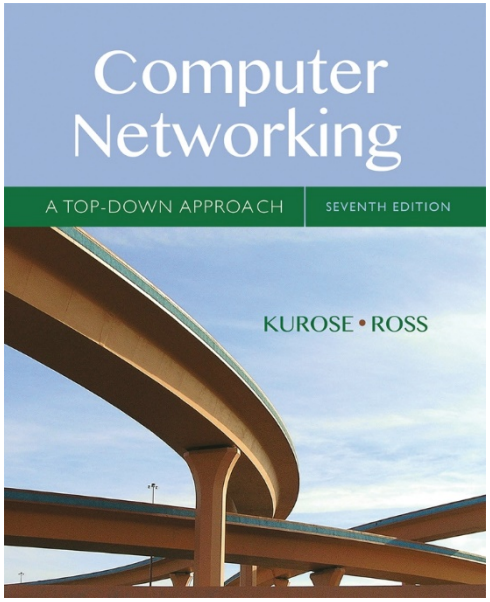
Chapitre 11
La couche application



Computer Networking: A Top Down Approach
7ème édition
Jim Kurose, Keith Ross
Addison-Wesley
2017

Chapitre II: plan

2.1 principes des applications réseaux

2.2 Web et HTTP

2.3 FTP

2.4 courriel

- SMTP, POP3, IMAP

2.5 DNS

2.6 applications P2P

DNS: domain name system

personne: plusieurs id:

- NAS, nom, passeport

terminaux, routeurs:

- adresse IP (32 bit) – pour adresser les datagrammes
- "nom", ex., www.yahoo.com – pour les utilisateurs

Q: comment faire correspondre une adresse IP à un nom, et vice versa?

Domain Name System:

- *base de données distribuée* implémenté dans une structure hiérarchique de serveurs de noms
- *protocole de couche application:* hôtes et serveurs de noms communiquent pour traduire les noms

DNS: services, structure

services DNS

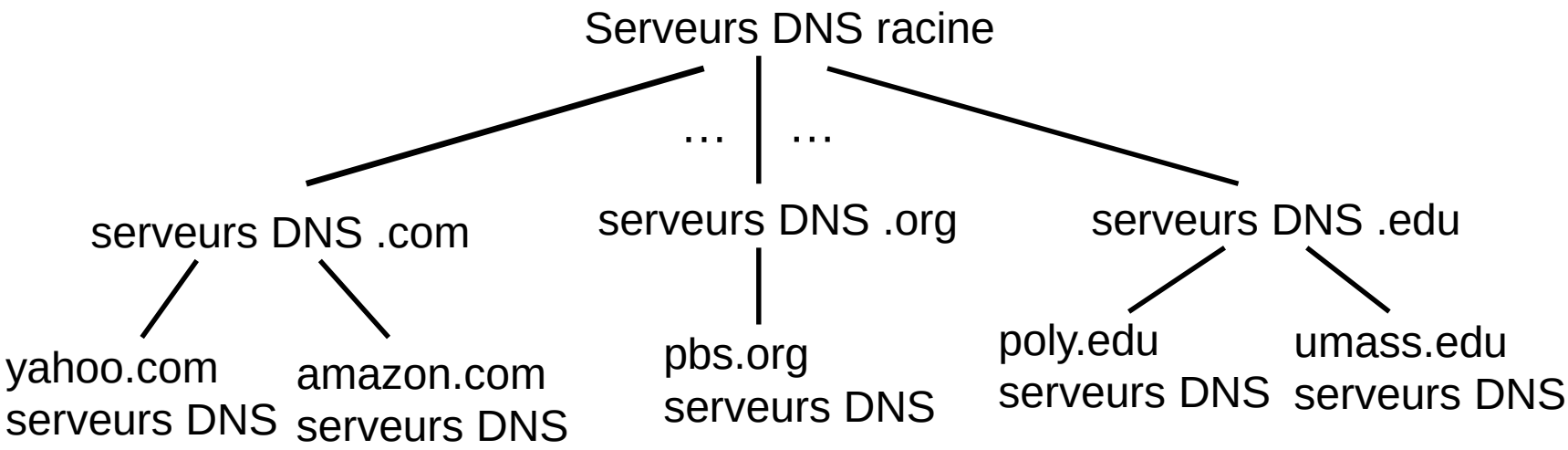
- traduction nom de hôte vers adresse IP
- traduction d'alias de serveurs web
  - canonique et alias
- traduction d'alias de serveurs mail
- répartition de charges
  - par réplication: plusieurs adresses IP correspondent au même nom
  - réponse avec rotation

pourquoi DNS n'est pas centralisé?

- fragilité d'un site central unique
- volume de trafic
- base de données centralisée trop éloignée
- maintenance

R: problème d'extensibilité!

DNS: base de données distribuée et hiérarchique

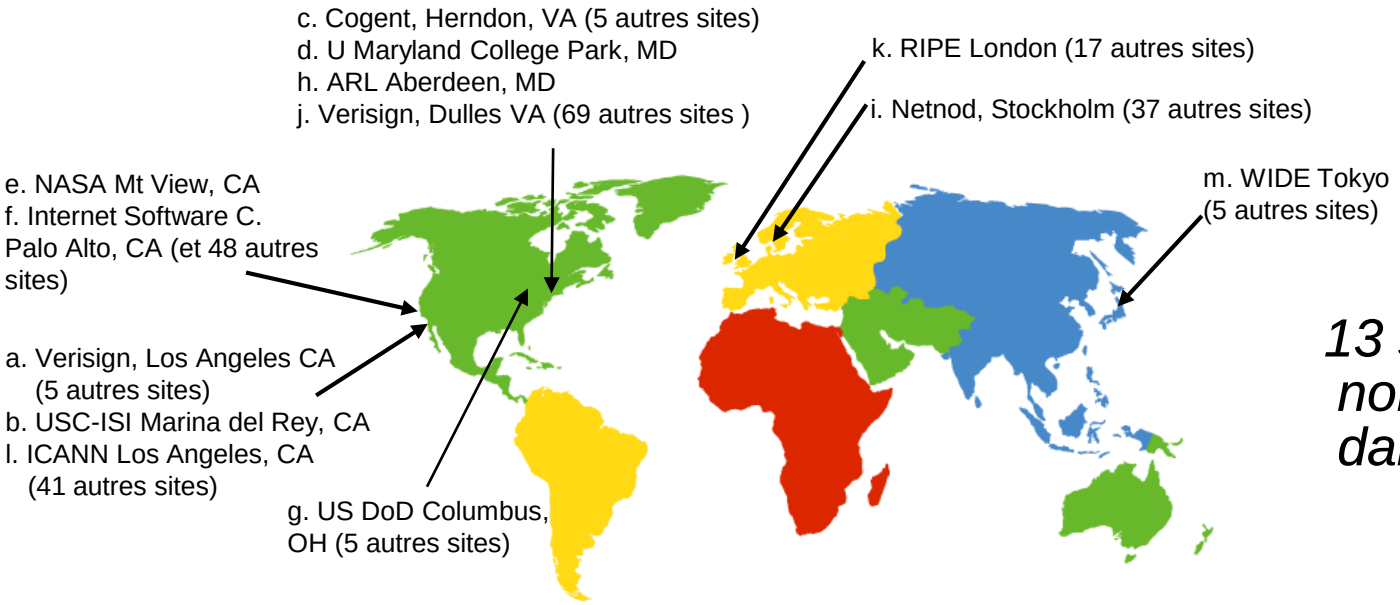


client veut l'@ IP de www.amazon.com; 1 ère possibilité:

- client contacte le serveur racine pour trouver le serveur DNS .com
- client contacte le serveur DNS .com pour trouver le serveur DNS amazon.com
- client contacte le serveur DNS amazon.com pour trouver le serveur DNS www.amazon.com

DNS: serveurs de noms “racine”

- contacté par un serveur de noms local qui ne peut satisfaire une requête
- serveur de noms “racine”:
  - contacte le serveur de nom de source autorisé s’il ne dispose pas de la traduction
  - obtient la traduction et la retourne au serveur de noms local



