Architectures «Peer 2 Peer» et EV Massivement Multi-utilisateurs

Patrice Torguet
torguet@irit.fr
Université Paul Sabatier

Partie 1 : P2P

- Introduction
- Problèmes des réseaux P2P
- Opérations protocolaires
- Démarrage d'un réseau P2P
- P2P sans structure
- P2P structurés et DHTs

Introduction

- Le précurseur : Napster
  - Partage de fichier avec transferts P2P
  - Mais : 1 serveur indexait tout (peers et fichiers) => problème de fiabilité et ... vis à vis de la loi
- Applications actuelles :
  - Partage de fichier (eMule, Bittorrent...)
  - Video conférences (Skype...)
  - Calcul réparti (SETI@home, bio-informatique...)
- Applications futures :
  - Réseaux de capteurs
  - Systèmes de fichiers distribués
  - MMVEs

Introduction

- Caractéristiques :
  - complètement réparti (pas de serveur) => P2P Total
  - ressources hétérogènes (P2P ≠ Grille)
  - hautement dynamique => Les peers peuvent disparaitre
- Similaire :
  - Réseaux ad-hoc
  - Grilles

Problèmes des réseaux P2P

- Problème principal : «churning»
  - Les peers apparaissent et disparaissent sans arrêt
  - Mais on doit maintenir la connectivité
- Autres problèmes :
  - Sécurité (exemple : TOR)
  - Routage efficace (structure d'un réseau P2P ≠ structure de l'Internet)
  - Traversée des pare-feux (Firewalls)

Opérations protocolaires

- Orientées réseau
  - Join : un peer rejoint le réseau P2P
  - Leave : un peer quitte le réseau P2P
  - Lookup/query: messages utilisés pour trouver des peers (lookup) ou des données (query)
- Orientées données
  - Put : une nouvelle donnée est rendue disponible
  - Get : récupération d'une donnée
  - Delete : effacement d'une donnée

Démarrage d'un réseau P2P

- On doit pouvoir trouver un des peers pour lui envoyer la requête de connexion :
  - Peers fiables : peers bien connus qui fonctionnent 24h/24
  - Cache : stocke les IPs des peers connus précédemment
  - broadcast/multicast IP : ne marche en général que sur LAN
  - Service de cache : serveur de nom spécifique
  - Manuel : liste d'adresses IP donnée par l'utilisateur (trouvée sur le web...)

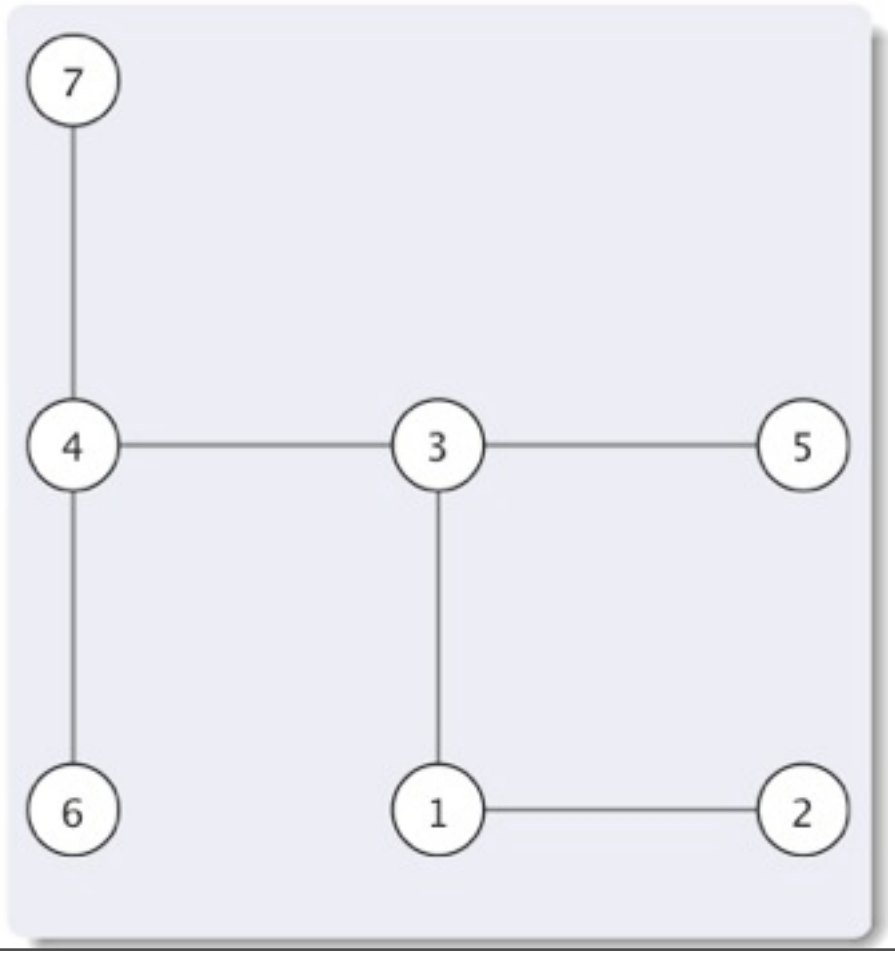
P2P non structuré

- Approche simple
  - connectivité aléatoire entre peers
  - utilisation de l'inondation (flooding) ou de l'exploration aléatoire pour trouver les données/peers
  - Peut avoir une efficacité très mauvaise
- Comment maintenir la connectivité ?

Gnutella (*newtella*)

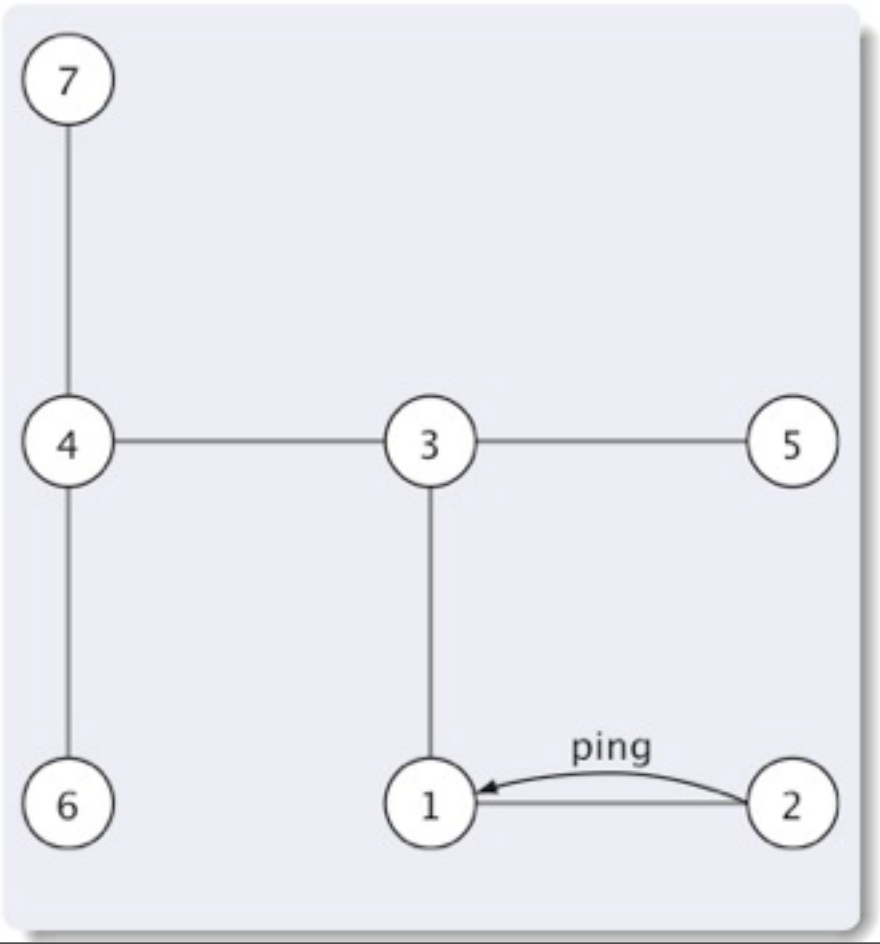
- Connexion utilisant une liste de peers bien connus
- Connectivité aléatoire utilisant TCP
- Les messages ont des id uniques et les ids et l'origine du message sont stockés dans des caches locaux (pour éviter la re-propagation et permettre la propagation inverse)
- Des messages PING/PONG sont utilisés pour trouver d'autres peers
  - PING envoyés au voisins et propagés
  - PONG sont rétro propagés et ramènent des adresses IP

Gnutella : exemple de réseau



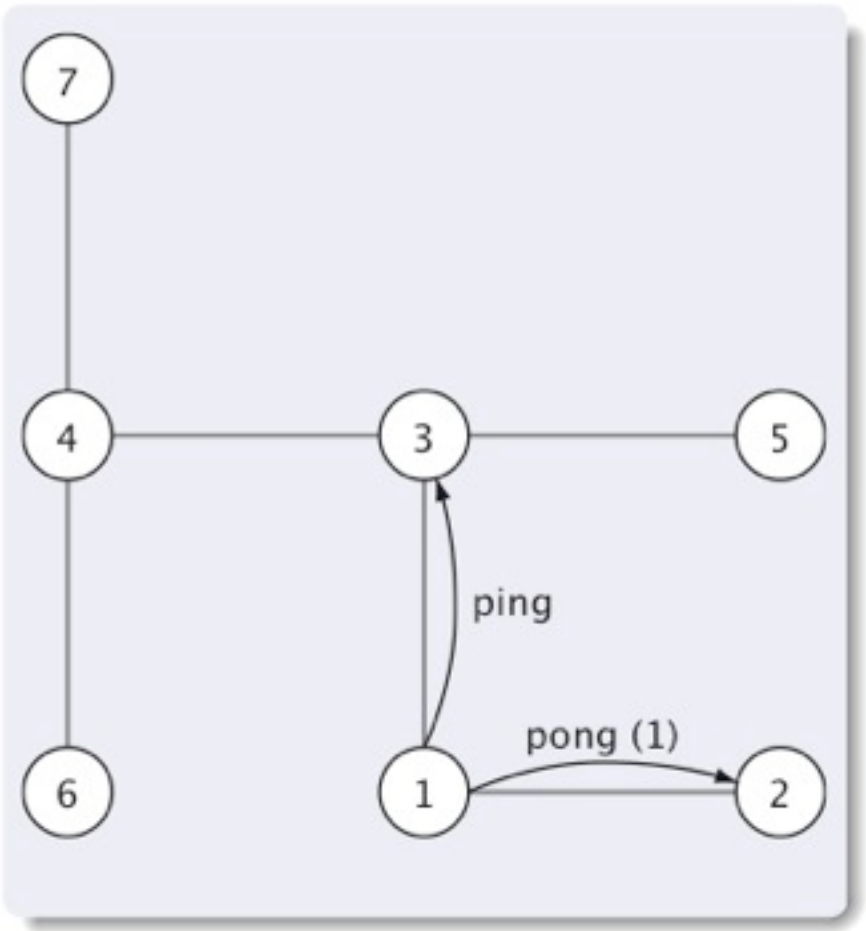
Figures d'Aaron Harwood du NICTA

Gnutella : PING/PONG



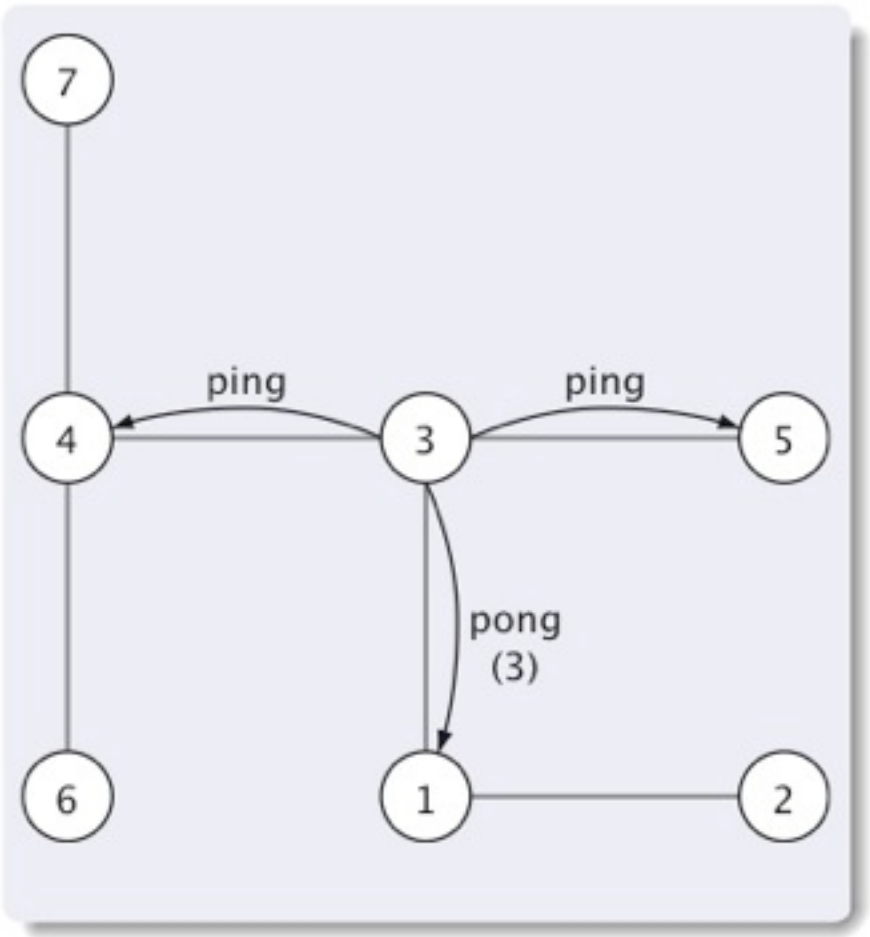
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



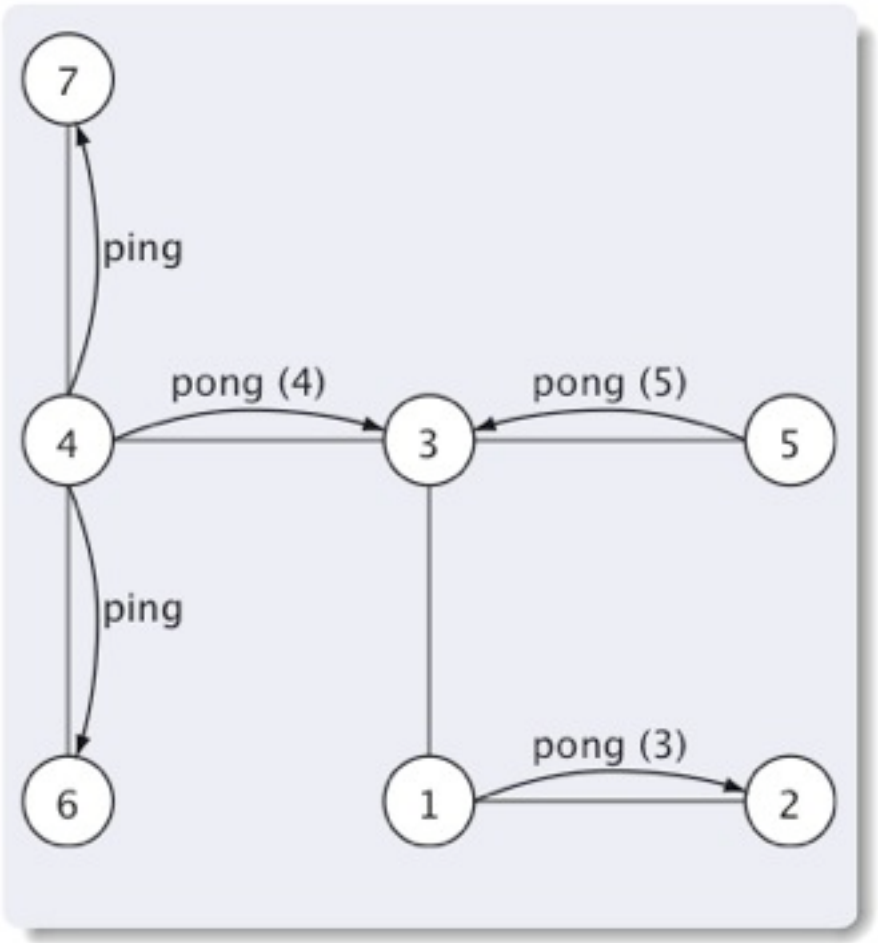
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



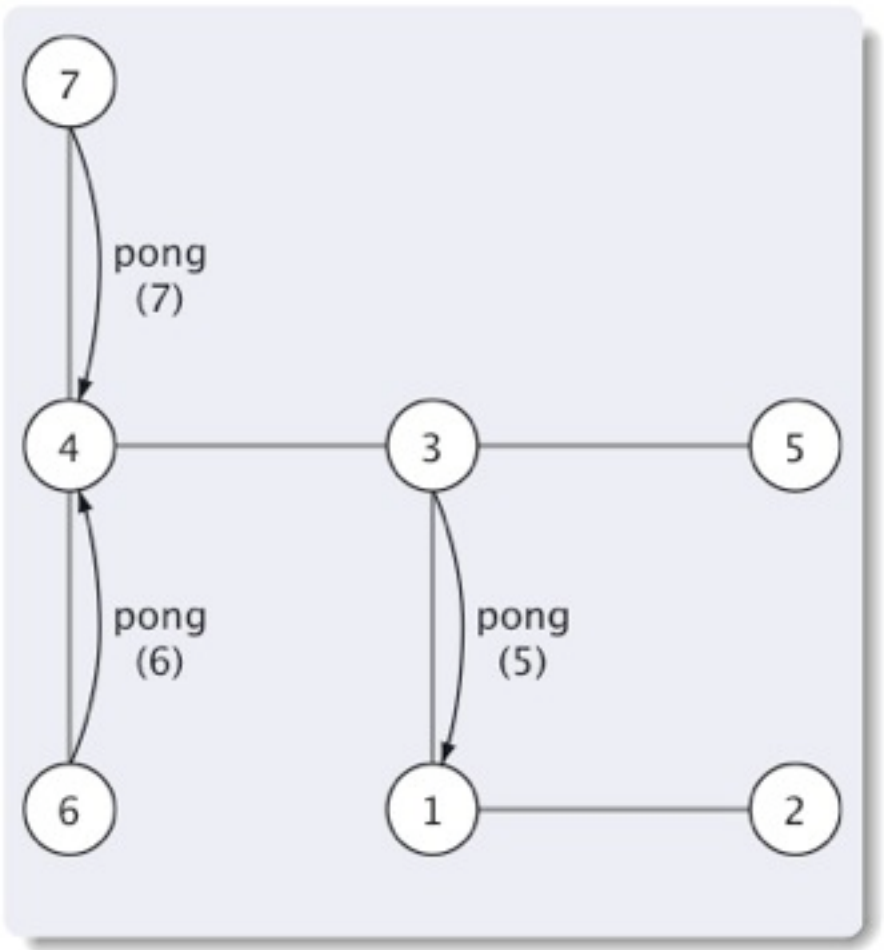
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



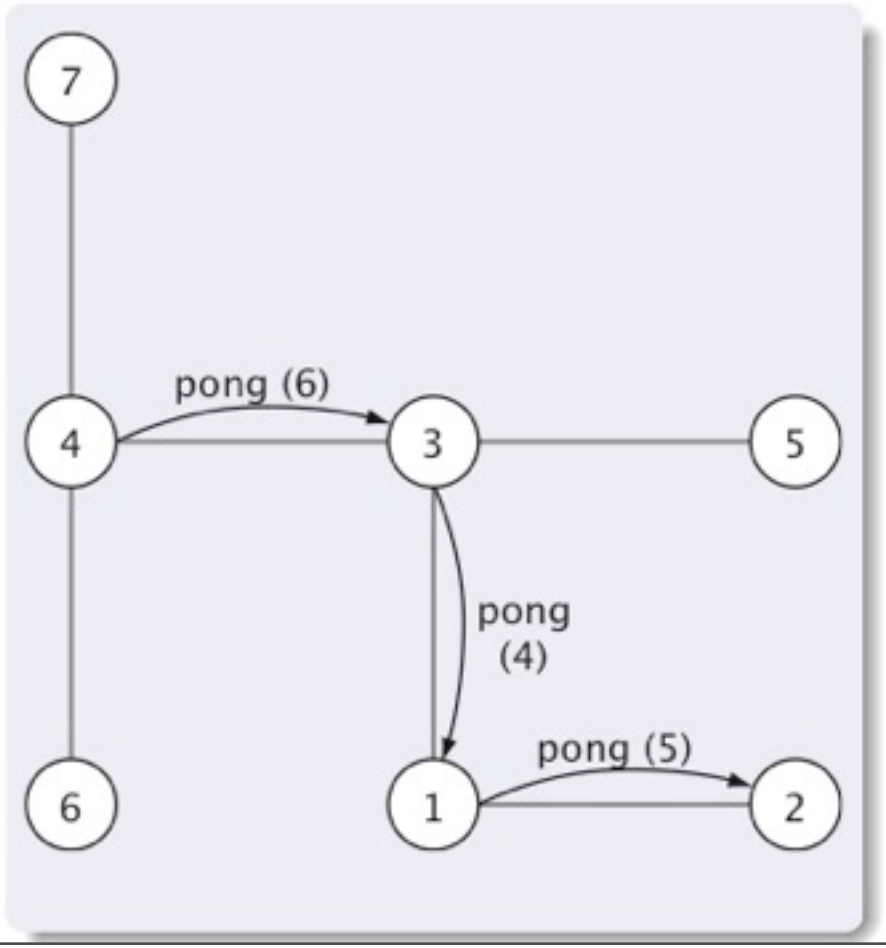
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



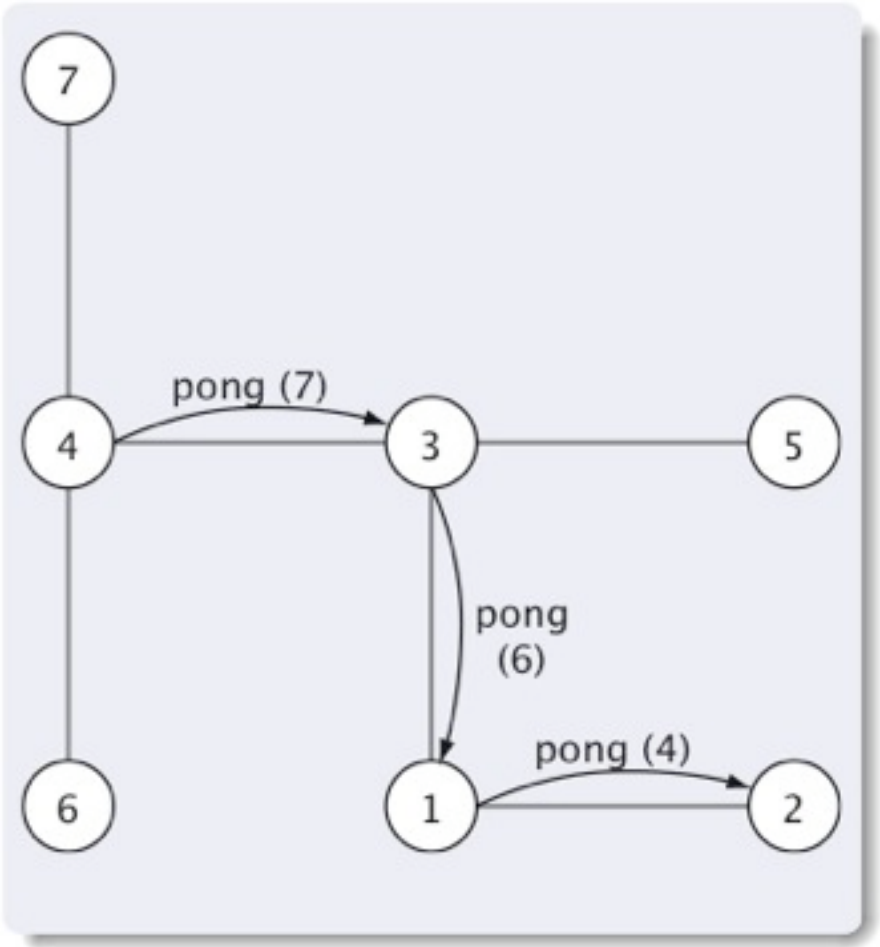
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



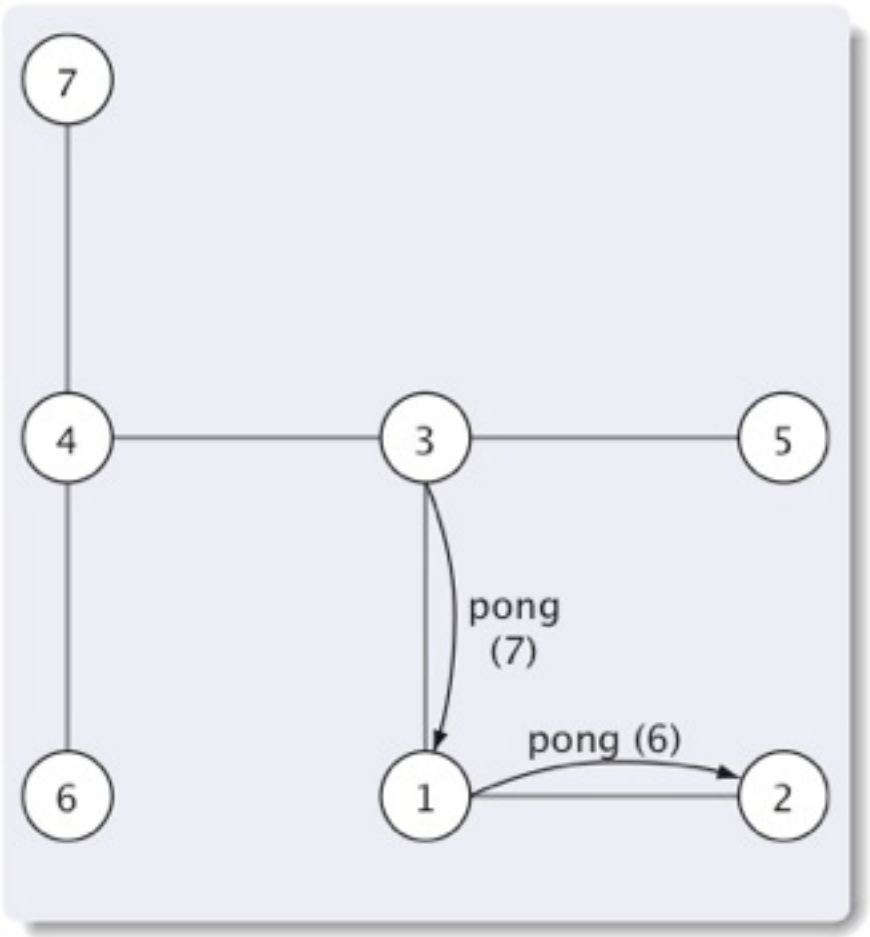
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



