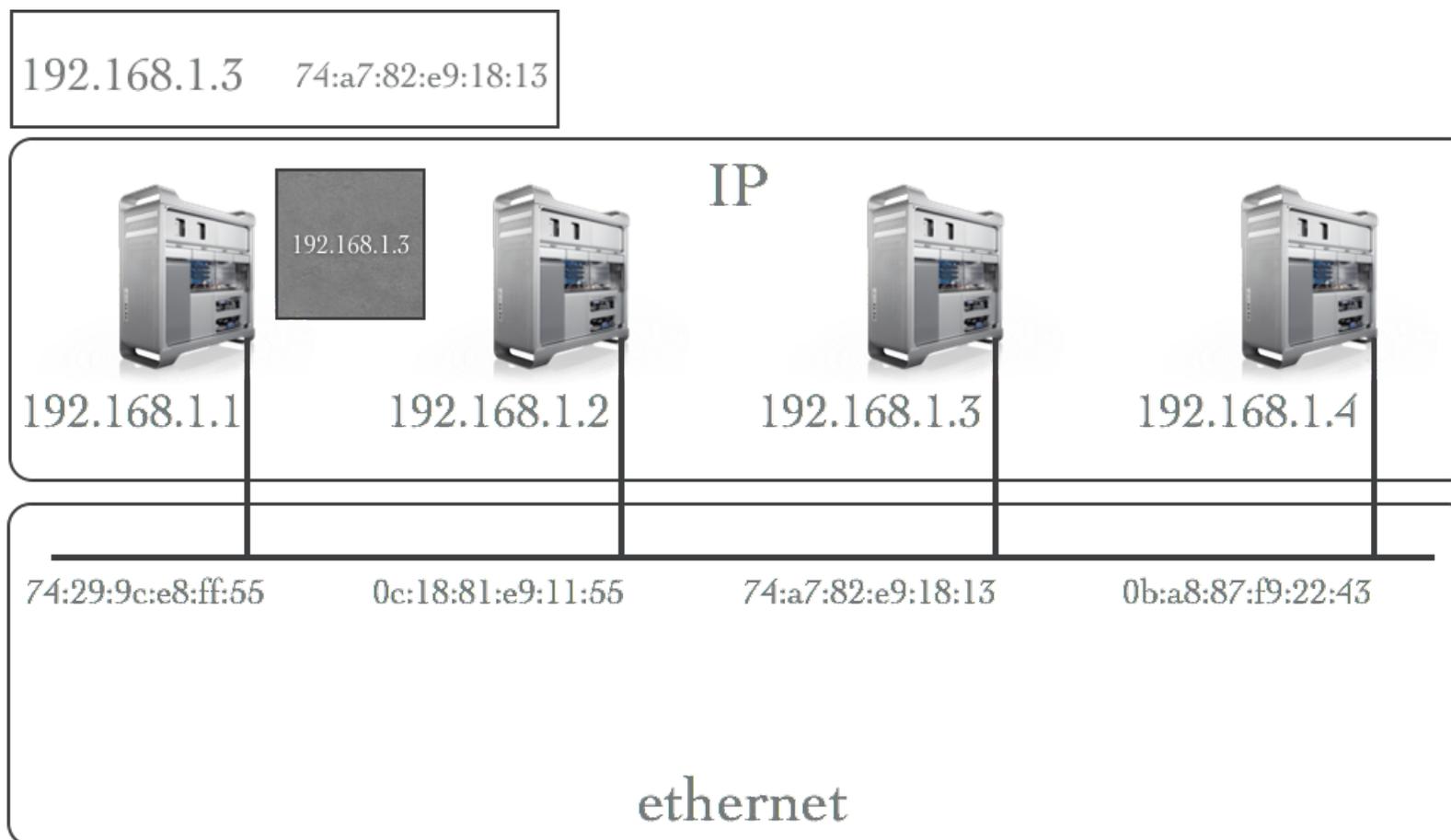
❖ Protocole ARP :