13 serveurs de noms « racine » dans le monde

TLD, serveurs de source autorisée

serveurs domaine de premier niveau (TLD) :

- responsable pour com, org, net, edu, aero, jobs, et uk, fr, ca, jp,…
- Network Solutions maintient les serveurs des TLD .com

serveurs de nom de source autorisée:

- Les institutions possèdent leurs propres serveurs DNS, et fournissent la correspondance entre adresse IP et nom d'hôte
- maintenue par l'institution ou un fournisseur de service

Serveurs de noms locaux

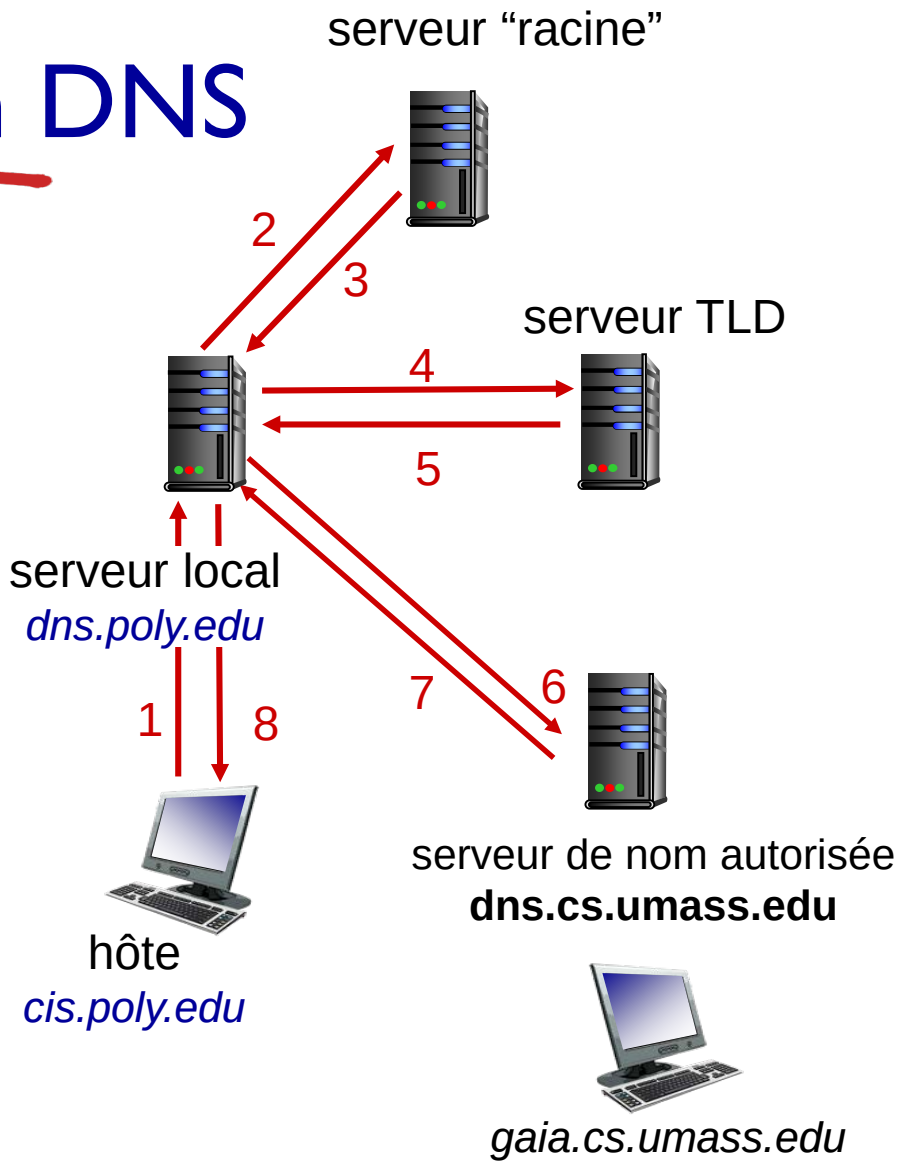
- n'appartient obligatoirement à l'hiérarchie
- chaque FAI (résidentiel, entreprise, université) possède un tel serveur
  - appelé aussi "default name server"
- lorsqu'un hôte formule une requête, elle est envoyée à son serveur DNS local
  - dispose d'un cache contenant les traductions récentes
  - fonctionne comme proxy et transfère les requêtes vers l'hiérarchie

Exemple de la traduction de nom DNS

- hôte (cis.poly.edu) veut l'adresse IP de (gaia.cs.umass.edu)

requête itérative:

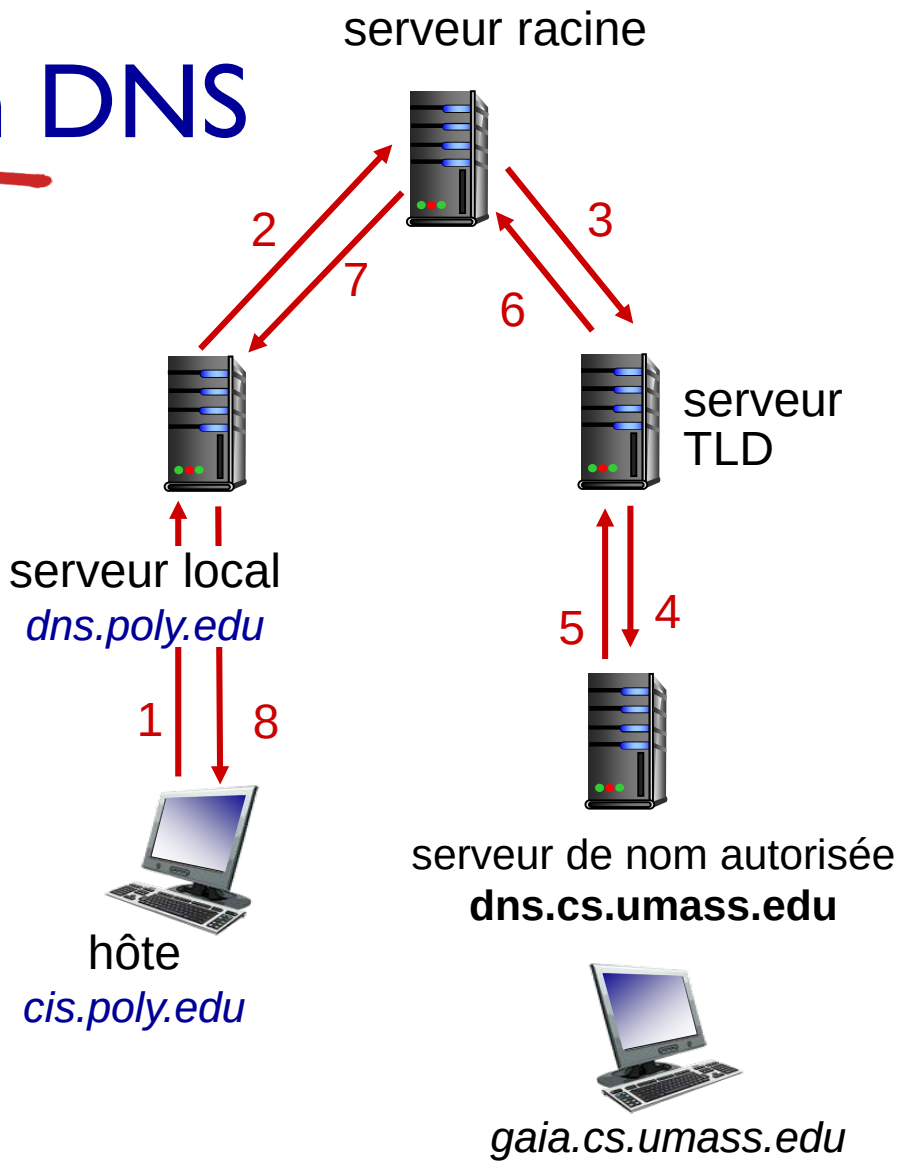
- le serveur contacté répond en fournissant le nom du serveur à contacter
- "je ne connais pas ce nom, mais demandez à ce serveur"



Exemple de la traduction de nom DNS

requête récursive:

- le serveur contacté doit lui même trouver une réponse
- plus de charge sur le haut de la hiérarchie?



