

# Pair à Pair ( P2P)

# Architecture du réseau

- ❑ **Centralisée**: un serveur ou un cluster sur lequel les clients se connectent (Napster)
- ❑ **Décentralisée**: il n'y a que des clients (Gnutella)
- ❑ **Faiblement centralisée**: des clients et des serveurs (Edonkey)
- ❑ **Hybride**: les clients peuvent devenir des serveurs

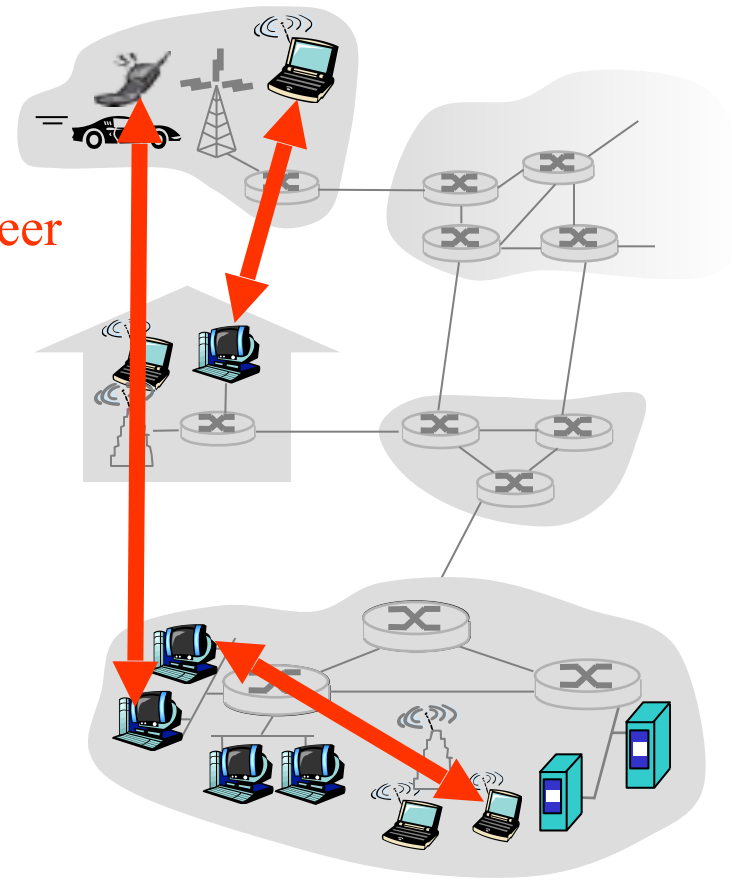
# P2P: problems with centralized directory

- ❑ Single point of failure
- ❑ Performance bottleneck
- ❑ Copyright infringement

file transfer is  
decentralized, but  
locating content is  
highly centralized

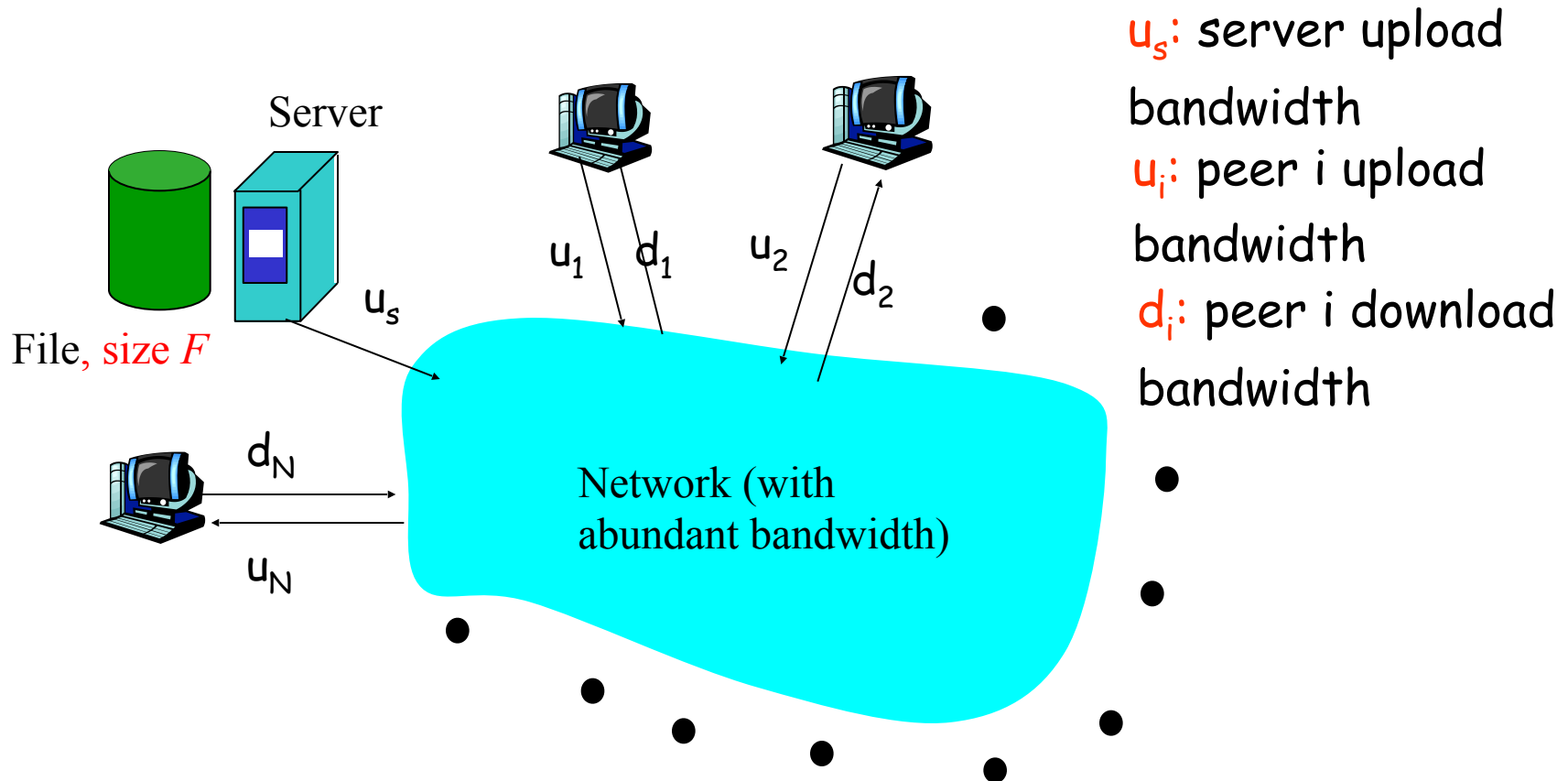
# Pure P2P architecture

- ❑ no always-on server
- ❑ arbitrary end systems directly communicate peer-peer
- ❑ peers are intermittently connected and change IP addresses



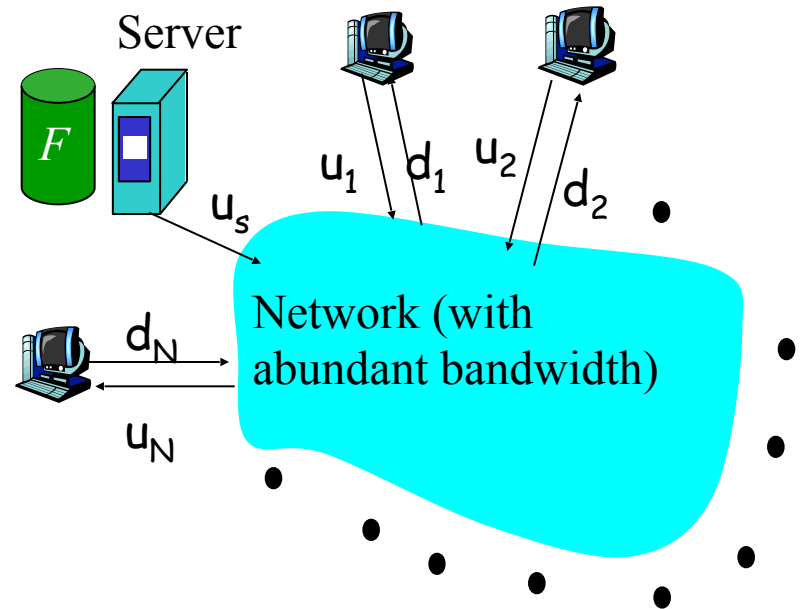
# File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to  $N$  peers?