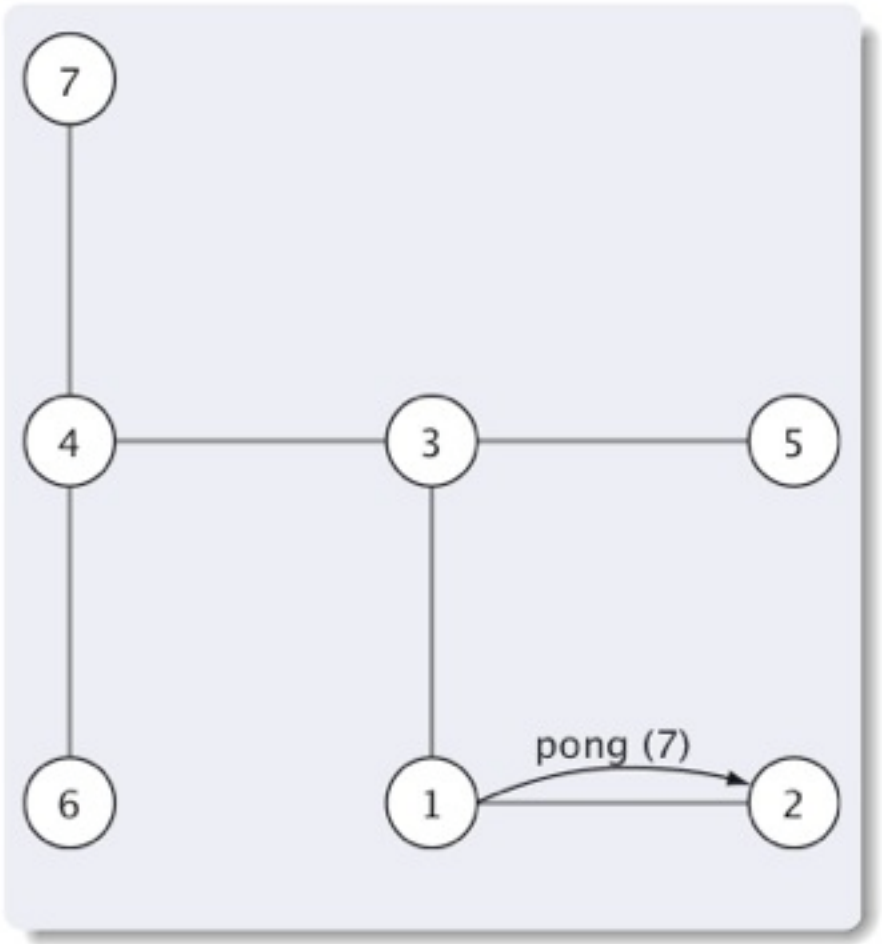
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



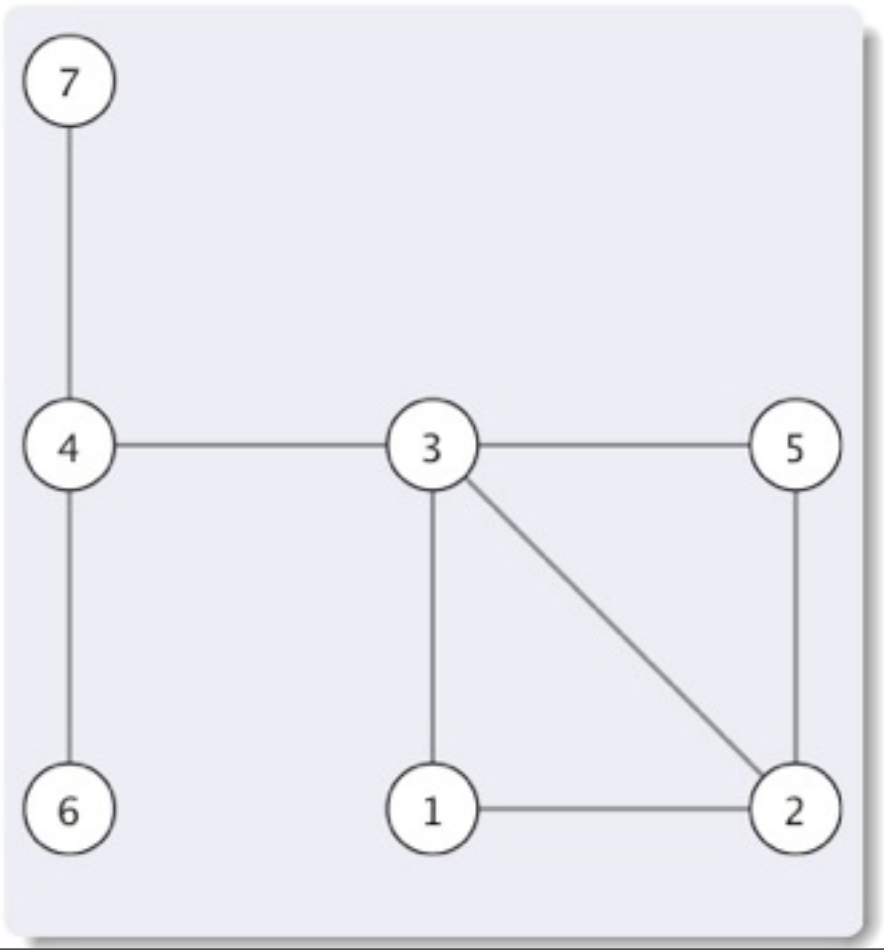
Figures d'Aaron Harwood du NICTA

Gnutella: PING/PONG



Figures d'Aaron Harwood du NICTA

Gnutella: new connections

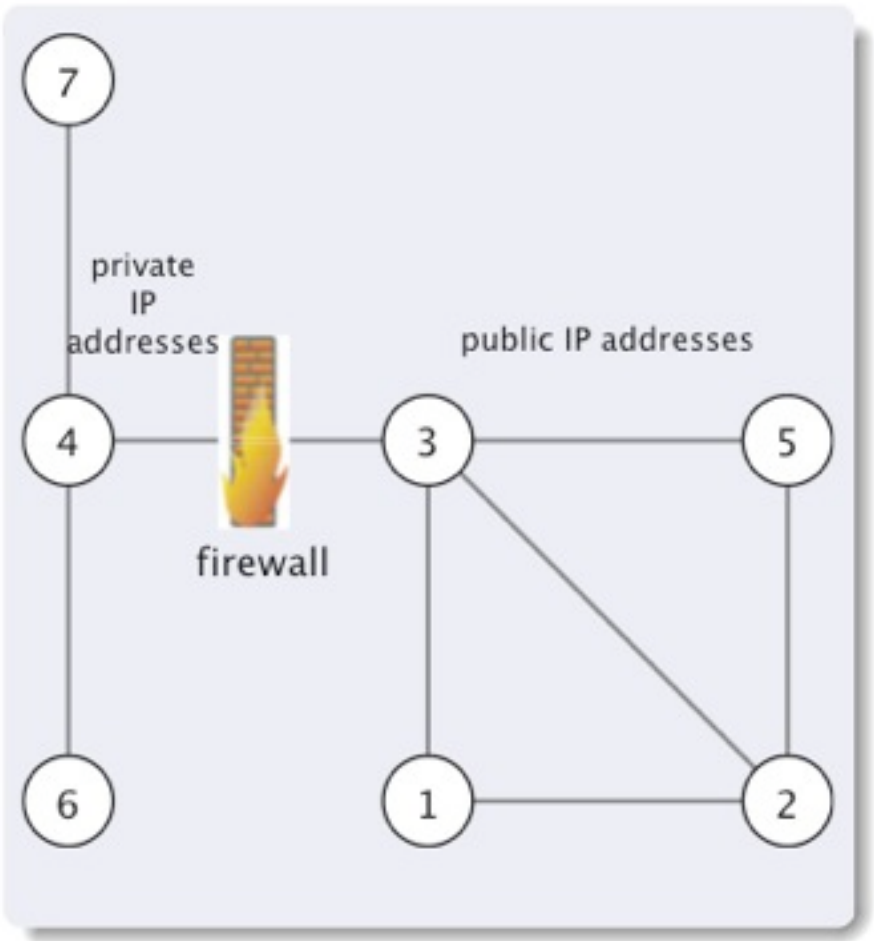


Figures d'Aaron Harwood du NICTA

Gnutella (*newtella*)

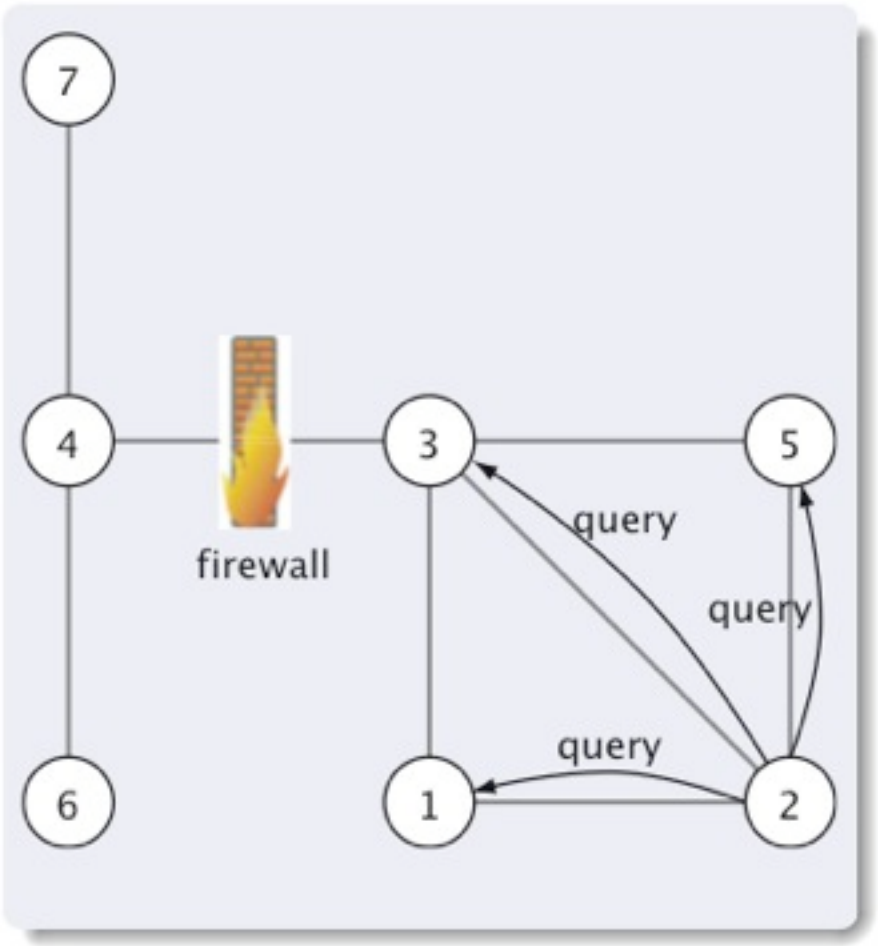
- Les messages QUERY sont envoyés par inondation
- Des messages HIT sont rétro propagés
- Des messages GET sont utilisés pour télécharger directement les fichiers
- Des messages PUSH sont utilisés pour demander l'envoi de données (pour la traversée des firewalls)