DNS: mémoire cache, mise à jour

- à chaque fois qu'un serveur apprend une correspondance, il l'a place en mémoire cache
  - les entrées en cache disparaissent après un TTL
  - les serveurs locaux mettent souvent les adresses des serveurs TLD
    - ainsi les serveurs "racine" sont rarement sollicités
- les entrées dans le cache peuvent être *obsolètes* (un service *best-effort*!)
  - si un hôte change d'adresse IP, elle ne sera actualisée partout qu'après que les TTLs expirent
- les mécanismes de mise à jour dans
  - RFC 2136

DNS records

DNS: base distribuée stockant les enregistrements de ressources (RR)

format du RR : **`(name, value, type, ttl)`**

type=A

- **`name`** nom de l'hôte
- **`value`** adresse IP

type=NS

- **`name`** domaine (ex., foo.com)
- **`value`** nom d'un serveur de source autorisée

type=CNAME

- **`name`** alias
- **`www.ibm.com`** est **`servereast.backup2.ibm.com`**
- **`value`** nom canonique

type=MX

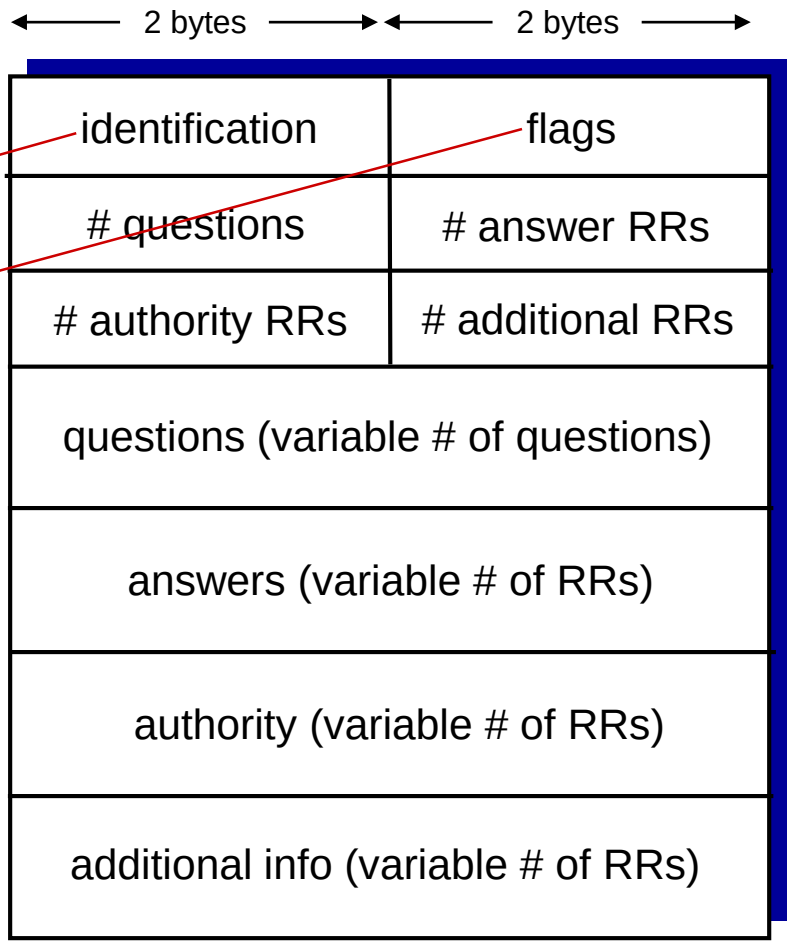
- **`value`** nom canonique d'un serveur mail ayant l'alias **`name`**

protocole DNS, messages

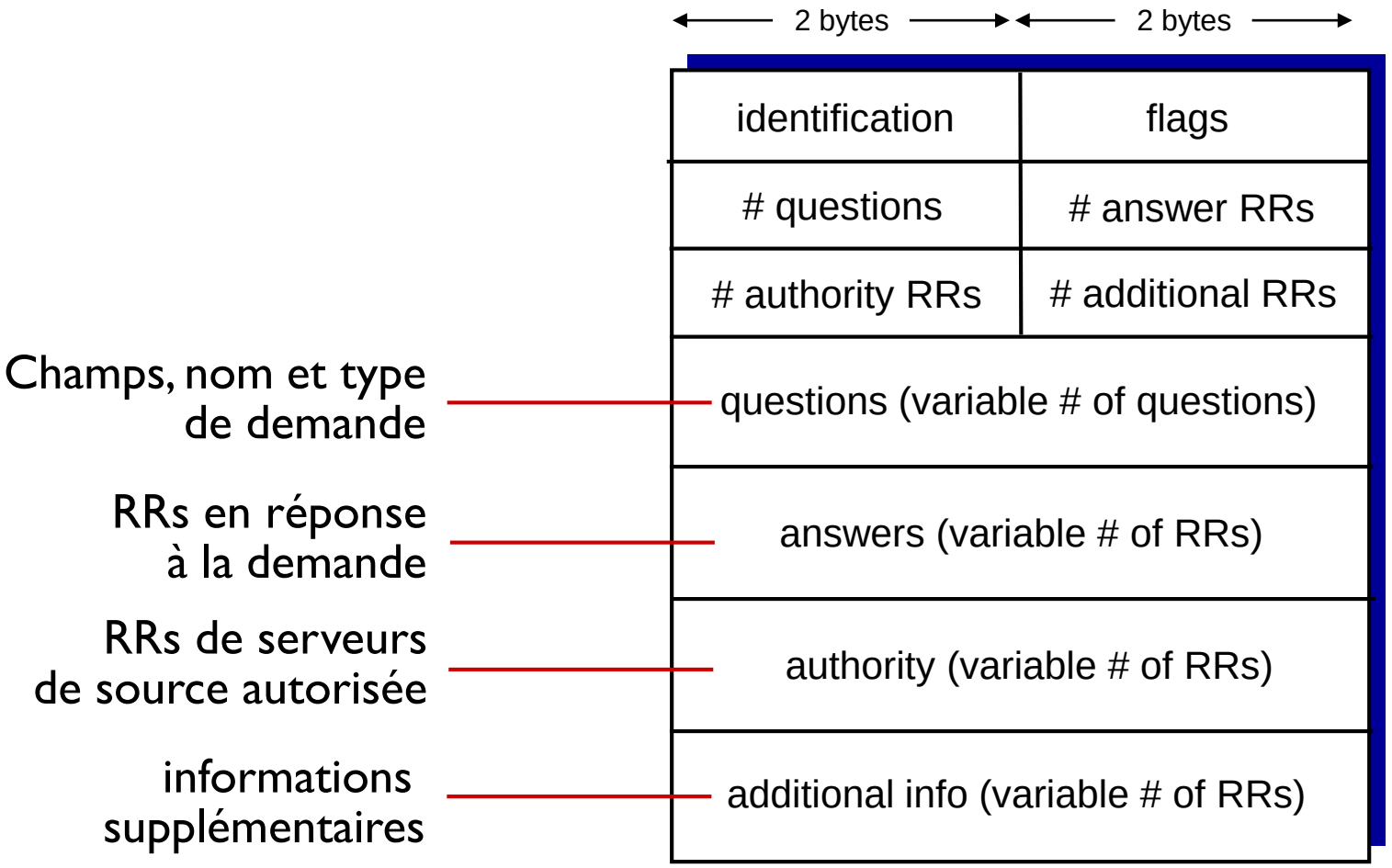
- messages requête et réponse ont le même *format*

en-tête

- identification: un # à 16 bit pour requête, réponse utilise le même # de requête
- fanions:
  - requête ou réponse?
  - récursion désirée?
  - récursion disponible?
  - le serveur a autorité?



