
Gestion de données dans les systèmes Pair-à-Pair

Bruno DEFUDE

Dept Informatique

GET- Institut National des Télécommunications

<http://www-inf.int-evry.fr/~defude/P2P>

Plan

1. Introduction
2. P2P pur : GNUTELLA
3. Super-peers
 1. SuperPeers
 2. Kazaa
4. P2P « structuré »
 1. Chord
 2. Autres (P-Grid, CAN, Pastry)
5. P2P sémantique
 1. Routing Indices
6. JXTA
7. Synthèse

Moteurs de recherche

- gestion centralisée (index) des ressources
- Index basé sur le contenu des ressources (texte), nom des ressources (images, vidéo), méta-données?
- indexation (robot se baladant sur le web et collectant les données) : index incomplet et pas à jour
- Recherche : algorithme de comparaison requête – document : Vector Space Model (le plus simple), PageRank (Google, tient compte de l'analyse des liens entre documents)

Moteurs de recherche (bilan)

- Difficulté à passer à l'échelle
 - Google est incomplet et nécessite plus de 15000 serveurs pour tenir la charge : difficile à mettre en place
- Pas forcément adapté pour autre chose que du texte
- Centralisé donc sensible aux attaques et aux fautes

Définition P2P

- Chaque nœud participant peut être client et serveur
- Chaque nœud paye sa participation en donnant accès à une partie de ses ressources
- Propriétés :
 - Pas de coordination centralisée
 - Pas de BD centralisée
 - Aucun nœud n'a une vision globale du système
 - Comportement global émerge à partir des interactions locales
 - Tous les services et données sont accessibles de n'importe quel nœud
 - Nœuds sont autonomes
 - Nœuds et connections sont non fiables

Classes de systèmes P2P

- P2P Hybrides (e.g Napster)
 - Index centralisé (non tolérant aux fautes)
 - Échange d'information direct
- P2P « Purs » (e.g Freenet, Gnutella)
- P2P Hiérarchiques ou « super-peers » (e.g Kazaa)
 - Mélange C/S et P2P
- P2P structurés (e.g Chord, P-Grid)
 - Structuration espace
- P2P sémantiques (e.g Routing Indices)
 - P2P « pur » avec routage basé sur une information sémantique

Fonctionnalités d'un système P2P

- Découverte de ressources
- Gestion des mises à jour
- Passage à l'échelle
- Tolérance aux fautes
- sécurité



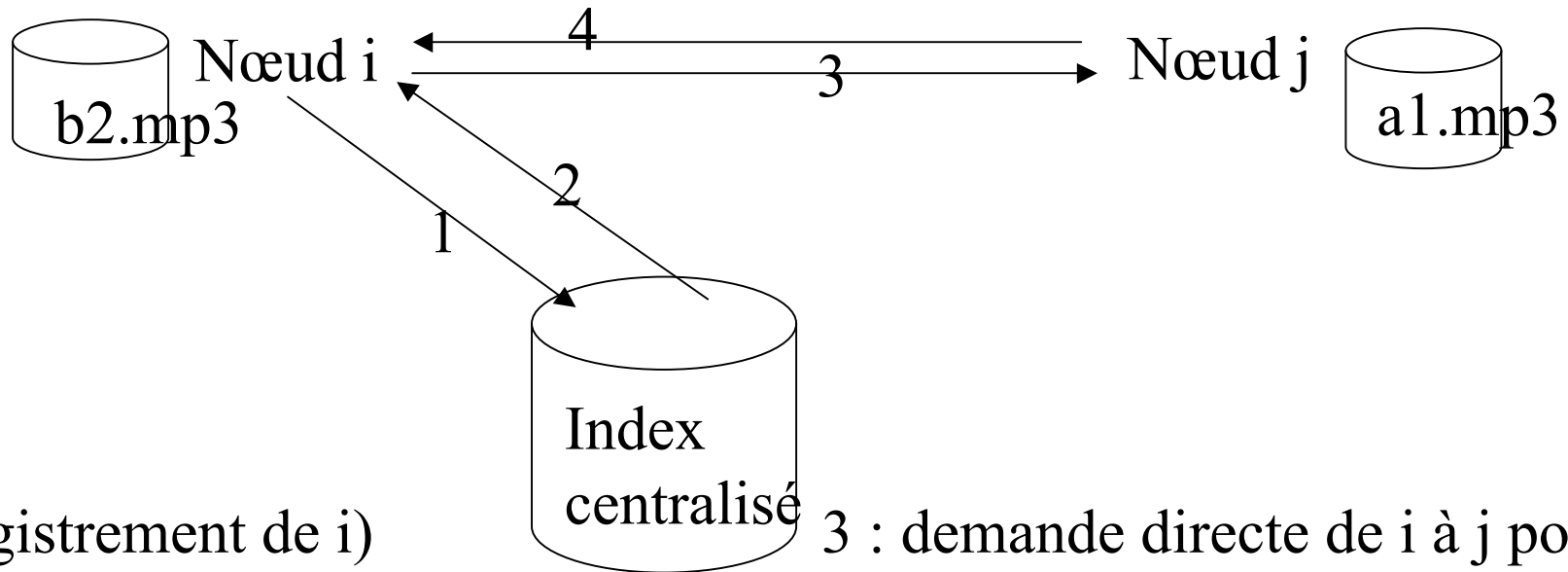
Classes d'applications

- Partage de fichiers : Napster, Gnutella, Freenet, KaZaa
- Système de stockage persistant à grande échelle : OceanStore, Pastis
- Grid computing : Seti@home

Critères de comparaison

- Recherche de ressource :
 - Topologie du réseau : ouverte (Gnutella) ou contrôlée (Chord)
 - Placement des données et méta-données : libre (Gnutella) ou dirigé (Chord)
 - Routage des messages : fonction de choix des successeurs
- Besoins applicatifs :
 - Expressivité du « langage de requêtes » : égalité (Chord), préfixe (P-Grid), SQL (?), ...
 - Complétude des résultats : une réponse, toutes les réponses, les k meilleures réponses, ...
 - Autonomie des nœuds : choix des ressources à stocker, choix des nœuds successeurs, ...

Napster



1: (enregistrement de i)
recherche de a1.mp3

(ajout des fichiers de i dans index)

2 : retour des nœuds possédant
a1.mp3

3 : demande directe de i à j pour
télécharger a1.mp3

4: téléchargement
de a1.mp3 et ajout dans la base de
ressources partagées

Bilan Napster

- Fonctionne bien à l'échelle d'Internet
- Accès en P2P mais recherche centralisée
- Serveur doit être bien dimensionné et tolérant aux fautes (cluster avec 100, 1000, ... nœuds)
- Sensible aux partitionnements du réseau (serveur inatteignable) et aux attaques

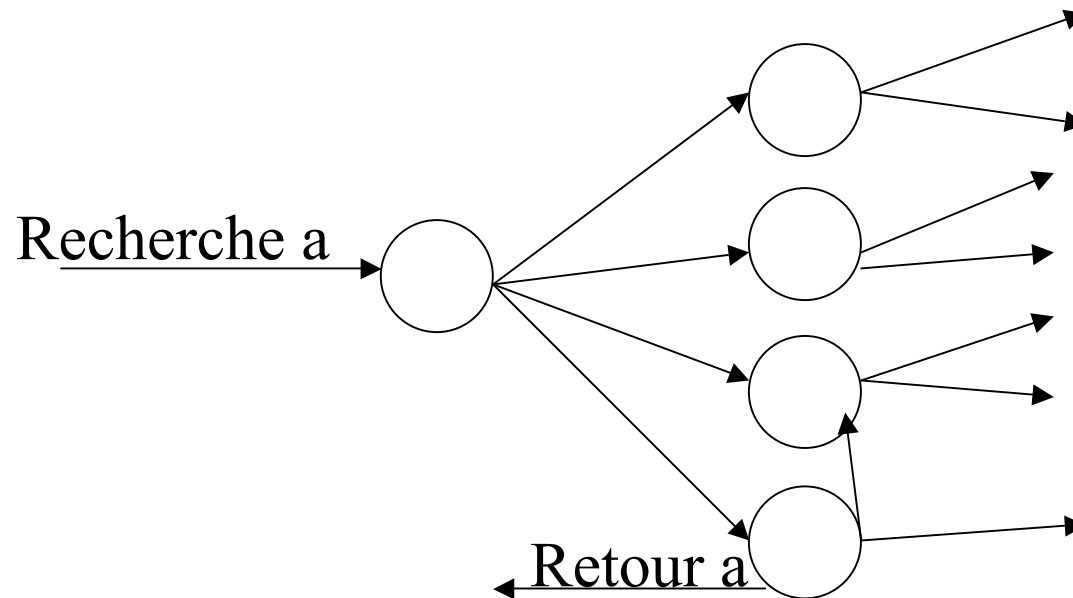
Plan

1. P2P pur : GNUTELLA
2. Super-peers
3. P2P « structuré »
4. P2P sémantique
5. Synthèse

P2P « pur »

GNUTELLA

Gnutella



Chaque nœud propage la requête à k voisins (4)

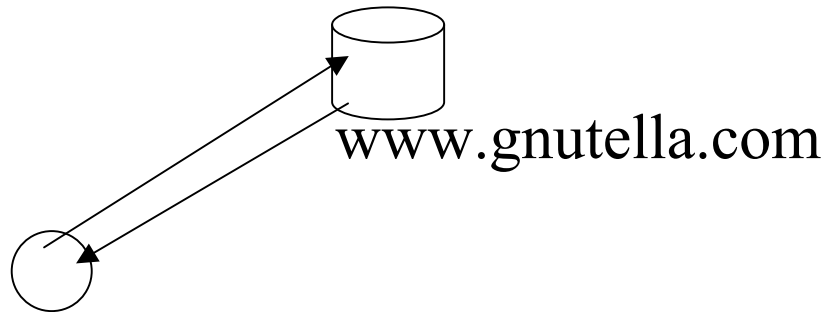
Nombre de propagation limité (7)