# File distribution time: server-client

- server sequentially sends  $N$  copies:
  - ❖  $N.F/u_s$  time
- client  $i$  takes  $F/d_i$  time to download

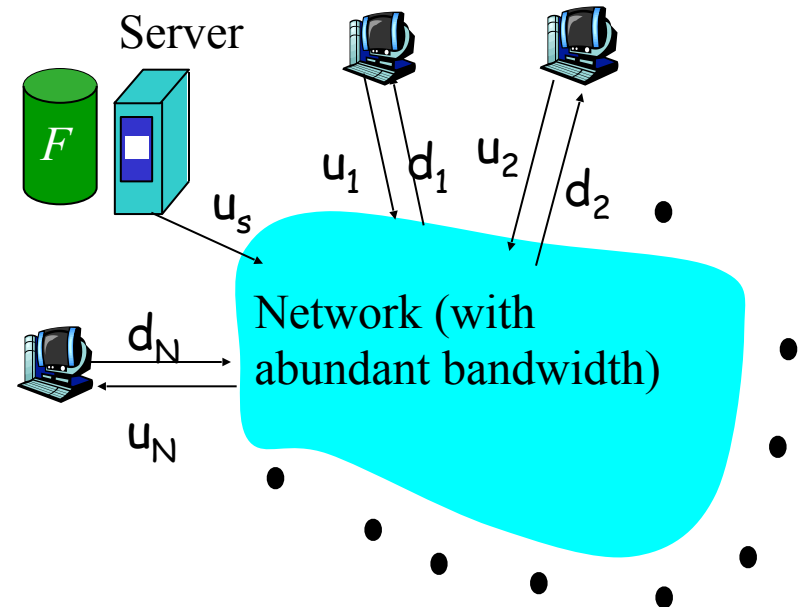


Time to distribute  $F$  to  $N$  clients using client/server approach =  $d_{cs} = \max \left\{ N.F/u_s, F/\min(d_i) \right\}$

increases linearly in  $N$   
(for large  $N$ )

# File distribution time: P2P

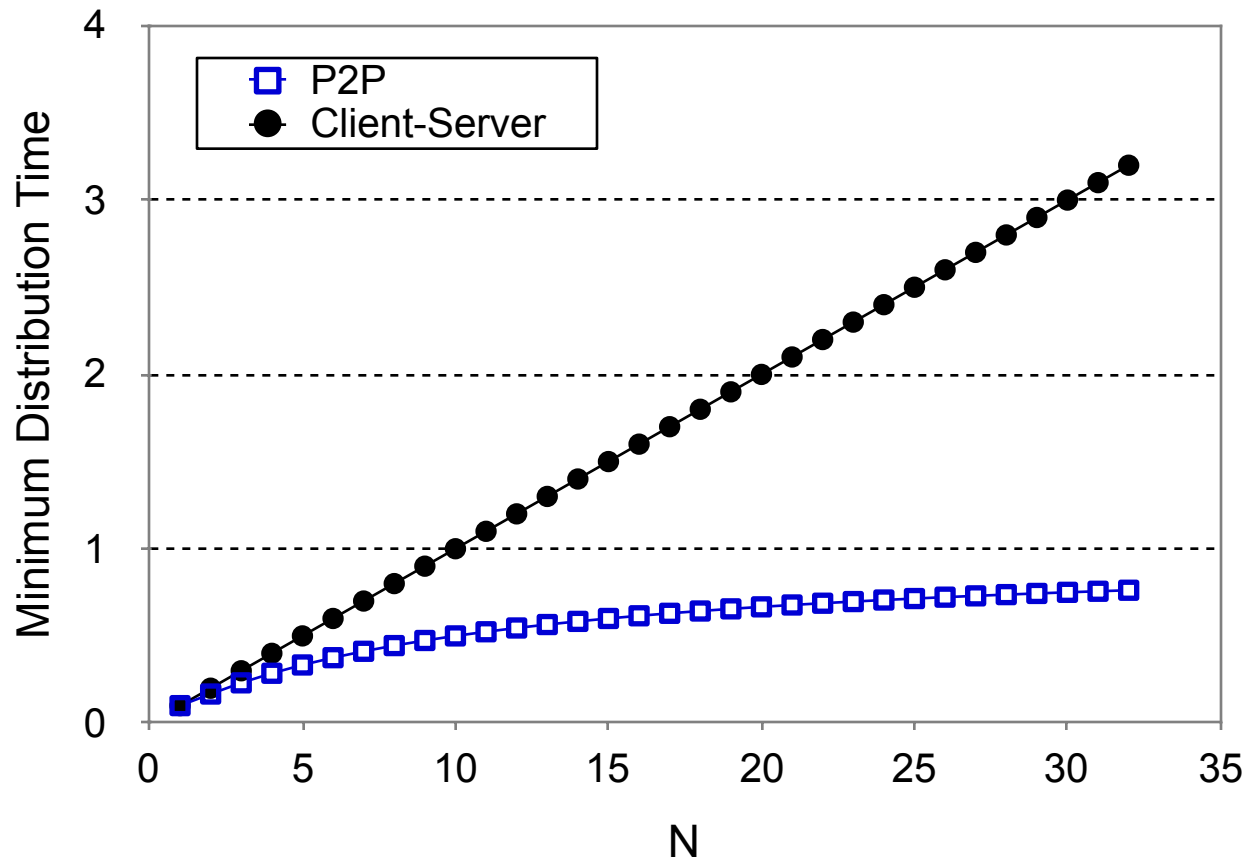
- ❑ server must send one copy:  $F/u_s$  time
- ❑ client  $i$  takes  $F/d_i$  time to download
- ❑  $N.F$  bits must be downloaded (aggregate)
  - ❑ fastest possible upload rate:  $u_s + \sum u_i$



$$d_{\text{P2P}} = \max \left\{ F/u_s, F/\min(d_i), N.F/(u_s + \sum u_i) \right\}$$

# Server-client vs. P2P: example

Client upload rate =  $u$ ,  $F/u = 1$  hour,  $u_s = 10u$ ,  $d_{\min} \geq u_s$





# Partage de fichiers

- Communication basée sur deux types de protocoles différents
  - ❖ découverte et localisation des fichiers
    - recherche des données
    - mettre en contact deux (ou plusieurs) utilisateurs parmi des millions
  - ❖ téléchargement des fichiers

# Découverte-localisation

## □ Base

- ❖ publication des fichiers partagés avec des méta-données
- ❖ découverte des fichiers disponibles
- ❖ localisation des sources à télécharger
- ❖ (En plus
  - identification des doublons
  - détection des fichiers corrompus
  - forums)