protocole DNS, messages



Insertion des enregistrements

- exemple: nouvelle startup “Network Utopia”
- enregistrer le nom networkuptopia.com au *registraire DNS* (ex., Network Solutions)
  - fournir noms et adresses IP des serveurs de source autorisée (primaire et secondaire)
  - registraire insère deux RRs au serveur TLD .com :
    (networkutopia.com, dns1.networkutopia.com, NS)
    (dns1.networkutopia.com, 212.212.212.1, A)
- créer des enregistrements type A pour www.networkuptopia.com et type MX pour networkutopia.com au niveau du serveur de source autorisée

Chapitre II: plan

2.1 principes des applications réseaux

2.2 Web et HTTP

2.3 FTP

2.4 courriel

- SMTP, POP3, IMAP

2.5 DNS

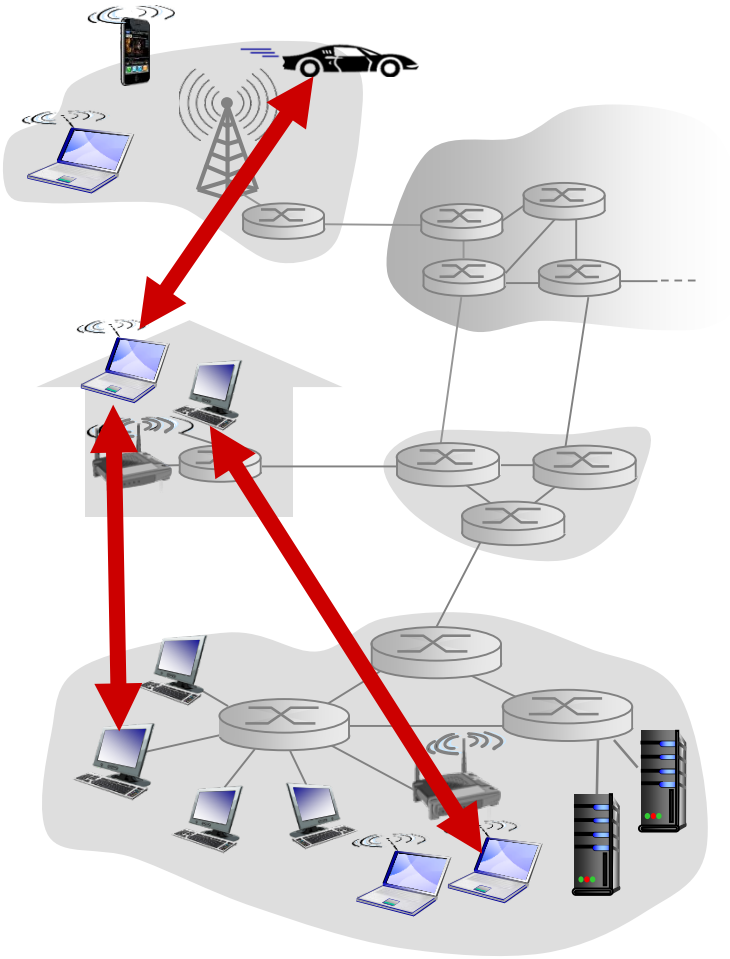
2.6 applications P2P

Architecture P2P pure

- *pas de besoin d'un serveur (toujours en marche)*
- terminaux communiquent directement
- pairs changent d'adresses IP (se connectent/se déconnectent)

exemples:

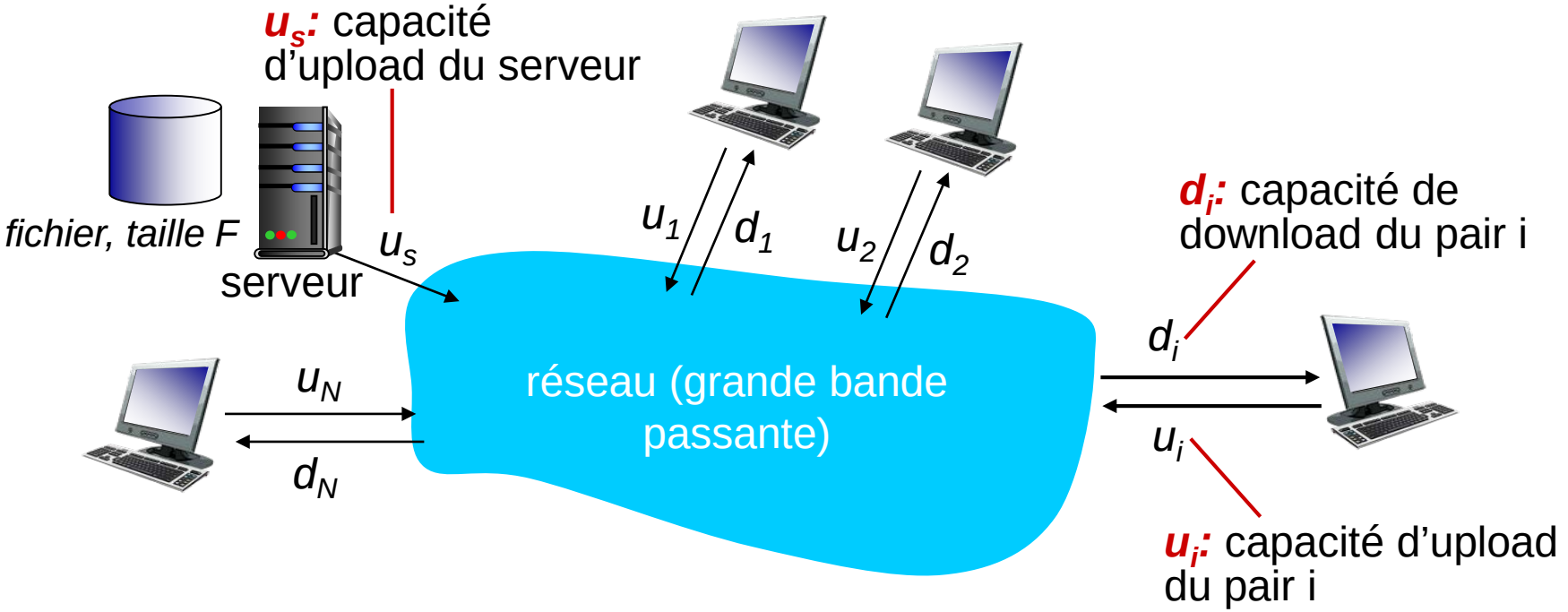
- distribution de fichiers (BitTorrent)
- Streaming (Sopcast)
- VoIP (Skype)



distribution de fichiers: client-serveur vs P2P

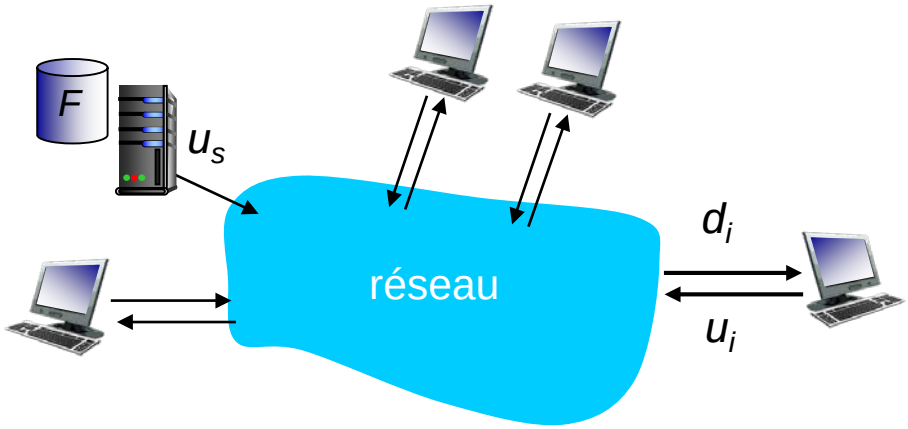
Question: temps nécessaire pour distribuer un fichier (de taille *F*) d'un serveur vers *N* *pairs*?