- A veut envoyer un datagramme à B
 - L'adresse MAC de B ne figure pas dans la table ARP de A.
- A diffuse un paquet de requête ARP, contenant l'adresse IP de B
 - adresse MAC de destination = FF-FF-FF-FF-FF-FF
 - tous les nœuds du réseau local reçoivent la requête ARP
- Evidemment, B reçoit le paquet ARP, répond à A avec son adresse MAC (B)
 - trame de réponse envoyée à l'adresse MAC de A de façon unicast
- A met en cache la paire d'adresses IP-MAC dans sa table ARP jusqu'à ce que les informations deviennent anciennes (timeout dépassé)
 - Les informations expirent (disparaissent) sauf si actualisées
- Aussi, des messages ARP gratuits (*gratuitous ARP*) sont envoyées au démarrage de certains systèmes d'exploitation pour annoncer aux machines sa paire d'adresses IP-MAC
- ARP est «plug-and-play» :
 - Les nœuds créent leurs tables ARP sans intervention de l'administrateur du réseau.

Protocole ARP/RARP [RFC 826]

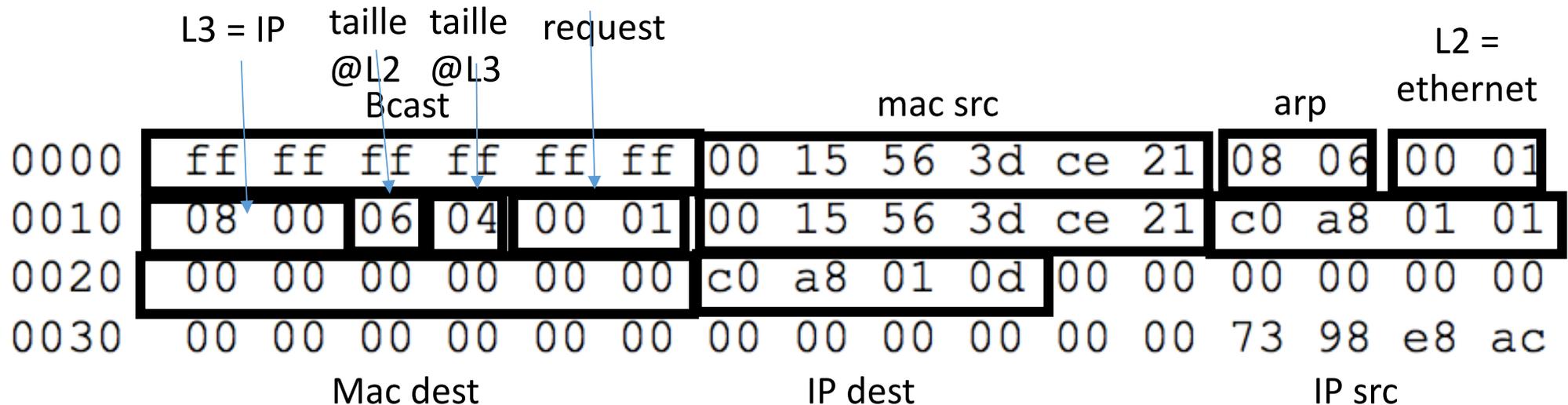


Protocole ARP/RARP [RFC 826]



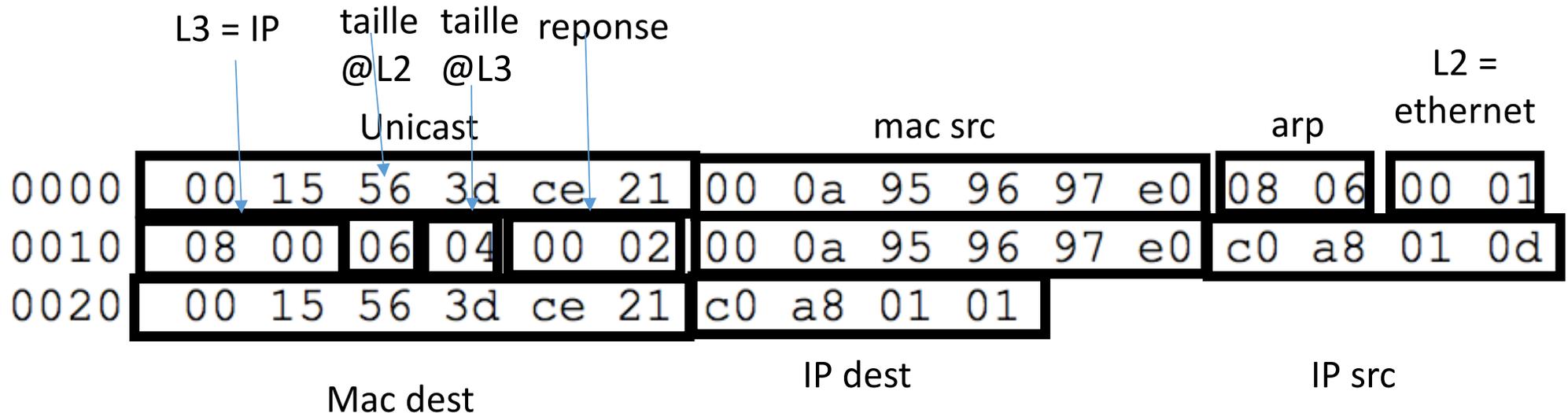
Protocole ARP/RARP [RFC 826]