# Quelques exemples

# Query flooding: Gnutella

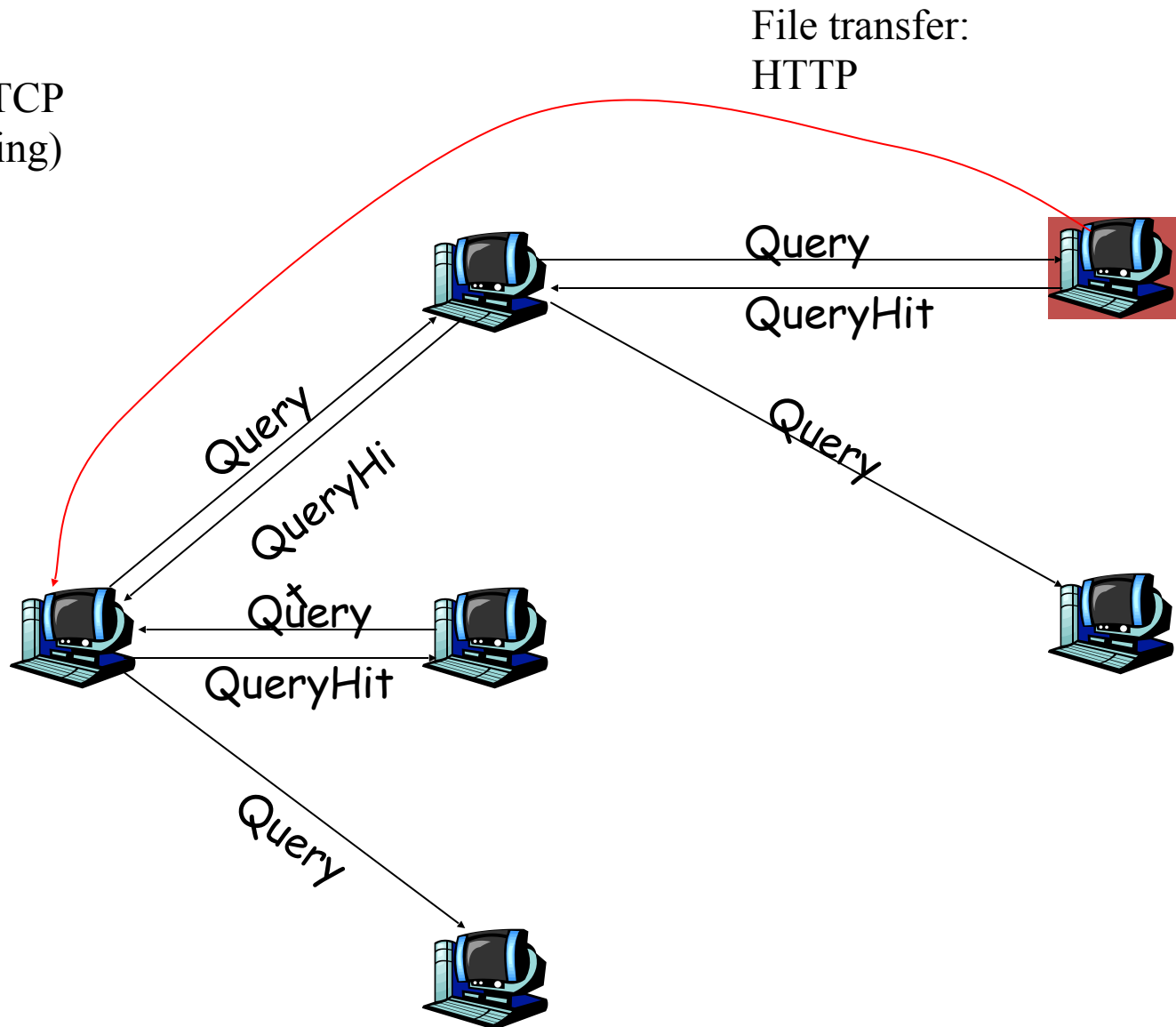
- ❑ fully distributed
  - ❖ no central server
- ❑ public domain protocol
- ❑ many Gnutella clients implementing protocol

## overlay network: graph

- ❑ edge between peer X and Y if there's a TCP connection
- ❑ all active peers and edges is overlay net
- ❑ Edge is not a physical link
- ❑ Given peer will typically be connected with  $< 10$  overlay neighbors

# Gnutella: protocol

- ❑ Query message sent over existing TCP connections (flooding)
- ❑ peers forward Query message
- ❑ QueryHit sent over reverse path



Scalability:  
limited scope  
flooding

# Gnutella: Peer joining

1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

## version suivante : ultra peers

- » Since version 0.6 (2002),
  - » leaf nodes and
  - » ultra nodes (also called ultrapeers).
- » The leaf nodes are connected to a small number of ultrapeers (typically 3) while each ultrapeer is connected to more than 32 other ultrapeers by TCP.

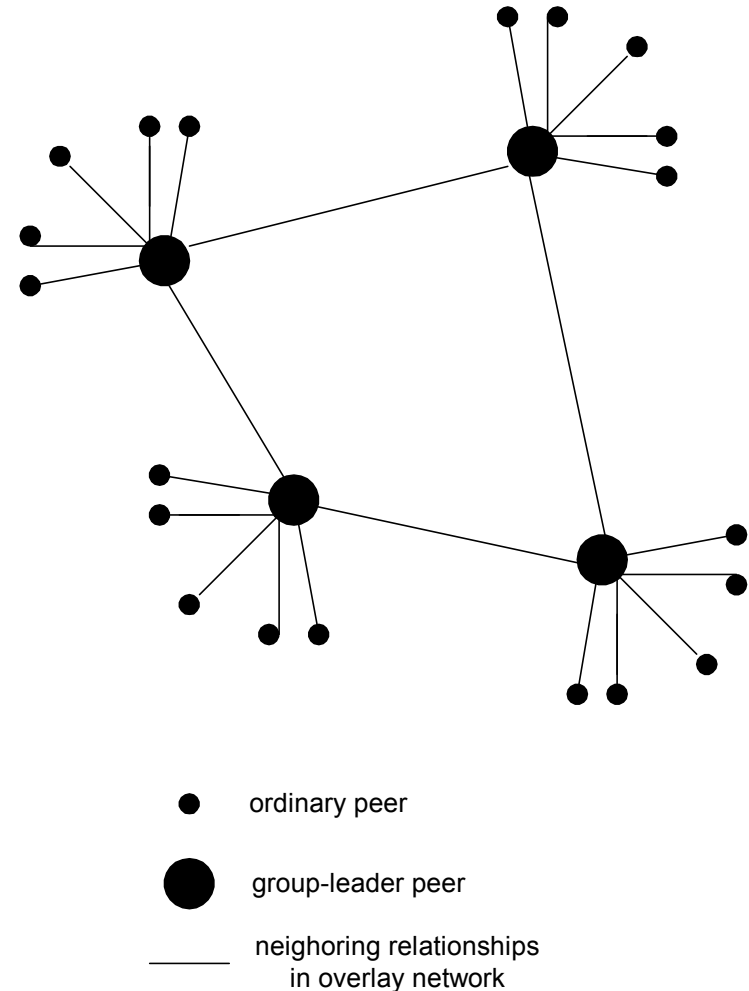
## version suivante : ultra peers

- » search request: Query routing protocol
  - » Query routing table (QRT) : table of hashed keyword
- » result of the search sent by UDP to the node that initiated the search (the ultra node of the requester)
  - » queries contain IP+port of the requester



# Exploiting heterogeneity: KaZaA

- Each peer is either a group leader or assigned to a group leader.
  - ❖ TCP connection between peer and its group leader.
  - ❖ TCP connections between some pairs of group leaders.
- Group leader tracks the content in all its children.



# KaZaA: Querying

- ❑ Each file has a hash and a descriptor
- ❑ Client sends keyword query to its group leader
- ❑ Group leader responds with matches:
  - ❖ For each match: metadata, hash, IP address
- ❑ If group leader forwards query to other group leaders, they respond with matches
- ❑ Client then selects files for downloading
  - ❖ HTTP requests using hash as identifier sent to peers holding desired file