- les capacités de téléchargement des pairs sont limitées



distribution de fichiers: client-serveur

- *transmission du serveur :* doit envoyer N copies en série:
  - temps envoi d'une copie: F/u_s
  - temps pour N copies: NF/u_s



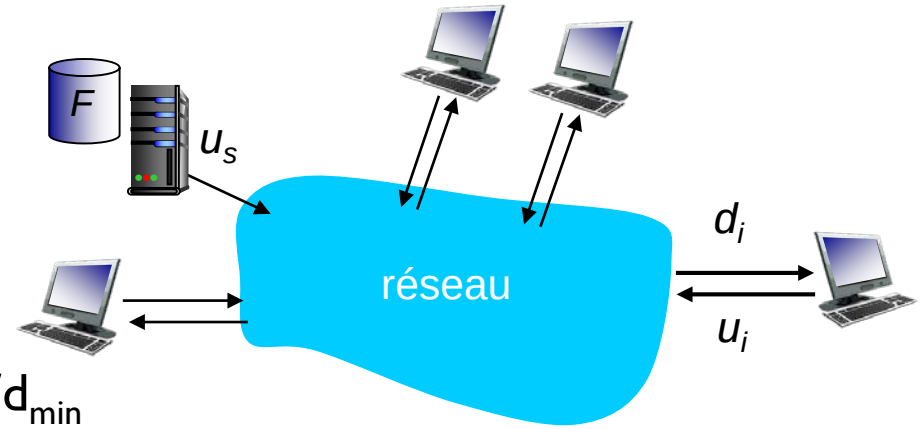
- *client:* chaque client doit télécharger une copie
  - d_{min} = taux min. de téléchargement
  - temps min de téléchargement: F/d_{min}

temps pour distribuer F en utilisant client/serveur $D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$

croît linéairement avec N

distribution de fichiers: P2P

- *transmission du serveur :* doit uploader au moins une copie
  - temps envoie d'une copie: F/u_s
- *client:* chaque client doit télécharger une copie
  - temps min de téléchargement: F/d_{min}
- *clients:* téléchargent un total de *NF* bits
  - débit d'upload max est $u_s + \Sigma u_i$



temps pour distribuer F en utilisant P2P

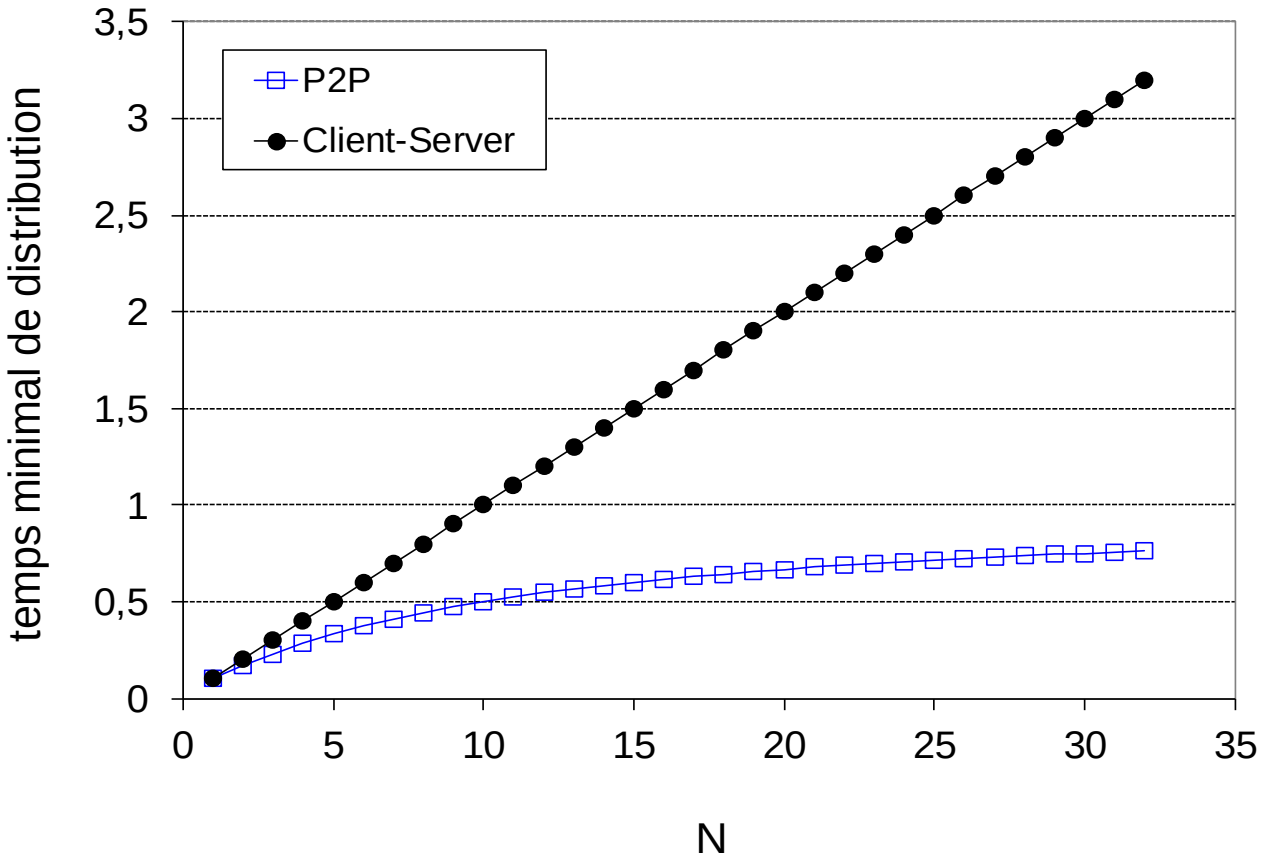
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

croît linéairement avec *N* …

… mais chaque pair ajoute de la capacité au réseau

Client-serveur vs. P2P: exemple

u = débit d'upload client, F/u = 1 heure, $u_s = 10u$, $d_{min} \geq u_s$

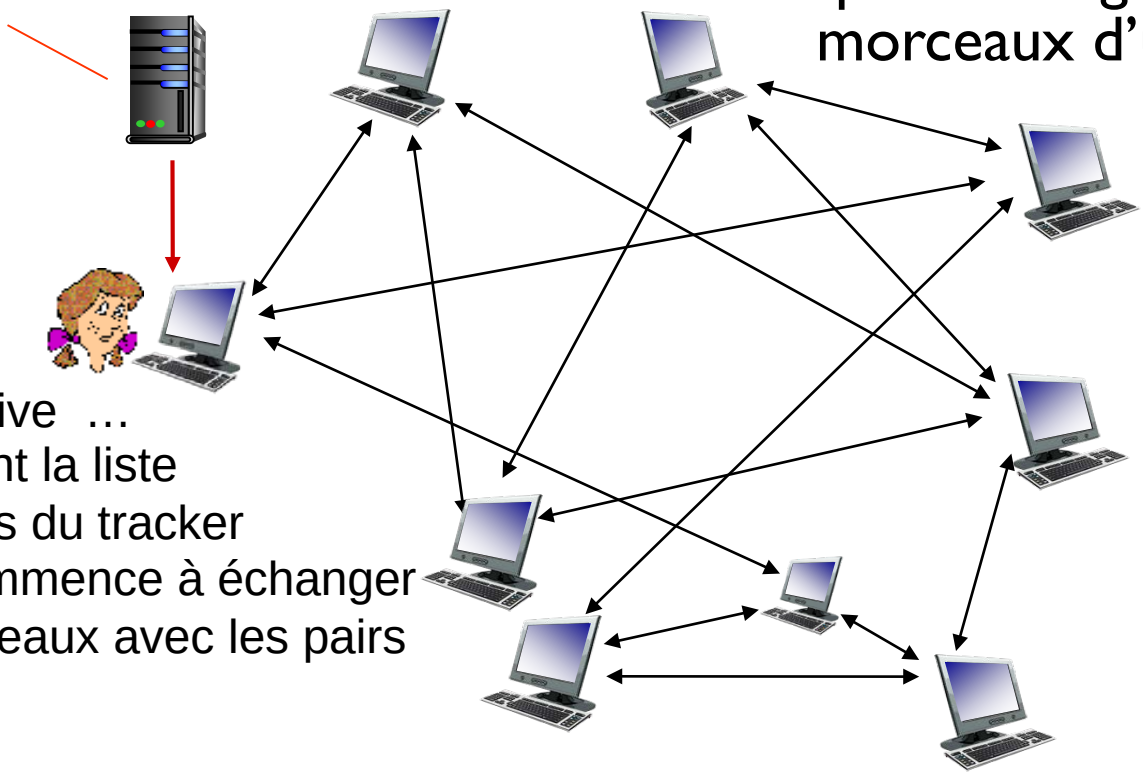


distribution de fichiers P2P : BitTorrent

- un fichier est divisé en plusieurs morceaux (chunks) de 256KO chacun
- les pairs dans un torrent envoie/reçoit des morceaux