Gnutella: QHGP



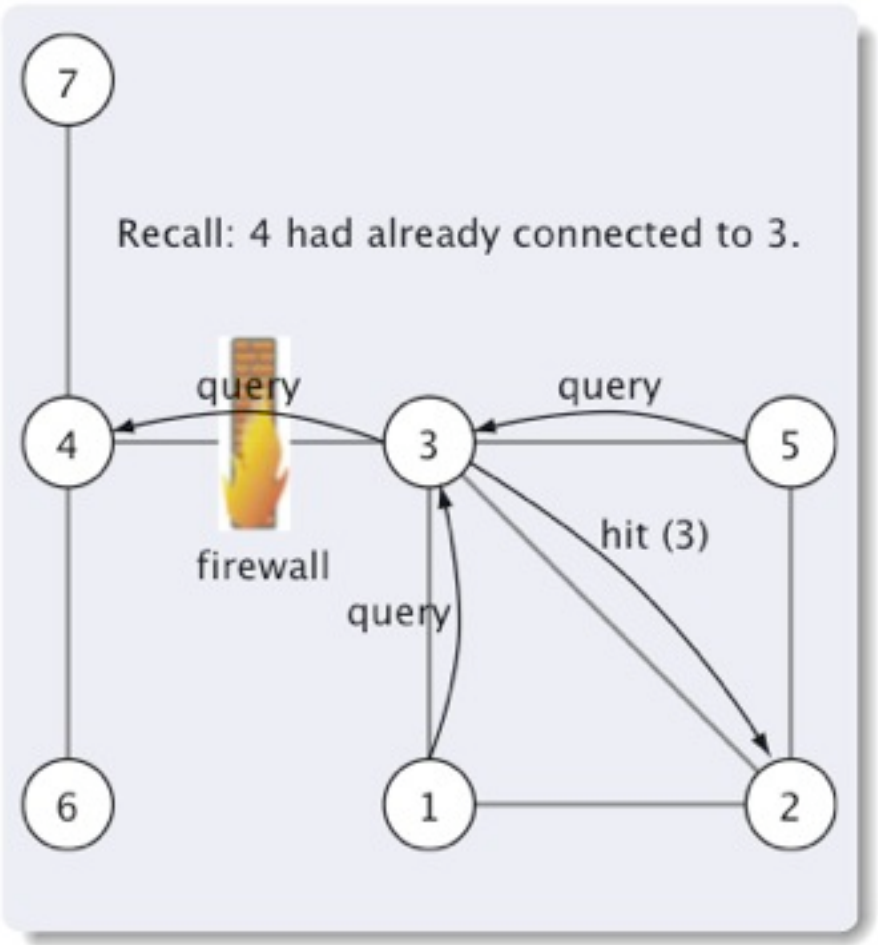
Figures d'Aaron Harwood du NICTA

Gnutella : inondation des requêtes



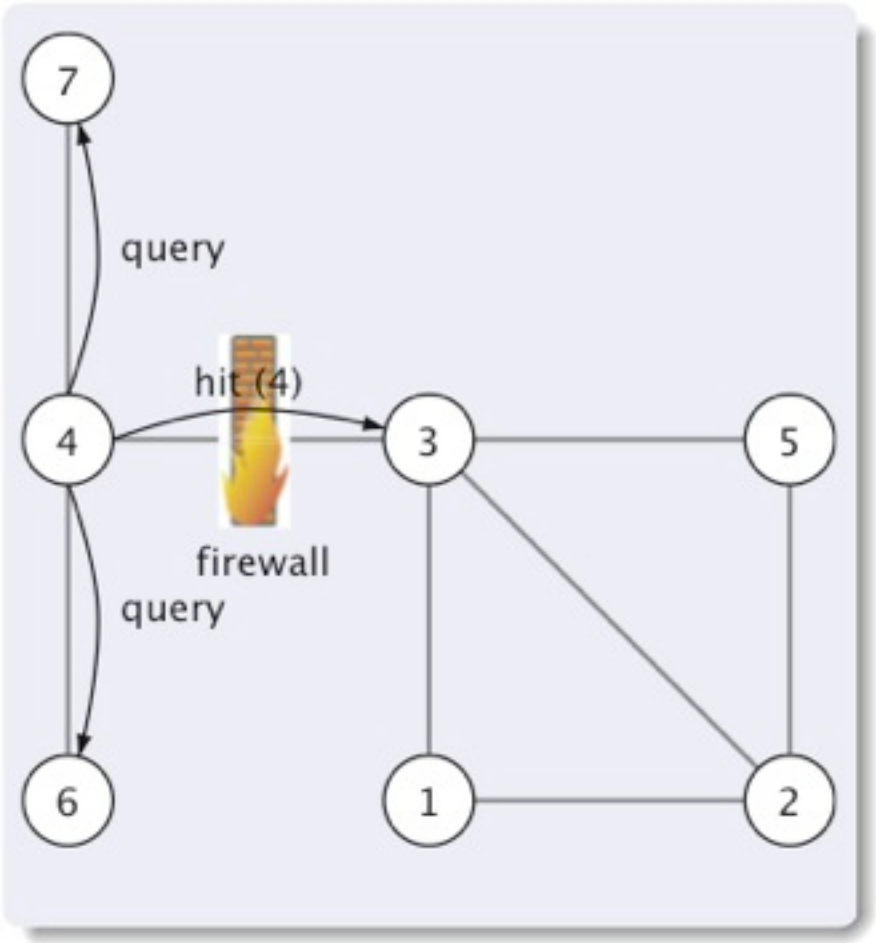
Figures d'Aaron Harwood du NICTA

Gnutella : traversée des pare-feux



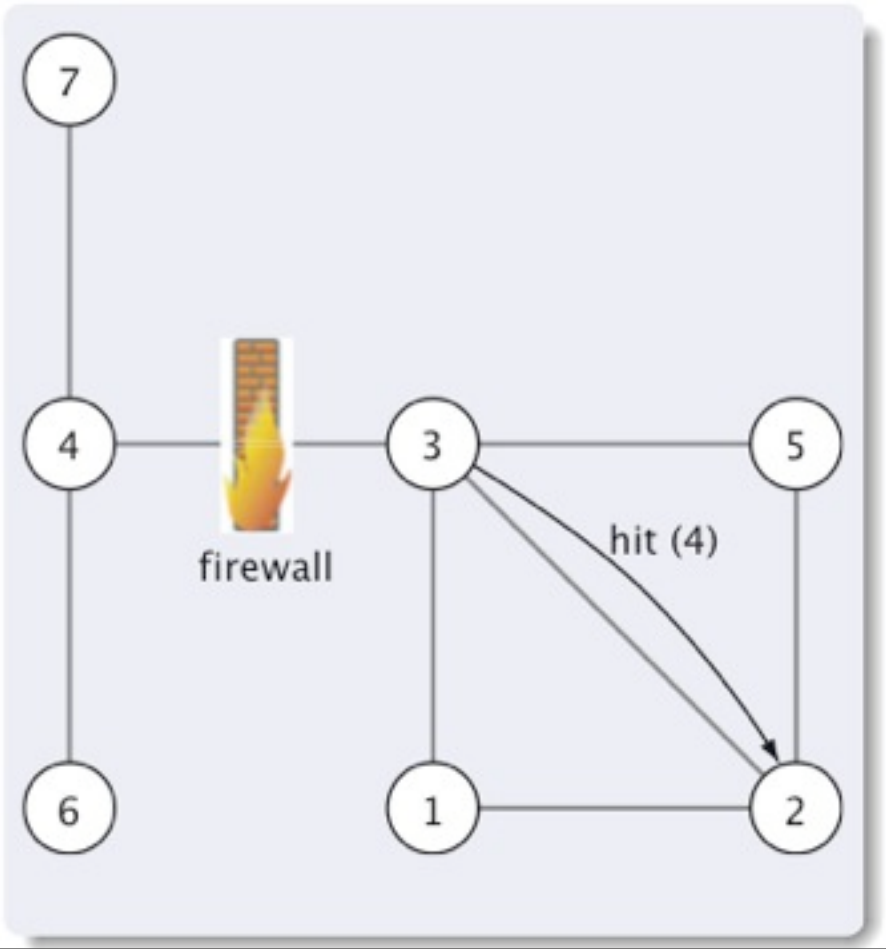
Figures d'Aaron Harwood du NICTA

Gnutella : les HIT reviennent de certains peers



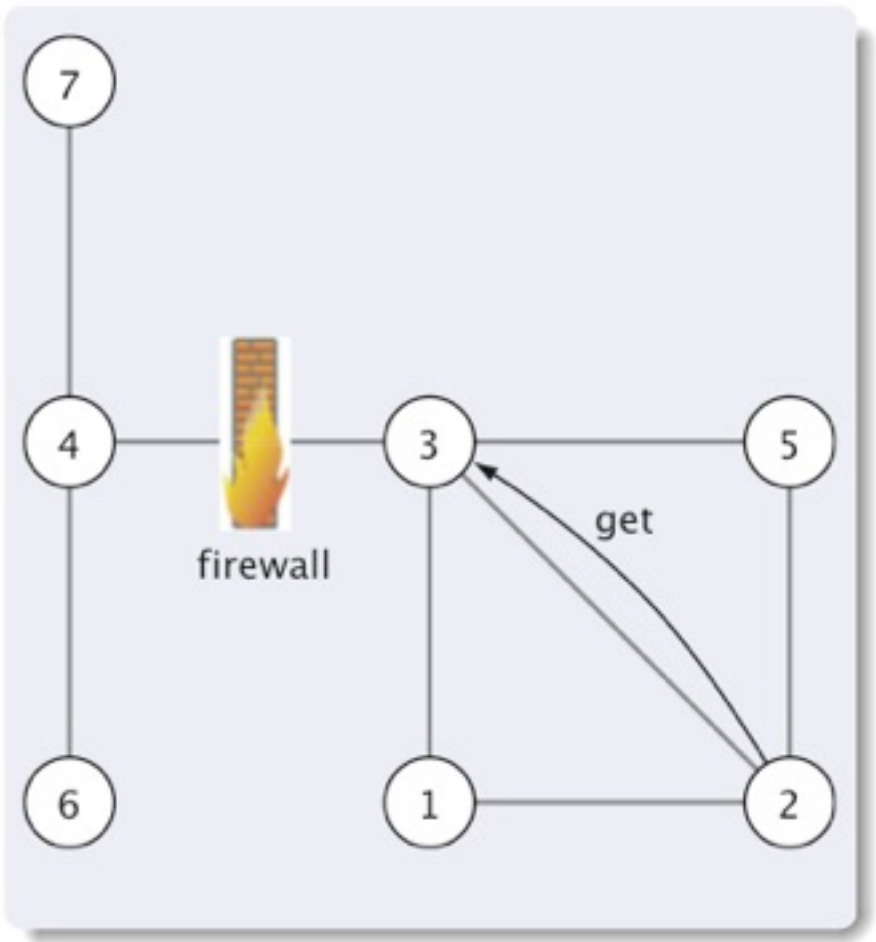
Figures d'Aaron Harwood du NICTA

Gnutella: les Hits sont rétro propagés



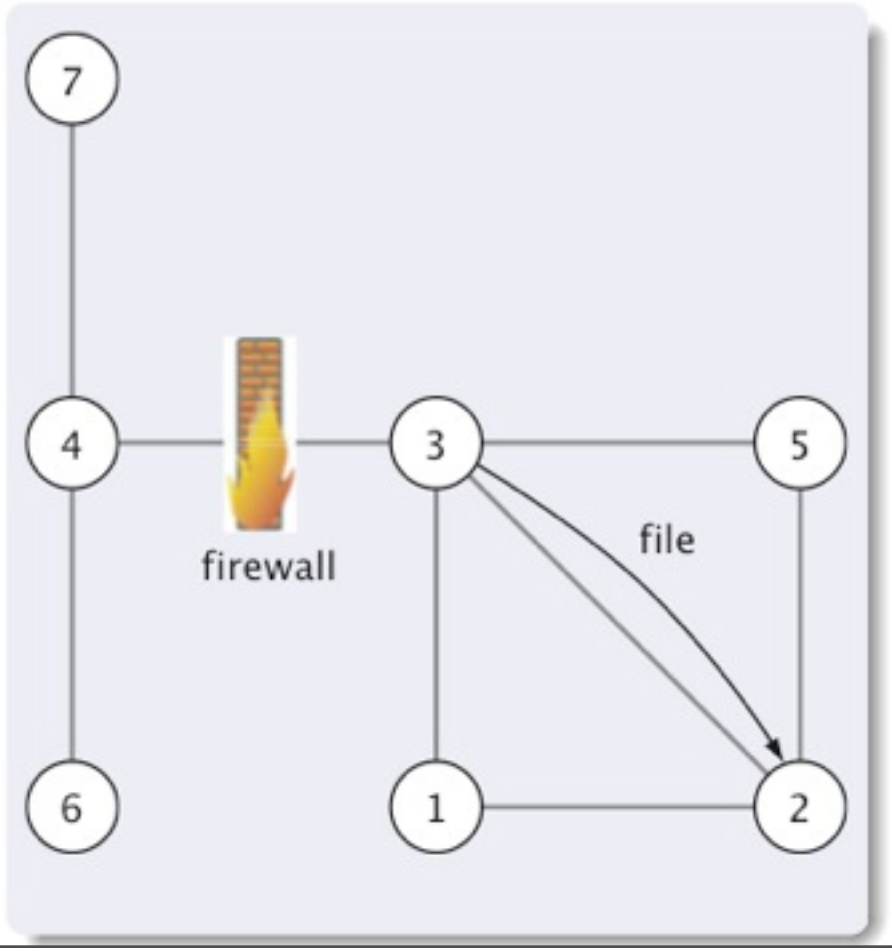
Figures d'Aaron Harwood du NICTA

Gnutella : Get vers une IP publique



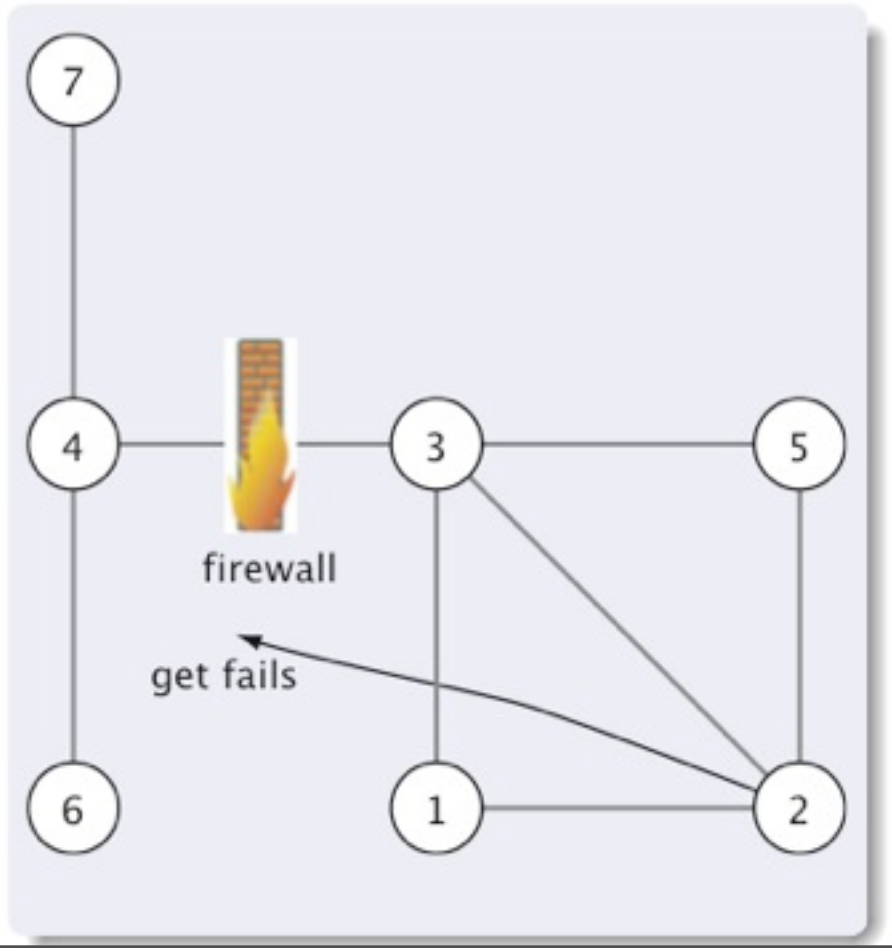
Figures d'Aaron Harwood du NICTA

Gnutella : Get vers une IP publique



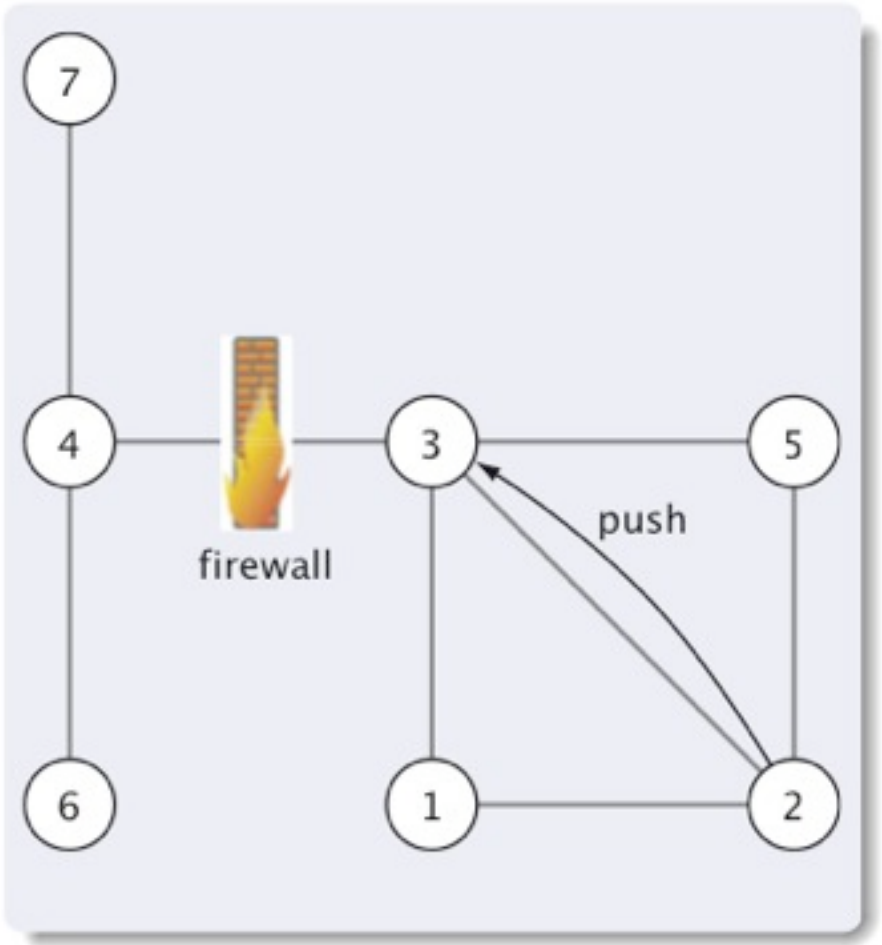
Figures d'Aaron Harwood du NICTA

Gnutella : Get ne marche pas pour une IP privée



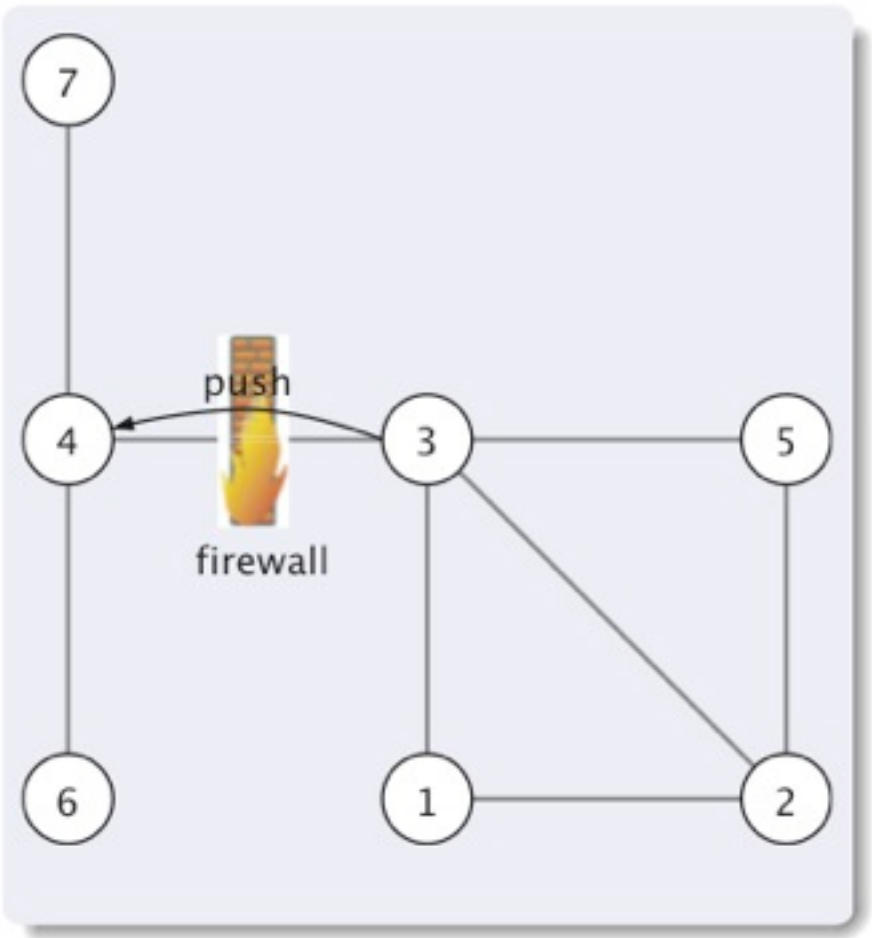
Figures d'Aaron Harwood du NICTA

Gnutella : on doit utiliser PUSH



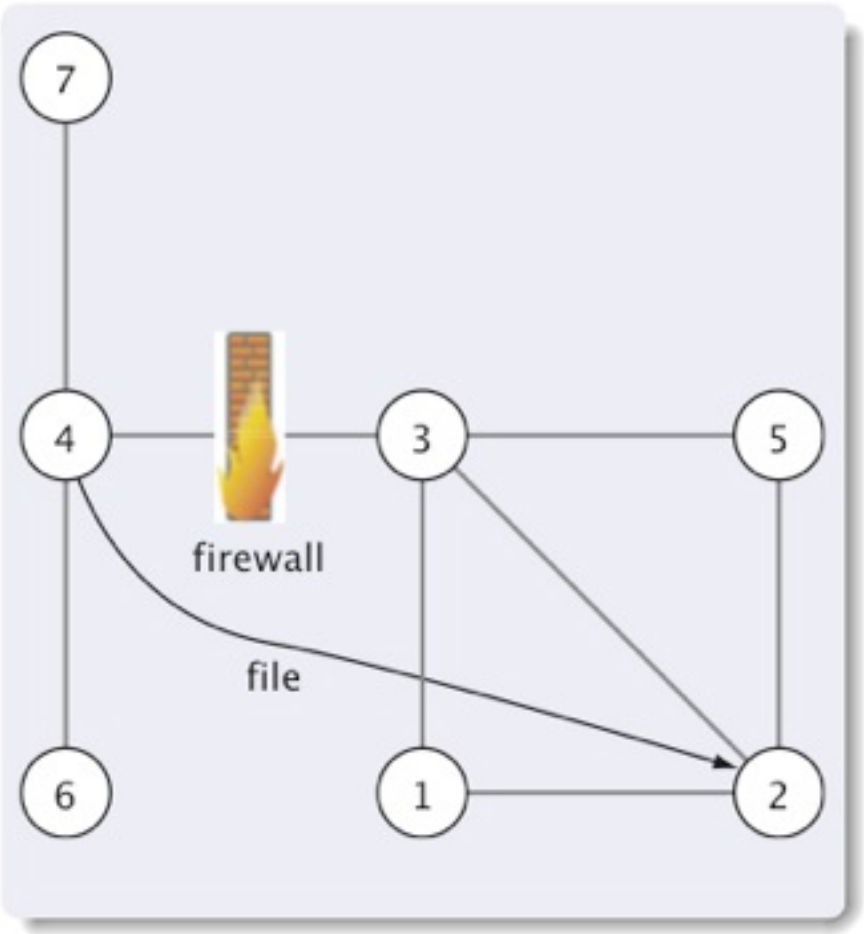
Figures d'Aaron Harwood du NICTA

Gnutella : on doit utiliser PUSH



Figures d'Aaron Harwood du NICTA

Gnutella : on doit utiliser PUSH



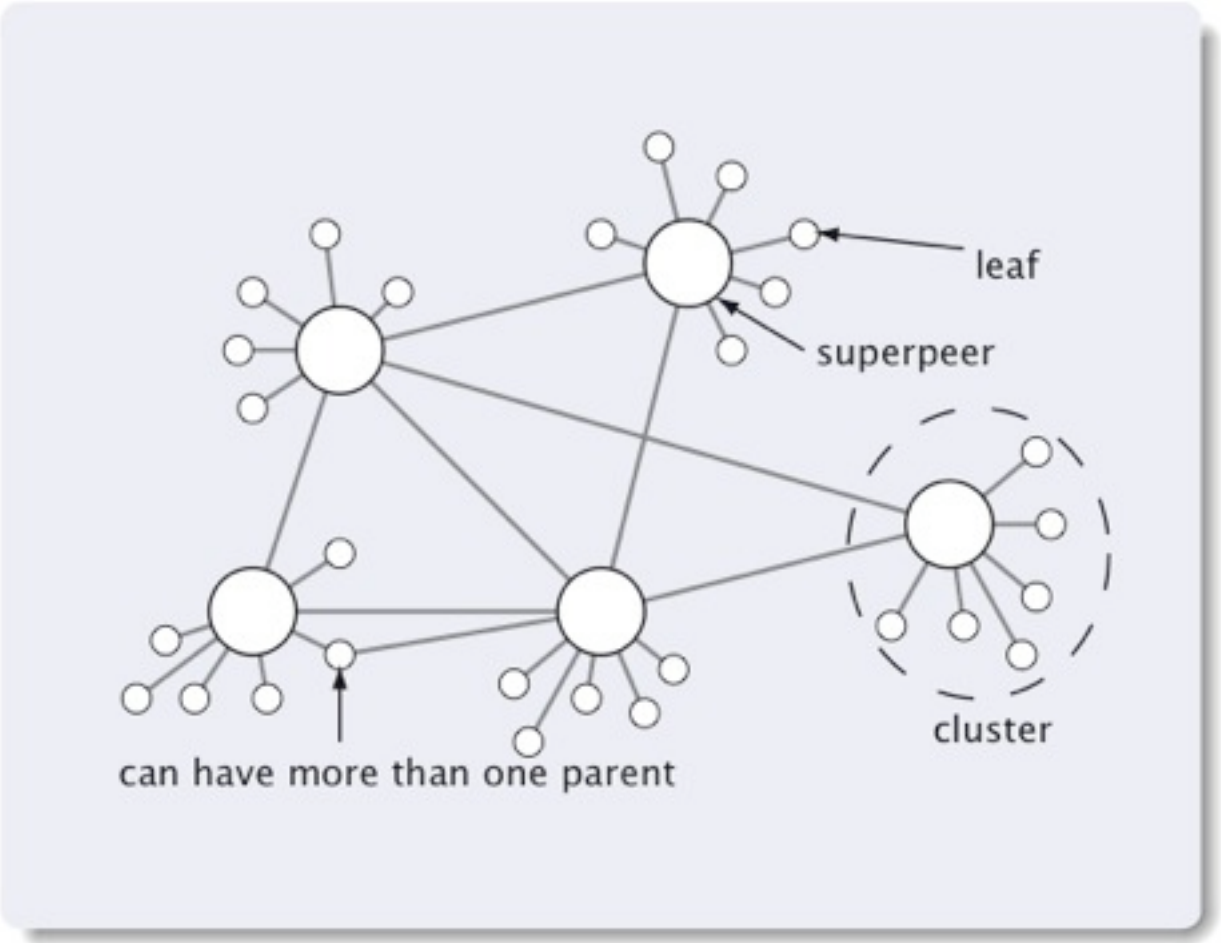
Figures d'Aaron Harwood du NICTA

Gnutella (*newtella*)

- Introduction de super-peers (ultra peers) pour optimiser Gnutella
  - Ils ont une bonne bande passante
  - Ils indexent les fichiers pour d'autres peers (et gèrent la recherche)
  - Un peer peut-être connecté à plusieurs super-peers (utilisé pour la fiabilité)

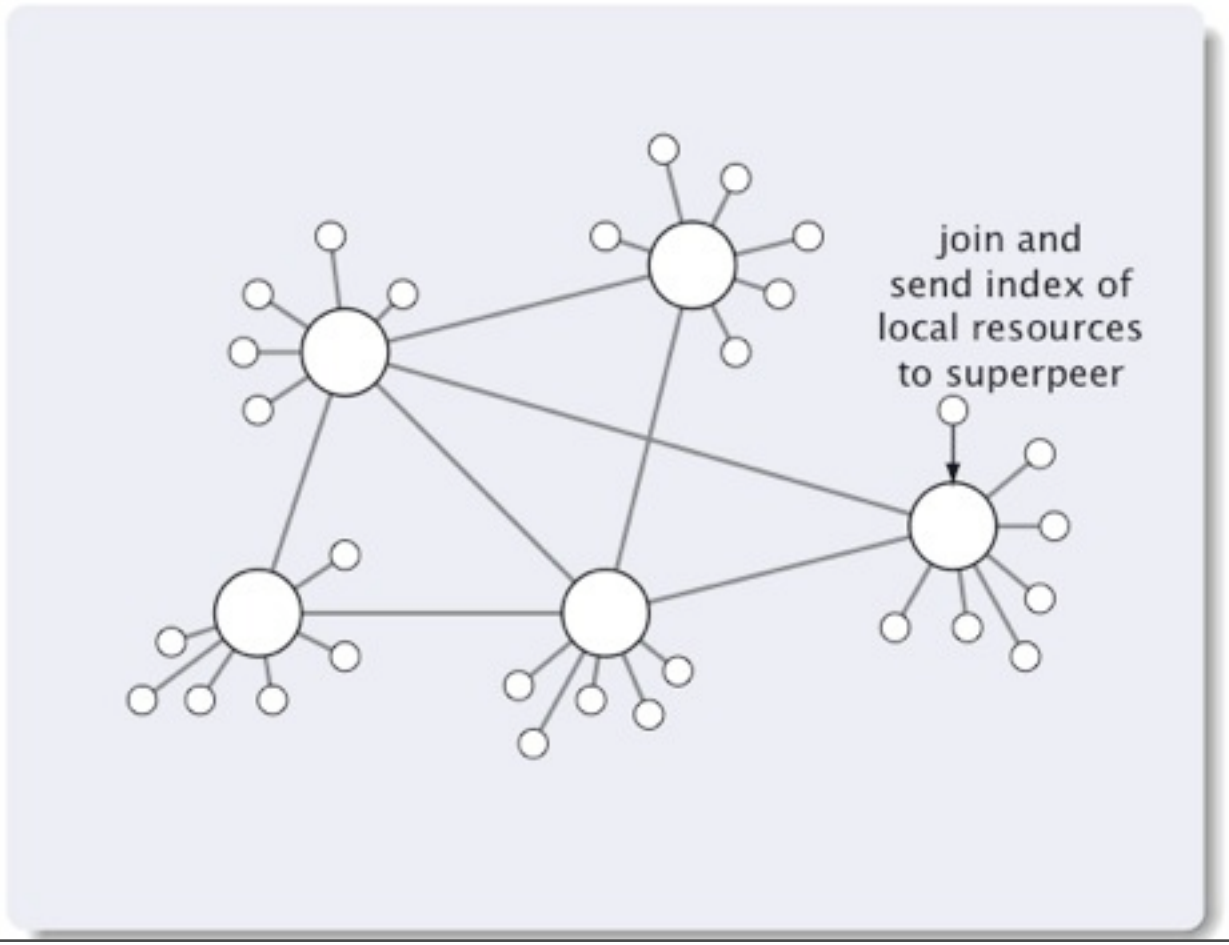
  - Mais il y a toujours de l'inondation au niveau des super-peers
  - La promotion vers un super-peer ne se fait qu'à la connexion

Gnutella : Super-peers



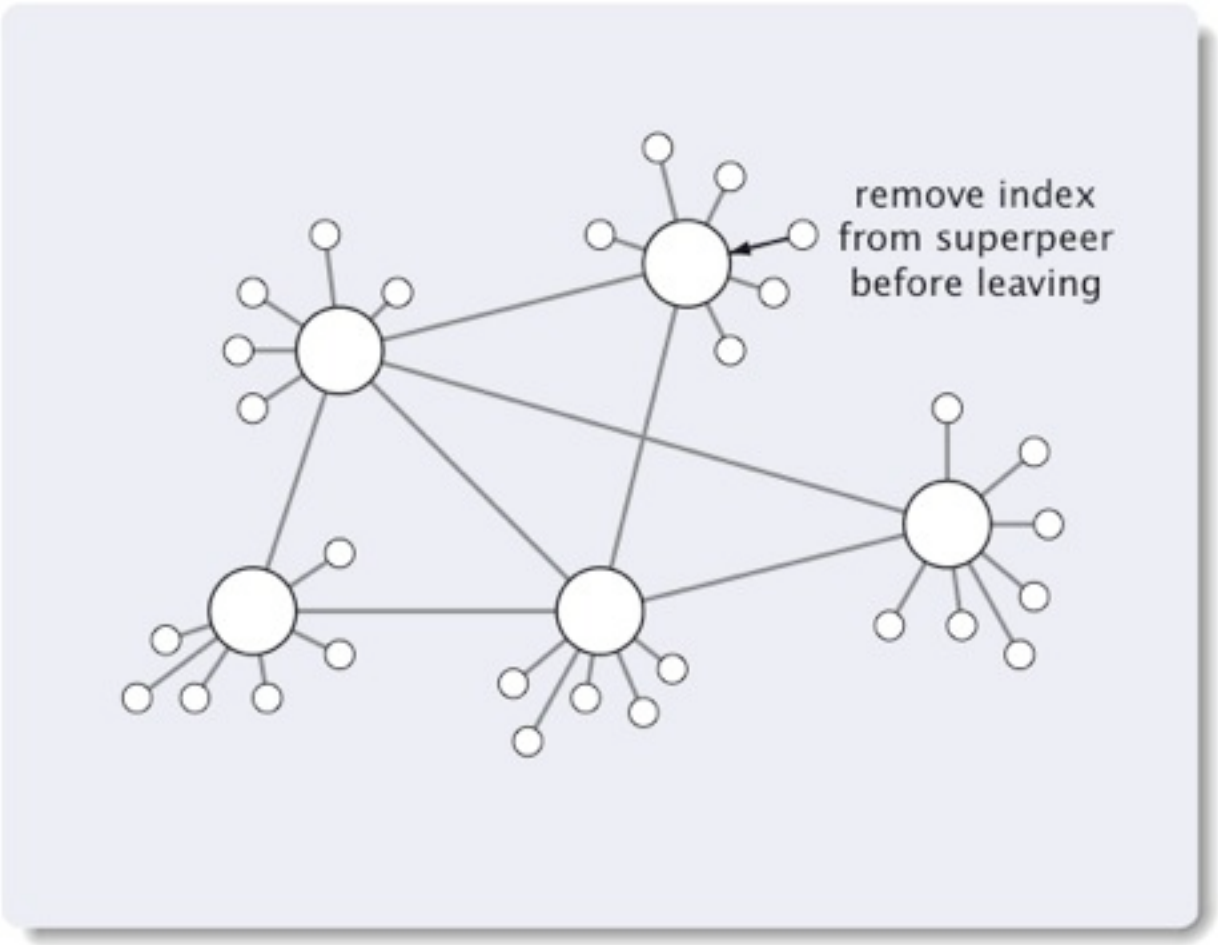
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



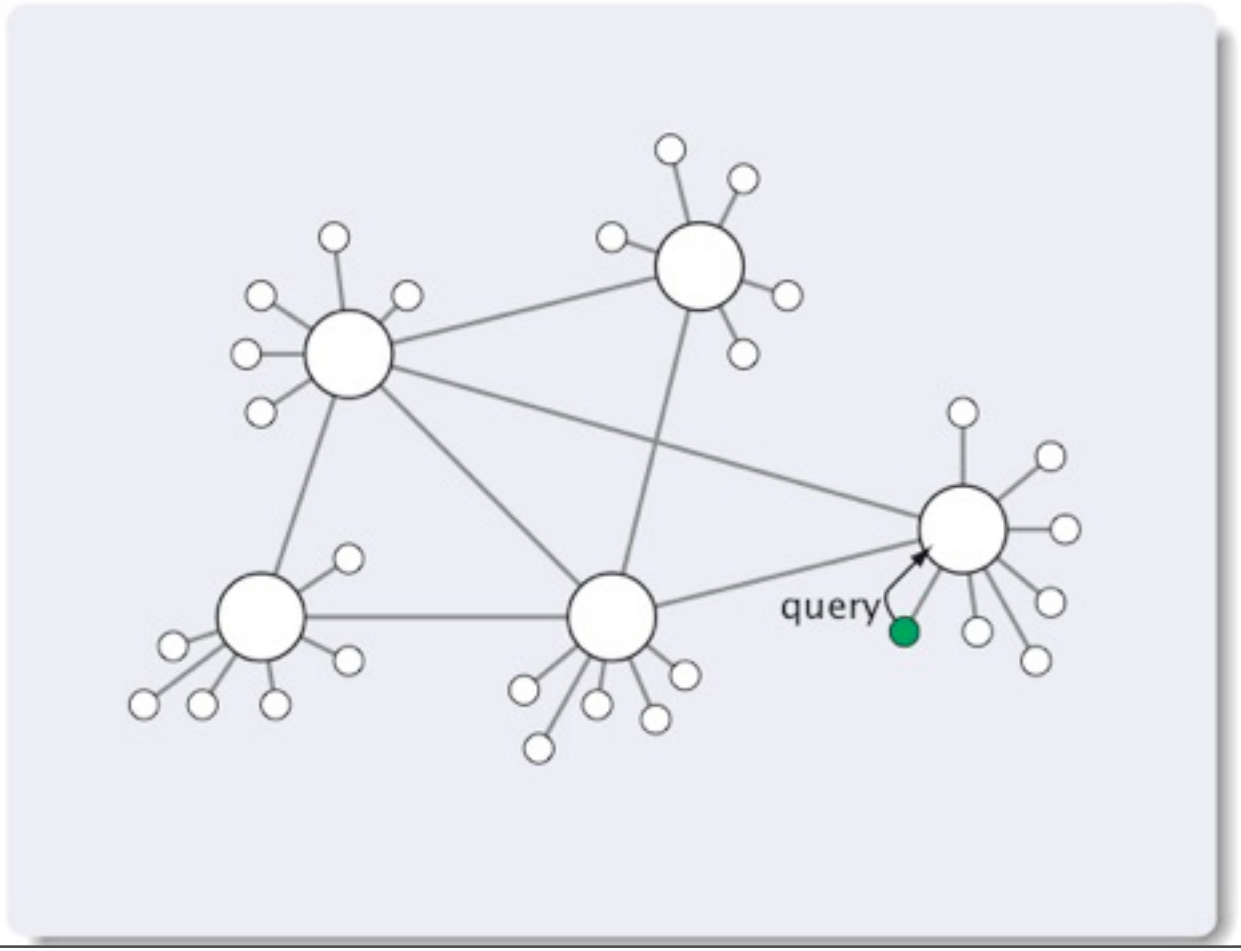
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