Détection de cycles

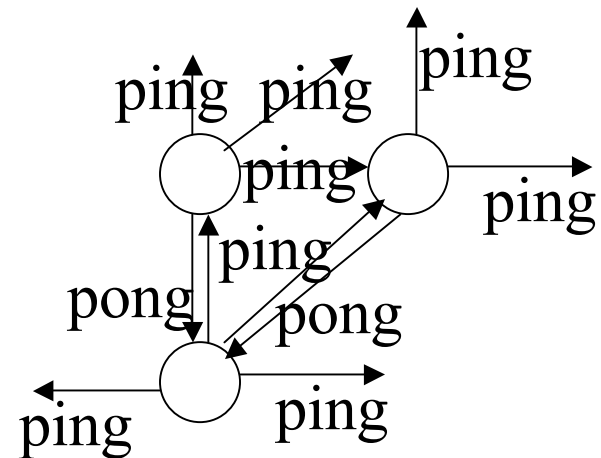
Gnutella : types de messages

Type	Description	Information
Ping	Annonce disponibilité et lance recherche nouveaux pairs	vide
Pong	Réponse à un ping	Adresse IP + No port; nombre et taille de fichiers partagés
Query	Requête	Bande passante minimum demandée; critère de recherche
QueryHit	Réponse à Query si on possède la ressource	Adresse IP + No port et bande passante; nombre de réponses + descripteurs réponses
Push	Demande de téléchargement pour pairs derrière un firewall	Identifiant du pair; index du fichier demandé; adresse IP No port où envoyer le fichier

Gnutella (ajout d'un noeud)

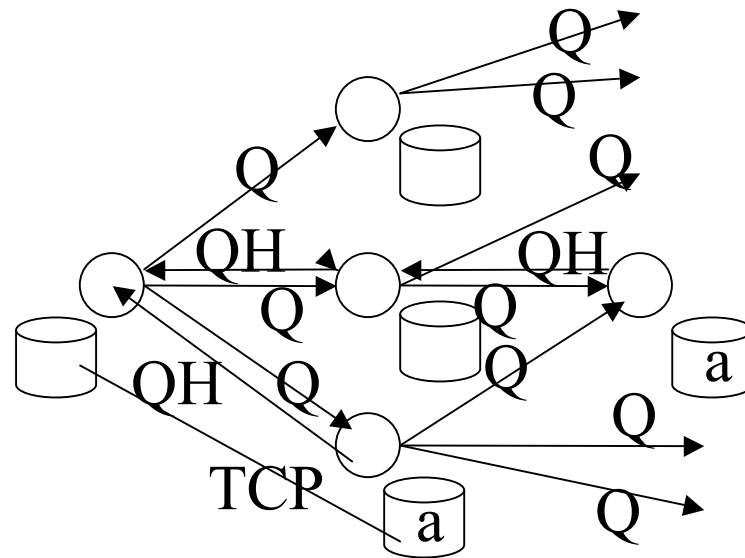


Initialisation de la
table des nœuds connus
(extérieur au protocole)



Compléter la table
des nœuds connus

Gnutella (recherche)



Les principes sous-jacents

- Principe d'égalité entre les nœuds
 - Même capacité (puissance, bande passante, ...)
 - Même comportement (également client et serveur) et bon comportement (pas de « mensonge »)
- Principe de requêtes « populaires »
 - Requêtes concernent principalement peu de ressources
 - Ressources très demandées sont très répliquées
- Principe de topologie du réseau
 - Graphe minimisant le nombre de chemins entre deux nœuds
 - Longueur du chemin minimum entre deux nœuds quelconque est faible (5 à 8)

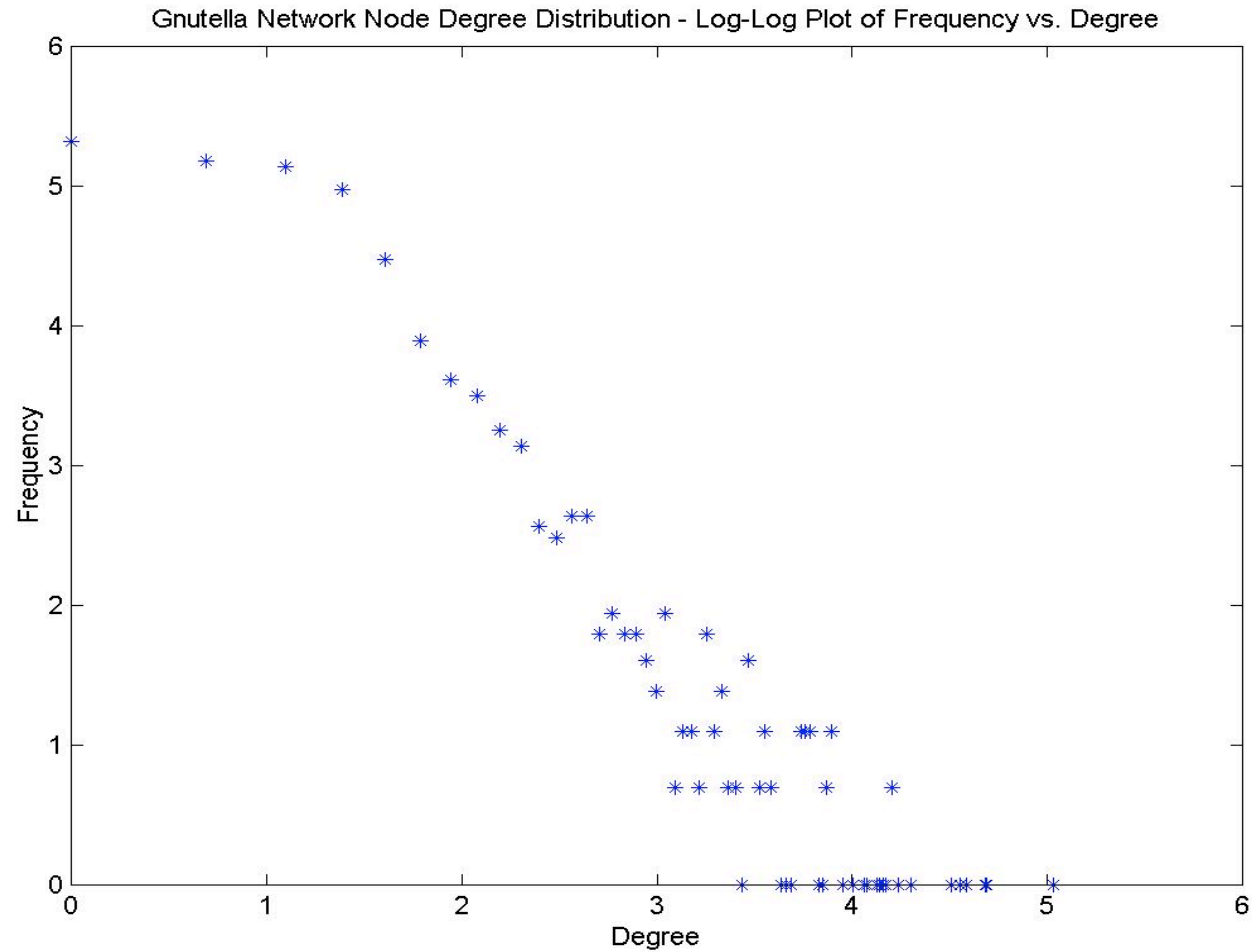
Quid des principes en réalité?

- Principe d'égalité entre les nœuds
 - A- [Sarioiu et al. 01] montrent un écart de 1 à 3 dans la bande passante disponible
 - B- [Adar, Huberman 00] montrent que 70% des utilisateurs ne partagent aucun fichier (ils n'en ont pas ou bien ils n'intéressent personne) et que 50% des résultats sont produits par 1% des nœuds
 - A peut perturber le réseau et produire des partitionnements (trop forte charge demandée à des nœuds connectés via des modems e.g)
 - B implique que d'une part ceux qui partagent n'y ont pas intérêt (pas de réciprocité) et d'autre part que le réseau est sensible aux pannes et aux attaques
 - Études montrent également que certains nœuds sous-évaluent leur bande passante disponible pour éviter d'être choisis

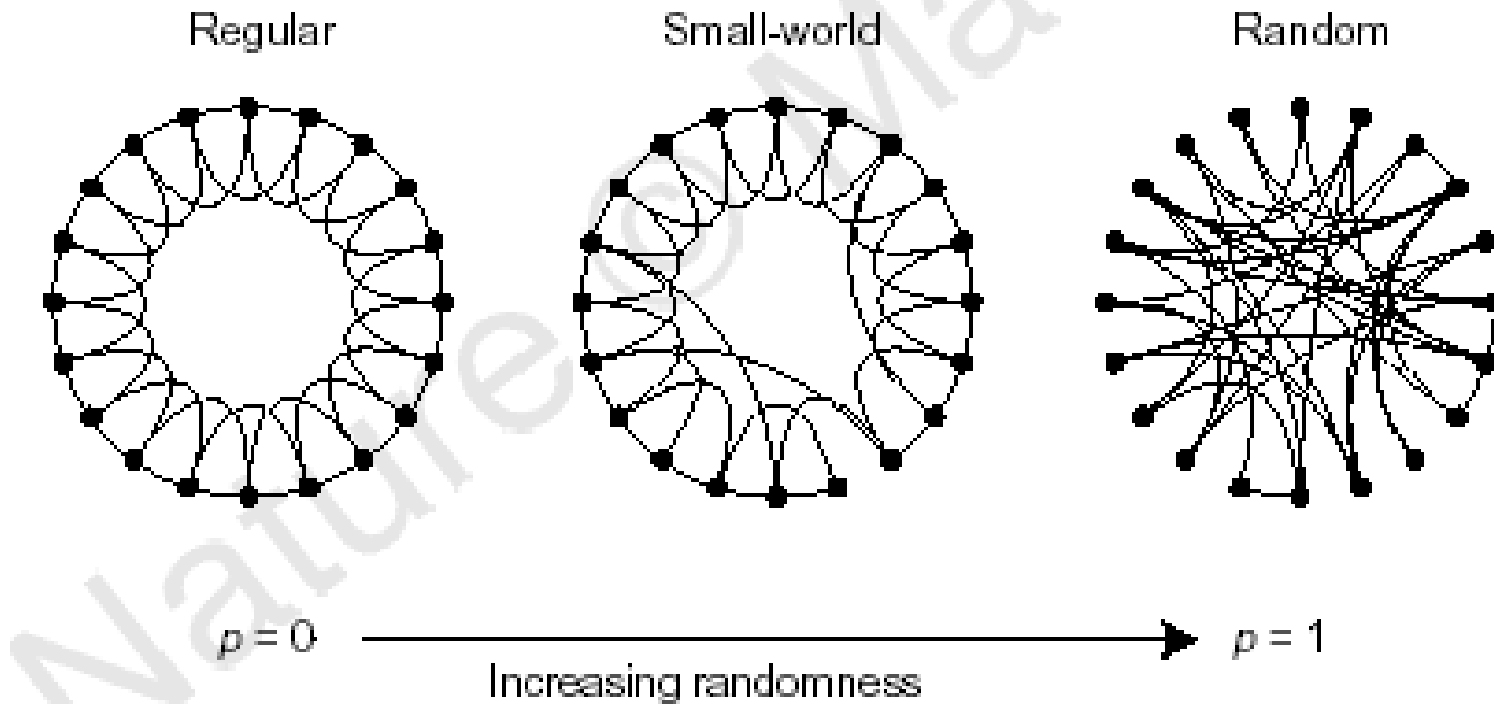
Quid des principes (2)