tracker: fait le suivi des pairs qui participent au torrent

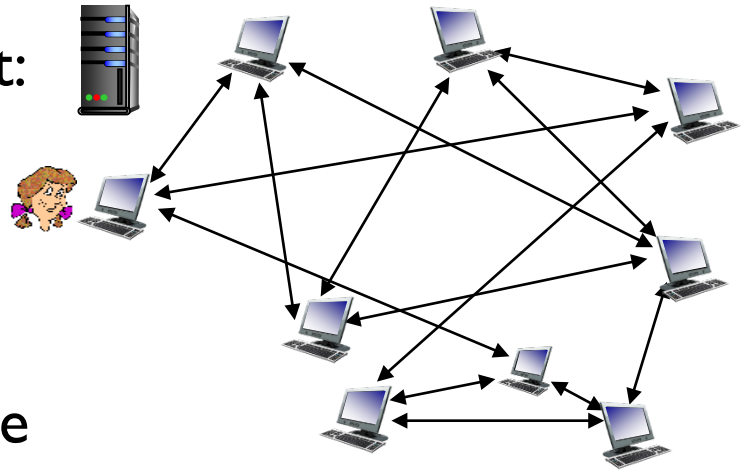
torrent: groupe des pairs qui s'échangent les morceaux d'un fichier



Alice arrive …
… obtient la liste des pairs du tracker
… et commence à échanger les morceaux avec les pairs

distribution de fichiers P2P : BitTorrent

- un pair qui entre dans un torrent:
  - n'a pas de morceaux, mais va les accumuler au fil du temps à partir des autres pairs
  - s'enregistre au niveau du tracker pour récupérer la liste de pairs, puis se connecte à un sous ensemble (les voisins)



- un pair télécharge et téléverse des morceaux
- un pair peux changer ses voisins
- *churn:* les pairs peuvent apparaître/disparaitre
- lorsque le pair termine le téléchargement, il peut quitter le torrent (égoïste) ou rester (altruiste)

BitTorrent: demande, envoie des morceaux

demander des morceaux:

- à chaque instant, chaque pair possède un ensemble diffèrent de morceaux
- périodiquement, Alice demande aux pairs la liste des morceaux qu'ils possèdent
- Alice demande les morceaux manquants, commençant par le plus rare (rarest first)

envoie des morceaux: donnant-donnant (tit-for-tat)

- Alice sert les quatre pairs qui lui envoie des morceaux avec le *plus grand débit*
  - les autres pairs sont ignorés par Alice (zéro envoi)
  - réévalué le top 4 chaque 10 secs
- chaque 30 secs: choisir aléatoirement un autre pair, et le servir
  - un pair peut entrer dans le top 4 (optimiste)

BitTorrent: tit-for-tat

(1) Alice choisit Bob "de manière optimiste"

(2) Alice devient dans le top 4 de Bob; Bob réagit en servant Alice

(3) Bob entre au top 4 de Alice

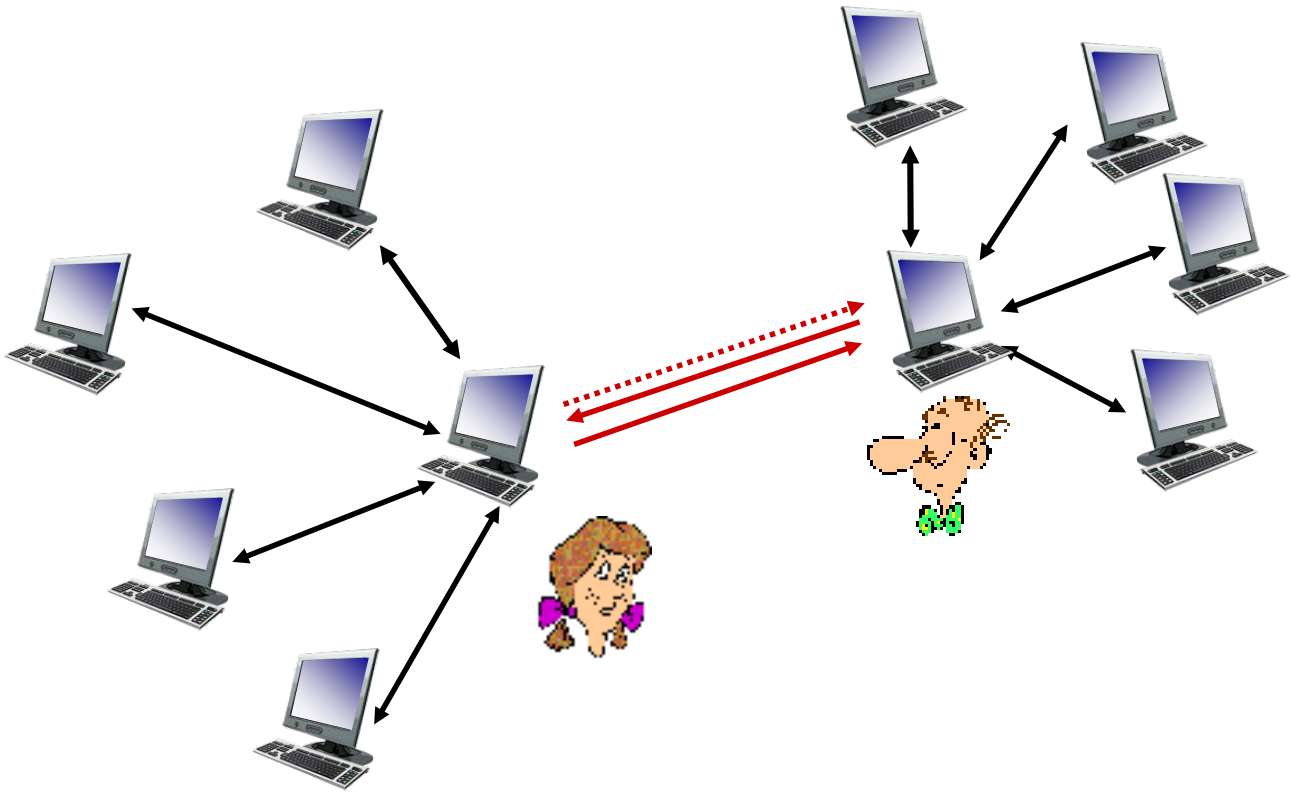


Table de hachage distribuée (DHT)

- table de hachage
- paradigme DHT
- DHT circulaire et le réseau transparent
- Peer churn

Base de données simple

Une simple base avec des paires (clé, valeur):

- clé: nom; value: NAS

Key	Value
John Washington	132-543-570
Diana Louise Jones	761-553-791
Xiaoming Liu	385-410-902
Rakesh Gopal	441-891-956
Linda Cohen	217-665-609
.....	.....
Lisa Kobayashi	177-230-199