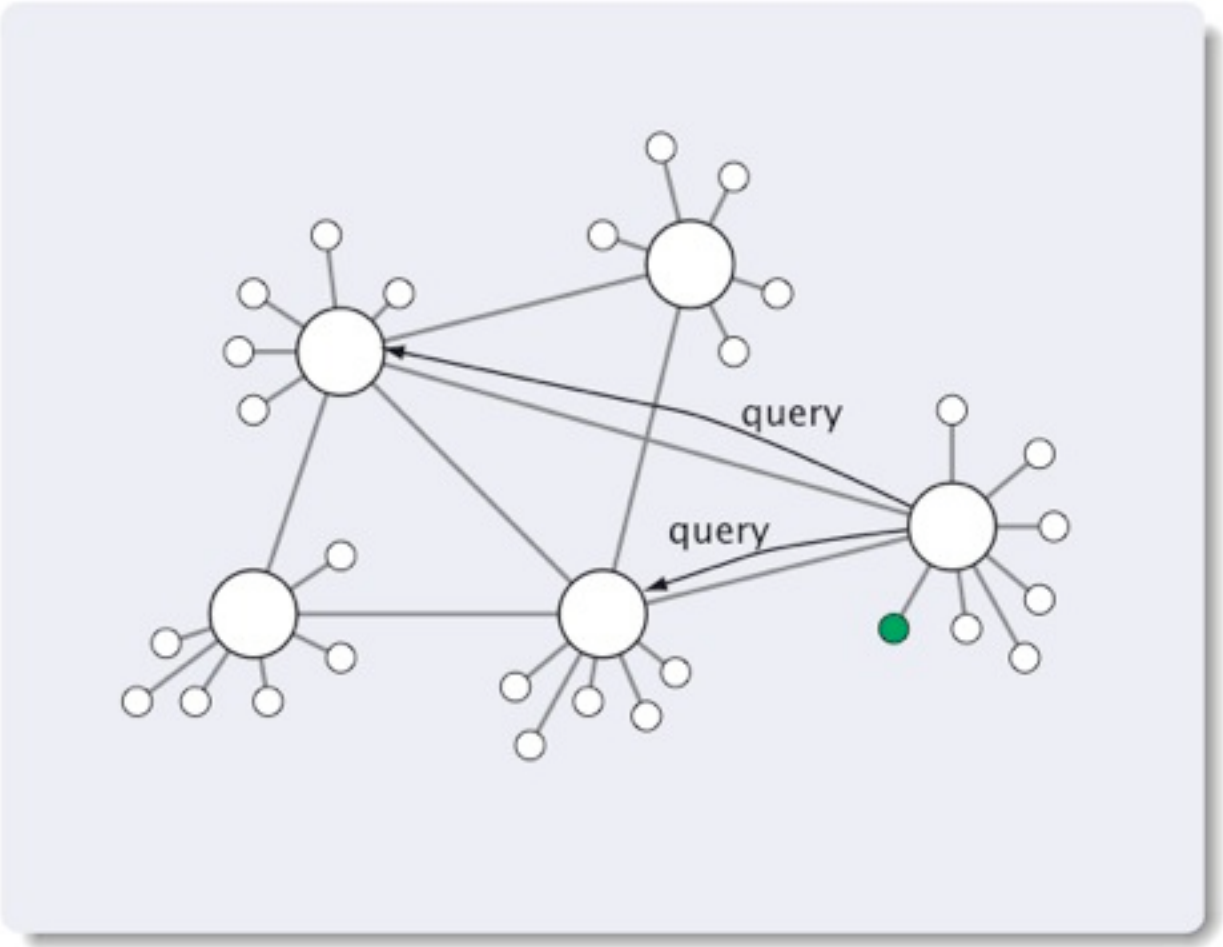
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



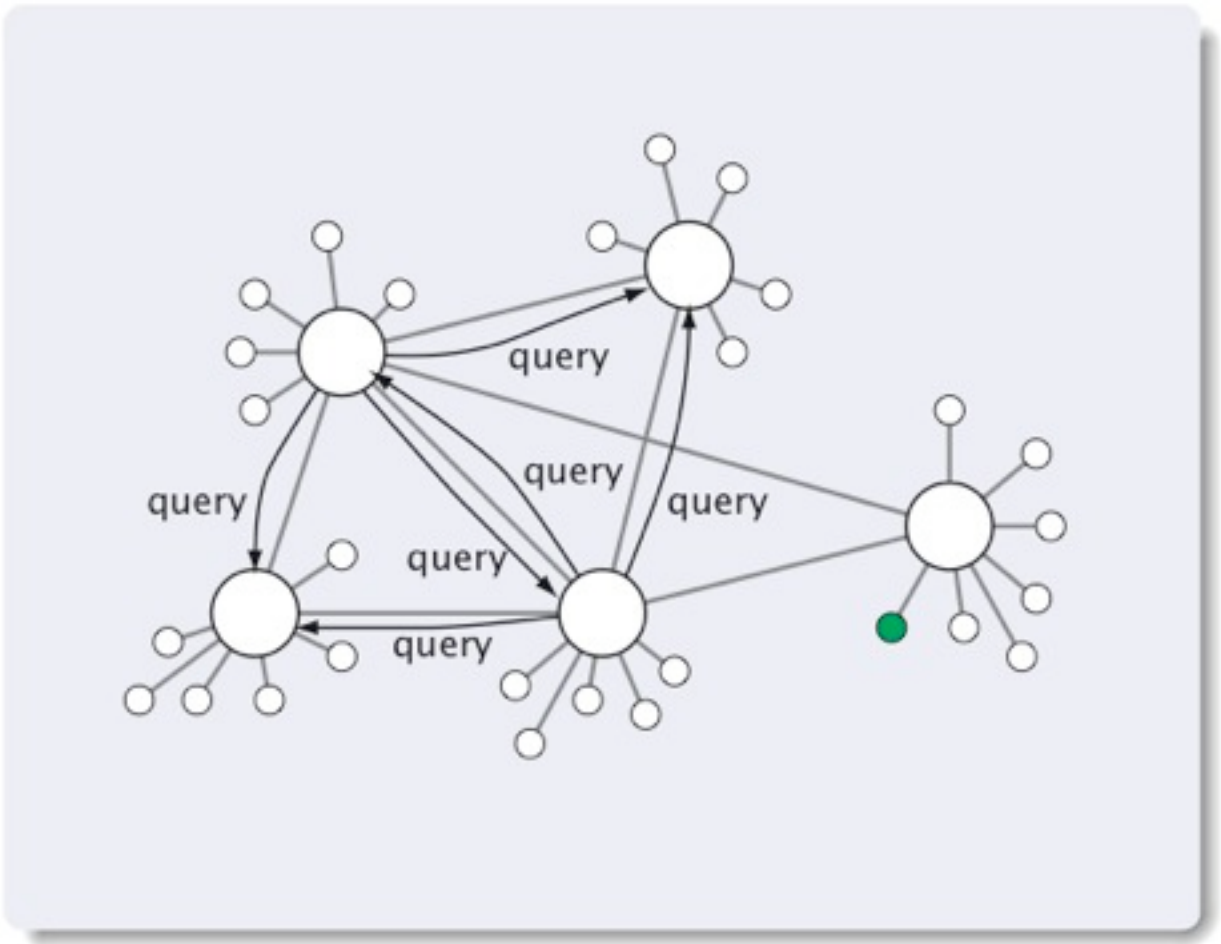
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



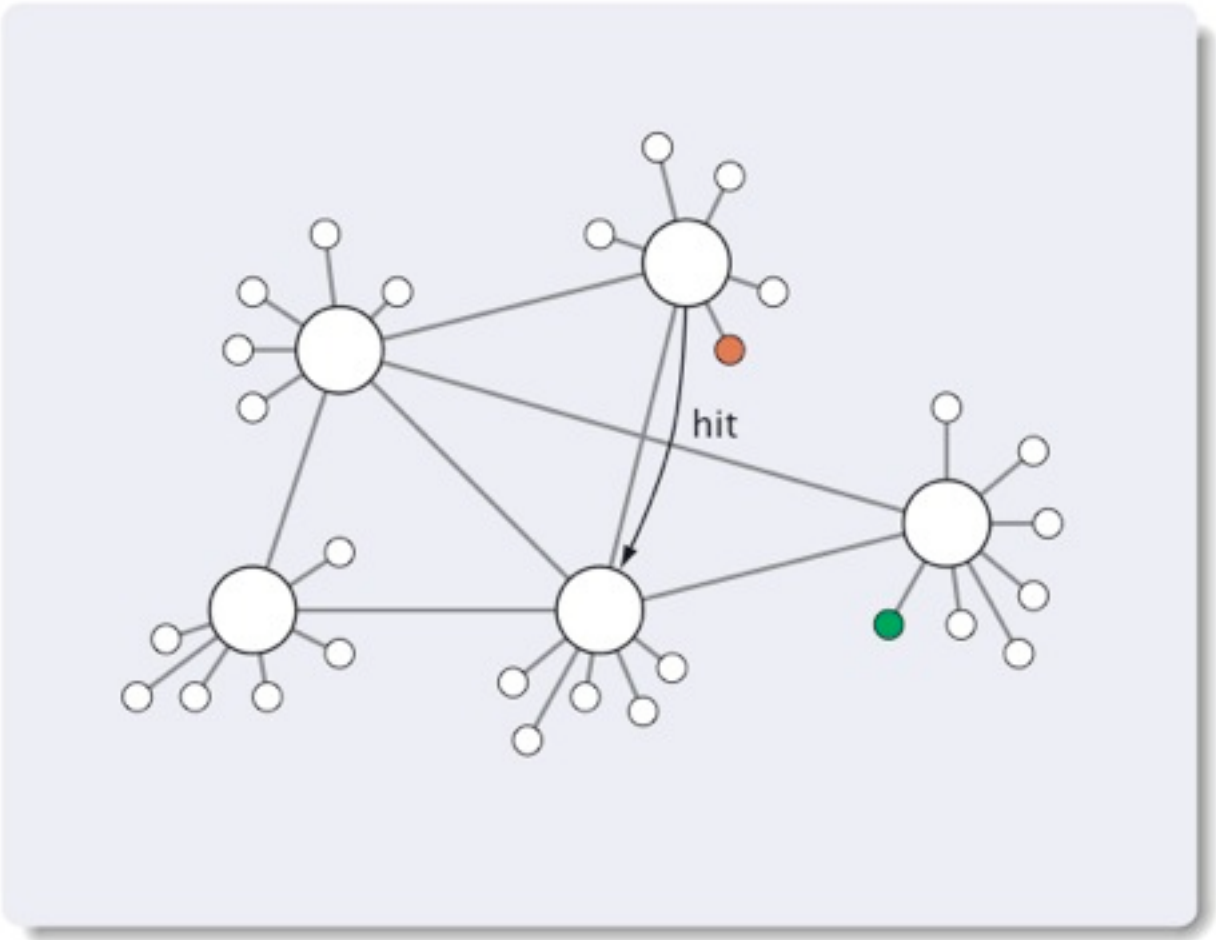
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



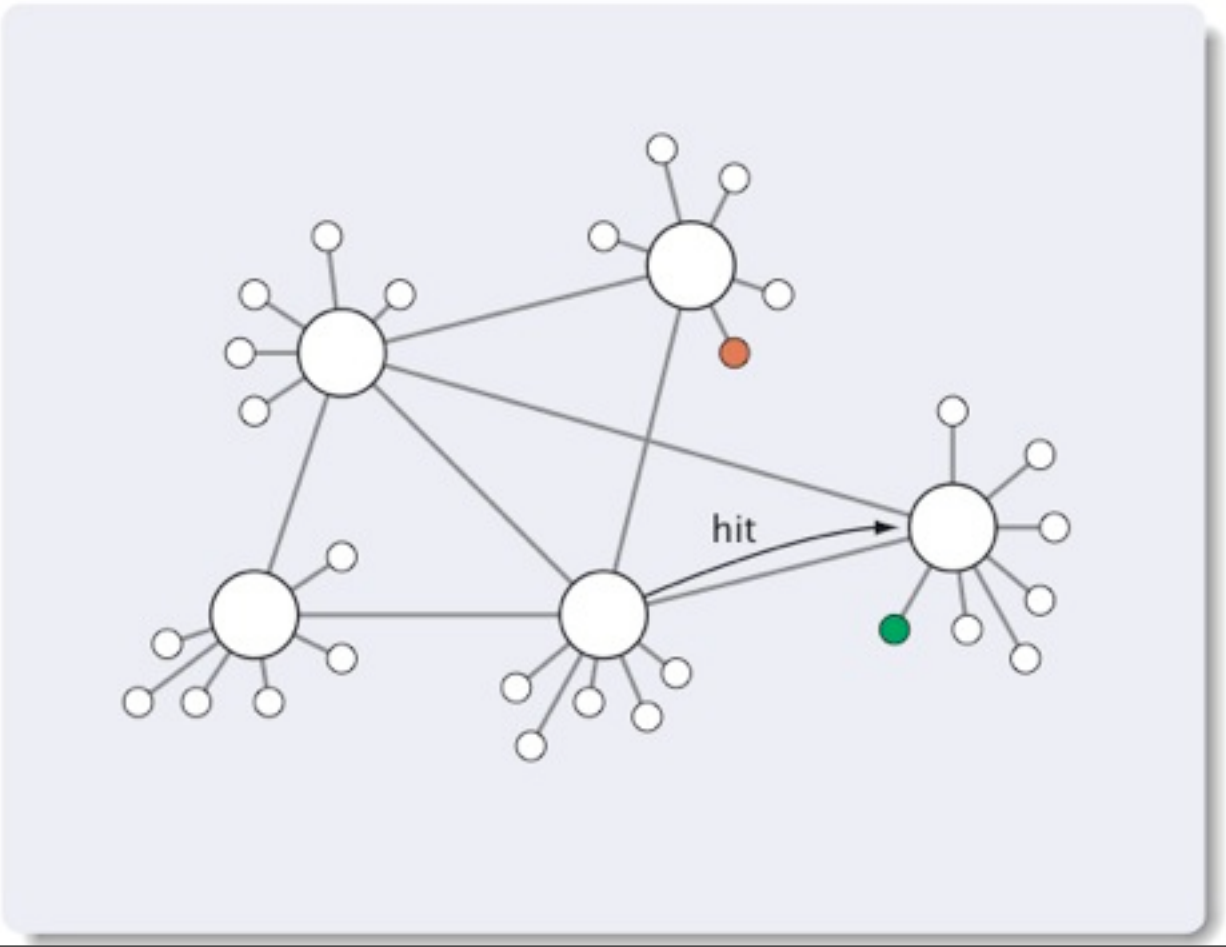
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



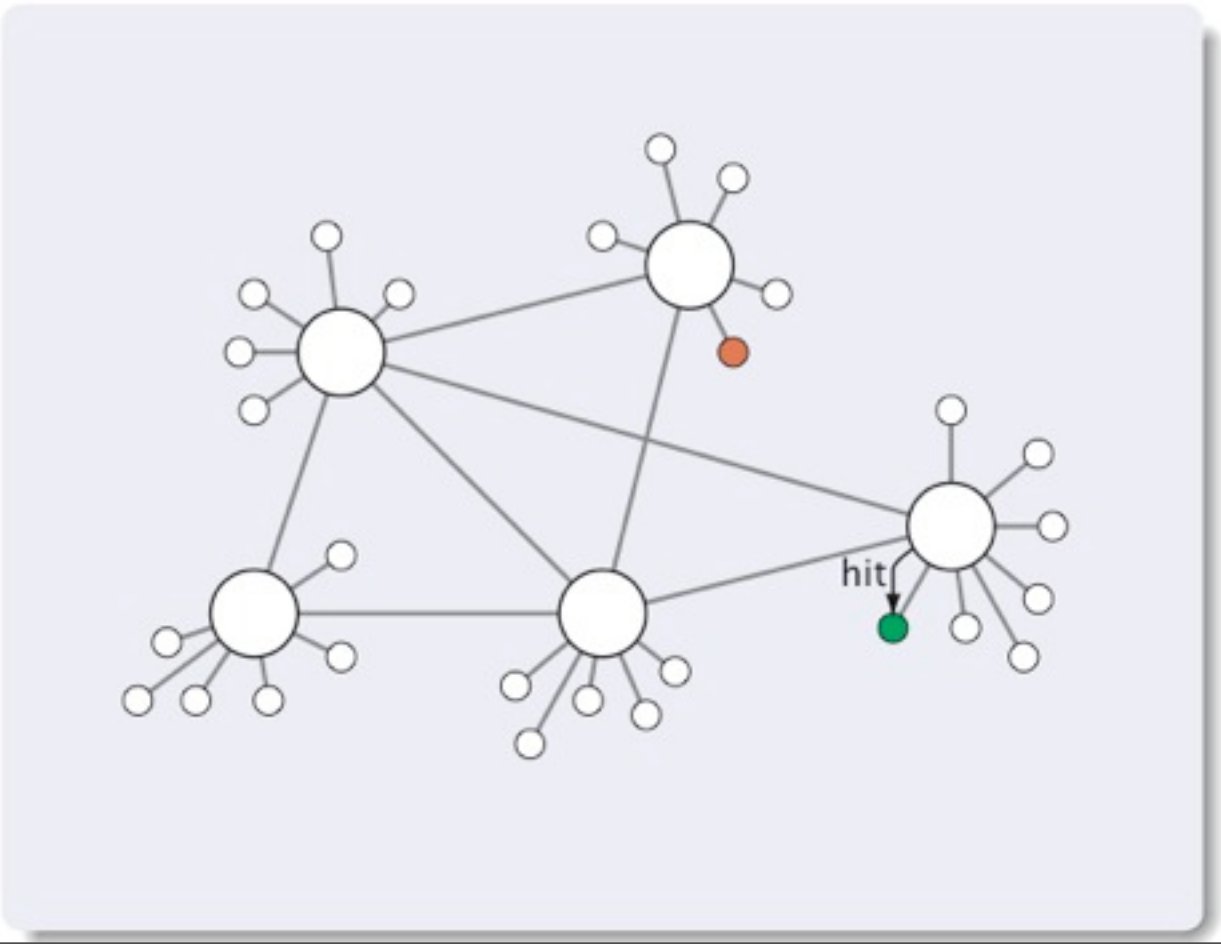
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



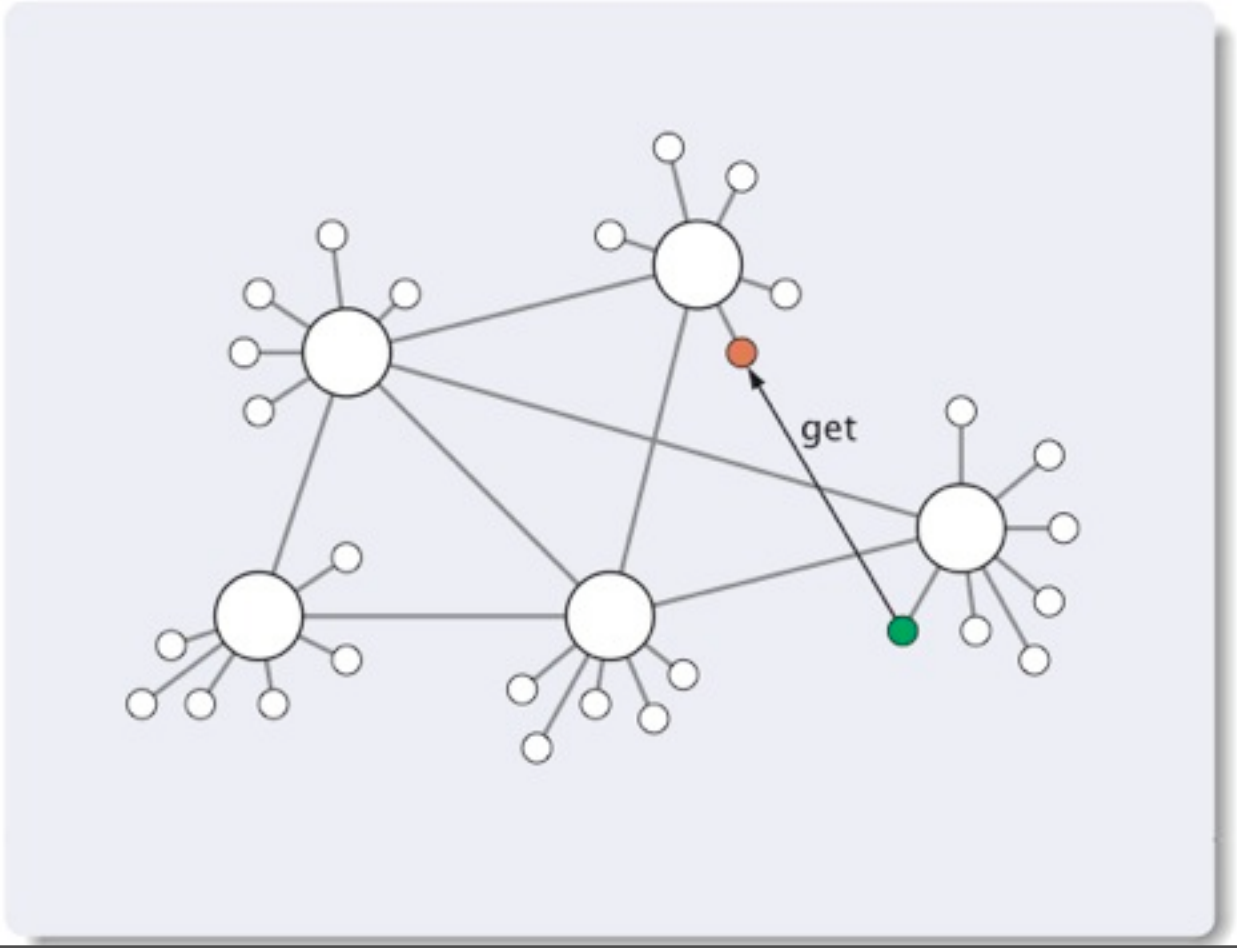
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



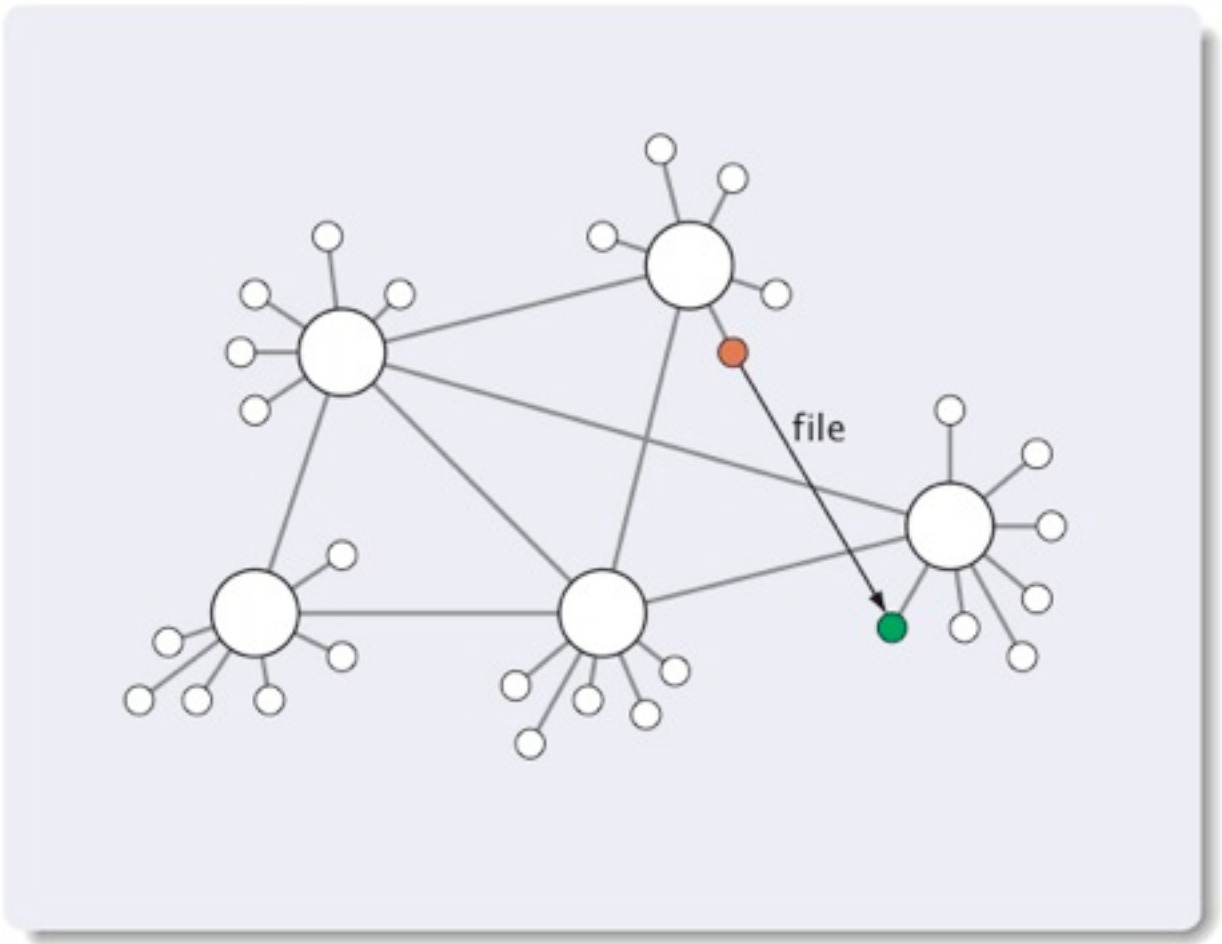
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