- ❖ Trame ARP-Requête: Broadcast : on ne connaît pas encore la destination



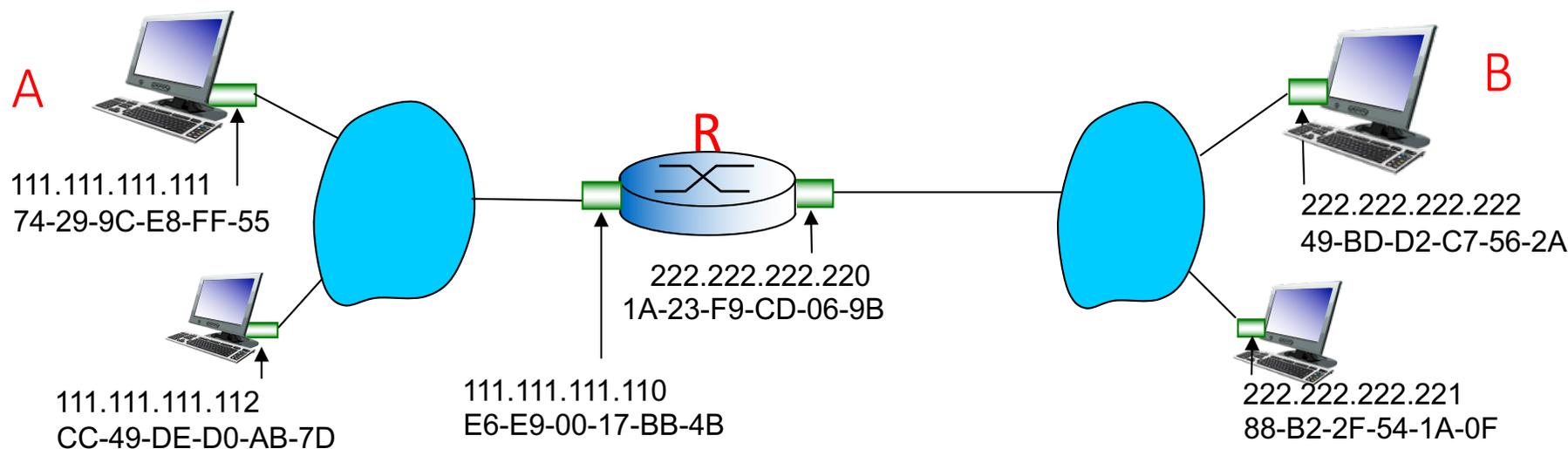
Protocole ARP/RARP [RFC 826]

- ❖ Trame ARP – Réponse: Unicast : On connaît la machine qui avait demandé en premier lieu