- Principe des requêtes « populaires »
 - Les 100 requêtes les plus fréquentes sont distribuées uniformément
 - Les autres suivent une distribution de type Zipf
 - Les techniques de cache de résultats s'appliquent bien et peuvent apporter une amélioration notable
- Principe de topologie du réseau
 - Plusieurs études montrent que le graphe sous-jacent de Gnutella est de type « small-world » et que le degré des nœuds suit une distribution « power law »
 - Le principe de diffusion de Gnutella ne s'adapte pas à SW (beaucoup de messages redondants)

Topologie Réseau gnutella



Graphes Small World (Watts, Strogatz, Nature)



Observations sur la bande passante Gnutella

- Sur une période de un mois
 - Requête = 560 bits (y compris headers TCP/IP)
 - Requêtes 25% du trafic, pings 50% et reste 25%
 - En moyenne un pair est connecté activement à 3 autres
- Limite de la dégradation à 10 requêtes / seconde
 - $10 \text{ req.} \times 560 \text{ bits} \times 4 \times 3 \text{ connections} = 67200 \text{ b/s}$
 - Au-dessus de la capacité des modems
- Gnutella ne passe pas l'échelle

Bilan de Gnutella

- Complètement décentralisé
- Très tolérant aux fautes
- S'adapte bien à la dynamique du réseau
- Simple, robuste et passe l'échelle (pour le moment)
- Gros consommateur de bande passante
- Pas de garantie de succès, ni d'estimation de la durée des requêtes
- Pas de sécurité, ni de réputation
- Problème du « free riding »

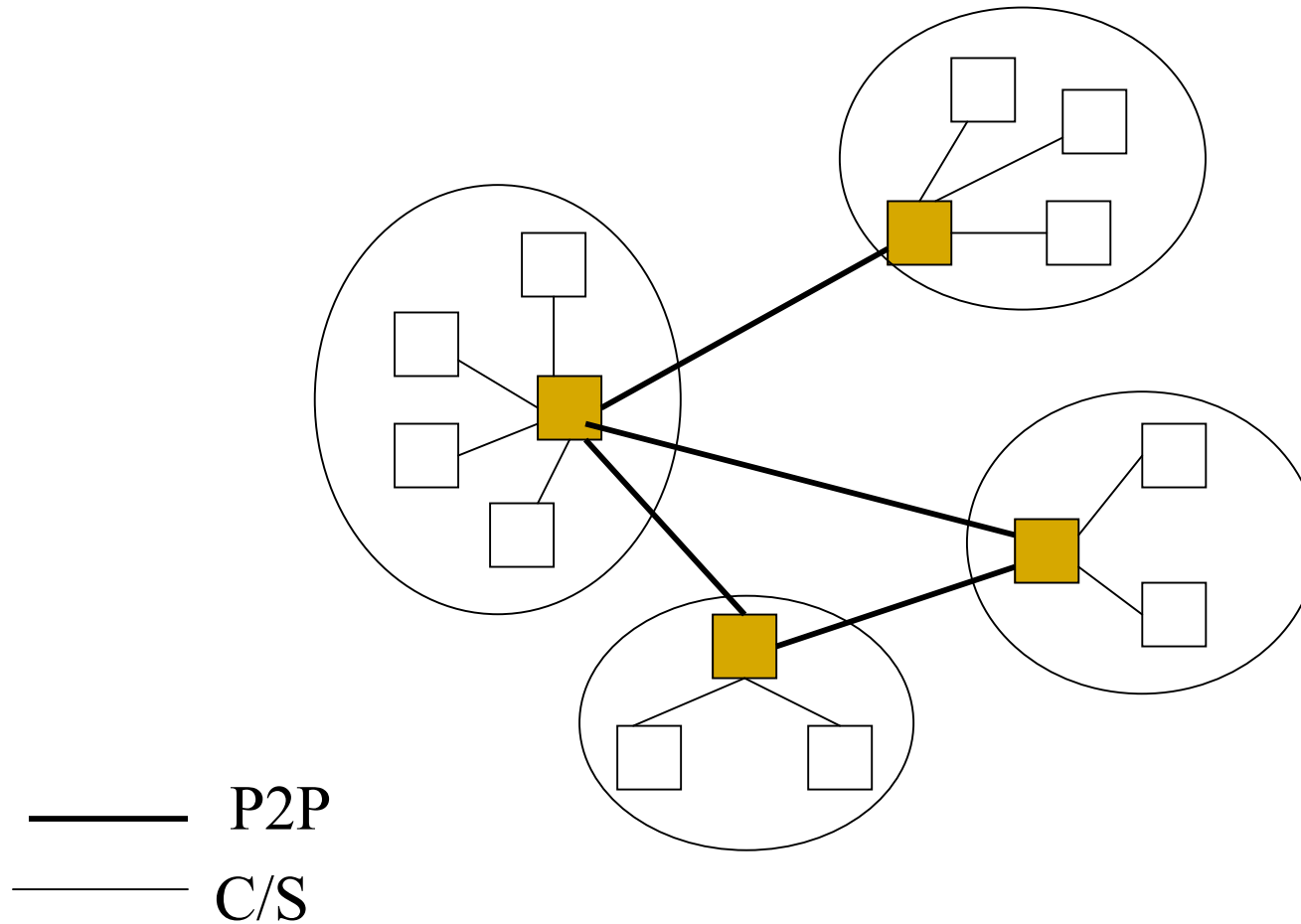
Super-peers

Client/serveur + P2P

Super-peers

- Éviter les problèmes dus à l'hétérogénéité de la bande passante des nœuds
- Tous les nœuds ne sont plus égaux
 - Nœuds avec bonne bande passante sont organisés en P2P : les super-peers
 - Nœuds avec faible bande passante sont rattachés en mode client/serveur à un super-peer (cluster)
 - Super-peers disposent d'un index des ressources de leur cluster
- Utilisé dans KaZaa

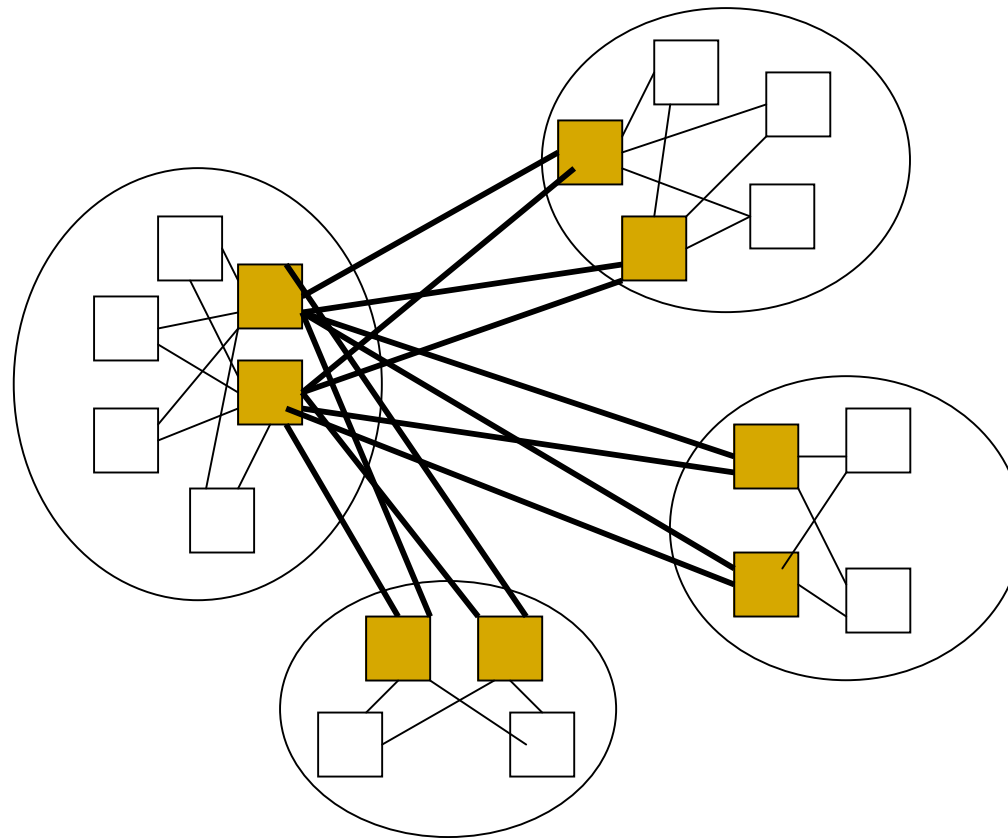
Exemple de super-peer



Super-peer redondant

- Super-peer introduit de la sensibilité aux fautes
- Amélioration possible, choisir k super-peers (partenaires) au lieu d'un dans un cluster
- Chaque partenaire est connecté à chaque client et possède un index de toutes leurs ressources
- Clients envoient leurs requêtes aux partenaires selon un principe de « round-robin »
- Les voisins d'un partenaire distribuent également leurs requêtes équitablement
- Fait baisser la charge du partenaire d'un facteur k
- Augmente le coût d'entrée d'un nouveau client d'un facteur k
- Augmente le nombre de connections ouvertes de k^2

Exemple de super-peer redondant



Kazaa

- Plus de 3 millions d'utilisateurs, partageant plus de 3000 To de contenu
- Plus populaire que Napster
- Plus de 50% du trafic Internet ?
- Mp3, vidéos, jeux
- Possibilité de téléchargement en parallèle
- Récupération automatique du téléchargement si un des serveurs tombe
- Donne une estimation du temps de téléchargement

Kazaa (2)

- Super-pair sans redondance (mais si un super-pair tombe, les membres du groupe en choisissent un autre)
- Taille d'un groupe : 100-150 pairs
- Environ 30000 super-pairs
- Chaque super-pair est connecté via TCP à environ 30-50 super-pairs

Règles de conception d'un super-peer [Yang 2003]

- R1 : augmenter la taille d'un cluster augmente la charge individuelle (bande passante) mais diminue la charge agrégée
 - Peu de gros clusters : sensibilité aux fautes, attaques, ...
- R2 : la redondance de super-peer est favorable
 - Amène la bonne charge agrégée des gros clusters avec une faible charge individuelle et une bonne tolérance aux fautes

Règles de conception d'un super-peer

- R3 : maximiser le nombre de connections des super-peers (diminue le nombre de sauts pour atteindre les résultats). Stratégie doit être choisie par tous les super-peers
- R4 : minimiser le TTL (Time To Live)

P2P « Structuré » (DHT)

Distributed Hash Table :
Chord, P-Grid

Chord [Stoica et al. 2001]

- Table de hachage distribuée
- Les nœuds sont répartis sur un anneau
- Les ressources sont réparties sur les différents nœuds de l'anneau
- Structure dynamique
 - Ajout/retrait de nœud
 - Panne d'un nœud
- Peut être utilisée pour construire des applications au-dessus (DNS, ...)