- clé: titre d'un film; valeur: adresse IP

Table de hachage

- Plus pratique de stocker et chercher des clés représentées numériquement
- clé = hash(clé d'origine)

Clé d'origine	Clé	Valeur
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....		.....
Lisa Kobayashi	9290124	177-23-0199

Table de hachage distribuée (DHT)

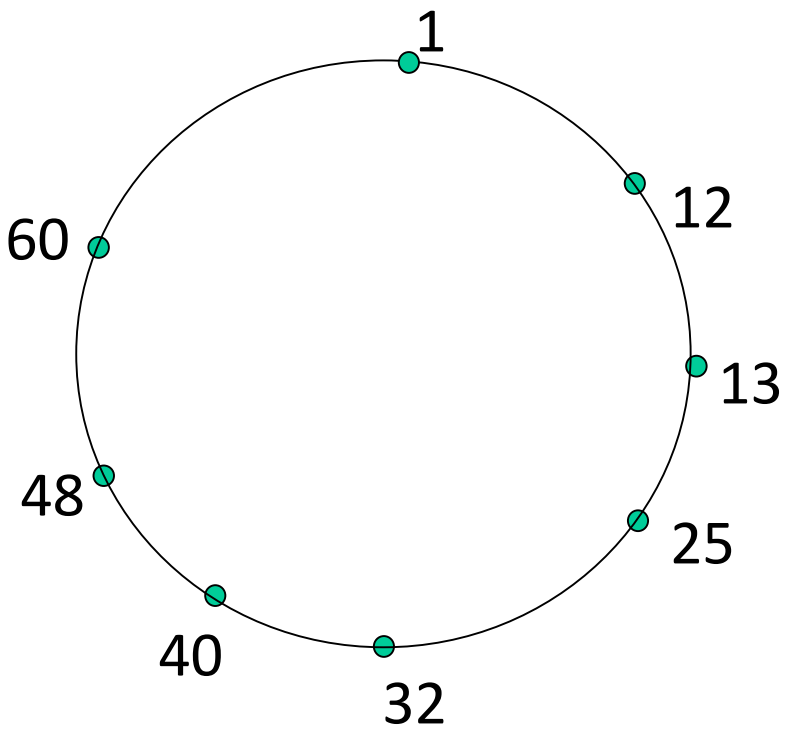
- distribuer les paires (clé, valeur) sur des millions de pairs
- un pair peut formuler une requête en utilisant la clé
  - la base retourne la valeur correspondante
  - pour répondre à la requête, un petit nombre de messages est échangé entre les pairs
- chaque pair possède une information concernant un petit nombre de pairs
- robuste à l'apparition/disparition des pairs (churn)

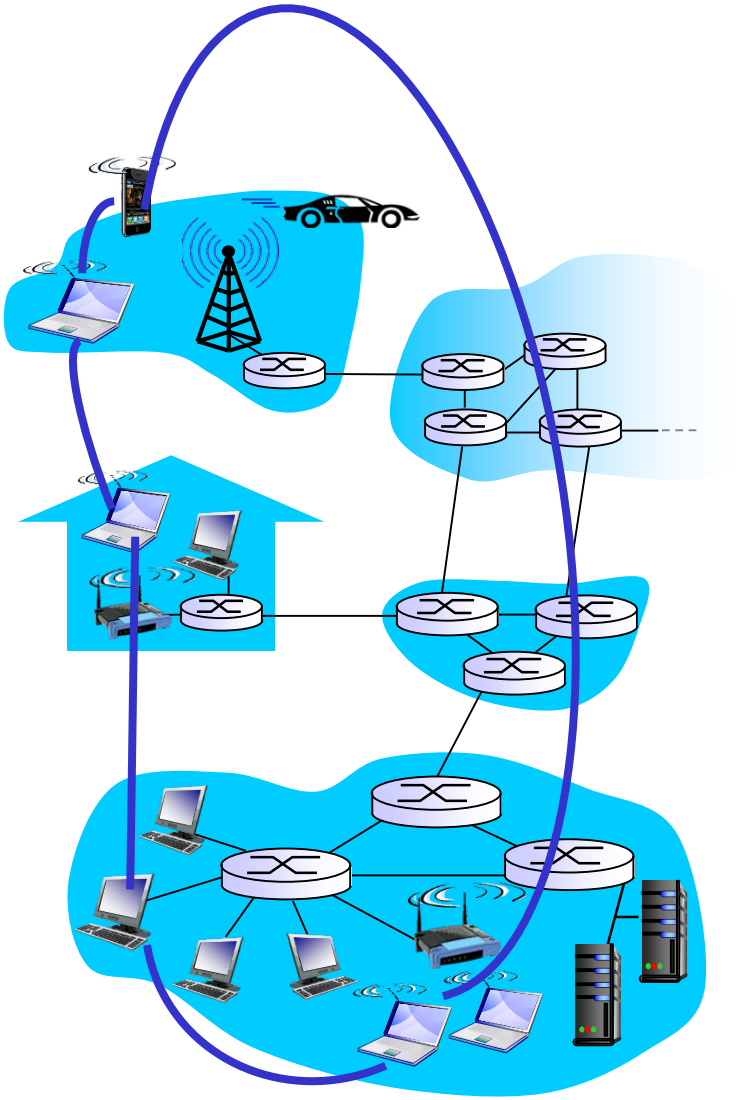
Assigner paires clé-valeur aux pairs

- règle: assigner une paire (clé, valeur) au pair ayant le ID le plus proche à la clé.
- convention: plus proche est le *successeur immédiat* à la clé.
- ex., espace des IDs {0,1,2,3,…,63}
- supposons 8 pairs: 1,12,13,25,32,40,48,60
  - Si clé = 51, alors assigner au pair 60
  - Si clé = 60, alors assigner au pair 60
  - Si clé = 61, alors assigner au pair 1