# KaZaA tricks

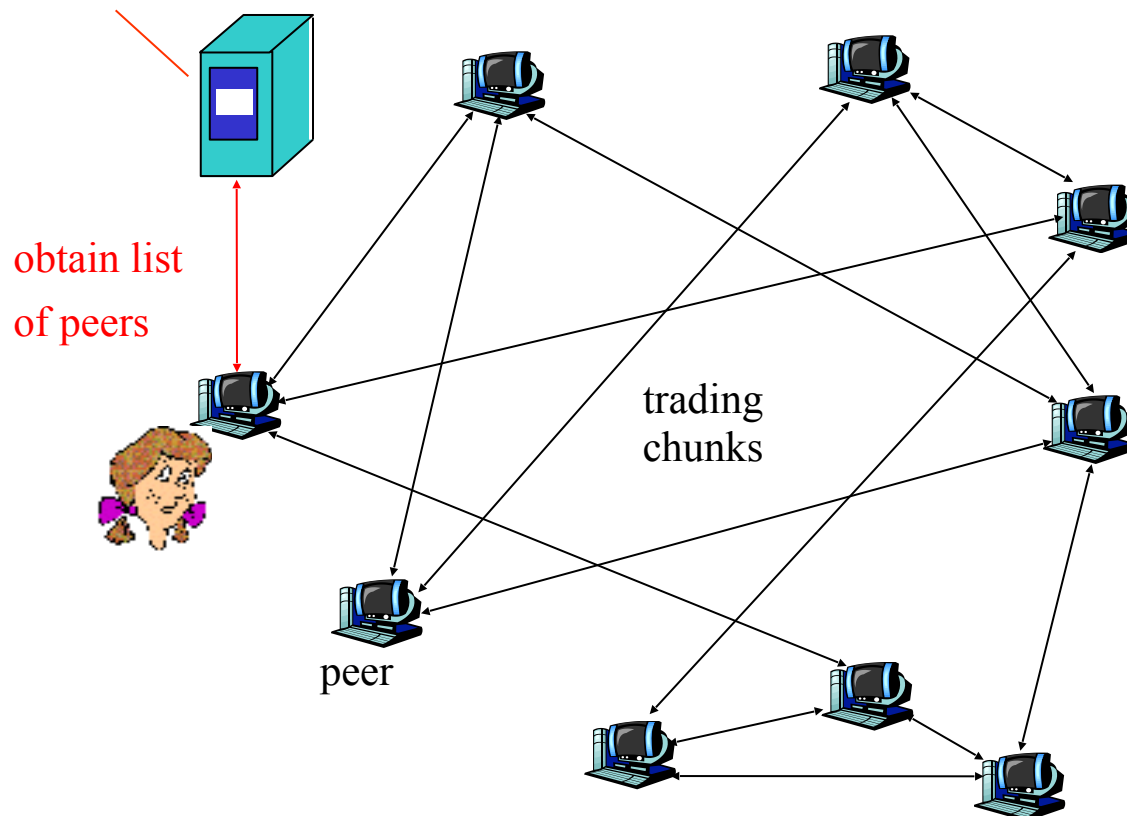
- ❑ Limitations on simultaneous uploads
- ❑ Request queuing
- ❑ Incentive priorities
- ❑ Parallel downloading

# File distribution: BitTorrent

## ❑ P2P file distribution

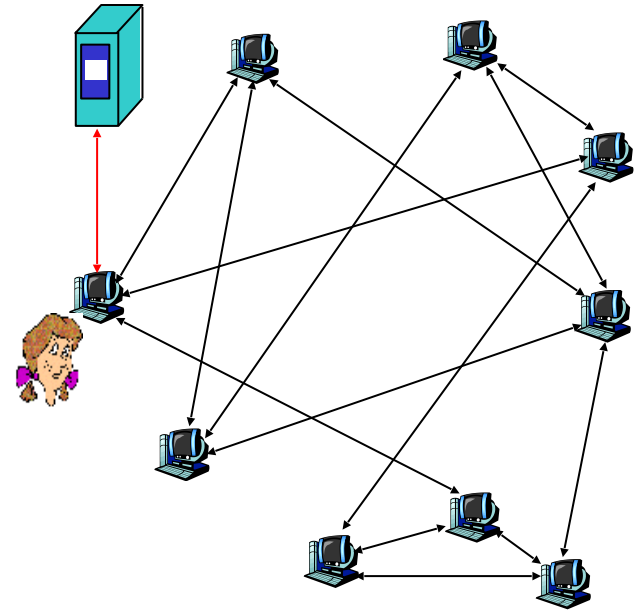
tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



# BitTorrent (1)

- ❑ file divided into 256KB **chunks**.
- ❑ peer joining torrent:
  - ❖ has no chunks, but will accumulate them over time
  - ❖ registers with tracker to get list of peers, connects to subset of peers ("neighbors") by TCP
- ❑ while downloading, peer uploads chunks to other peers.
- ❑ peers may come and go
- ❑ once peer has entire file, it may (selfishly) leave or (altruistically) remain



# BitTorrent (2)

## Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - ❖ rarest first

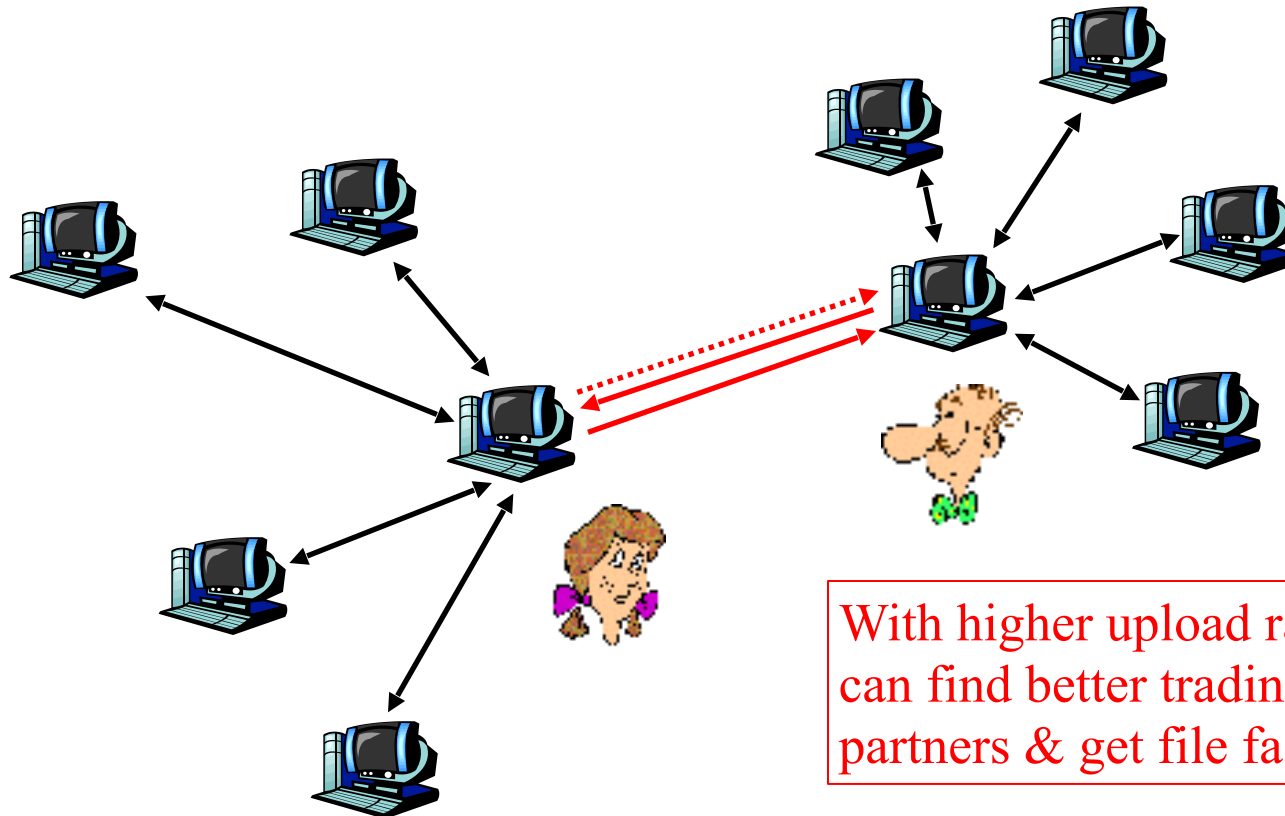
## Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks at the highest rate
  - ❖ re-evaluate top 4 every 10 sec
- every 30 secs: randomly select another peer, starts sending chunks
  - ❖ newly chosen peer may join top 4
  - ❖ "optimistically unchoke"

4 unchoked +1 optimistically uncooked  
all the other are choked

# BitTorrent: Tit-for-tat

- (1) Alice “optimistically unchokes” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers



With higher upload rate,  
can find better trading  
partners & get file faster!

# BitTorrent clients

- » ( free) BitTorrent,  $\mu$ Torrent, BitComet, BitLord, Deluge, Transmission, qBittorrent, Leap, Vuze Free,...