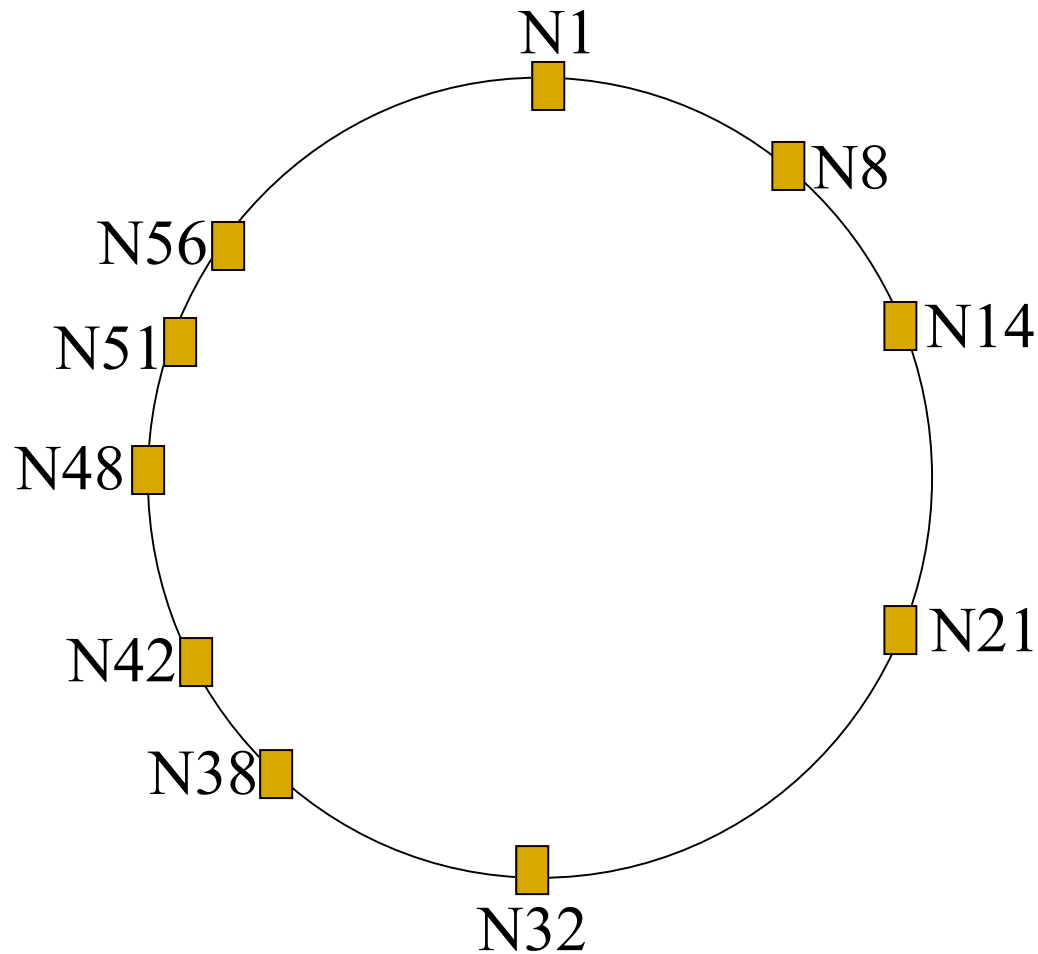
Consistent Hashing

- Chord alloue les ressources aux nœuds en utilisant une fonction de hachage
- Consistent Hashing garantit avec une probabilité élevée
 - Ressources sont distribuées uniformément sur l'ensemble des nœuds
 - L'ajout/retrait d'un Nème nœud n'oblige à déplacer que $O(1/N)$ ressources

Structure en anneau

Chaque nœud est
alloué sur l'anneau en
fonction de hash(IP)

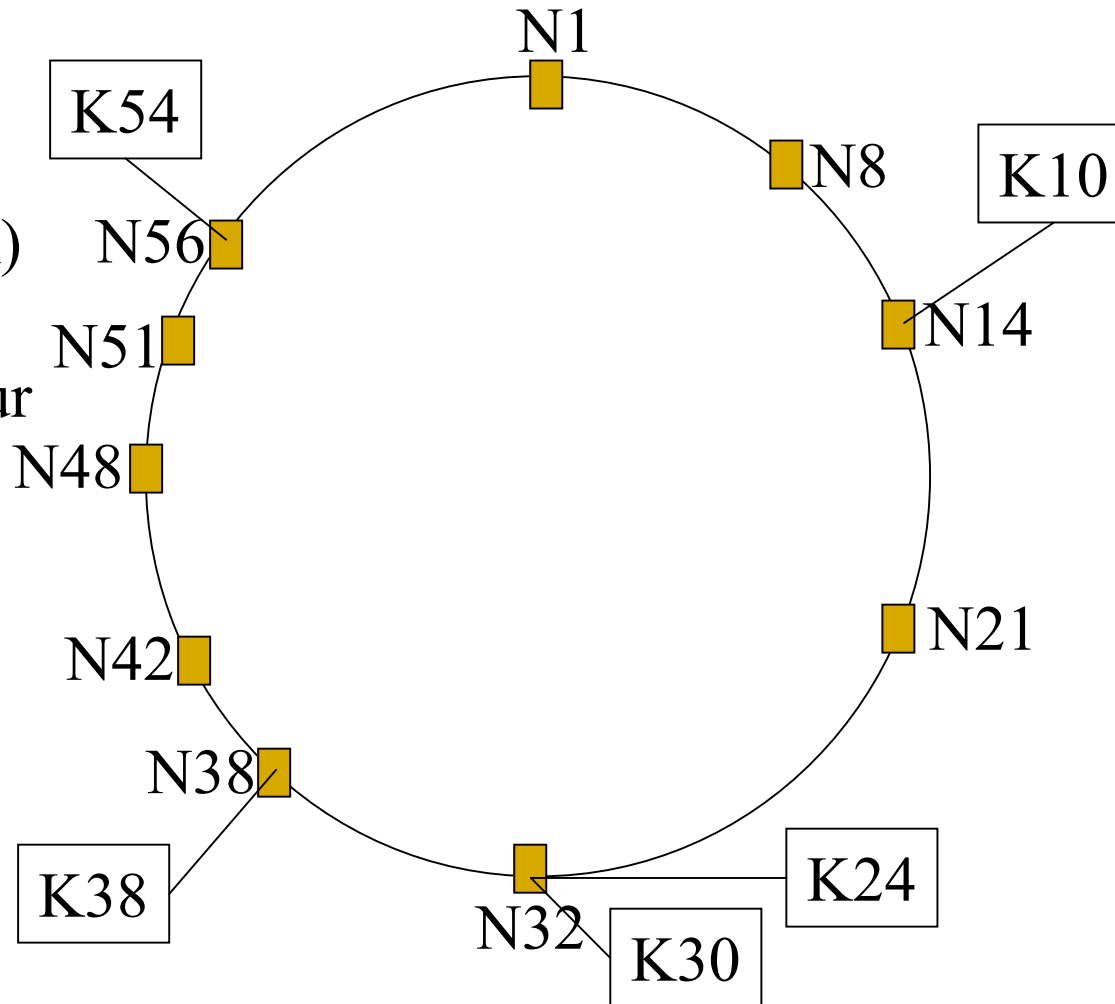
Au plus 2^m nœuds



Placement des ressources

Hash(ressource)=k

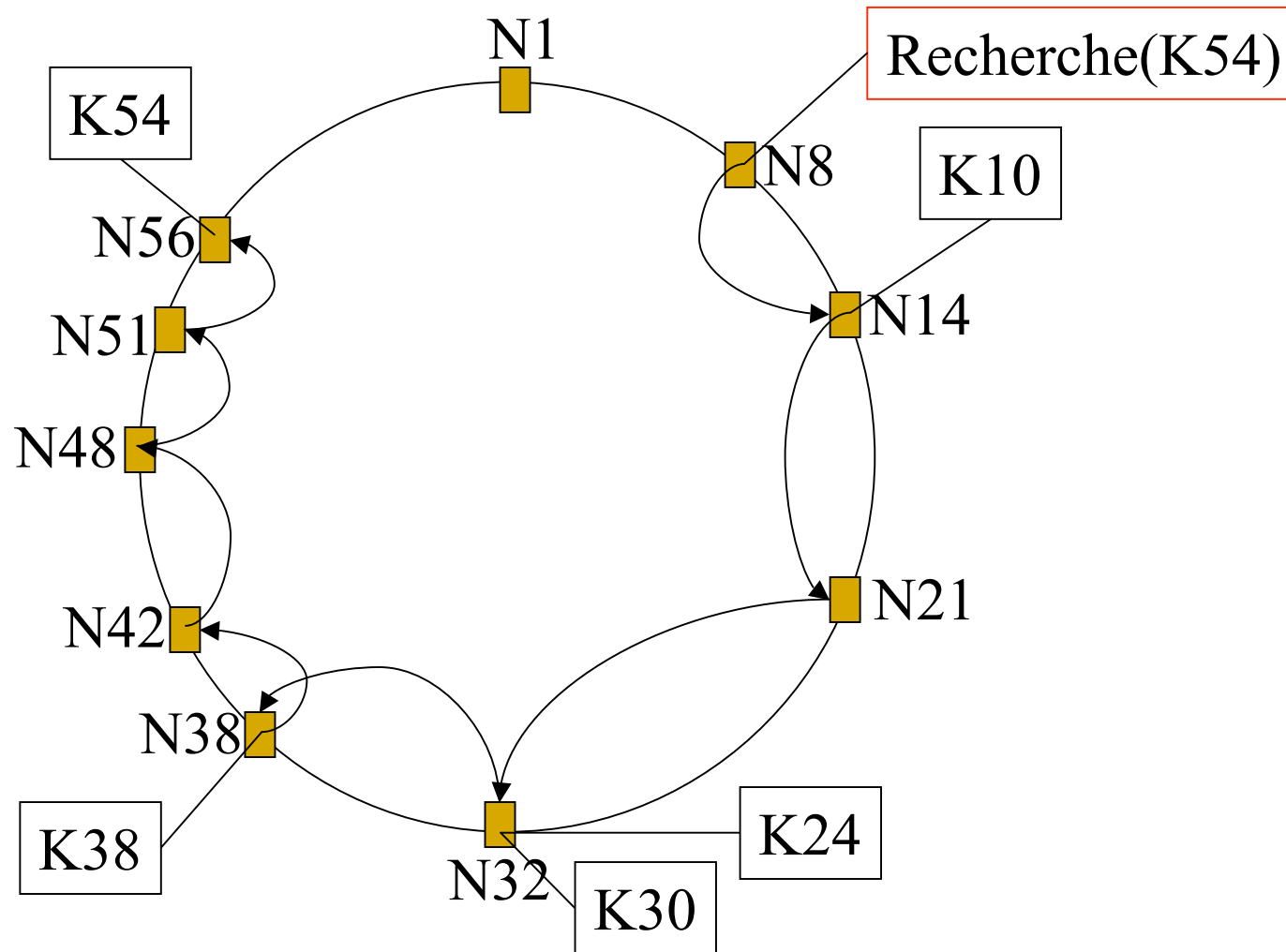
k placé sur successeur(k)
successeur(k)=nœud
immédiatement supérieur
(ou égal) à k