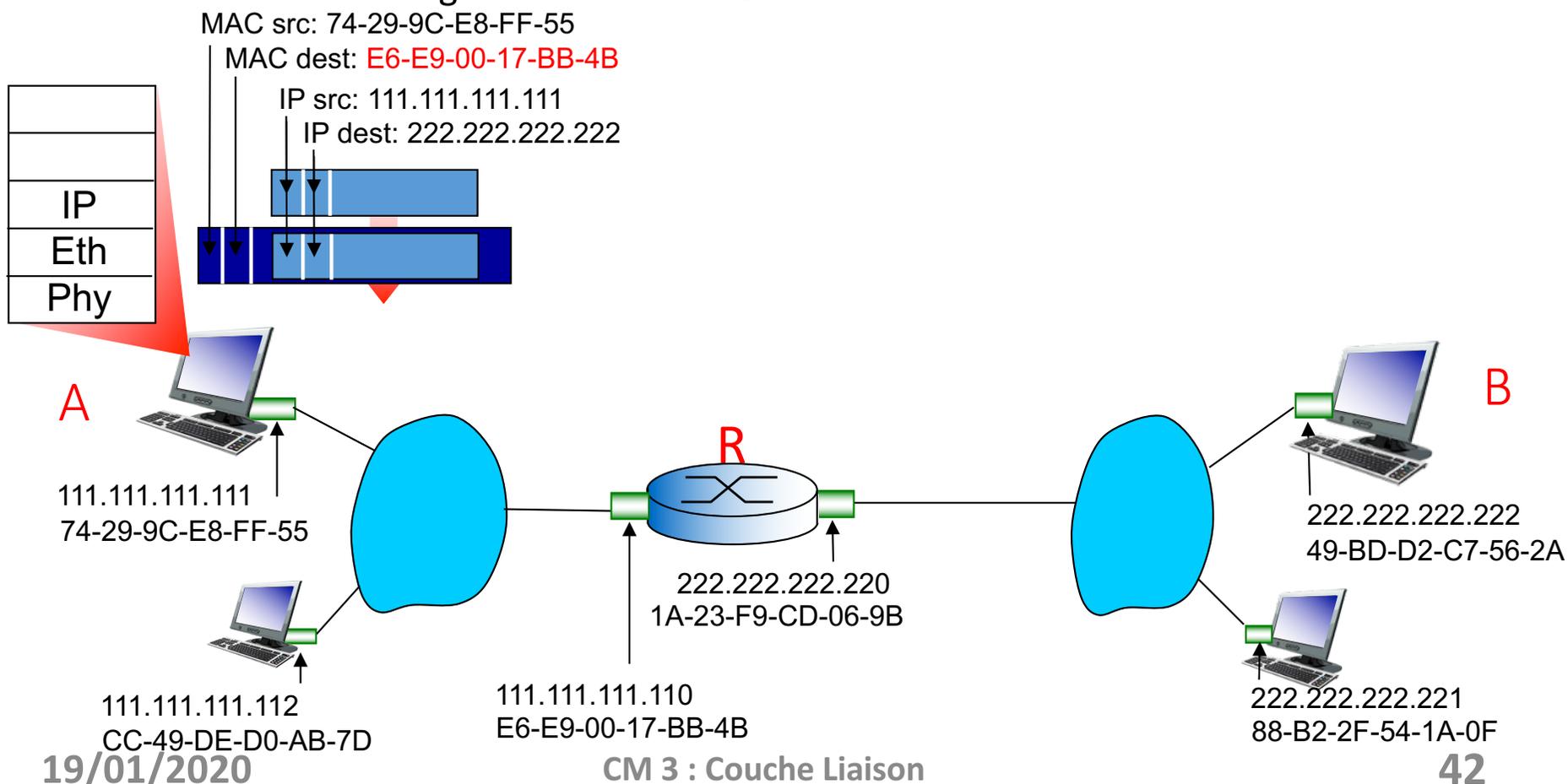
Procédure pas à pas: envoyez un datagramme de A à B via R

- ❖ se concentrer sur l'adressage - au niveau IP (datagramme) et MAC (trame)
- ❖ supposer que A connaît l'adresse IP de B
- ❖ supposons que A connaisse l'adresse IP du premier routeur de saut, R (comment ?)
- ❖ supposer que l'adresse MAC de R est connue (comment ?)