# Rechercher

# Recherche par diffusion simple

- les clients ne publient rien et ne font que des requêtes
  - ❖ une requête est diffusée à tous les clients:
    - avec une limite de portée (TTL)
    - en évitant les cycles (identification unique de la requête, la requête contient son chemin ...)
    - recherche en profondeur ou en largeur
  - ❖ un client répond par le chemin emprunté par la requête (ou directement)

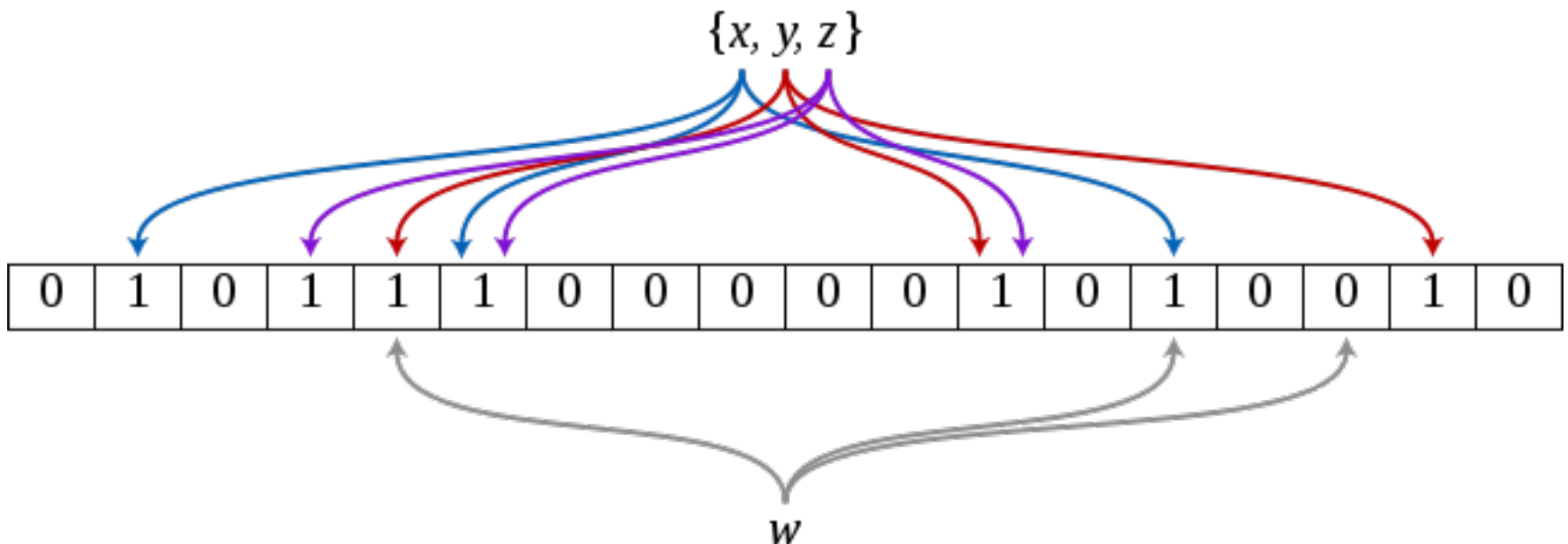
# Filtres de Bloom

## □ Principe:

- ❖ un ensemble  $E$  de  $n$  éléments
- ❖  $k$  fonctions de hachage  $h_i$  de  $E$  sur  $\{1, m\}$
- ❖ un vecteur de  $m$  bits (le filtre)
- ❖ tester si  $x$  appartient à  $E$  si
  - si le bit  $h_i(x)=0$  pour un  $i$  non
  - si le bit  $h_i(x)=1$  pour tout  $i$  oui (mais peut être faux-positif)
- ❖ On diminue les faux positifs en utilisant plusieurs fonctions de hachage
  - Pour  $m$  et  $n$  donnés, le  $k$  optimal est  $m/n \ln 2$  et la probabilité de faux-positifs est  $(1/2)^k$

# Exemple: Filtres de Bloom

- ❑ Trois fonctions de hachage
- ❑  $W$  n'appartient pas à  $E$



# Filtres de Bloom

## □ Utilisation:

- ❖ Si le client possède le document les bits des mot-clés correspondant sont à 1
- ❖ Les clients s'échangent les filtres
- ❖ Les requêtes ne sont propagées que vers les clients qui ont le bon filtre

# Recherche par indexation

- ❑ Les clients publient les fichiers qu'ils partagent sur les serveurs sur lesquels ils sont connectés
- ❑ Les serveurs indexent les descriptions des fichiers
- ❑ Les clients envoient des requêtes aux serveurs pour trouver les fichiers et les localiser

# Recherche par DHT

## □ Distributed Hash Table:

### ❖ Des nœuds et des données

- Un espace de clés (exemple: clés de 160 bits)
- Une répartition des clés (qui possède les clés?)
- Un "overlay" de connexion entre les nœuds qui permet de savoir quel nœud possède quelle clé.

# Distributed Hash Table (DHT)

- ❑ DHT = base de données pour P2P
- ❑ Contient des couples (key, value)
  - ❖ key: numero SS; value: nom
  - ❖ key: content type; value: adresse IP
- ❑ Un pair **query** la DHT par une key
  - ❖ retourne les valeurs correspondant à la key
- ❑ Un pair peut aussi **insérer** des couples (key, value)



# DHT

- Stocker un fichier  $f$  de contenu data:
  - ❖ Un hachage (SHA1 par exemple) produit une clé de 160 bits  $k$  à partir du nom  $f$
  - ❖ Par un  $\text{put}(k, \text{data})$  sur l'overlay trouver le nœud responsable pour la clé  $k$  et lui transférer les données
- Rechercher un fichier  $f$ :
  - ❖ Par hachage (SHA1) obtenir la clé  $k$  associée à  $f$
  - ❖ Rechercher sur l'overlay le nœud responsable de  $k$  et récupérer les données

# DHT: hachage consistant

- ❑ On associe à chaque document (ou mot-clé) un identificateur unique (hachage)
- ❑ On associe à chaque client un identificateur unique de même taille ID
- ❑ On définit une métrique pour définir la distance  $d$  entre les clés-ID
- ❑ Un nœud ID  $i$  possède les clés  $j$  telles que  $d(j,i)$  est minimal parmi les ID (chaque nœud possède les clés les plus proches)

Le retrait ou l'ajout d'un nœud ne modifie que les voisins

# DHT

- ❑ notion d'overlay: chaque nœud maintient des liens vers d'autres nœuds, on obtient ainsi un overlay.

## Comment obtenir un routage efficace

- ❑ Un nœud soit possède la clé, soit connaît un nœud plus près de la clé: on peut utiliser algorithme glouton basé sur la clé.
- ❑ On peut aussi définir d'autres algorithmes de routage sur des overlays (et définir des topologies d'overlays ayant de bonnes propriétés -degré et taille des routes).

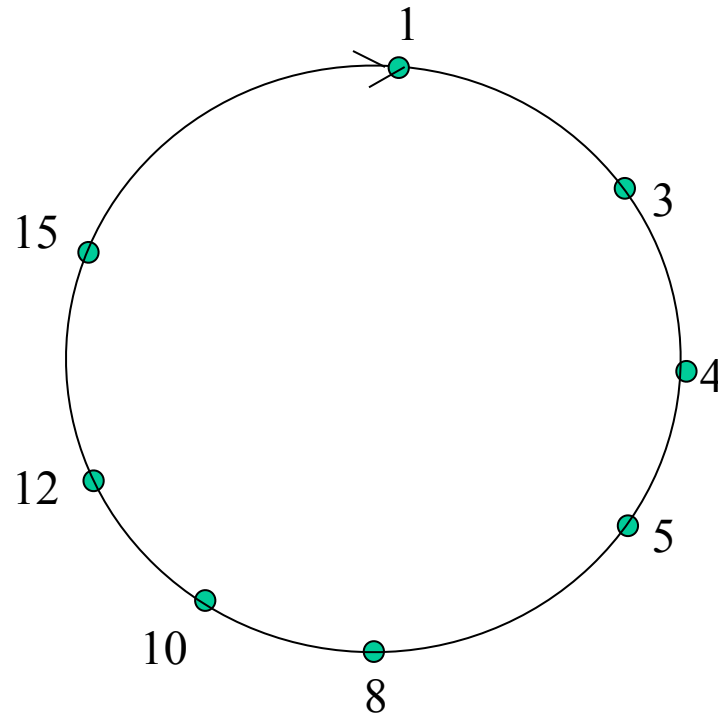
# DHT Identifiers

- Assign integer identifier to each peer in range  $[0, 2^n - 1]$ .
  - ❖ Each identifier can be represented by  $n$  bits.
- Require each key to be an integer in **same range**.
- To get integer keys, hash original key.
  - ❖ eg, key =  $h(\text{"Led Zeppelin IV"})$
  - ❖ This is why they call it a distributed "hash" table