Recherche (naïve) d'une ressource

- Sur le nœud i on reçoit la requête : recherche(k)
- Si i possède k , il retourne k
- Sinon, il propage la requête à son successeur (chaque nœud doit stocker l'identification de son successeur)
- Le résultat suit le chemin dans l'ordre inverse
- Recherche linéaire en nombre de nœuds

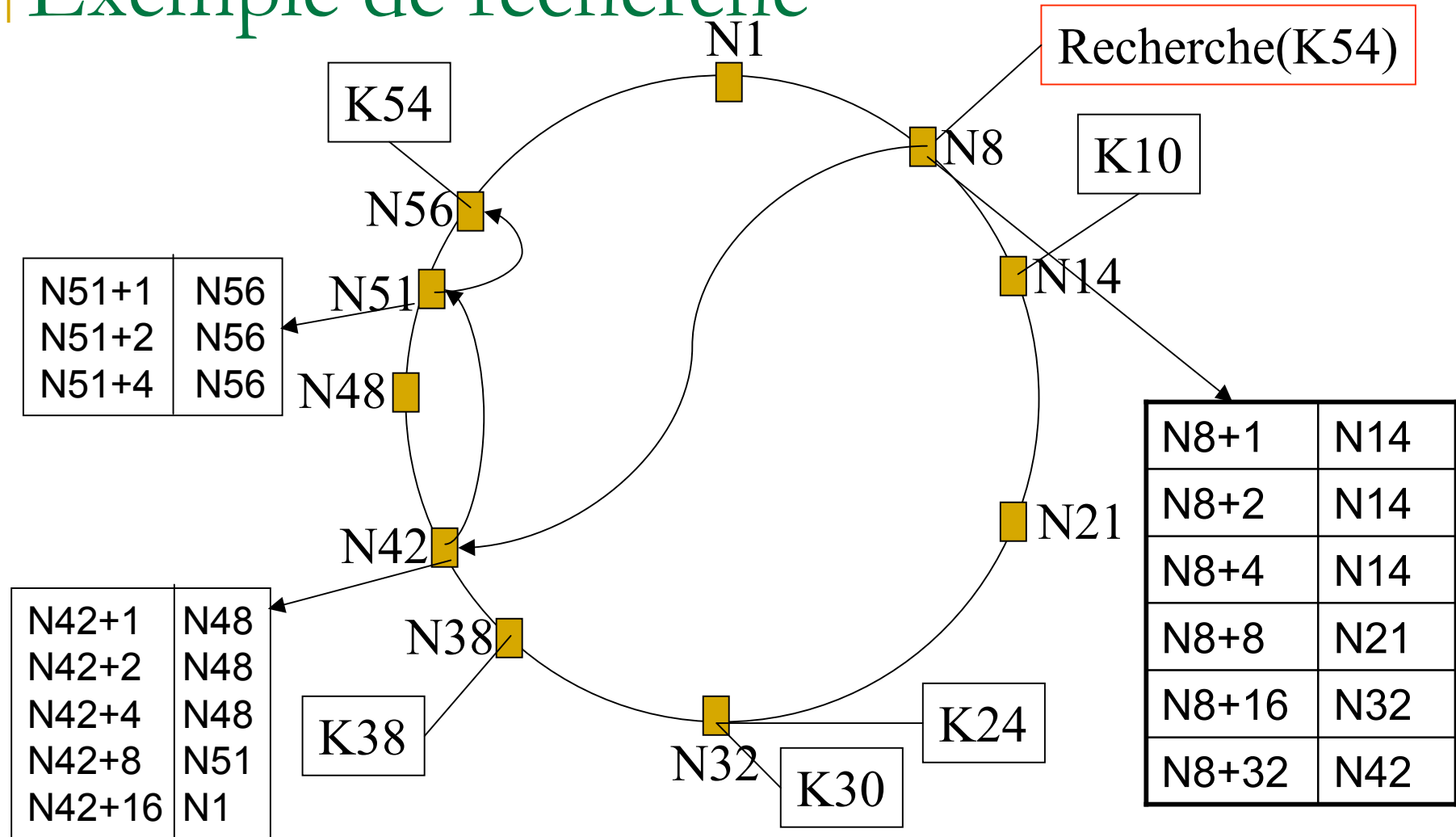
Exemple de recherche naïve



Amélioration de la recherche

- Avoir une table de routage plus complète
- Pour chaque nœud i :
 - $\text{Succ}[k] = \text{premier nœud sur l'anneau qui vérifie } (i + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$
 - Successeur = $\text{succ}[1]$
 - m entrées dans la table
 - Prédecesseur (utilisé pour la maintenance dynamique du réseau)
 - Donne adresse IP et No de port du nœud

Exemple de recherche



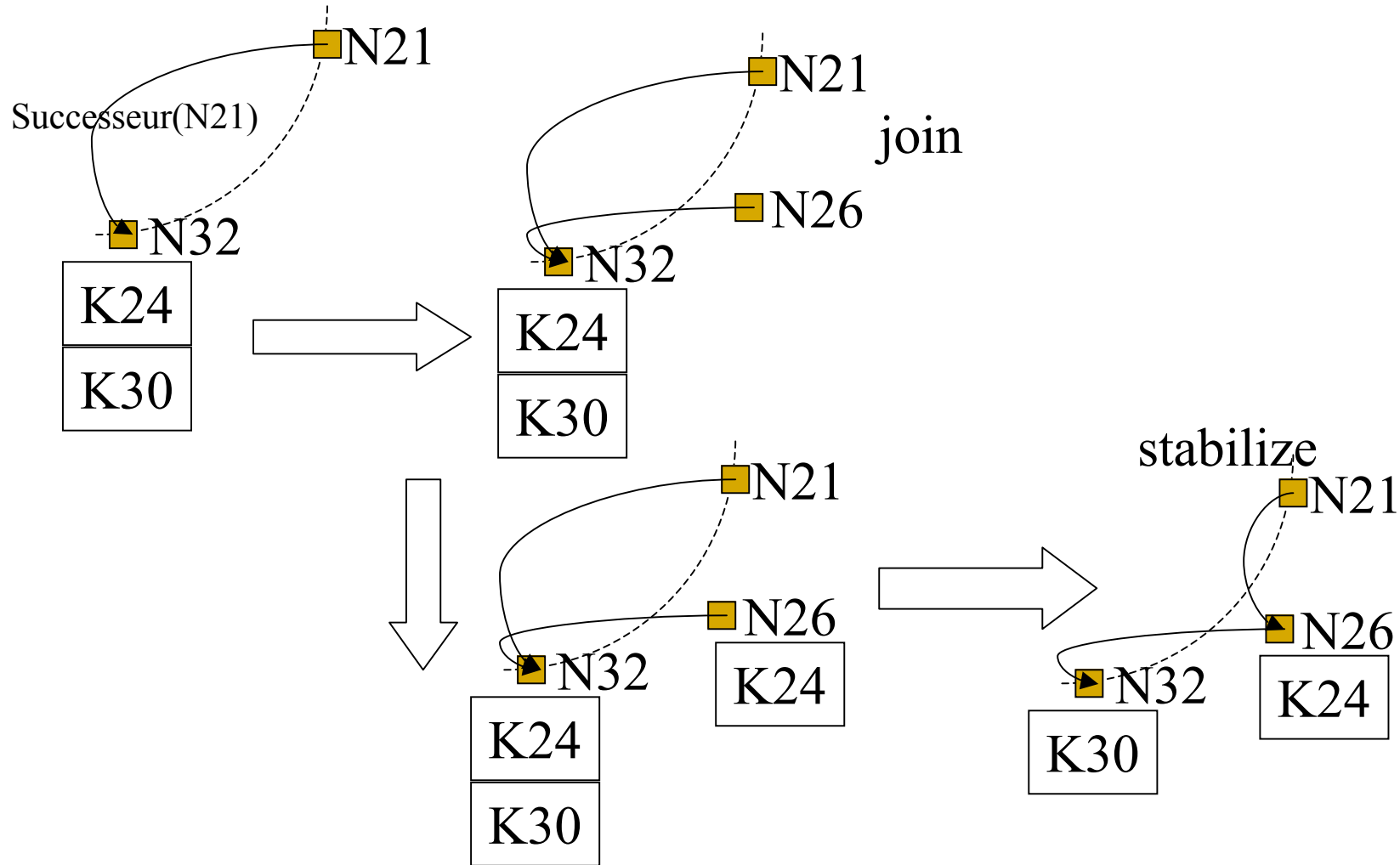
Algorithme de recherche

- Rechercher si la clé existe localement. Si oui on renvoie la valeur associée sinon
- Rechercher dans table routage nœud avec plus grande valeur inférieure ou égale à la clé cherchée
- Transmettre la requête au nœud sélectionné et appliquer récursivement
- Nombre de sauts moyen : $O(\log_2(N))$

Ajout d'un nœud

- Repose sur la combinaison d'opérations élémentaires
 - N.join(n') : nœud n annonce au nœud n' qu'il rentre dans le réseau et lui demande de lui fournir son successeur
 - N.stabilize() : lancé périodiquement, permet à n et à son successeur de vérifier qu'ils forment un couple correct (il n'y a pas de nouveaux nœuds entre les 2)
 - N.majentrées() : lancé périodiquement, permet d'initialiser la table de routage pour les nouveaux nœuds ou de la mettre à jour pour les nœuds existants
 - N.testpredecesseur() : lancé périodiquement, vérifie que le prédecesseur est toujours là

Exemple d'ajout N26



Propriétés de l'algorithme d'ajout

- L'algorithme garantit que n'importe quelle séquence de join entrelacée avec des stabilize converge vers un état stable (tous les nœuds restent atteignables)
- Même en présence d'un ajout (limité) de nœud, le coût de la recherche reste en $O(\log(N))$
- Une recherche peut échouer si le réseau n'est pas complètement stabilisé (il faut relancer la recherche un peu plus tard)