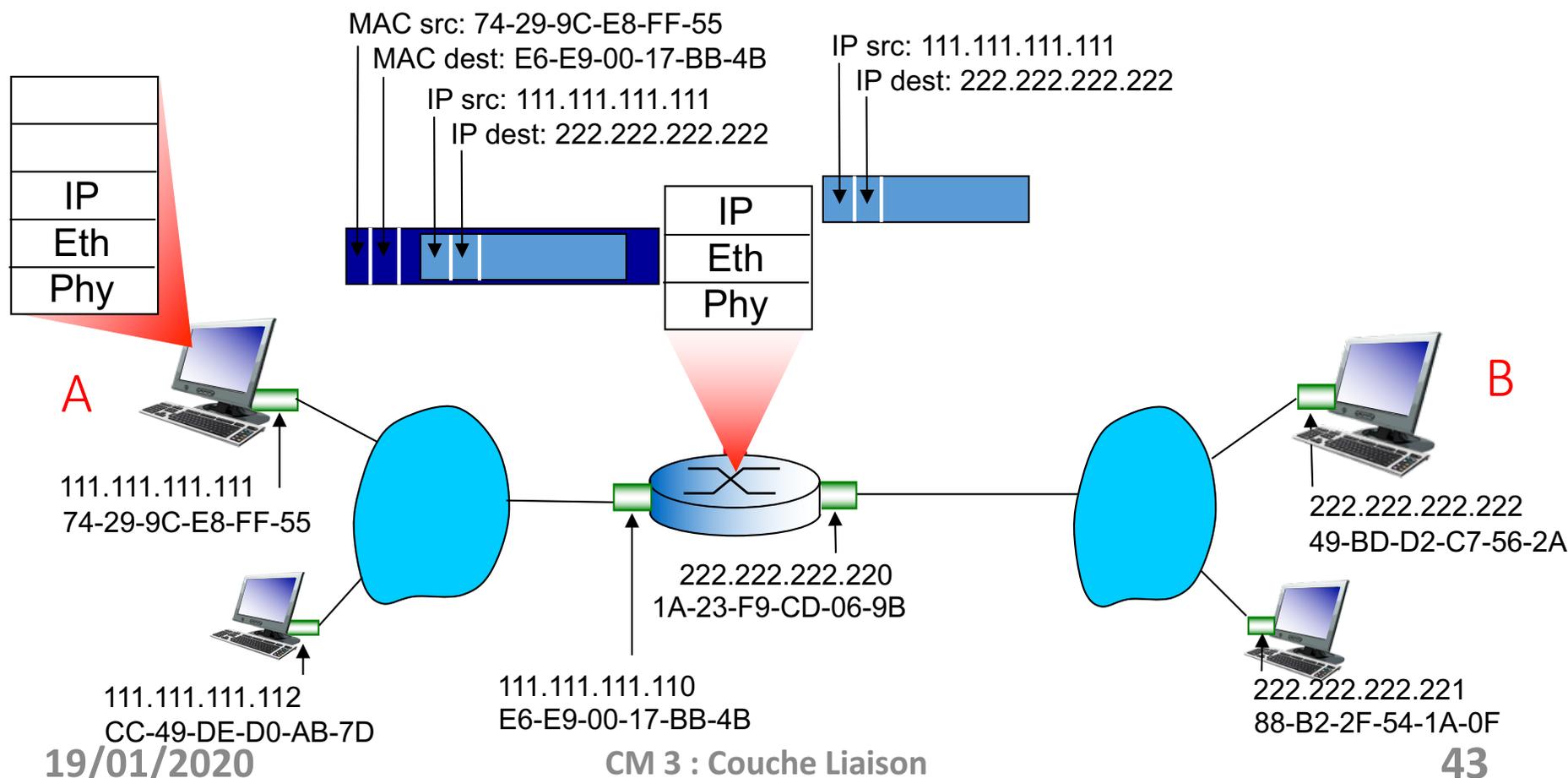
Procédure pas à pas: envoyez un datagramme de A à B via R

- ❖ A crée un datagramme IP avec les adresses IP <source A, dest. B >
- ❖ A crée une trame de couche liaison avec l'adresse MAC de R comme destination, la trame contient le datagramme IP de A à B