# How to assign keys to peers?

- Central issue:
  - ❖ Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the **closest** ID.
- Convention in lecture: closest is the **immediate successor** of the key.
- Ex:  $n=4$ ; peers: 1,3,4,5,8,10,12,14;
  - ❖ key = 13, then successor peer = 14
  - ❖ key = 15, then successor peer = 1

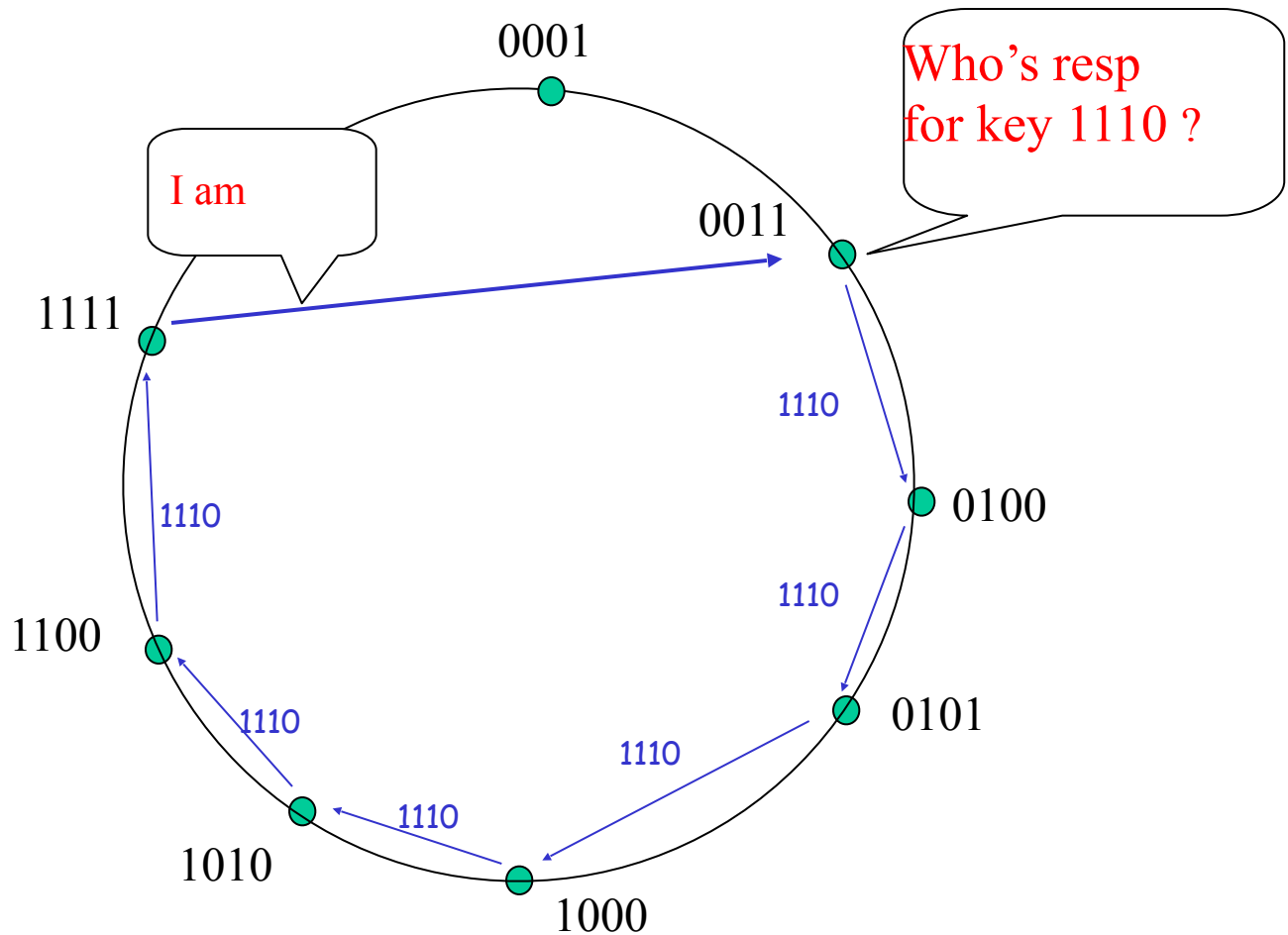
# Circular DHT (1)



- ❑ Each peer only aware of immediate successor and predecessor.
- ❑ "Overlay network"

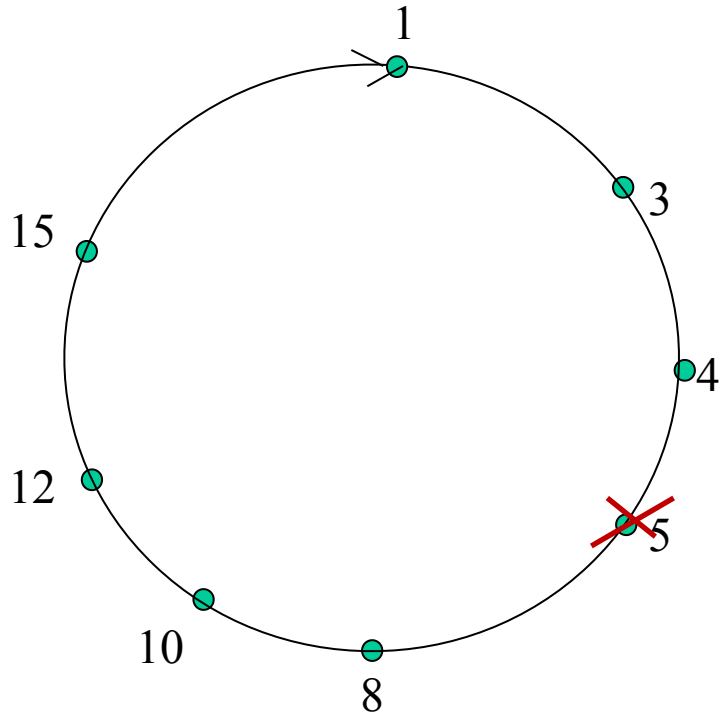
## Circle DHT (2)

$O(N)$  messages  
on avg to resolve  
query, when there  
are  $N$  peers



Define closest  
as closest  
successor

# Peer Churn

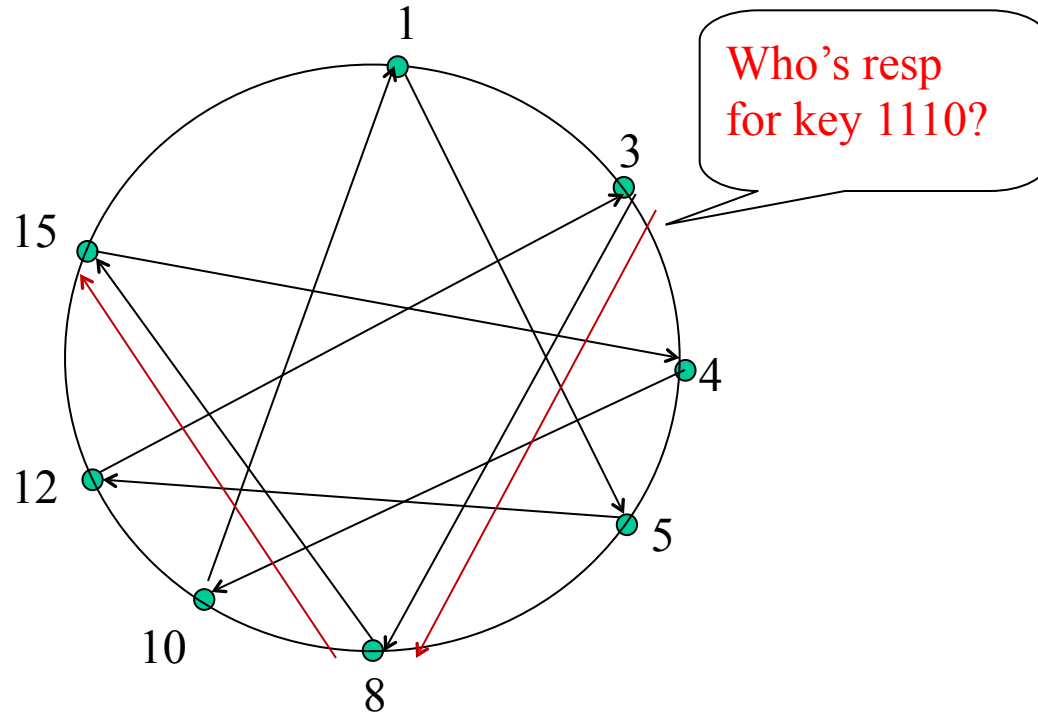


- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- ❑ Peer 5 abruptly leaves
- ❑ Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- ❑ What if peer 13 wants to join?