DHT circulaire

- chaque pair connait seulement son successeur et prédécesseur immédiats

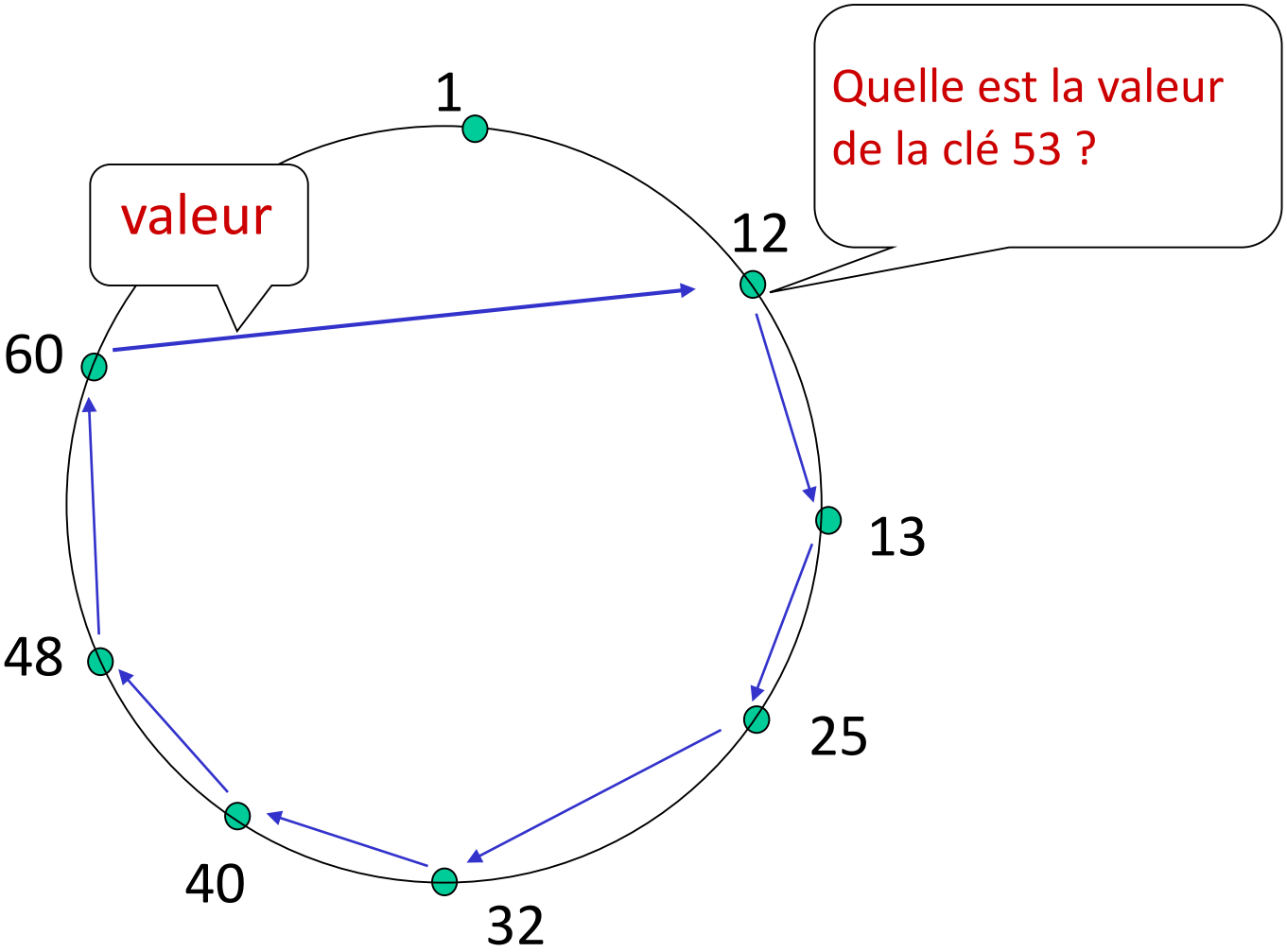




réseau transparent

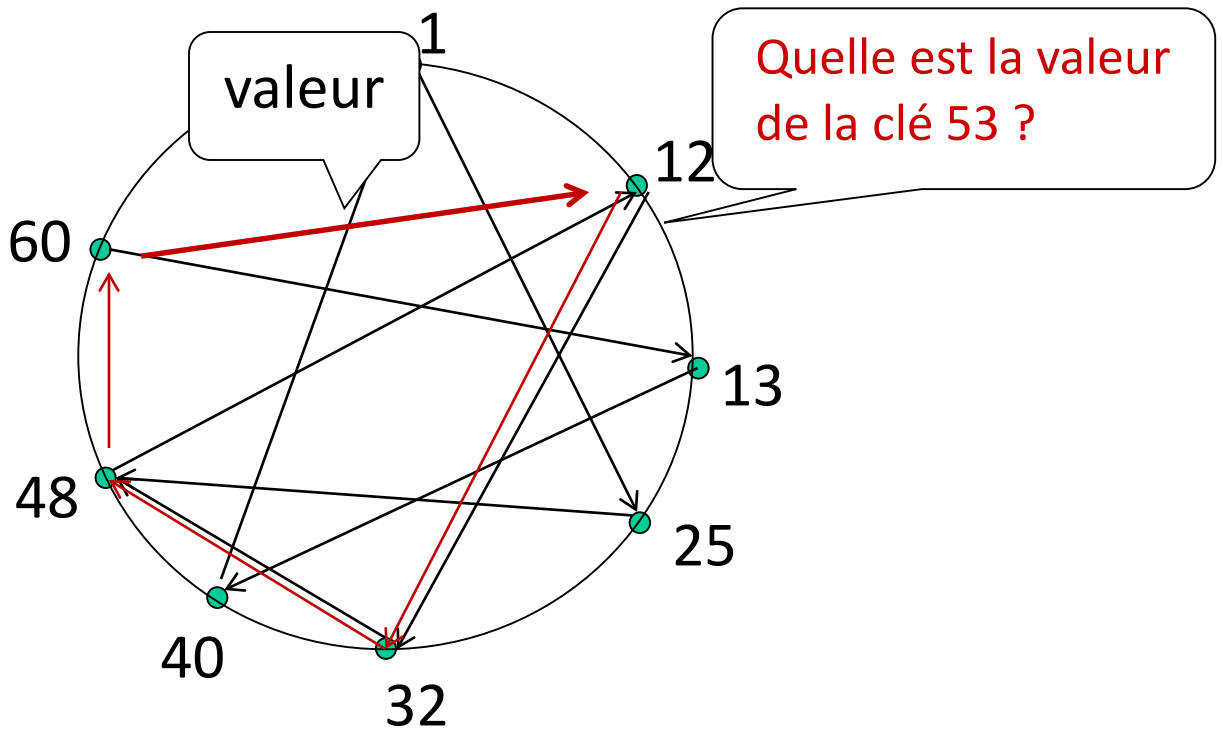
“overlay network”

Répondre à une requête



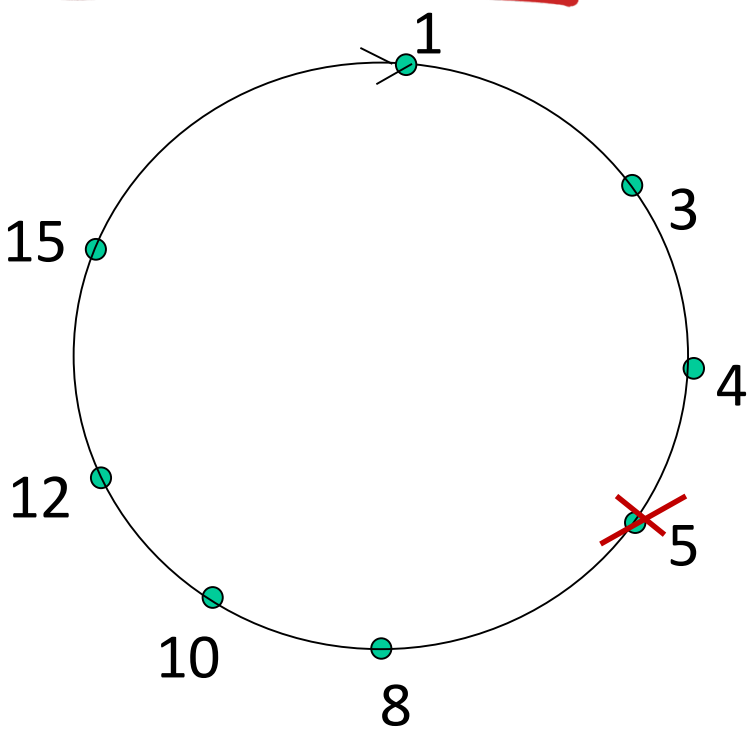
O(N) messages pour répondre à une requête pour *N* pairs

DHT circulaire avec raccourcis



- chaque pair maintient une information sur les adresses IP de son prédécesseur, successeur, raccourcis.
- le nombre des messages réduit de 6 à 3.
- si *O(log N)* raccourcis, *O(log N)* messages pour répondre à une requête

Peer churn



exemple: pair 5 quitte sans préavis

solution au "peer churn":

- les pairs apparaissent/ disparaissent (churn)
- chaque pair connaît les adresses de ses deux successeurs
- il "ping" périodiquement ces successeurs pour vérifier s'ils sont toujours "vivants"
- si le successeur immédiat quitte, il choisit le prochain comme nouveau successeur immédiat

Chapitre II: en résumé

ce que nous avons appris dans ce chapitre

- architectures
  - client-serveur
  - P2P
- exigences des applications:
  - fiabilité, débit, délai
- service de transport
  - orienté connexion, fiable: TCP
  - non fiable: UDP
- protocoles spécifiques:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT

Chapitre II: en résumé

ce que nous avons appris dans ce chapitre

- échange de messages requête/réponse :
  - client demande une info ou service
  - serveur répond par données, code d'état
- formats des messages :
  - en-tête
  - données

important:

- contrôle vs. données
  - dans la bande, hors bande
- sans état vs. avec état
- transfert fiable vs. non fiable