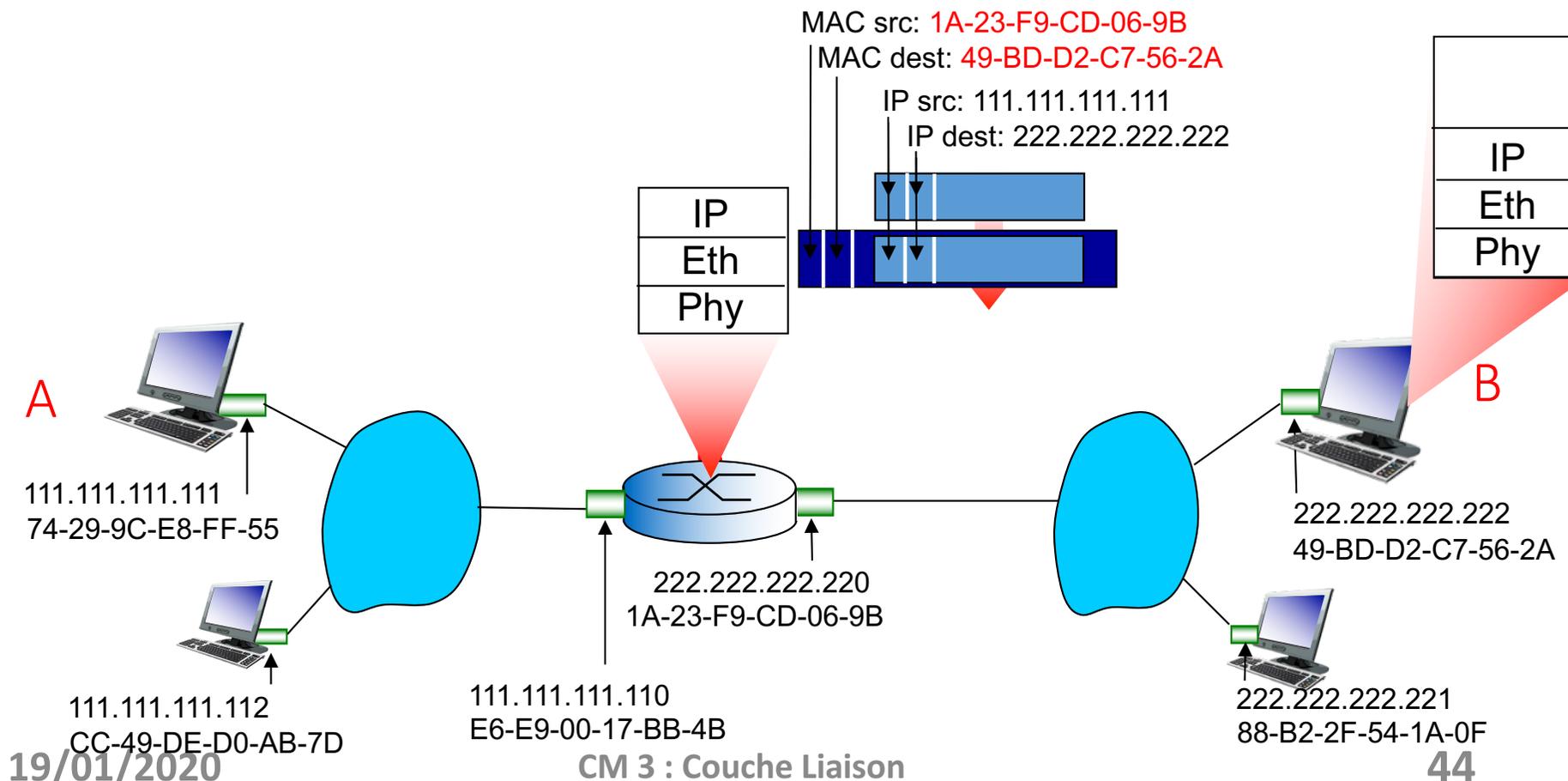
Procédure pas à pas: envoyez un datagramme de A à B via R

- ❖ Trame envoyée de A à R
- ❖ Trame reçue en R, datagramme enlevé, transmis à la couche IP



Procédure pas à pas: envoyez un datagramme de A à B via R

- ❖ R transmet le datagramme avec les adresses IP <source A, la destination B>
- ❖ R crée une trame de couche liaison avec l'adresse MAC de B comme destination, la trame contient le datagramme IP de A à B encore une fois



Procédure pas à pas: envoyez un datagramme de A à B via R

- ❖ R transmet le datagramme avec les adresses IP <source A, la destination B>
- ❖ R crée une trame de couche liaison avec l'adresse MAC de B comme destination, la trame contient le datagramme IP de A à B encore une fois