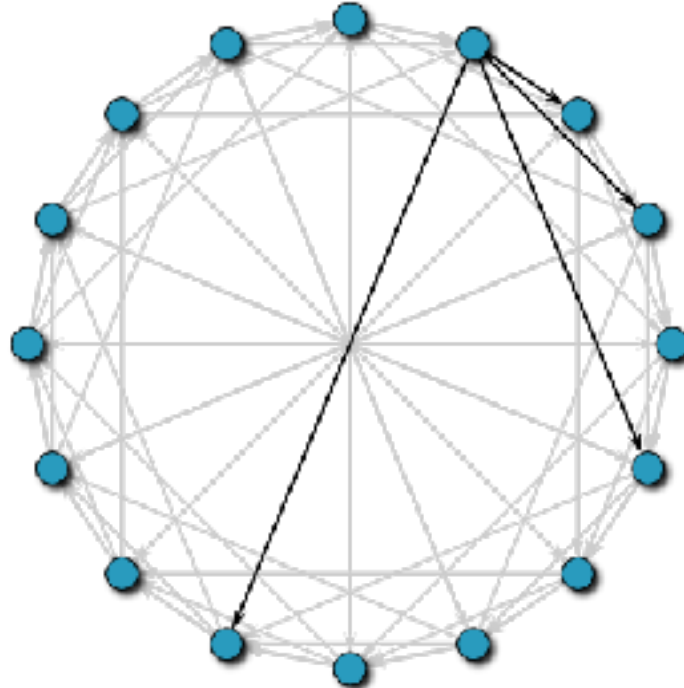
# Circular DHT with Shortcuts



- ❑ Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- ❑ Reduced from 6 to 2 messages.
- ❑ Possible to design shortcuts so  $O(\log N)$  neighbors,  $O(\log N)$  messages in query

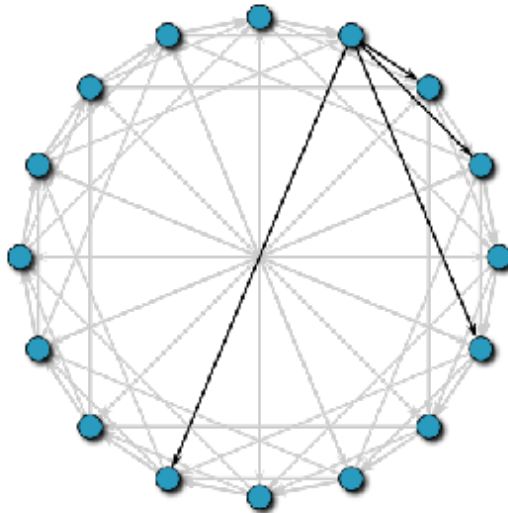
# Chord

- ❖ points sur un cercle la distance est la longueur de l'arc orienté. L'espace des clés est décomposé en segments. Si  $i$  et  $j$  sont des nœuds adjacents,  $j$  a toutes les clés comprises entre  $i+1$  et  $j$ .

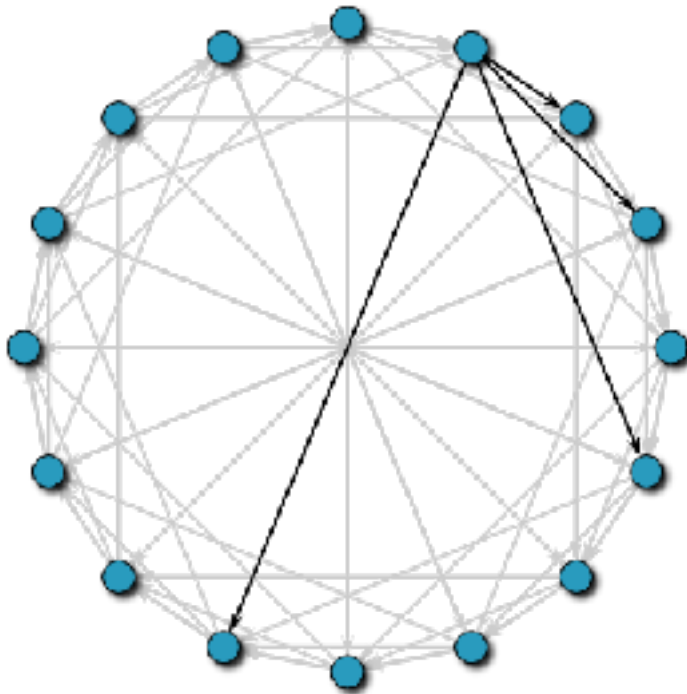


# Chord

- ❖ Anneau de taille  $2^m$  (clés de 0 à  $2^m-1$ )
- ❖ Hachage par SHA1 (ID-clés réparties de façon uniforme)
- ❖ Chaque nœud maintient une liste de K successeurs et K prédécesseurs sur l'anneau



# Chord



Chaque nœud maintient une  
table de nœuds à distance  
 $(n + 2^i) \bmod 2^m$

# Chord

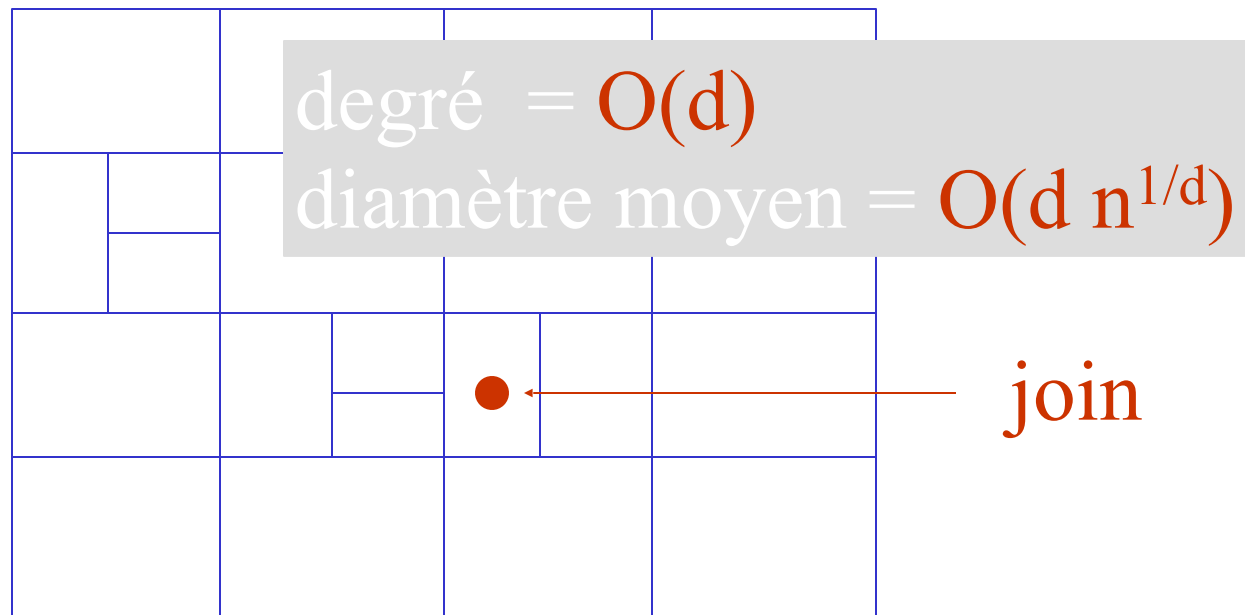
Chercher le nœud responsable de  $k$  à partir du nœud  $n$ :

- ❑ Soit  $c(k)$  le premier nœud successeur de  $(n + 2^{k-1}) \bmod 2^m$
- ❑ Si  $k$  appartient à  $]n, \text{successor}]$ : trouvé
- ❑ Sinon transmettre la demande au plus près prédécesseur  $f$  de  $k$  dans  $C(k)$
- ❑ Si les nœuds sont répartis uniformément on y arrive en  $m$  étapes (et donc en  $\log$  de la taille  $= (2^m)$ ) à chaque appel la distance entre le nœud et  $k$  divisée par 2.