Sensibilité aux fautes

- Algorithme de recherche repose sur la notion de successeur (sensible à la panne de celui-ci)
- Une solution consiste à gérer une liste de r successeurs

Evaluation Chord

- Type de recherche : égalité
- Coût de la recherche : $O(\text{Log}(n))$
- Coût de la mise à jour : $O(\text{Log}(n))$
- Coût de l'ajout d'un nœud : $O(\text{Log}_2(n))$
- Pas d'autonomie de stockage et de routage

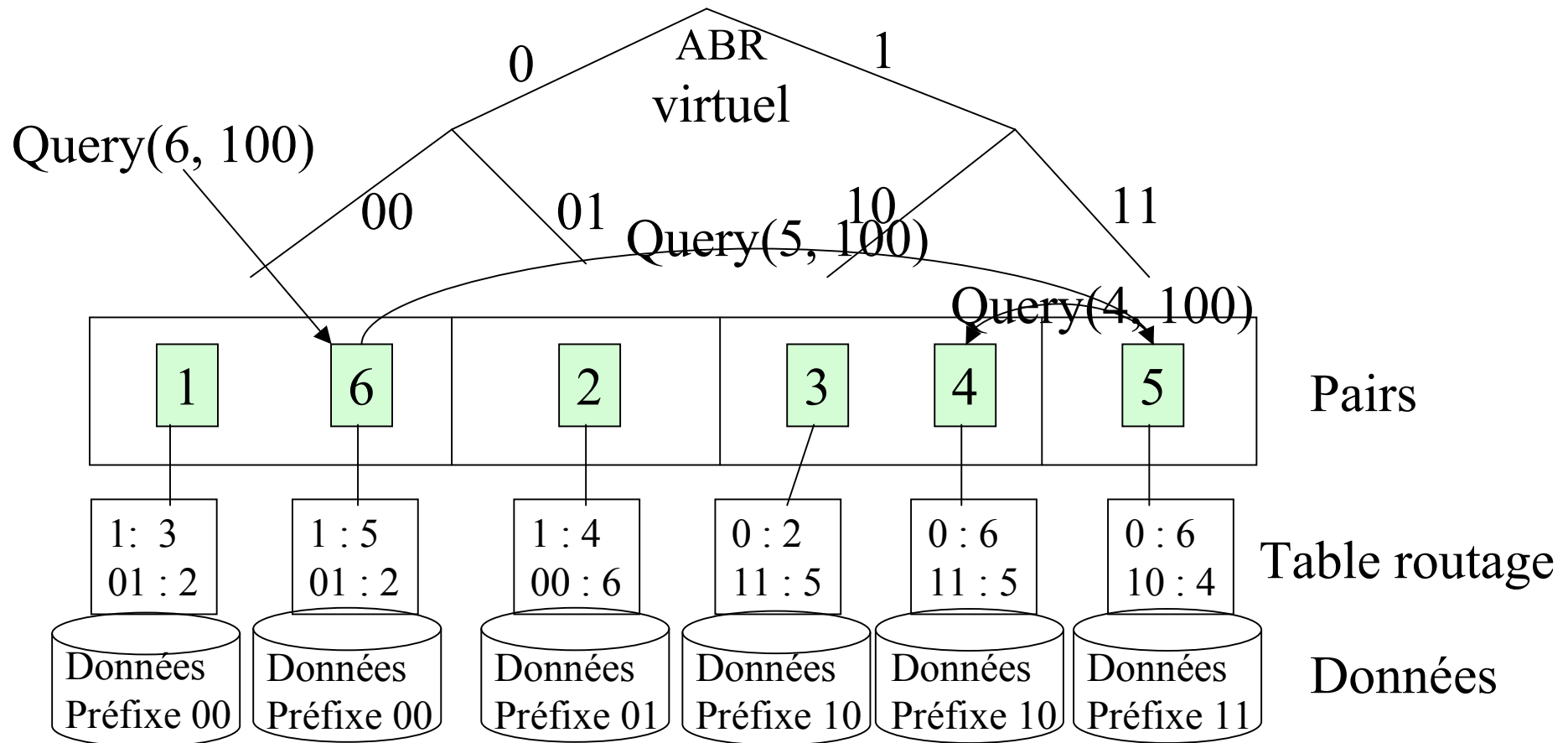
Bilan Chord

- Algorithme assez simple, avec de bonnes propriétés démontrables
- Résultats expérimentaux confirment
- Problème de latence :
 - Fonction de recherche minimise le nombre de sauts, mais tous les sauts n'ont pas forcément le même prix (traversée transatlantique e.g)
 - Besoin d'utiliser de l'information sur la distance entre les nœuds (on choisira parmi les successeurs possibles celui à distance minimale) -> Global Network Positioning

P-Grid [Aberer 01]

- Table de hachage distribuée
- Nœuds se répartissent les ressources selon une structure d'arbre binaire de recherche
- Repose sur une fonction de hachage préservant les préfixes et distribuant uniformément les ressources sur l'arbre binaire
- Structure auto-adaptative, avec réplication
- Permet les recherches basées sur les préfixes (pas seulement égalité stricte)

Exemple d'un P-Grid



Bilan de P-Grid

- Structure assez simple, tolérante aux fautes, avec passage à l'échelle
- Évite de structurer les nœuds selon une fonction de hachage
- Peut intégrer facilement des optimisations sur la latence (choix d'un pair)
- N'est pas restreint aux recherches basées sur l'égalité stricte

Autres approches de DHT

- Freenet [Clarke 01]
- CAN [Ratsanamy 01] : espace à n-dimensions
- Pastry [Rowstron 01] : hypercube

P2P « sémantique »

Routing Indices

P2P « sémantique »

- Ajouter de l'information aux tables de routage
- Généraliste vs spécifique (à une application)
- Information doit pouvoir être maintenue dynamiquement
- Équilibre entre taille et précision

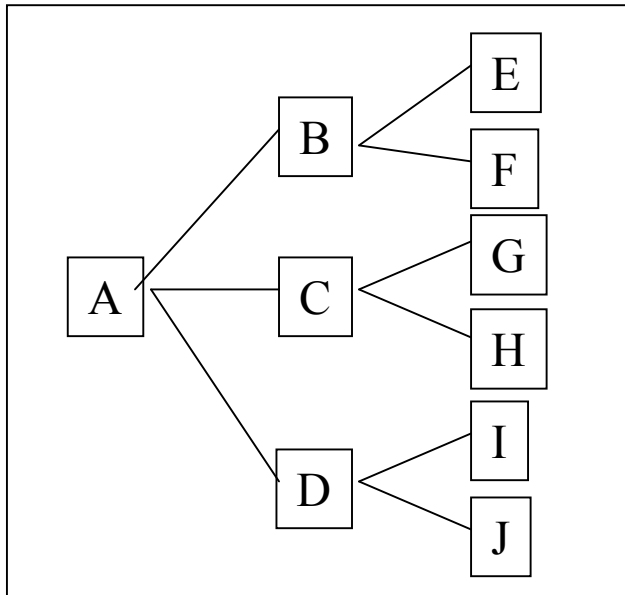
Quelle information ?

- Sur le contenu des nœuds (index)
 - Suppose que les nœuds aient un contenu homogène (pas de placement aléatoire)
 - Routage par comparaison requête-index
 - Même information partagée par tous, ou bien information construite localement
 - Exemple « routing indices »
- Sur le réseau (clustérisation)
 - Regrouper logiquement les nœuds avec même « sémantique »
- Sur les requêtes
 - Suppose qu'il y ait peu de requêtes différentes
 - Routage par comparaison requête-requête
- Sur les utilisateurs
 - Suppose que les utilisateurs aient toujours le même besoin
 - Routage par comparaison utilisateur-utilisateur

« Routing Indices » [Crespo 02]

- Introduire de l'information sur le contenu des nœuds (index)
- Analogue aux systèmes d'index répartis et hiérarchisés pour moteurs de recherche sur Internet
- Trouver l'équilibre entre la taille de l'index et le gain

Exemple de « RI »



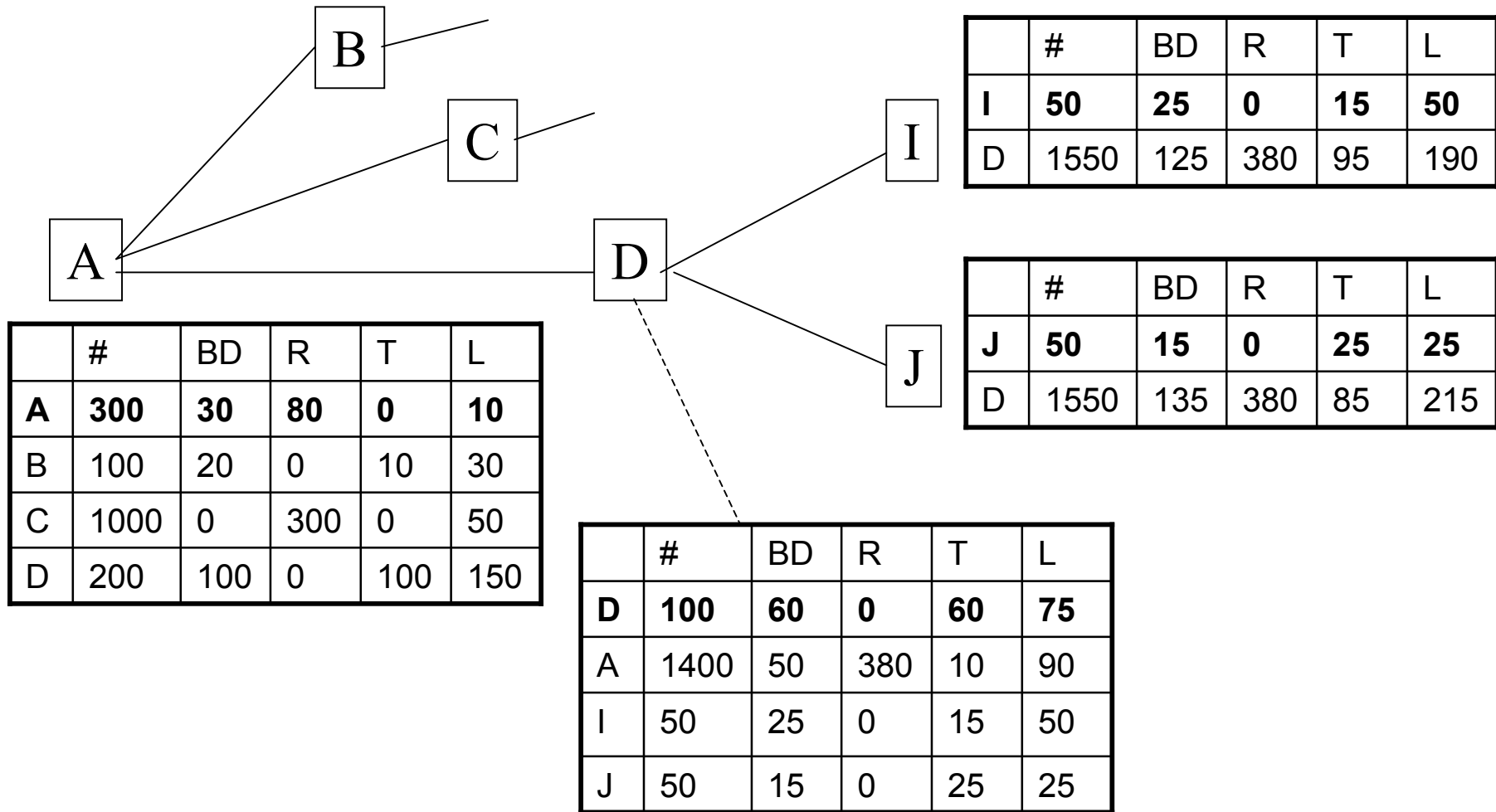
RI pour nœud A

Chemin	Nb doc.	BD	Réseaux	Théorie	langages
B	100	20	0	10	30
C	1000	0	300	0	50
D	200	100	0	100	150