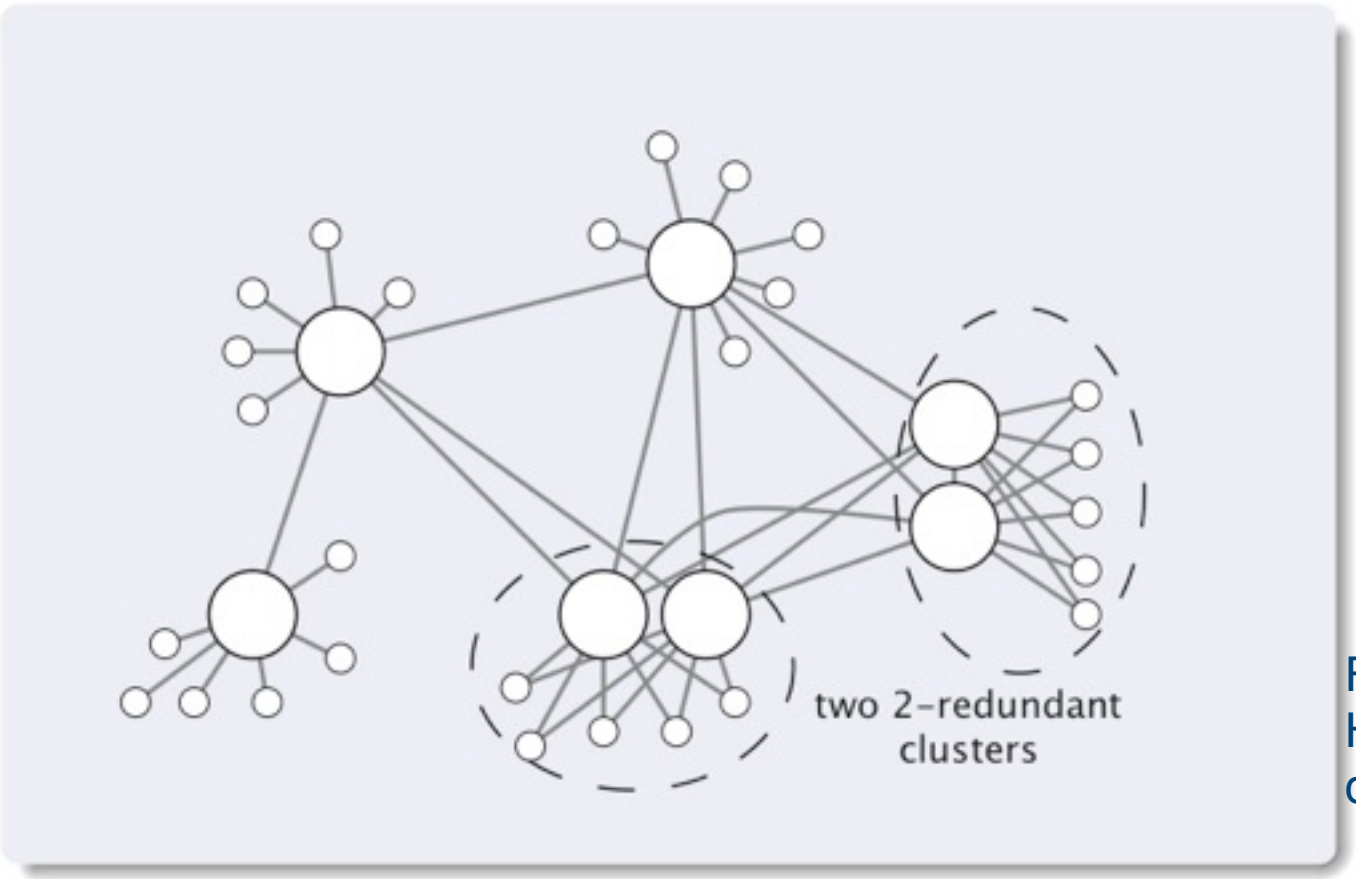
Figures d'Aaron Harwood du NICTA

Gnutella: Super-peers



Figures d'Aaron Harwood du NICTA

Gnutella: Redondance



Figures d'Aaron Harwood du NICTA

Gnutella (*newtella*)

- Autres problèmes
  - pour éviter de trop inonder on peut utiliser un TTL mais avec ça il n'y a plus aucune certitude de trouver un fichier
  - si trop de peers (super-peers) disparaissent on peut se retrouver avec une partition du réseau. Mais le loi de puissance s'applique et fait que le système est plutôt robuste aux disparitions aléatoires
  - les adresses IP, les associations peer/fichier, et me nom des fichiers circulent en clair => il peut y avoir des attaques de déni de service avec de faux messages push et pong. On peut aussi trouver de faux fichiers.

ed2k

- eDonkey 2000 est un réseau P2P avec des super-peers (appelés serveurs) comme pour gnutella mais il n'y a pas de connexions entre super-peers
- Les serveurs indexent les fichiers des peers qui leur sont connectés et permettent la recherche de fichiers
- Les téléchargements sont fait entre peers
- Un peer est typiquement connecté à de nombreux serveurs et en obtient de nouveaux via l'échange de listes de serveurs
- Le client le plus populaire du réseau ed2k est eMule

ed2k

- L'architecture ed2k gère
  - la recherche de fichiers via des mots-clés
  - les téléchargements réalisés en parallèle depuis plusieurs peers
  - les fichiers partagés pendant leur téléchargement
  - la vérification des données téléchargées grâce à des hashes (clés de hachage)

Bittorrent

- Bittorrent est un réseau P2P avec de nombreux serveurs (de type ed2k) gérant les peers intéressés par un même fichier (ou un ensemble de fichiers). Les serveurs sont appelés des «trackers».
- Les trackers sont trouvés soit via le web ou au travers du réseau Kad
- Le protocole favorise les peers qui envoient des données par rapport à ceux qui ne font que télécharger (leechers) en utilisant une technique appelée «choking»
  - Le «choking» est un refus temporaire d'envoyer
  - Le peer pourra peut-être télécharger plus tard

Bittorrent

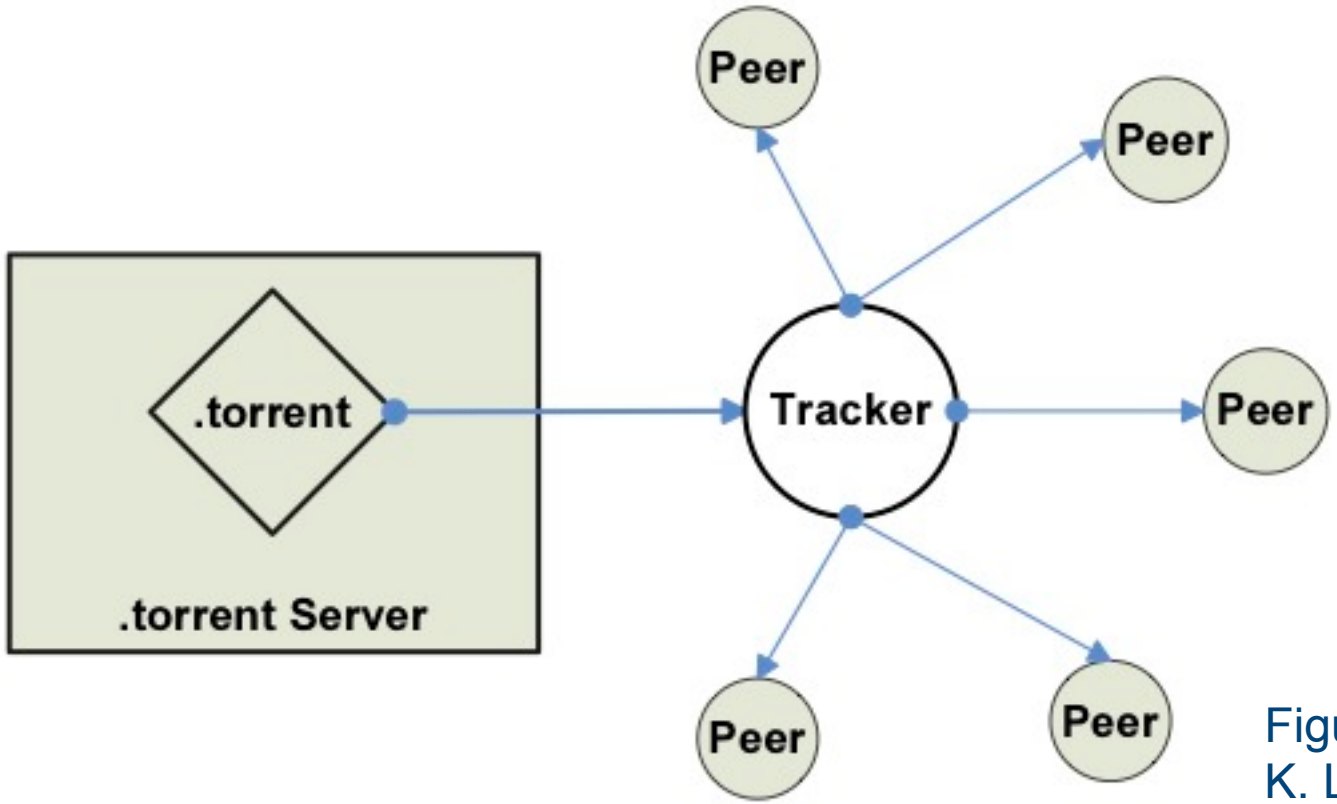


Figure de E. K. Lua et al.

Bittorrent

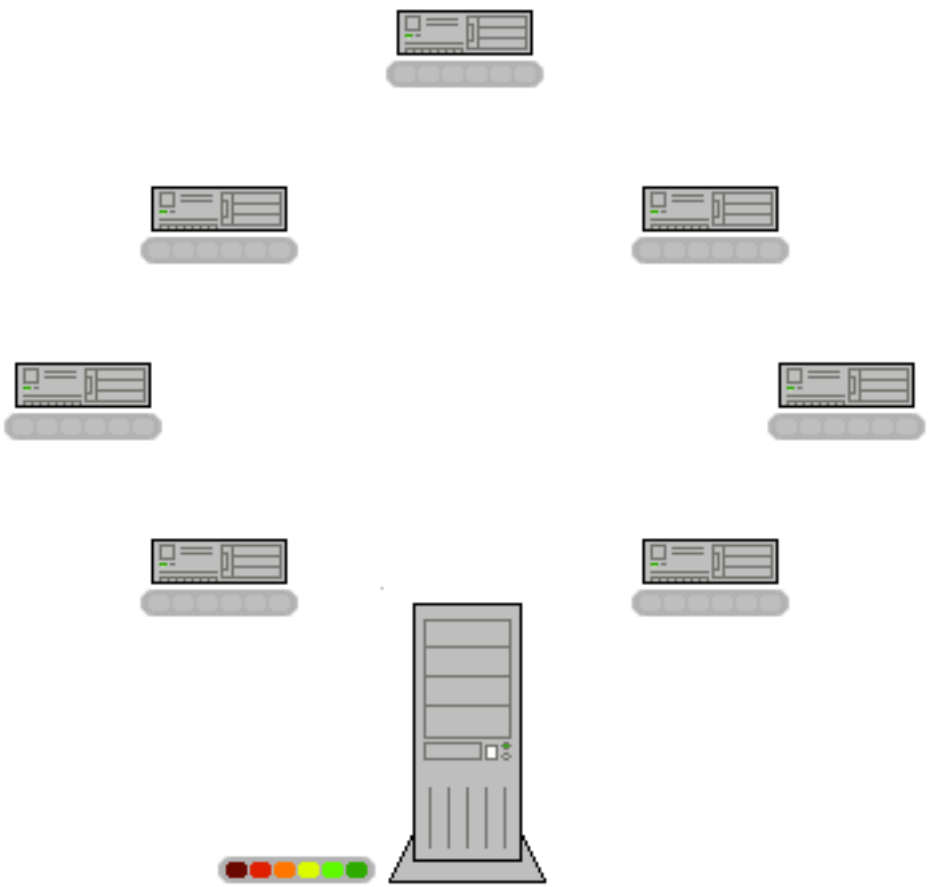


Figure de Wikipedia

Freenet

- Freenet est un réseau d'anonymat
  - il offre un système de fichier distribué virtuel qui stocke des fichiers de façon anonyme (et aussi des freesites et des «chats»)
  - chaque peer amène une part de l'espace de stockage
  - la réplication est utilisée pour s'assurer qu'il n'y a pas de perte de donnée quand des peers disparaissent
  - le cryptage est utilisé pour s'assurer que les données ne peuvent être modifiées que par leur auteur et pour échanger les données de façon anonyme
  - l'architecture est complètement distribuée (comme gnutella) mais Freenet choisit où les données sont stockées

Freenet

- GUIDs
  - A chaque fichier est associé un id global unique qui est utilisé comme une clé pour trouver les données du fichier
  - Les id sont calculés en utilisant des hash de type SHA-1
  - Il y a principalement 2 types de clés
    - CHK (content hash keys) sont calculées à partir du contenu du fichier après cryptage et sont utilisées par freenet pour trouver un fichier
    - SSK (signed subspace keys) sont utilisées par les utilisateurs pour gérer un ensemble de fichiers. Seul le créateur du SSK peut le modifier mais il peut être lu par tout le monde. En fait il s'agit d'un fichier texte qui liste les CHK de certains fichiers. Ils peuvent être utilisés pour créer une hiérarchie de fichiers ou pour découper des gros fichiers.

Freenet

- Freenet gèrent principalement deux opérations :
  - PUT qui envoie les données vers un peer qui stockera les données. Elles seront ensuite déplacées ou dupliquées par Freenet. Les données sont attachées à un GUID (CHK ou SSK).
  - GET qui, pour un GUID donné, est envoyé au travers du réseau vers un peer qui gère les données associées. Les données sont renvoyées par rétro propagation.

Freenet

- Anonymat
  - Les messages sont cryptés pour chaque paire de peers qui les échangent.
  - Le routage n'est que local est basé sur les GUID, ainsi aucun peer ne peut savoir si le précédent (resp. le suivant) est l'origine (ou la destination) du message.

Freenet : routage

- Les tables de routage ne stockent que les GUIDs et le peer connu qui est sensé être plus proche des données
  - Elles ne sont que locales
  - Elles sont construites dynamiquement
  - Au début, un peer n'est connecté qu'à un seul peer et lui demande tout
  - Quand des données sont trouvées tous les noeuds qui les ramènent mettent à jour leurs tables
  - Freenet utilise le pliage de chemin (path folding) de sorte que deux peers non connectés qui échangent des données peuvent se connecter. Ainsi le réseau s'auto organise.

Freenet : routage

- Le routage fonctionne ainsi :
  - A chaque peer est associé un lieu ($L \in \{0 \text{ and } 1\}$)
  - Le GUID de la requête est transformé en une valeur entre 0 et 1
  - La requête est d'abord envoyée vers le peer dont le lieu est le plus proche de la valeur du GUID
  - Puis Freenet utilise le routage «greedy routing» (algorithme glouton) comme suit :

Freenet : routage

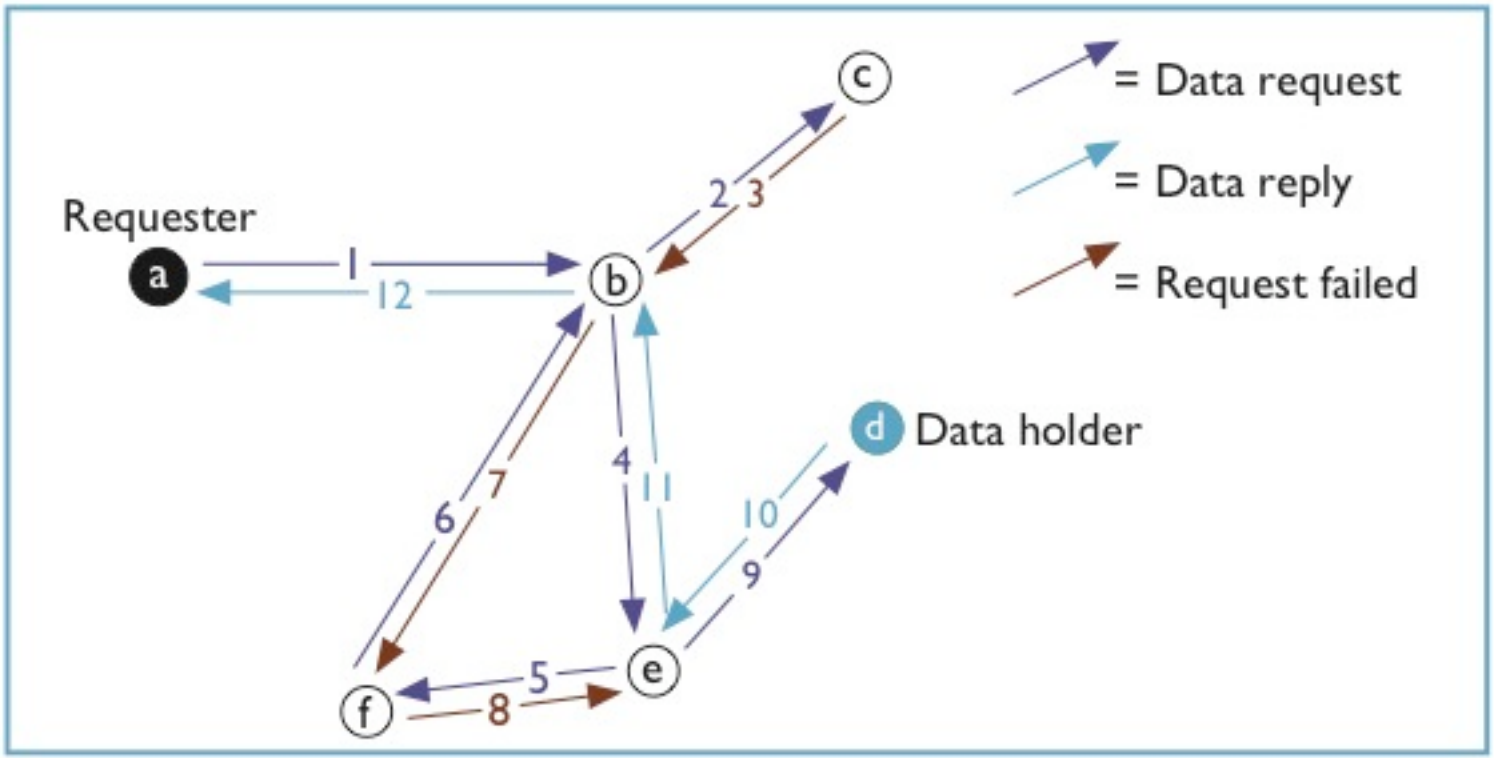


Figure de
Ian Clarke

Freenet : routage

- Aux requêtes de recherche est associé un HTL (hops to live) pour éviter une attente trop longue
- Quand les données sont rétro propagées elles peuvent être répliquées
- D'autre part, le lieu des peers peut changer à cause du pliage de chemin (deux peers connectés auront des lieux similaires)

Freenet : routage

- Pour stocker des données on procède ainsi :
  - La requête est transmise dans le réseau
  - Soit un noeud possède déjà les données et on s'arrête. Note : les données originales sont rétro propagées et sont donc dupliquées sur le chemin.
  - Soit le HTL descend jusqu'à 0. Dans ce cas, les données sont envoyées au peer qui a décrémenté le HTL à 0. Les données peuvent aussi être dupliquées sur le chemin.

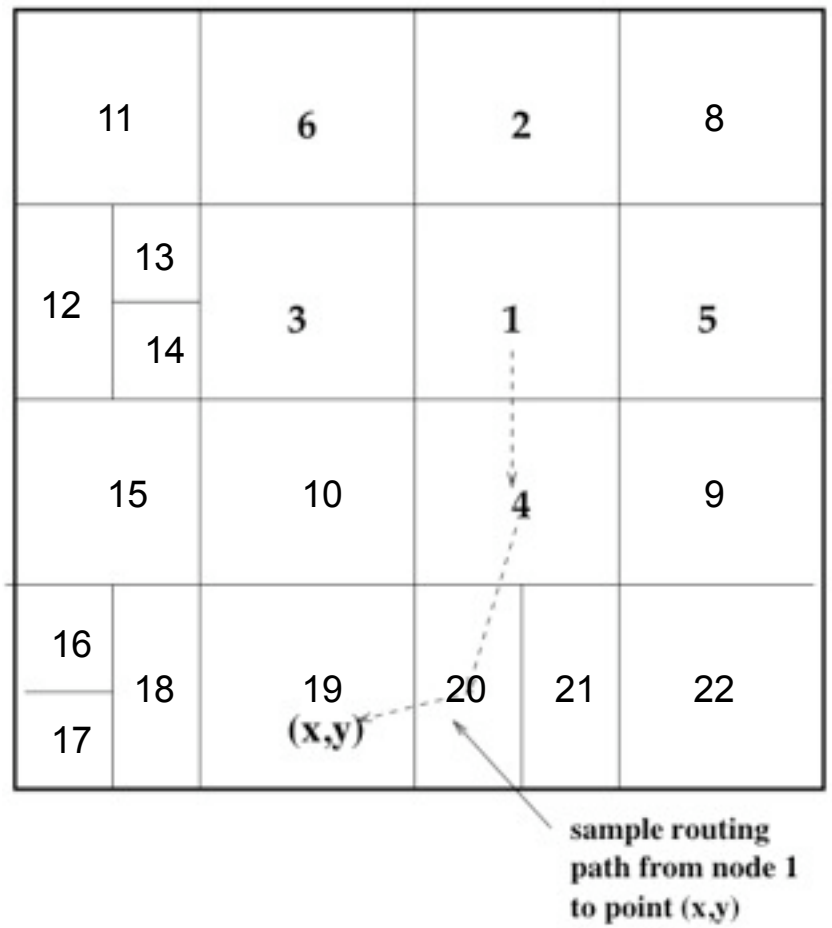
P2P structuré

- 4 articles de recherche sont apparus quasi simultanément : CAN, Chord, Pastry et Tapestry
- Ils proposent les même propriétés
  - la position des données dans le réseau est contrôlée (comme pour Freenet)
  - Mais la structure du réseau est elle aussi contrôlée
- Ils gèrent deux principales opérations qui sont similaires à celles des tables de «hashing» (d'où le nom DHT : distributed hash tables)
  - PUT : introduit des données, liées à une clé, dans le réseau
  - GET : récupère des données en précisant leur clé

CAN

- CAN (Content Addressable Network) est fondé sur une association entre les clés et un espace Cartésien à d dimensions géré comme un hyper tore (d>1)
- A chaque peer est associé une région rectangulaire de l'espace à d dimensions et gère toutes les données dont les clés sont dans la région
- Chaque peer est connecté aux peers qui gèrent les régions voisines
- Le routage est simplement réalisé en choisissant le voisin qui est le plus proche de la clé destination

CAN



voisinage de 1 = {2,3,4,5}

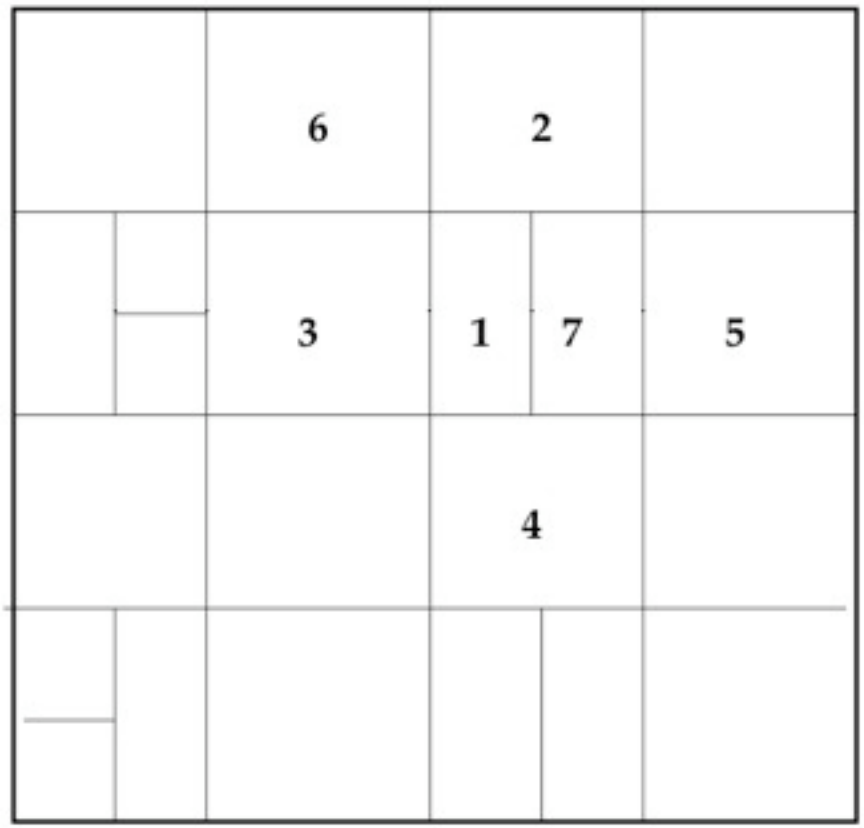
voisinage de 8 = {22,2,5,11}

Figure de S. Ratnasamy et al.