# Un autre exemple: CAN

"Content-Addressable Network"

tore de dimension  $d$



# Téléchargement

- ❑ protocoles ad-hoc (Edonkey, Bittorrent) ou http
- ❑ Swarming (essaim)
  - ❖ téléchargement d'un fichier en téléchargeant différentes parties en parallèle depuis plusieurs clients
- ❑ téléchargement multiple
  - ❖ le fichier est décomposé en  $n$  blocs
  - ❖ calcul du hachage de chaque bloc
  - ❖ fichier est identifié par le hachage des blocs

# Quelques réseaux

## □ Napster

- ❖ (historique)
- ❖ Protocole:
  - architecture centralisée
  - recherche par indexation

## □ Fasttrack

- ❖ clients: Kazaa Mldonkey
- ❖ architecture hybride
- ❖ recherche par indexation et diffusion entre ultrapeers
- ❖ identification faible des fichiers (MD5 sur 300ko puis hachage sur 32 bits)
- ❖ http (+ swarming)



# Quelques réseaux

## □ Edonkey

- ❖ Clients: Edonkeys, Emule
- ❖ protocole
  - faiblement centralisée
  - recherche par indexation TCP
  - recherche multi-serveurs par UDP
  - téléchargement en duplex avec bitmaps, streaming et swarming (Bittorrent dans les dernières versions)
  - système de crédits pour inciter au partage

## □ Overnet/Kad

- ❖ téléchargement Edonkey
- ❖ décentralisé
- ❖ DHT (Kademlia)

# Quelques réseaux

## ❑ Gnutella

- ❖ architecture hybride
- ❖ Recherche par diffusion courte (TTL <8) avec filtres de Bloom
- ❖ Gnutella et Gnutella2
- ❖ téléchargement par http avec possibilité de swarming

# Quelques réseaux

## ❑ Bittorrent

- ❖ pas de découverte, un fichier .torrent contient les infos sur le fichier
- ❖ localisation centralisée pour chaque fichier (tracker)
- ❖ Téléchargement avec bitmaps
  - le client commence à fournir du contenu à ses voisins
  - au bout d'un certain temps il bloque les voisins qui ne répondent pas et conserve les quatre meilleurs

- ❖ BitTorrent (client original) · ABC · BitComet · BitTornado · Deluge · KTorrent · Popcorn Time · OneSwarm · Shareaza · Transmission ·  $\mu$ Torrent · Vuze (anciennement Azureus)

.....

# Quelques réseaux

## □ Freenet

- ❖ complètement distribué
- ❖ publication par copie
- ❖ recherche par diffusion en profondeur
- ❖ protocole et contenu cryptés
- ❖ documents signés par signature digitale
- ❖ axé sur l'anonymat
- ❖ Ants, GNUnet, Marabunta, Mute, Nodezilla, OFF System, Perfect Dark, Rodi, Rshare, Share, Sumi, Winny

# I2P (invisible Internet project)

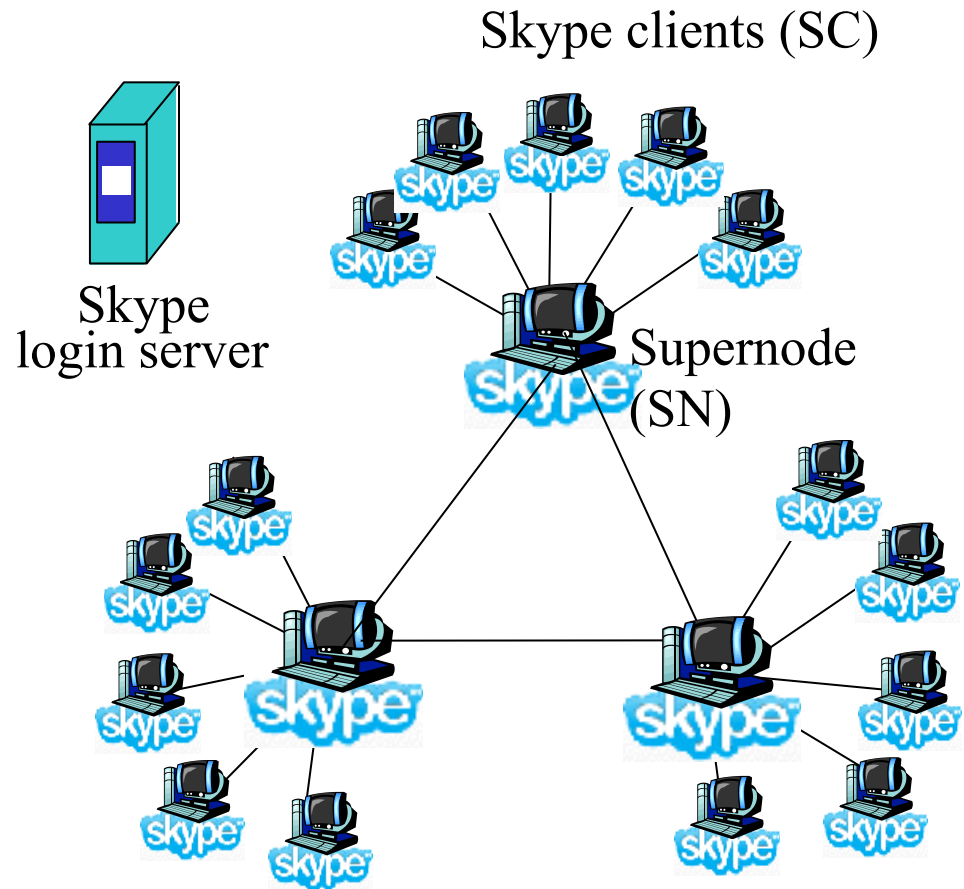
- ❑ Réseau anonyme pour préserver la confidentialité des communications sur internet
- ❑ Pour communiquer avec un pair, le trafic passe par une série d'autre pairs (constitution d'un tunnel - I2PTunnel):
  - ❖ Un observateur ne peut savoir qui est la source et le destinataire final
  - ❖ Chaque pair peut prétendre que le message ne lui est pas destiné
- ❑ Identification grâce à une clef
- ❑ I2PSnark (client BitTorrent), Susimail, SusiDNS

# Friend-to-Friend ( F2F)

- Réseau P2P anonyme, échange de fichiers uniquement avec des utilisateurs de confiance (amis)
- Échange indirecte avec les amis de mes amis (en passant par amis à chaque fois)
- Reconnaissance d'amis par adresse IP ou signature numérique
- anoNet, OneSwarm, Freenet, Retroshare, Galet, Turtle F2F, GNUnet (avec activation F2F), Waste (avec ping désactivé), Kerjodando

# P2P Case study: Skype

- ❑ inherently P2P: pairs of users communicate.
- ❑ proprietary application-layer protocol (inferred via reverse engineering)
- ❑ hierarchical overlay with SNs
- ❑ Index maps usernames to IP addresses; distributed over SNs



# Peers as relays

- ❑ Problem when both Alice and Bob are behind "NATs".
  - ❖ NAT prevents an outside peer from initiating a call to insider peer
- ❑ Solution:
  - ❖ Using Alice's and Bob's SNs, Relay is chosen
  - ❖ Each peer initiates session with relay.
  - ❖ Peers can now communicate through NATs via relay