Utilisation de l'index

- Soit Q une requête, conjonction de plusieurs termes de recherche (t_Q^1, \dots, t_Q^k)
- $\text{Proximité}(Q, \text{chemin}) = \text{Nbdocuments} \times \prod_i (\text{RI}(t_Q^i) / \text{Nbdocuments})$
- Q émise sur A = ('BD', 'langages')
- $\text{Proximité}(Q, B) = 100 \times 20/100 \times 30/100 = 6$
- $\text{Proximité}(Q, C) = 1000 \times 0/1000 \times 50/1000 = 0$
- $\text{Proximité}(Q, D) = 200 \times 100/200 \times 150/200 = 75$
- Permet d'ordonner les nœuds successeurs

Exemple complet de RI



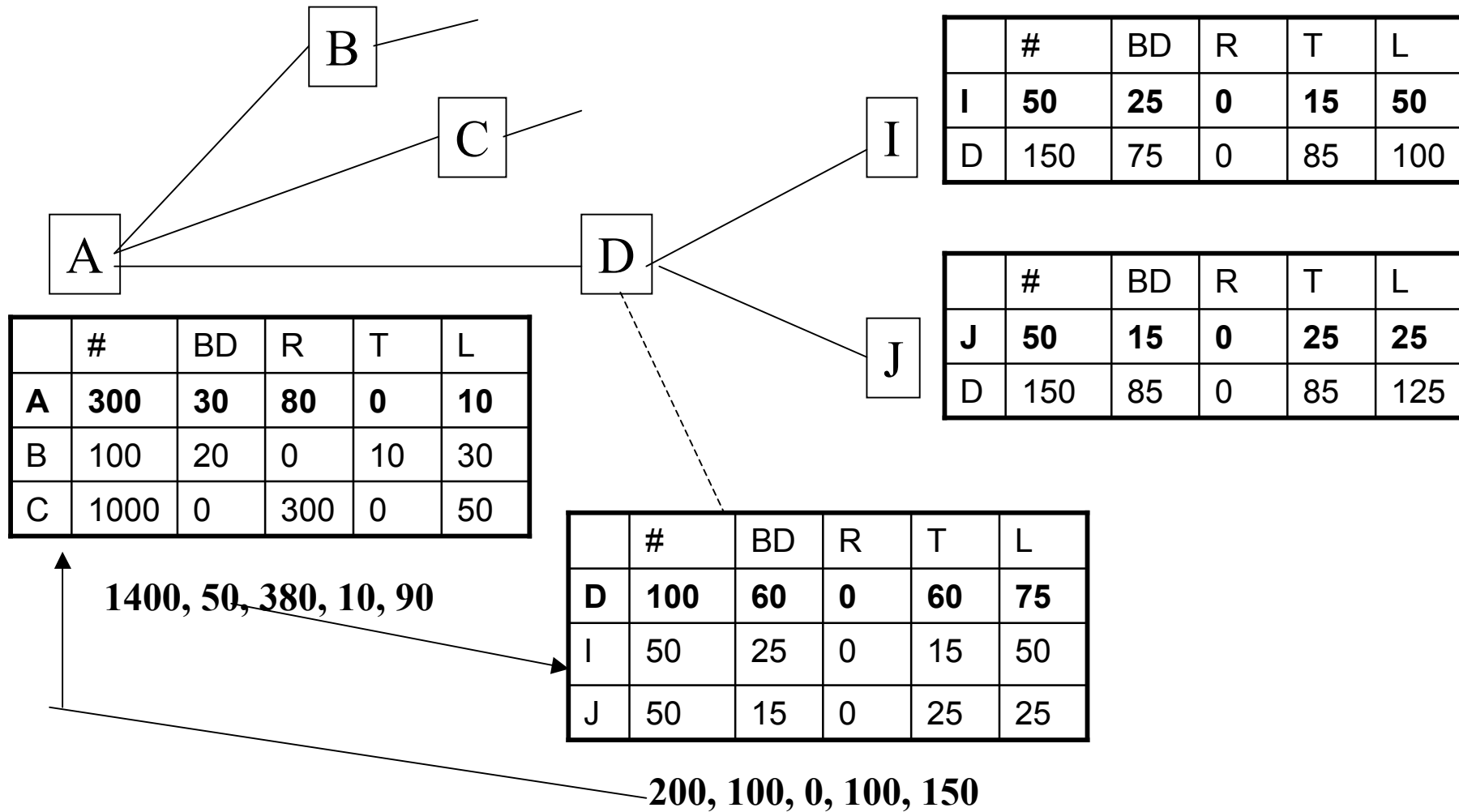
Algorithme de recherche

- Résoudre Q localement. Si suffisamment de résultats OK, sinon
- Tant qu'il n'y a pas assez de résultats
 - Évaluer proximité des successeurs
 - Prendre le successeur S le plus proche (et non encore exploré), si vide retour
 - Recherche(Q, S)

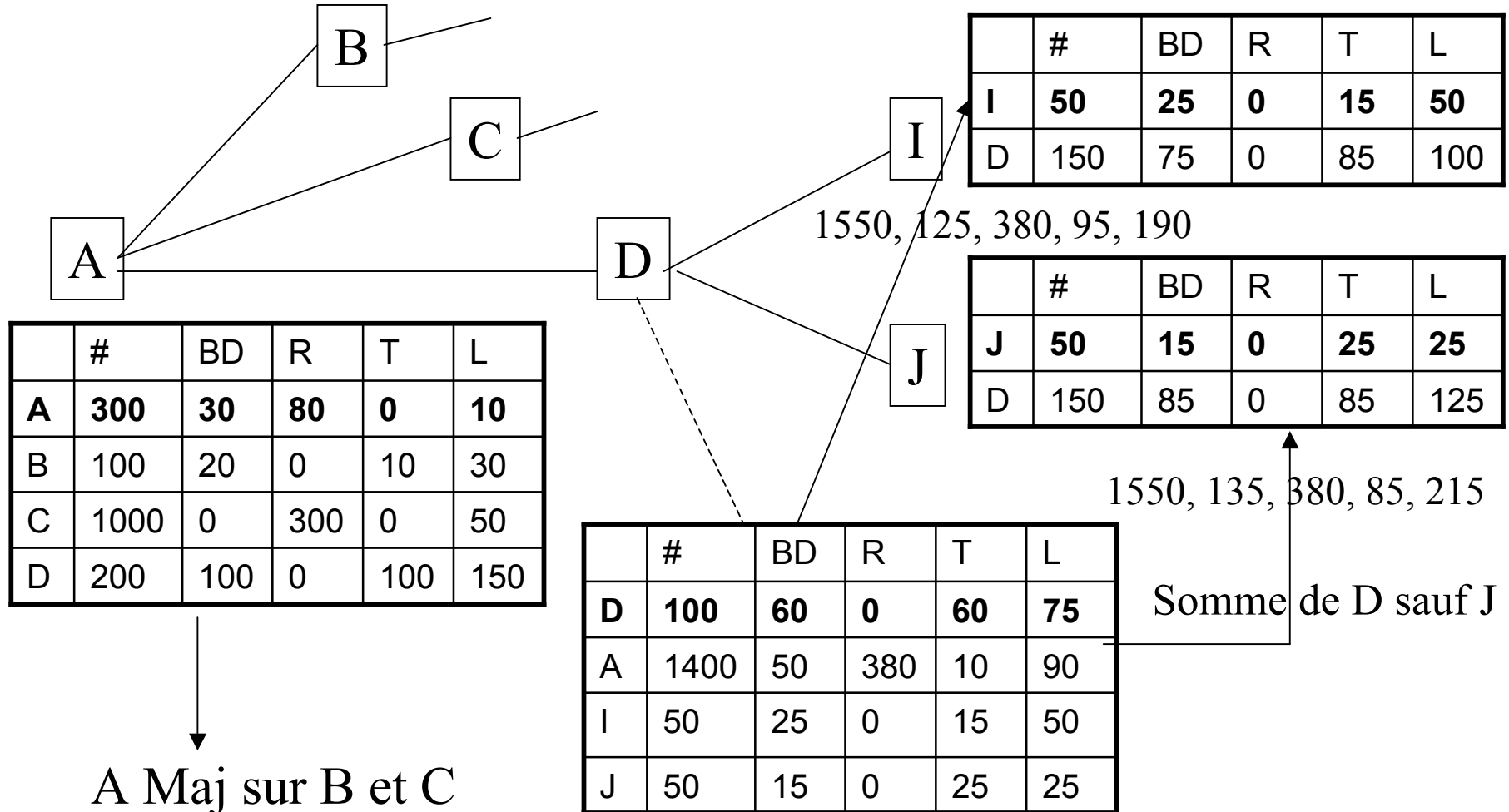
Performances de la recherche

- Par rapport à Gnutella diminue le nombre de messages
- Exploration restreinte aux nœuds ayant la plus grande probabilité de succès
- Pas d'information sur le nombre de sauts nécessaires (améliorations possibles avec d'autres RI)
- Pas de garantie d'avoir tous les résultats
- Plutôt orienté recherche des k meilleurs résultats

Création d'un RI (ajout de AD)



Création d'un RI (2)