CAN

- Quand un nouveau peer rejoint le réseau
  - il choisit aléatoirement un point de l'espace
  - il se connecte à n'importe quel peer du CAN
  - le CAN route ensuite la requête «join» vers le peer qui gère actuellement la région qui contient se point aléatoire
  - cette région sera ensuite coupée en 2 et une des sous-régions sera donnée au nouveau peer. Toutes les données de cette sous-région seront transmises au nouveau peer.
  - les voisinages sont ensuite mis à jour

CAN



Après l'ajout du peer 7

voisinage de 1 = {2,3,4,7}

voisinage de 7 = {1,2,4,5}

Figure de S. Ratnasamy et al.

CAN

- Quand un «peer» quitte le réseau
  - si la région qu'il gérait peut être fusionnée avec la région d'un de ses voisins pour créer une région valide (rectangulaire), ce voisin récupère la nouvelle région fusionnée et toutes les données sont transférées.
  - si ce n'est pas possible alors le voisin qui a la plus petite région recevra la région et les données. Ensuite le réseau sera recalculé, en tâche de fond, de façon à ré-attribuer des régions valides.
  - Dans les deux cas, les voisins communiquent pour reconstruire leurs voisinages.

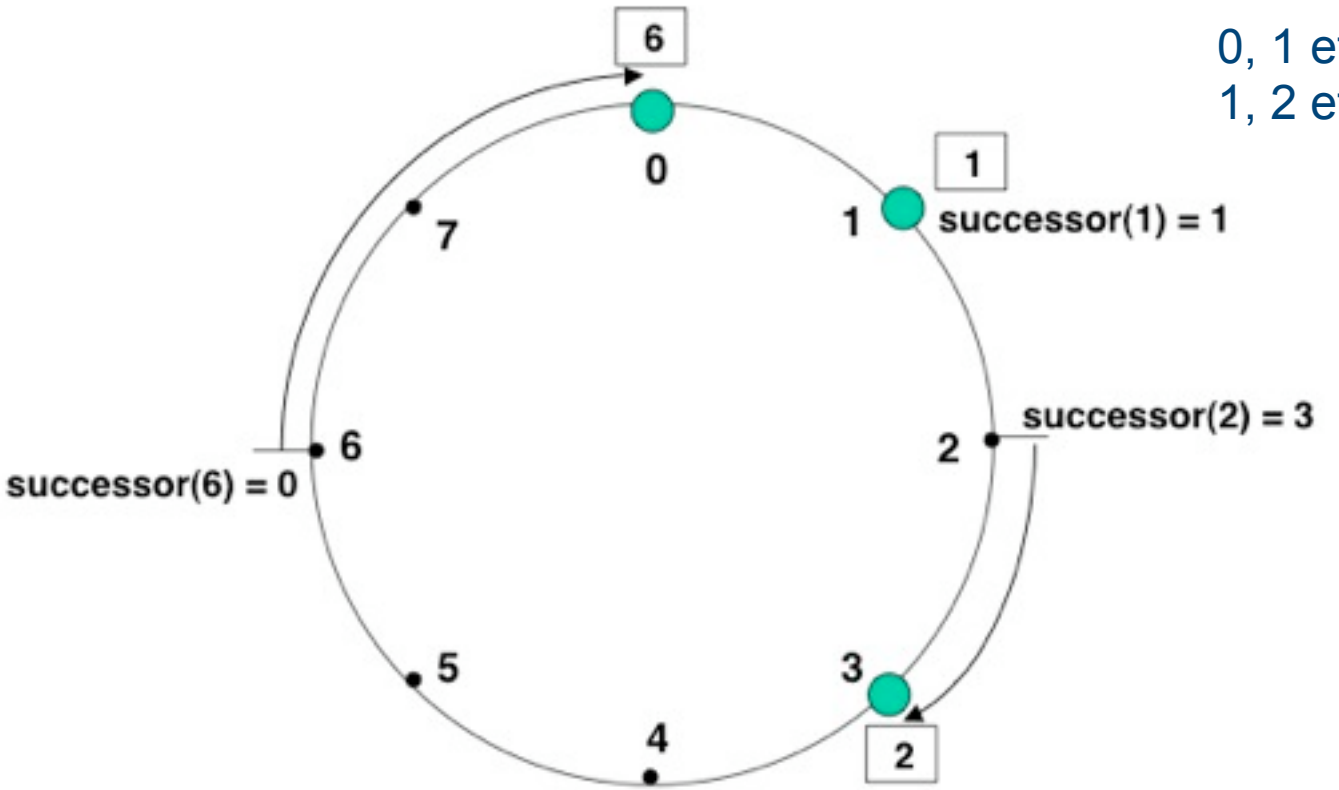
CAN

- Pour plus de fiabilité plusieurs réalités (chacune ayant son espace à d dimensions) peuvent être mises en place de façon à ce que les données soient présente sur différents peers.
- Ainsi si on a k réalités il y aura k copies des données dans le CAN
- Les réalités peuvent aussi être utilisées pour optimiser le routage (en choisissant la plus courte distance) et/ou pour paralléliser les requêtes.

Chord

- Chord utilise un hachage consistant pour affecter les clés aux peers
  - chaque peer reçoit à peu près le même nombre de clés
  - quand un peer s'en va il y a le moins possible de transferts
- Le hachage consistant affecte à chaque clé une valeur sur m-bits en utilisant SHA-1
- A chaque peer est aussi affecté une valeur sur m-bits en hachant son adresse IP avec SHA-1
- Chaque clé est associé au premier peer dont l'id est égal ou suit la valeur de k. Ce peer est appelé le successeur de k. Si on dessine les ids sur un cercle de 0 à 2^m-1, successeur(k) est le premier noeud dans le sens des aiguilles d'une montre après k

Chord



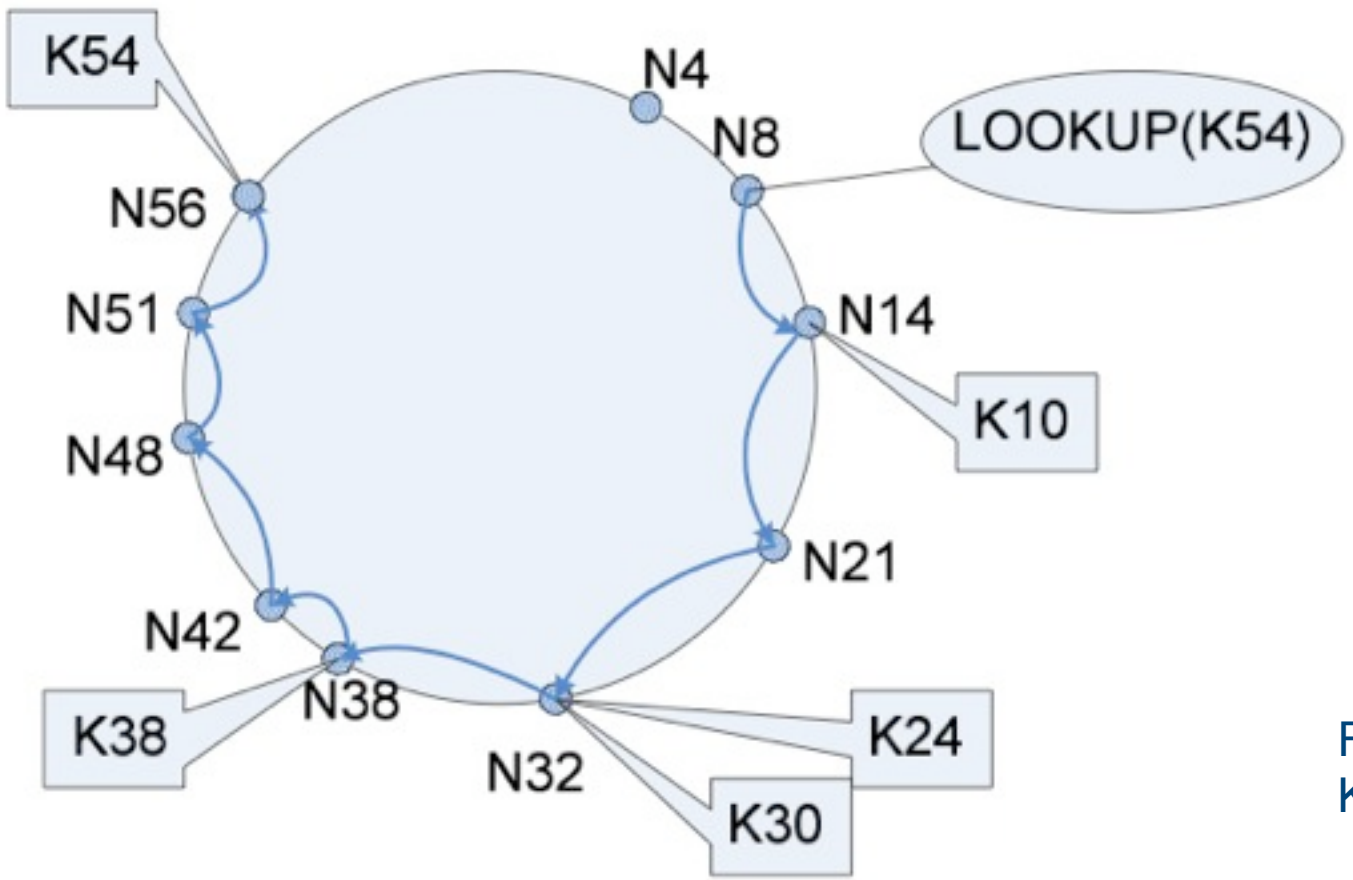
0, 1 et 3 sont des peers
1, 2 et 6 sont des clés

Figure de I. Stoica et al.

Chord

- Quand un nouveau noeud N rejoint le réseau certaines des clés gérées par le successeur de N vont être transmises à N
- Quand un noeud N quitte le réseau toutes ces clés sont transmises à son successeur
- Chaque peer doit savoir contacter son successeur
- Les requêtes de recherche sont transmises dans l'anneau Chord de successeur en successeur jusqu'à que successeur(n) > k (k est gérée par le successeur(n))
- Les données sont ensuite renvoyées en utilisant la rétro propagation

Chord



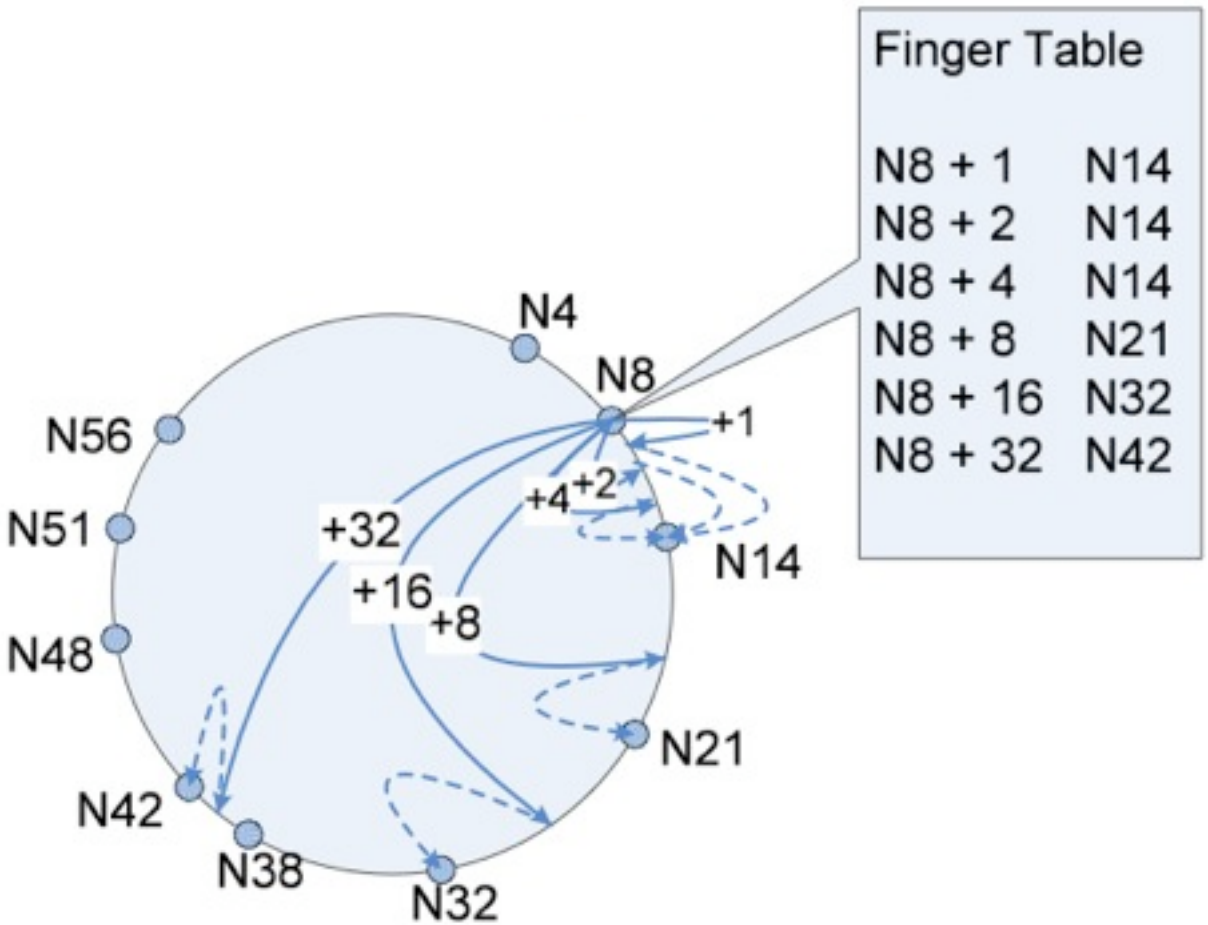
Un anneau Chord pour m = 6

Figure de E. K. Lua et al.

Chord

- Pour optimiser le routage, chaque noeud a une table de routage (appelée finger table) qui contient au max m entrées
  - l'entrée i stocke le successeur($n+2^{i-1}$) où n est l'id du noeud courant
  - en plus de l'id on stocke son IP et son numéro de port
- Les tables doivent être mises à jour à chaque arrivée/départ de noeud
  - C'est géré en tâche de fond
- Quand un peer disparait il est possible qu'un noeud ne connaisse plus son successeur
  - Pour éviter cela, chaque noeud stocke la liste de ces r successeurs

Chord



Un anneau Chord pour m = 6

Figure de E. K. Lua et al.

Pastry

- Pastry et Tapestry sont similaires
- Ils utilisent un routage de préfixe de type Plaxton (un graphe de Plaxton est une structure de donnée distribuée optimisée pour gérer un réseau de recouvrement, structuré à partir d'une racine, permettant de localiser des objets nommés - Pastry et Tapestry utilisent en fait plusieurs racines pour améliorer la fiabilité et le passage à l'échelle)

Pastry

- Pastry utilise une métrique de routage externe pour optimiser le système (ce peut être la latence - en utilisant ping - le nombre de sauts - en utilisant traceroute - la bande passante...)
- C'est un DHT dans lequel les clés et les id de noeuds sont des entiers non signés sur 128 bits organisés en anneau comme pour Chord
- Les id des noeuds sont choisis aléatoirement et uniformément de façon à ce que des noeuds proches en id soient géographiquement répartis
- Les peers maintiennent 3 structures de données :
  - un ensemble de noeuds feuilles
  - un voisinage
  - une table de routage

Pastry

- Les feuilles : il s'agit de L/2 noeuds avant et après N sur l'anneau
- Le voisinage : il s'agit des M plus proches peers selon la métrique. Il n'est pas utilisé directement mais permet de réparer la table de routage
- La table de routage contient une entrée pour chaque bloc d'adresse
  - Les blocs d'adresse sont formés en découpant les entiers sur 128 bits en blocs de b bits (en général 4 bits => 32 chiffres hexadécimaux)
  - Cela partage les adresses en niveaux. Le niveau 0 contient 0 chiffres commun avec l'adresse du noeud, le niveau 1, 1 chiffre commun...
  - La table contient l'adresse du plus proche peer connu pour chaque chiffre à chaque niveau d'adresse (à l'exception du chiffre du noeud). Elle contient normalement 15 contacts par niveau.

Figure de E. K. Lua et al.

Pastry

Routing Table of a Pastry peer with NodeID 37A0x, b = 4, digits are in hexadecimal, x is an arbitrary suffix

0x	1x	2x	**3x**	4x	...	Dx	Ex	Fx
30x	31x	32x	...	**37x**	38x	...	3Ex	3Fx
370x	371x	372x	...	**37Ax**	37Bx	...	37Ex	37Fx
37A0x	37A1x	37A2x	...	37ABx	37ACx	37ADx	37AEx	37AFx

Pastry

Figure courtesy of E. K. Lua et al.

Example: Routing State of a Pastry peer with NodeID 37A0F1, b = 4, L=16, M=32

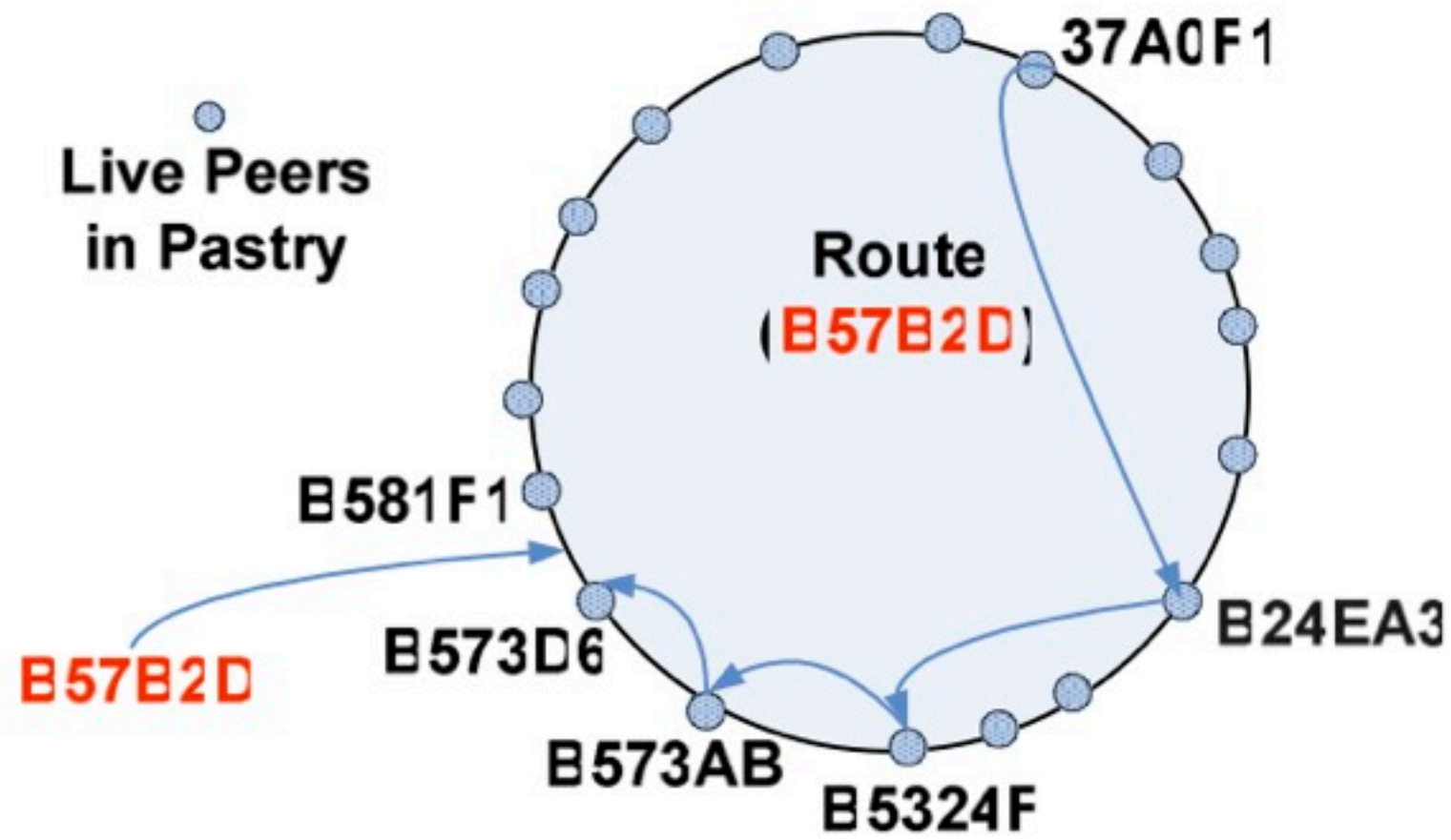
NodeID 37A0F1			
Leaf Set (Smaller)			
37A001	37A011	37A022	37A033
37A044	37A055	37A066	37A077
Leaf Set (Larger)			
37A0F2	37A0F4	37A0F6	37A0F8
37A0FA	37A0FB	37A0FC	37A0FE
Neighborhood Set			
1A223B	1B3467	245AD0	2670AB
3612AB	37890A	390AF0	3912CD
46710A	477810	4881AB	490CDE
279DE0	290A0B	510A0C	5213EA
11345B	122167	16228A	19902D
221145	267221	28989C	199ABC

Pastry

- Une requête peut être routée sur l'anneau vers le peer qui est le plus proche de l'adresse de destination (il peut ne pas y avoir de peer à l'adresse exacte)
- Le routage fonctionne ainsi :
  - On cherche d'abord parmi les feuilles pour trouver l'adresse
  - Si on ne trouve pas, on utilise la table de routage pour trouver un noeud plus proche de l'adresse de destination (ie il a au moins x chiffres en commun avec la destination tel que x > y : le nombre de chiffre en commun pour le noeud courant)
  - Si on ne trouve pas (il n'y a pas de noeud plus proche ou le noeud le plus proche n'existe plus), on envoie la requête à quelqu'un de l'ensemble des feuilles qui est plus proche

Figure courtesy of E. K. Lua et al.

Pastry



Pastry

- Quand des peers disparaissent, les peers qui les utilisent pour le routage ou pour un des autres ensembles demandent aux autres peers de leur voisinage ou de l'ensemble de feuilles des peers de remplacement

Kademlia

- Kademlia est une DHT qui utilise des clés et des id de noeuds sur 160 bits (clés et ids sont dans le même espace comme pour Chord et Pastry)
- Néanmoins elle utilise XOR comme métrique de distance pour le routage. XOR a les spécifités suivantes qui en font une distance géométrique (non Euclidienne) :
  - $XOR(a,a) = 0$ et $XOR(a,b)>0$ si $a \neq b$
  - $XOR(a,b) = XOR(b,a)$
  - $XOR(a,b)+XOR(b,c) \geq XOR(a,c)$
- Il est à noter que comme $XOR(a,b) = XOR(b,a)$ le routage de Kademlia est symétrique et donc que les noeuds reçoivent des informations importantes pour le routage quand ils reçoivent des messages ce qui n'est pas le cas pour Chord et pour Pastry

Kademlia

- Le routage de Kademlia utilise le concept des k-buckets (seaux)
- les K-buckets sont des listes d'au max k noeud (k vaut en général 20)
- Chaque noeud a jusqu'à 160 listes (une pour chaque bit de l'espace d'adressage)
- Le n-ème k-bucket contient des noeuds qui ont leurs bits de 0 à n-1 égaux à ceux du noeuds courant et le n-ème bit différent
- De nombreux k-buckets sont vides lorsque n augmente vers 160 car les partitions de l'espace des clés deviennent de plus en plus petites. Pour n=0 l'espace des clés est découpé en 2...
- L'arbre suivant montre 4 k-buckets pour un noeud dont l'adresse commence par 0011

Kademlia

Figure de P. Maymounkov et al.

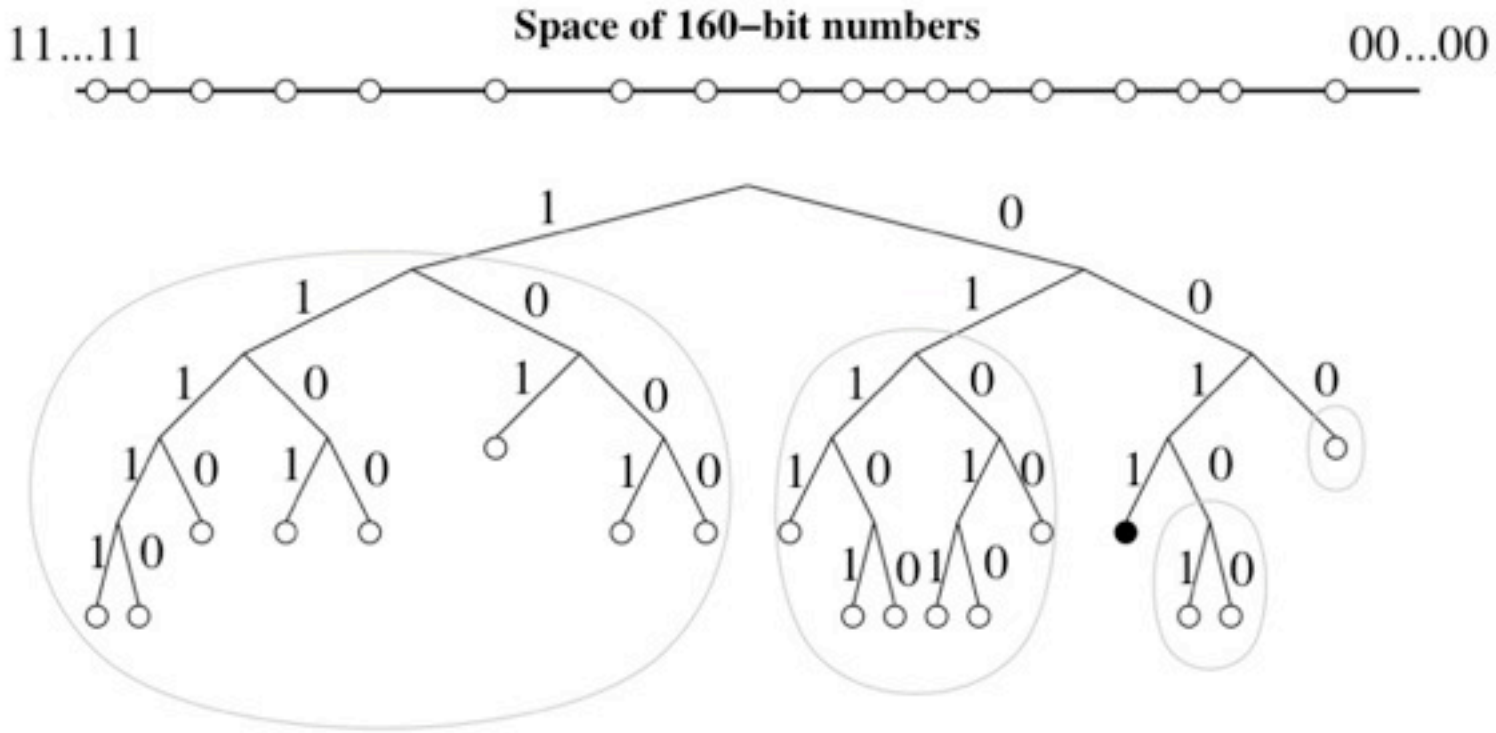
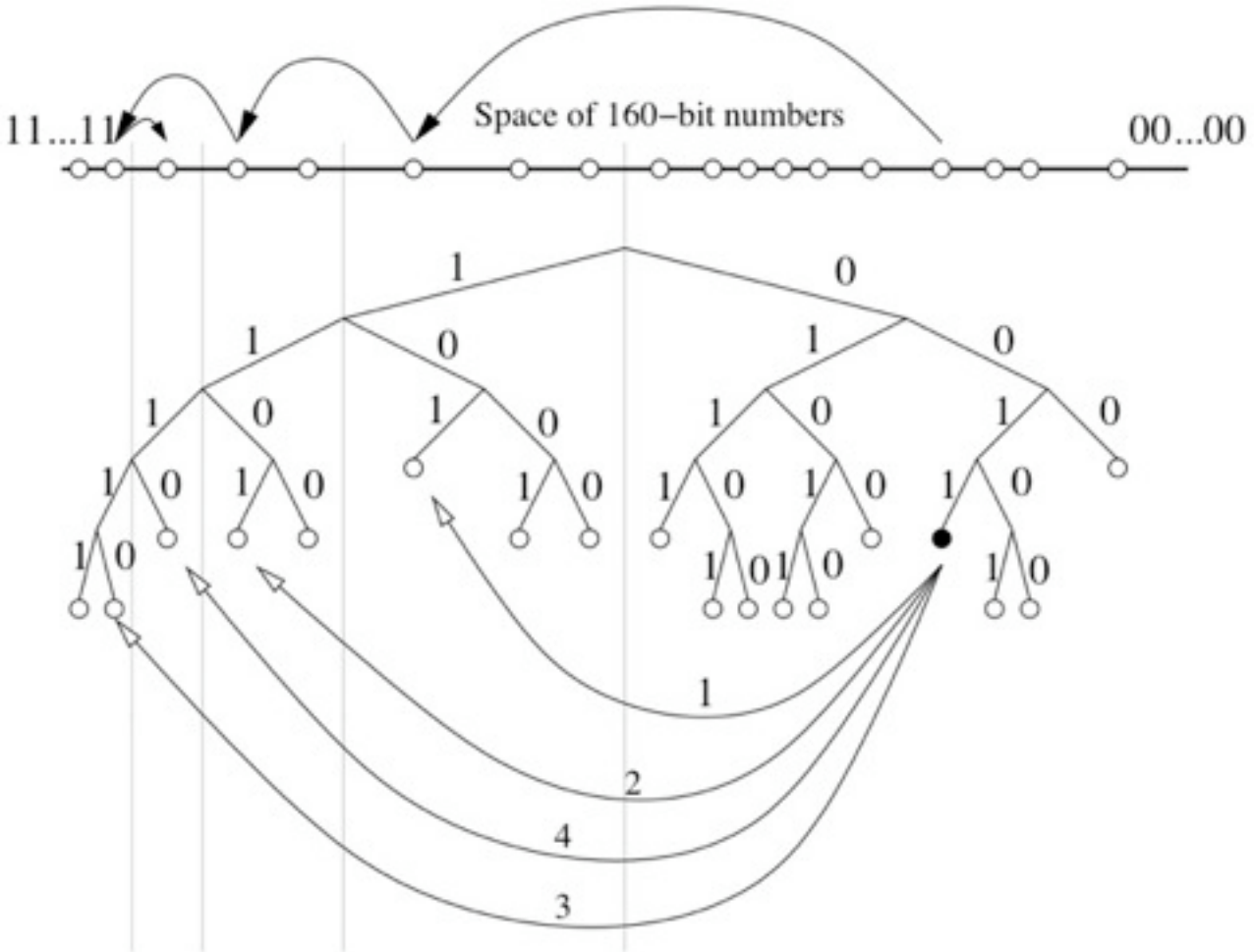


Fig. 1: Kademlia binary tree. The black dot shows the location of node 0011 ··· in the tree. Gray ovals show subtrees in which node 0011 ··· must have a contact.

Kademlia

- Pour trouver la route vers une clé, un noeud envoie des messages aux α (typiquement 3) noeuds qui sont plus proches de la destination. α permet de paralléliser la requête de façon à ce qu'elle ne soit pas bloquée par des noeuds disparus
- Les noeuds sont sélectionnés dans le k-bucket non vide le plus proche de la clé
- S'il y a moins de α noeuds dans ce k-bucket alors le système prends les autres noeuds dans un k-bucket plus éloigné
- Les noeuds contactés vont répondre à la requête en donnant les k noeuds les plus proches de la clé qu'ils connaissent
- Le noeud de départ va utiliser α de ces k noeuds pour la prochaine requête jusqu'à ce qu'il trouve les k noeuds les plus proches de la clé qui la gèrent

Kademlia



Kademlia

- A chaque fois qu'un noeud reçoit un message il met à jour ces k-buckets ainsi :
  - Si le k-bucket qui doit contenir le nouveau node a moins de k noeuds alors il y est ajouté
  - Si le k-bucket est plein alors le plus vieux noeud du k-bucket (ils sont triés par date de dernier contact) est contacté (avec un ping). S'il répond alors le nouveau noeud va dans une sous liste qui est utilisée quand des noeuds du k-bucket disparaissent.

Kademlia

- Quand un noeud rejoint le réseau il choisit sont id aléatoirement. Il doit connaitre un noeud quelconque du réseau (fourni par l'utilisateur). Ce noeud va dans un k-bucket (tous les autres sont vides).
- Le nouveau noeud va ensuite faire une recherche de son propre ID de façon à s'ajouter aux k plus proches voisins (il remplit aussi ces k-buckets)
- Par la suite il va continuer à remplir ces k-buckets en tâche de fond grâce à des recherches aléatoires (il commence par les plus gros k-buckets)

Kademlia

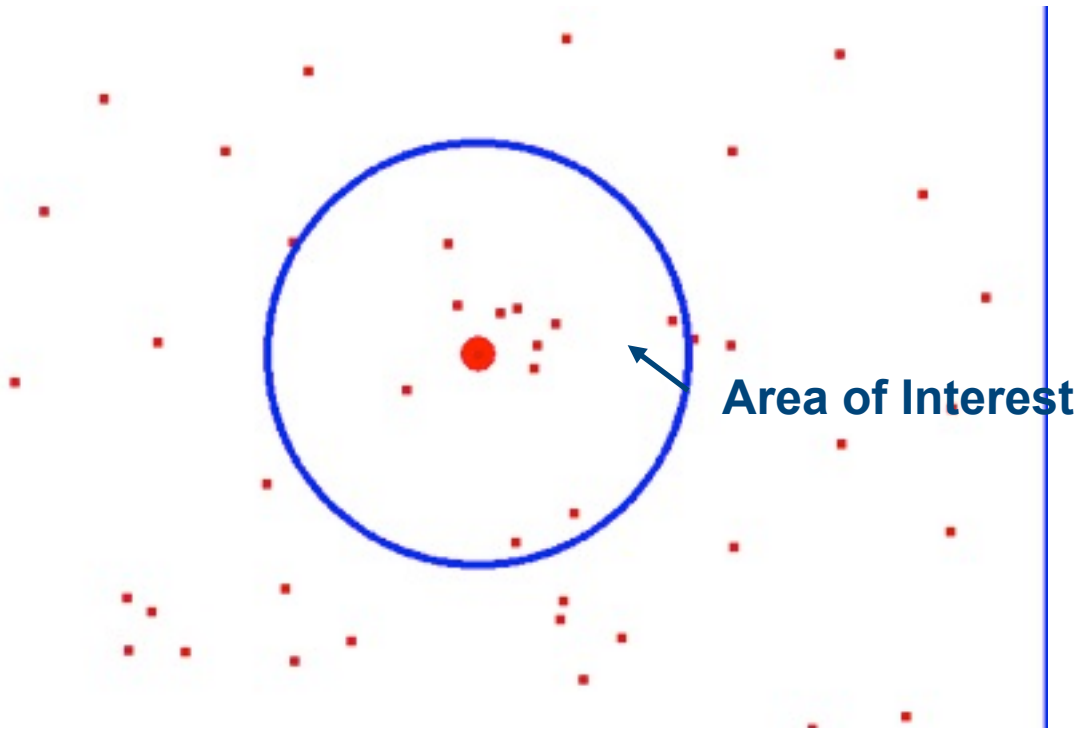
- Kademlia est le protocole de type DHT le plus utilisé pour le partage de fichier
- Il est utilisé par eMule et les clients Bittorrent
- Les données des peers (@IP et port) sont typiquement stockées en utilisant des hash de 160-bits du contenu des fichiers
- Comme chacun des k plus proches voisins peuvent stocker des données de peers différents, on peut trouver au plus k peers pour démarrer le téléchargement.
- Les recherches par mot clé sont réalisées en stockant un lien vers le fichier (qui contient au moins les noms du fichier et le hash du contenu) avec pour clé des hash 160-bits de chacun des mots des noms du fichier

Partie 2 : application aux MMVEs

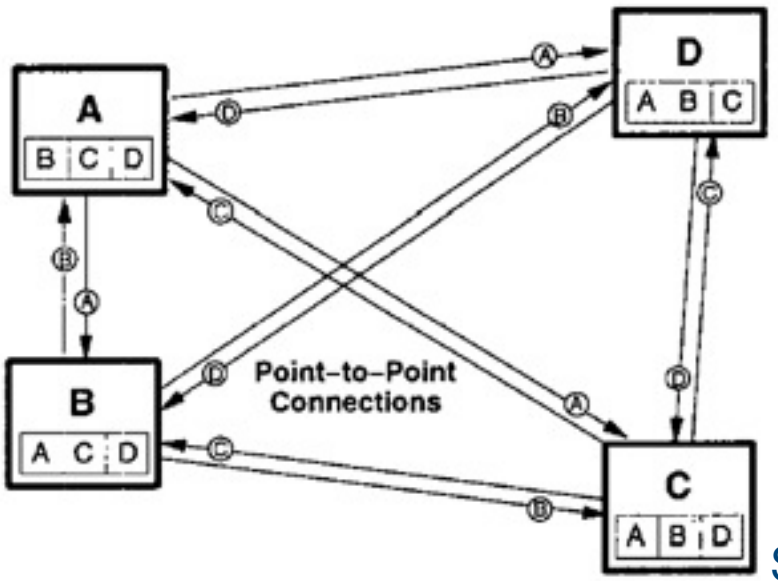
- Le problème du passage à l'echelle
- Approches hybrides C/S et P2P
- Coordinateurs de zones
- Point à point amélioré
- Echange de voisinages
- Notification mutuelle
- Notification mutuelle et multicast de recouvrement

Le problème du passage à l'échelle

- De nombreux (> millions) avatars sont répartis dans un monde virtuel
- Ils veulent communiquer avec les voisins qui sont dans leur zone d'intérêt (AOI)
- Comment s'assurer qu'ils reçoivent les messages qui les intéressent



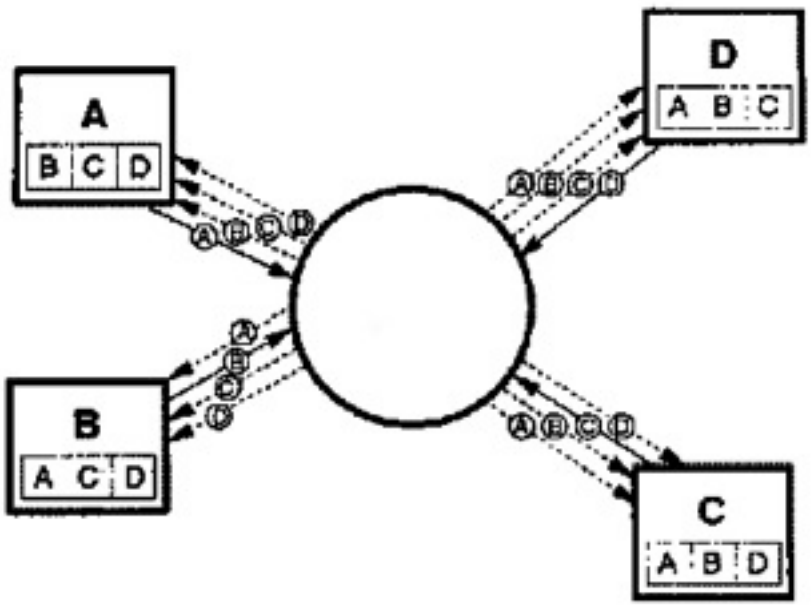
Une solution simple (point à point)



Source: [Funkhouser95]

N * (N-1) connexions ≈ $O(N^2)$ → Non passable à l'échelle !

Une meilleure solution (client-serveur)

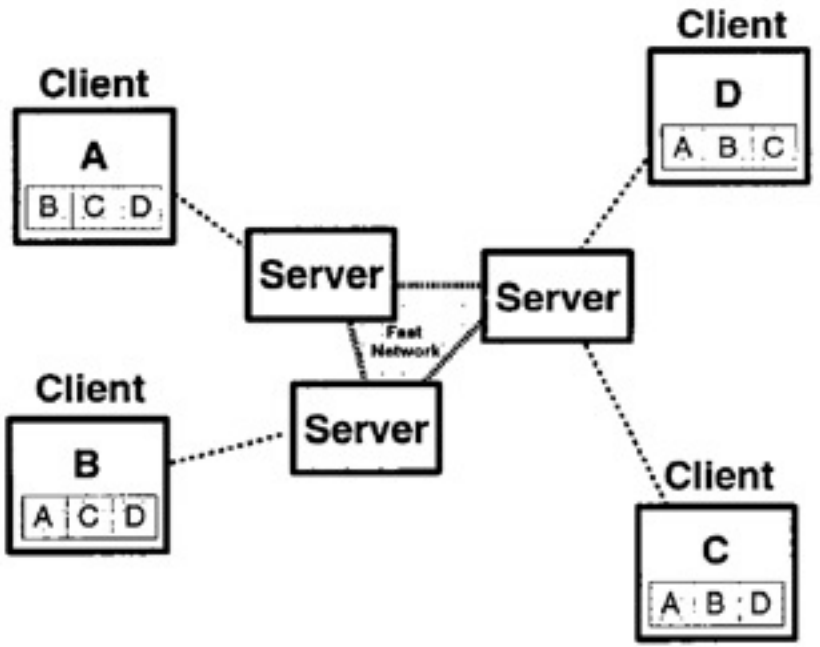


Source: [Funkhouser95]

Peut utiliser le filtrage de message sur le serveur pour réduire le traffic

N connexions = O(N) → mais le serveur est un goulet d'etranglement

Solution actuelle (cluster de serveurs)



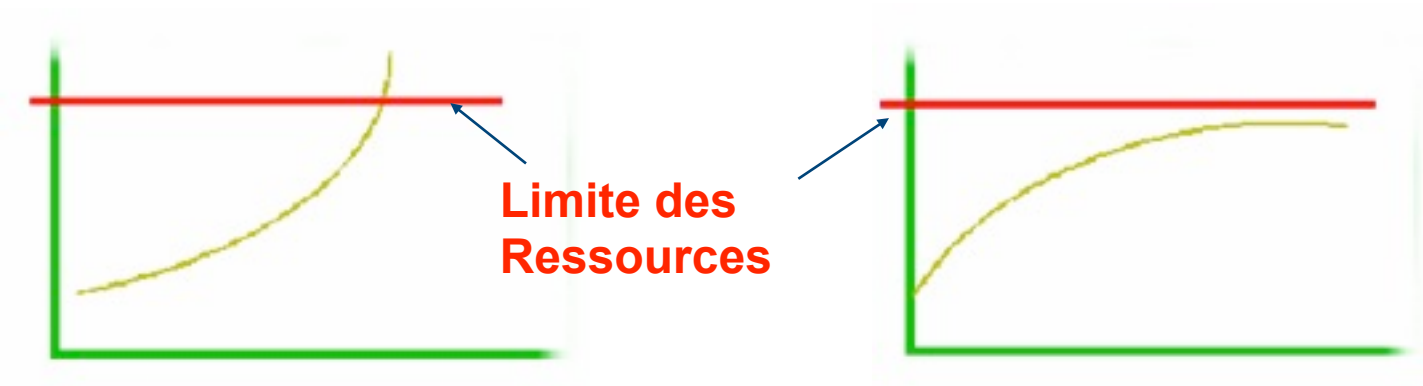
Source: [Funkhouser95]

Toujours limité par les serveurs. Couteux à déployer et maintenir.

Analyse du passage à l'echelle

- Contraintes
  - Ressources de calcul (CPU)
  - Ressources réseau (Bande passante)

Système non «scalable» vs. Système «scalable»



x : nombre d'entité

y : consommation des ressources à l'endroit qui limite le système

Solution ?

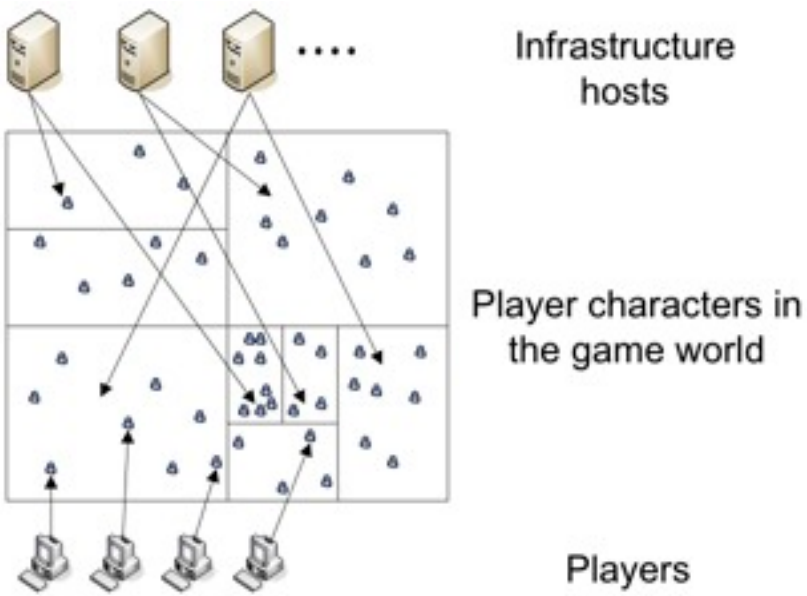
- Stratégies
  - Augmenter les ressources => Plus de serveurs
  - Diminuer leur consommation => Filtrage
- Architectures — Echelle
  - Point à point — dizaines — 10^1
  - Client-serveur — centaines — 10^2
  - Cluster de serveurs — milliers — 10^3
  - Peer-to-Peer ? — millions — 10^6
  - …

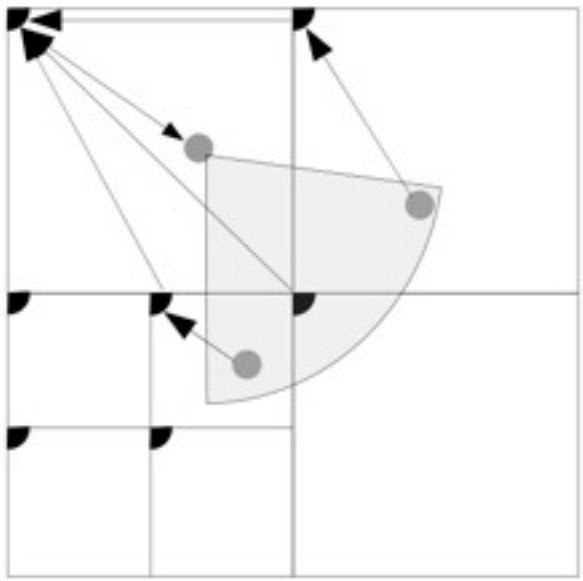
Approches hybrides C/S et P2P

- Mixer le C/S avec le P2P
  - P2P côté serveur
    - Les serveurs communiquent à travers un réseau P2P
  - ou côté client
    - Les clients forment un réseau P2P
  - ou des deux côtés

P2P côté serveur

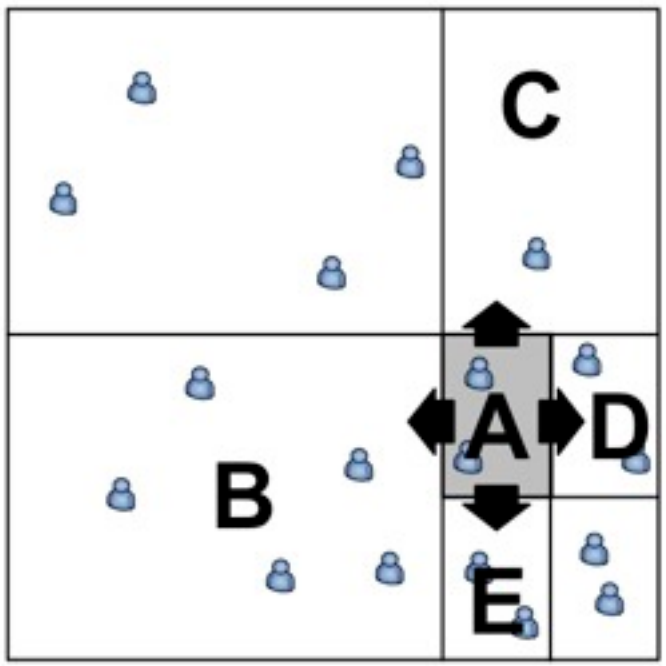
- Exemple : [Rieche et al. 2006] utilise CAN
  - Permet l'ajout de serveurs de façon dynamique quand le cluster de serveur sature





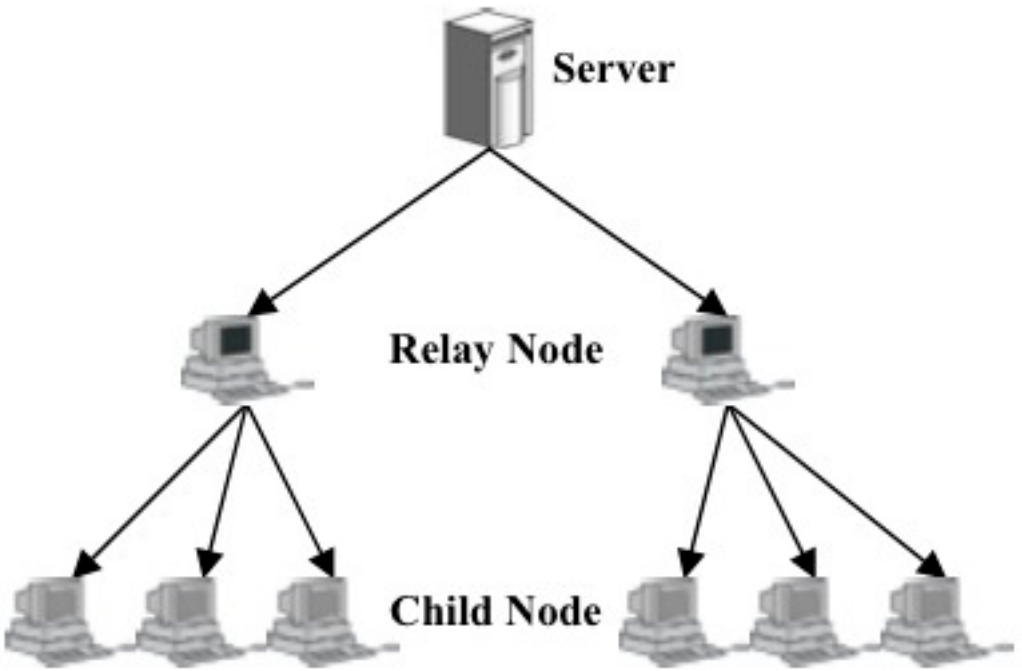
P2P côté serveur

- Exemple : [Rieche et al. 2006]
  - D'autre part, si un serveur s'arrête, une réplication sur des serveurs voisins permet de continuer à fonctionner



P2P côté client

- Exemple : [Ito et al. 2006]
  - Certains clients agissent comme des relay pour d'autres clients

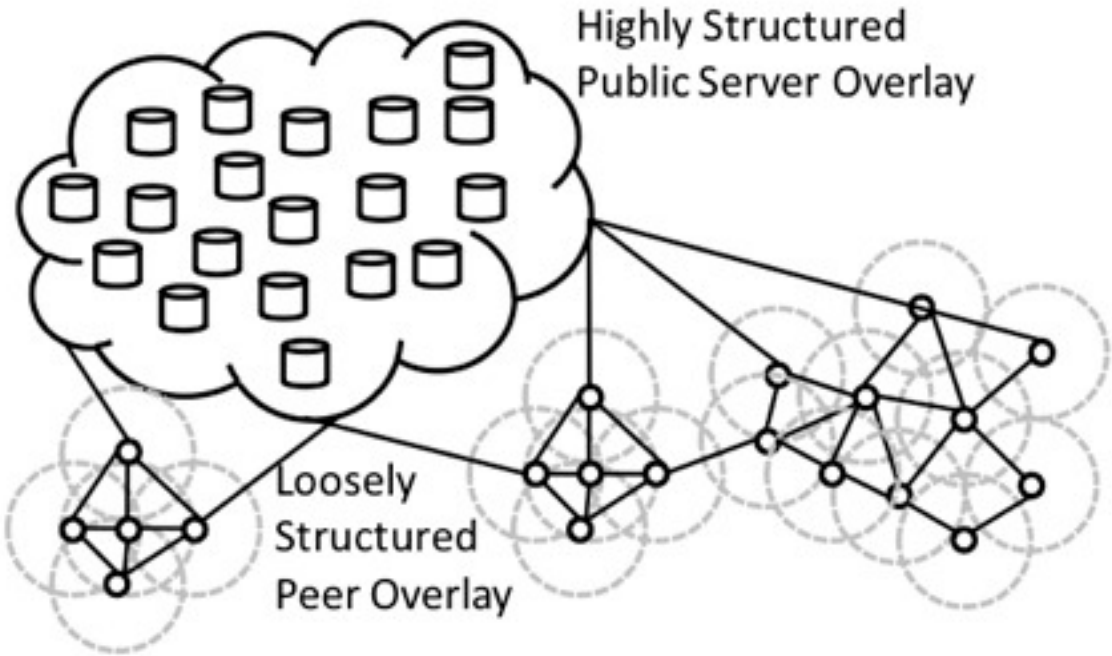


P2P côté client

- Exemple : Badumna
  - Les clients échangent les données les plus fréquentes directement en P2P
  - Ils utilisent les serveurs uniquement pour ce qui demande un arbitrage, une authentification ou une détection d'intrusion

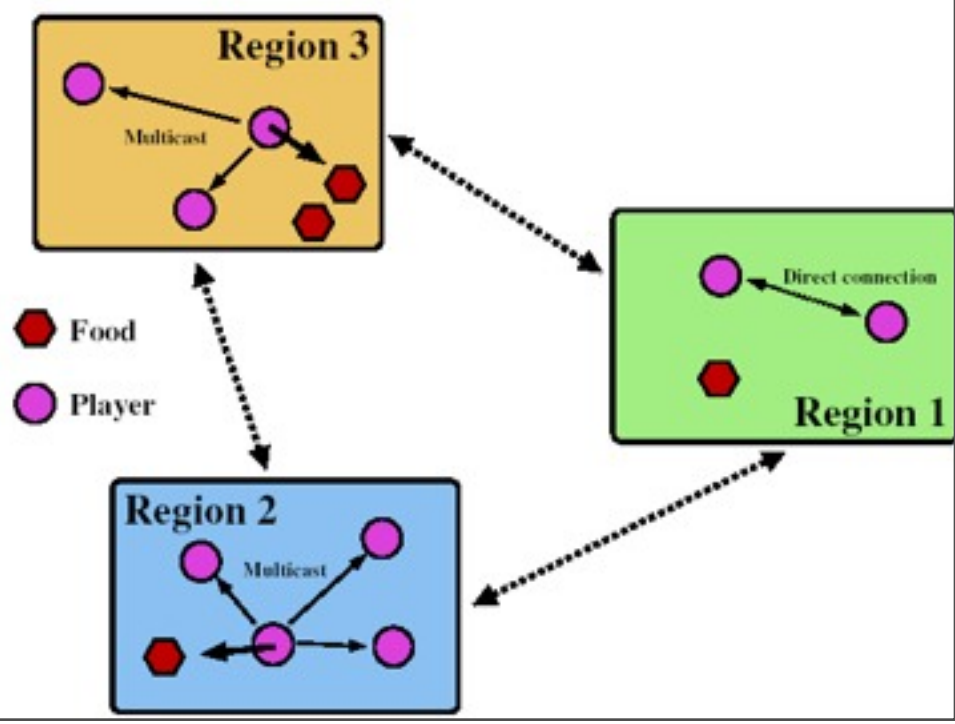
P2P des 2 côtés

- Exemple : HyperVerse/GP3 [Esch et al. 2008]
  - Utilise 2 réseaux P2P chacun avec ces caractéristiques propres



Coordinateurs de zones

- Un nombre fixe de zones
- Dans chaque zone, un super-peer coordonne l'activité
- Exemple : SimMud [Knutsson et al. 2004]
  - les coordinateurs gèrent des arbres multicasts construits sur Pastry et Scribe (multicast utilisant Pastry)



Point à point amélioré

- N*(N-1) connexions point à point
- Plus du filtrage
- Exemple:
  Update Free Regions (UFR) [Makbily et al. 1999]
  - définit des paires de régions mutuellement invisibles
  - les avatars qui sont dans des UFRs différentes n'ont pas à communiquer
  - Problèmes :
    - tous les noeuds doivent communiquer régulièrement pour calculer leurs UFRs
    - que se passe-t-il quand des foules apparaissent ?

Point à point amélioré

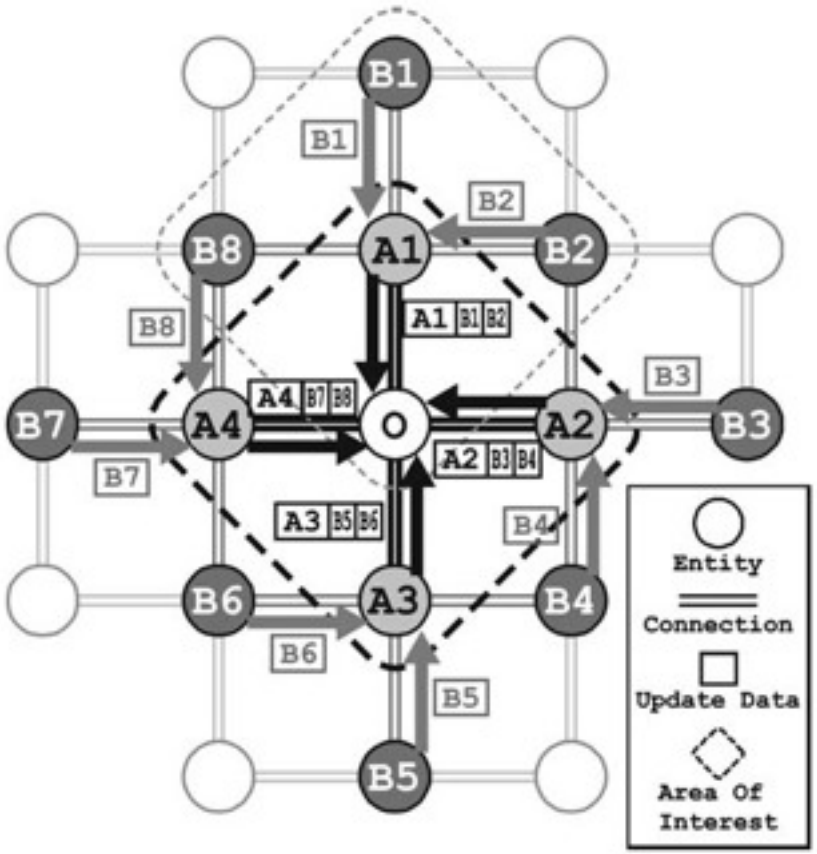
- Autre exemple : Frontier Sets [Steed et al. 2004]
  - Aussi longtemps que l'avatar vert reste dans la partie verte de la carte et que l'avatar rouge reste dans la partie rouge ils n'ont pas besoin de communiquer

Echange de voisinages

- Chaque noeud est connecté avec un nombre max de ses voisins les plus proches
- Les noeuds échangent constamment leurs voisinages pour découvrir de nouveaux voisins
- Pour éviter les partitions il y a toujours un nombre minimum de connexions à maintenir même si les voisins sont très éloignés

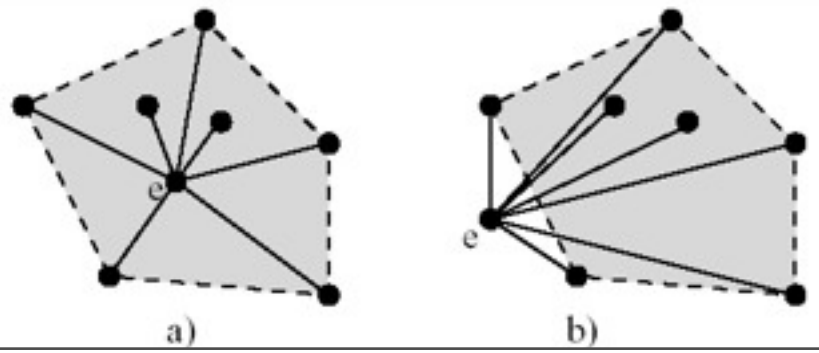
Echange de voisinages

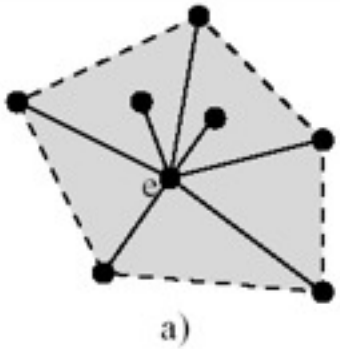
- Exemple: [Kawahara et al. 2004]

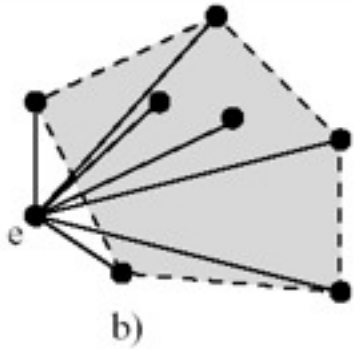


Notification mutuelle

- Chaque noeud est uniquement connecté à ces voisins d'AOI
  - ie les noeuds dont les avatars sont dans l'AOI de l'avatar local
- Quand les avatars se déplacent les noeuds notifient leurs voisins d'AOI en leur donnant les nouveaux voisins d'AOI
- Exemple : Solipsis 1 [Keller and Simon 2003]
  - utilise l'enveloppe convexe minimale

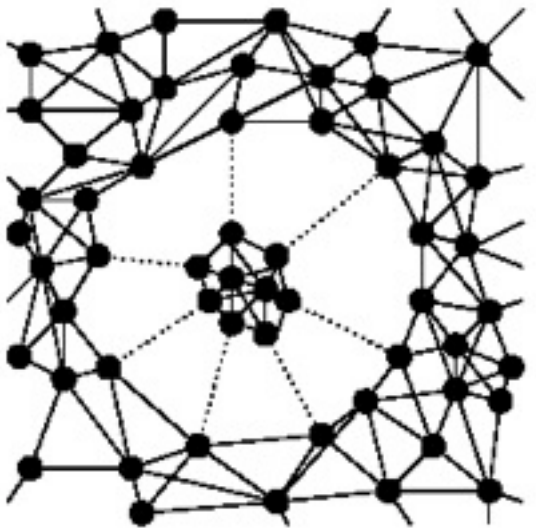






Notification mutuelle

- Exemple : Solipsis 1
  - si e est dans l'enveloppe convexe de ces voisins externes => pas de pb
  - si ce n'est pas le cas il doit se connecter à d'autres voisins
  - des connexions supplémentaires sont aussi maintenues pour éviter des partitions

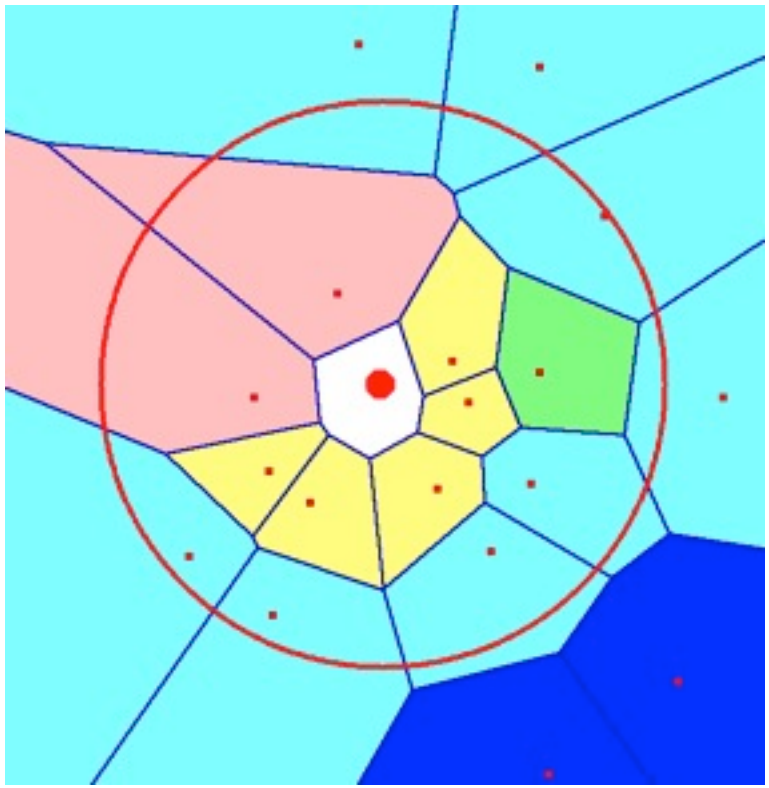


Notification mutuelle

- Exemple : VON [Hu et al. 2004]
  - Utilise les diagrammes de Voronoi pour solutionner le problème de découverte des voisins
  - Chaque noeud construit un diagramme de Voronoi de ces voisins
  - Ils collaborent pour trouver de nouveaux voisins

Notification mutuelle

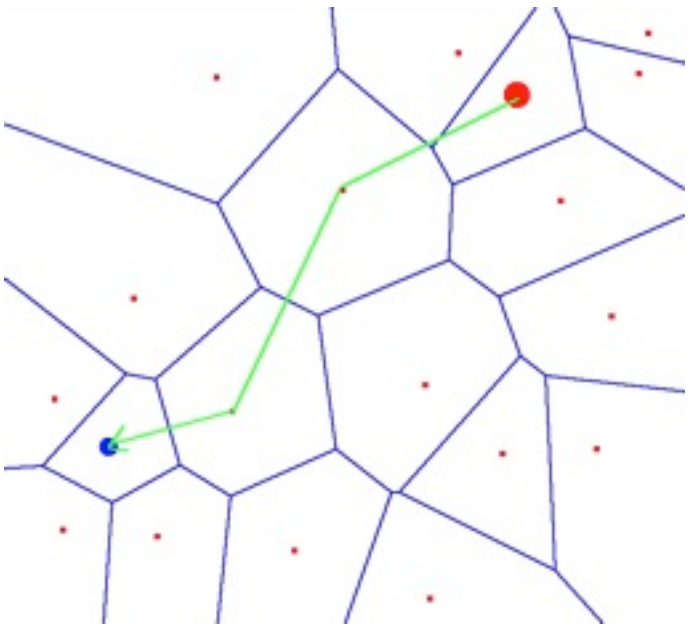
- Exemple : VON

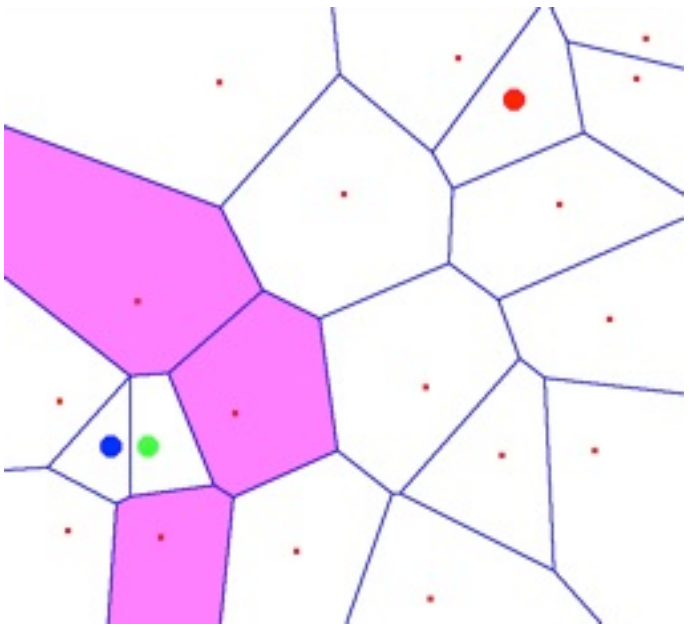


Cercle	Area of Interest (AOI)
Blanc	noeud courant
Jaune	voisin inclut (E.N.)
Cyan	voisin de bordure (B.N.)
Rose	E.N. & B.N.
Vert	voisin d'AOI
Bleu	voisin inconnu

Notification mutuelle

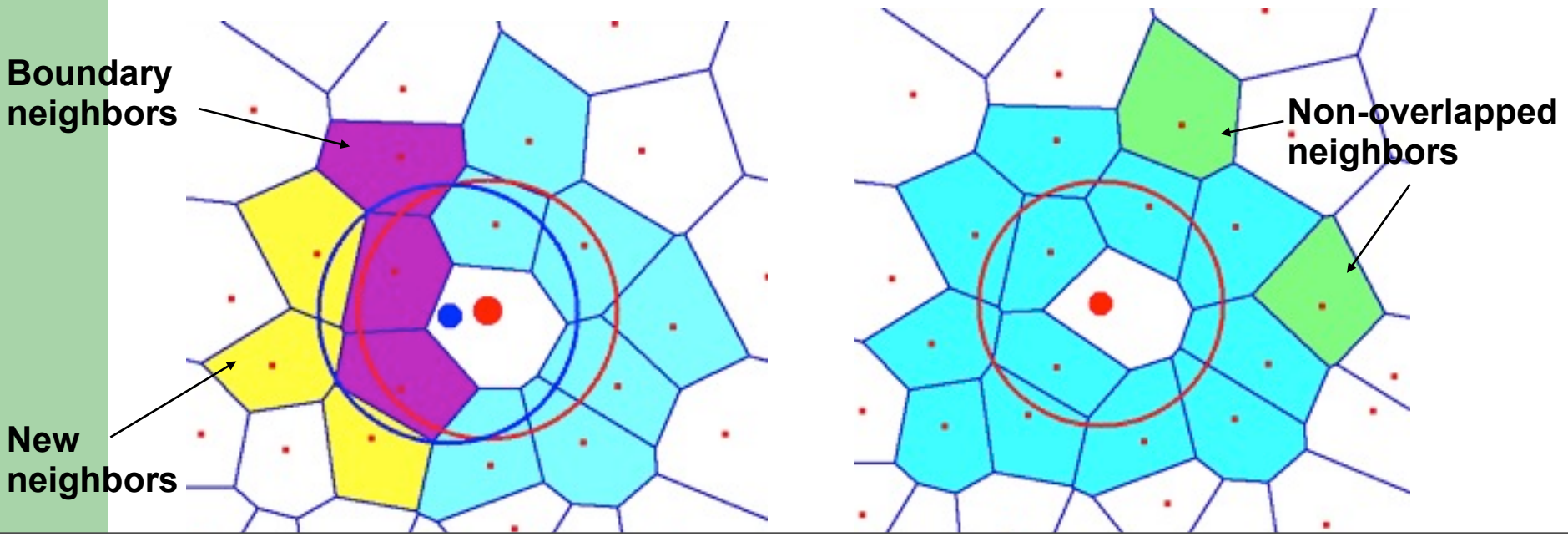
- Exemple : VON
  - quand un nouveau noeud apparait : une requête JOIN est propagée de n'importe quel noeud vers le noeud le plus proche





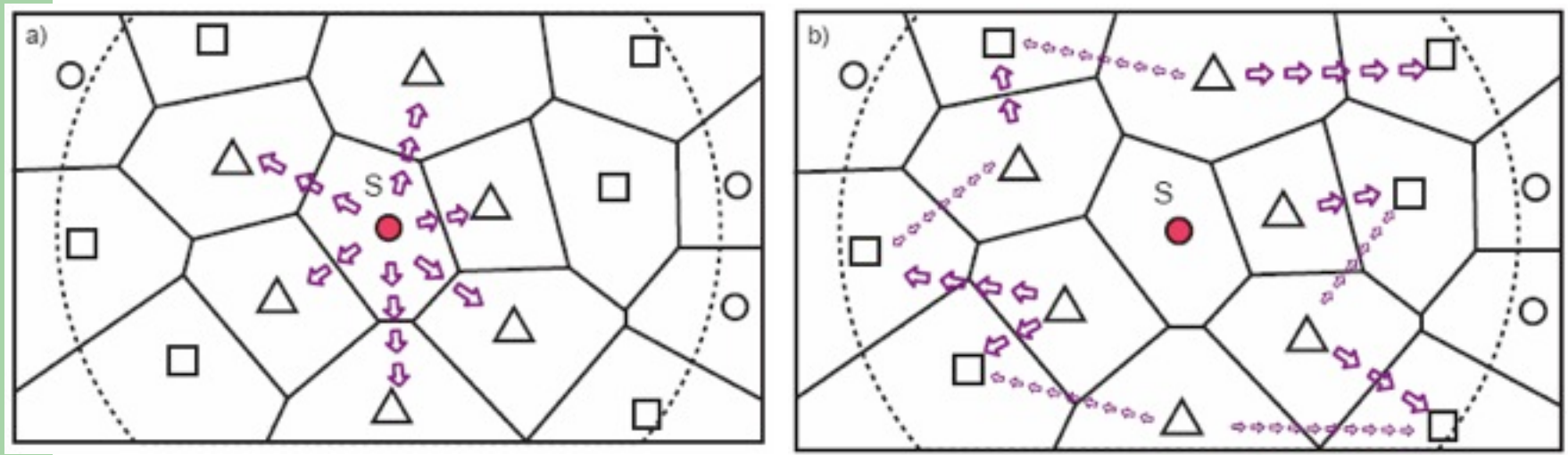
Notification mutuelle

- Exemple : VON
  - quand un noeud bouge, il envoie sa position à ces voisins connus. Les B.N. vérifient si l'AOI intersecte de nouveaux voisins. Les B.N. notifient le noeud qui bouge.



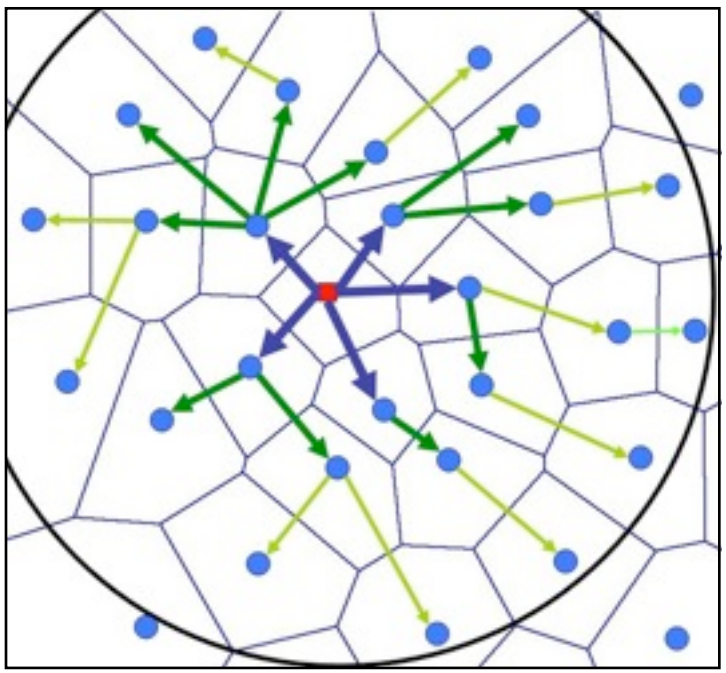
Notification mutuelle avec Mcast

- Si on ne peut pas maintenir des connexions directes avec tous les voisins => on utilise le multicast
- Exemple : VON forwarding [Chen et al. 2007]
  - propagation dans l'AOI



Notification mutuelle avec Mcast

- Exemple : VoroCast [Jiang et al. 2008]
  - Construit un arbre minimal dynamiquement



Notification mutuelle avec Mcast

- Exemple : FiboCast [Jiang et al. 2008]
  - Ajuste la fréquence de transmission des messages vers les noeuds éloignés

	Max_count	*Fibonacci*
1	2	0
2	3	1
3	3	1
4	4	2
5	5	3
6	7	5
7	10 > 8	8
8	∞	∞

hops	1	2	3	4	5	6	7	8	
	ˇ	ˇ	X	X	X	X	X	X	-6
	ˇ	ˇ	ˇ	X	X	X	X	X	-5
	ˇ	ˇ	ˇ	X	X	X	X	X	-5
	ˇ	ˇ	ˇ	ˇ	X	X	X	X	-4
	ˇ	ˇ	ˇ	ˇ	ˇ	X	X	X	-3
	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	X	-1
	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	0
	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	ˇ	0

Intergiciels intéressants

- JXTA (juxtapose) : offre un ensemble de bibliothèques utilisables pour faire des réseaux P2P http://jxta.dev.java.net/
- Le projet VAST : offre une implémentation de VON en C++ (il y a aussi une version Java non maintenue) http://vast.sourceforge.net/
- Badumna : offre un réseau P2P pour MMVE en C# (il peut aussi servir de proxy) http://www.badumna.com/

Remerciements

- Cette présentation utilise :
  - Eng Keong Lua; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S., "A survey and comparison of peer-to-peer overlay network schemes," Communications Surveys & Tutorials, IEEE , vol.7, no.2, pp. 72-93, Second Quarter 2005
  - Un tutorial sur le P2P présenté à PDCAT07 par Aaron Harwood http://p2p.csse.unimelb.edu.au/docs/PDCAT07-Tutorial.pdf
  - Les pages du projet VAST “publications” et “related work” (gérées par Shun-Yun Hu) http://vast.sourceforge.net/index.php
  - Les articles de Wikipedia dans la catégorie Distributed Data Sharing http://en.wikipedia.org/wiki/Category:Distributed_data_sharing