Maj d'un RI

- Analogue au processus de création
- Pour diminuer le coût, on peut accumuler les maj et les traiter par lots
- Suppression d'un nœud suit le même principe

Bilan RI

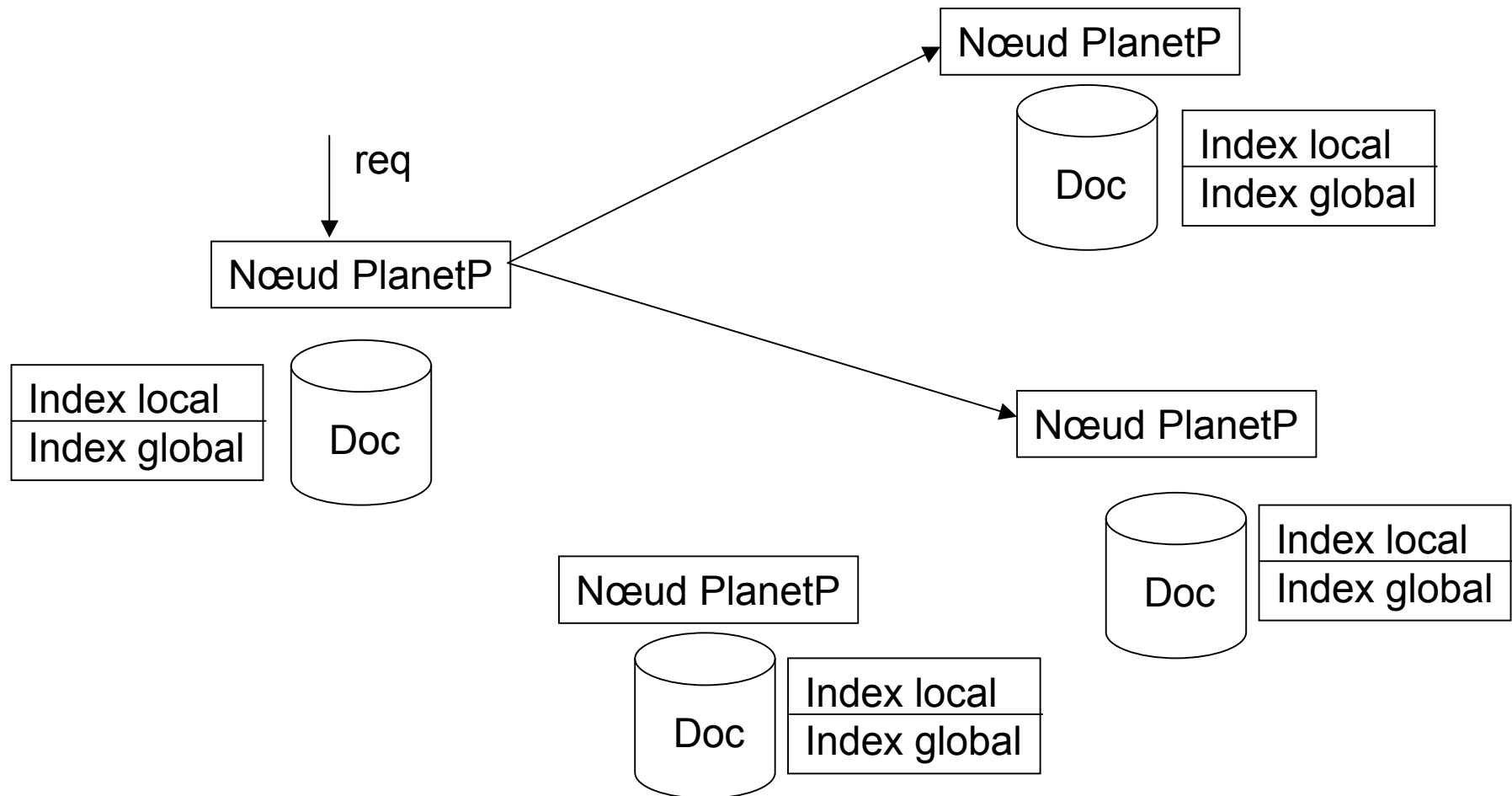
- Structure d'indexation assez simple, mais maj génère beaucoup de messages
- Fonctionne bien pour obtenir les **meilleurs** résultats, mais pas forcément **tous** les résultats
- Pour traiter des graphes généraux, il faut intégrer la gestion des cycles (détection ou prévention)
- S'applique à des langages types mots-clés mais pas généralisable à des langages plus complexes
- RI ne donne pas d'indications sur le nombre de sauts (d'autres RI sont proposées, e.g distinguer le nombre de sauts dans la RI)
- Classifier l'ensemble des nœuds via une classification « sémantique » (e.g genres de musique)

P2P et recherche d'information

PlanetP [Cuenca-Acuna 2003]

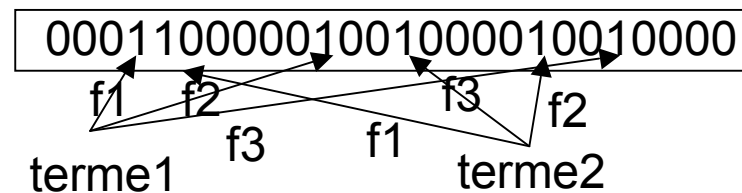
- Routage sémantique des requêtes via un index global du contenu des pairs
- Index est un Bloom filter
- Cohérence globale des index globaux assurée par un algorithme de rumeur (gossiping)

Principes de fonctionnement



Index

- Index local
 - Décrit le contenu des documents stockés sur le nœud
- Index global
 - Décrit le contenu de tous les nœuds (une entrée par nœud)
- Représentation compacte par filtre de Bloom



Gossiping PlanetP

- Algorithme de type épidémique
- Algo de base :
 - Toutes les T_g s, n envoie sa maj (rumeur) à un autre noeud m choisi au hasard (si m n'a pas déjà vu la rumeur il l'enregistre et la propage comme n)
 - N arrête ses envois au bout de x envois consécutifs sans enregistrement
 - Tous les T_r tours, x envoie une requête à un nœud y et lui demande le résumé de ses dernières modifications et le cas échéant récupère les maj manquantes
- Variante PlanetP :
 - Lors de l'envoi d'une rumeur par n , m répond avec x identifiants des rumeurs les plus récentes connues de m (mais plus propagées par m)

Maintenance dynamique du réseau

- Mise à jour du contenu d'un nœud : maj de l'index + propagation via l'algorithme de gossiping
- Entrée d'un nœud ou retour d'un nœud : gossiping
- Sortie temporaire d'un nœud : échec d'une communication donne un marquage à offline du nœud. Si au bout d'un timeout, le retour du nœud n'a pas été notifié via gossiping le nœud est considéré comme parti

Recherche d'informations PlanetP

- Vector Space Model adapté (IPF remplace IDF)
- Classement des nœuds :
 - $IPF_t = \log(1 + N/N_t)$ avec N nombre total de nœuds et N_t nombre de nœuds avec au moins un document contenant t
 - $Classement(Q, n) = \sum IPF_t$ pour tous les t présents dans n
- ✓ Algo de recherche : Q, k
 - ⊖ Prendre les m premiers nœuds et leur envoyer Q
 - ⊖ Récupérer les résultats classés selon VSM et garder les k meilleurs
 - ⊖ Itérer sur m autres nœuds et s'arrêter lorsque p nœuds consécutifs ne contribuent pas aux k meilleurs résultats

Résultats expérimentaux

- Comparaison PlanetP avec évaluation centralisée montre des résultats comparables
- Montée en charge sans problème jusqu'à 1000 nœuds
- Dynamique du réseau bonne si connexions de type DSL

Bilan PlanetP

- Idée assez simple et semble marcher sur des réseaux de taille moyenne
- S'adapte à l'aspect dynamique du réseau
- Nécessite la connaissance globale de tous les nœuds ainsi qu'une taille d'index fixe qq soit le nombre de documents dans le nœud

Recherche avec expansion de requêtes

[Nakauchi 2003]

- Travaille sur des méta-données associées à des documents multimédia
- Intègre de l'expansion de requêtes par une base de connaissances locale à chaque pair
- Utilise du feedback pour mettre à jour les BD locales

Bases de connaissances locales

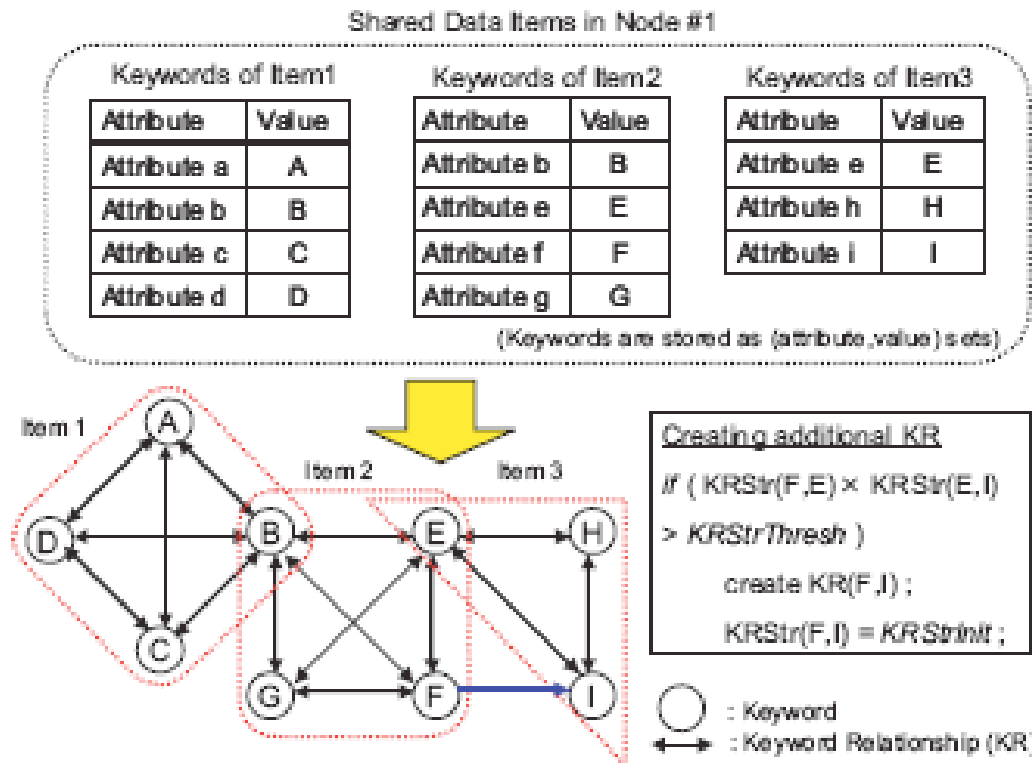


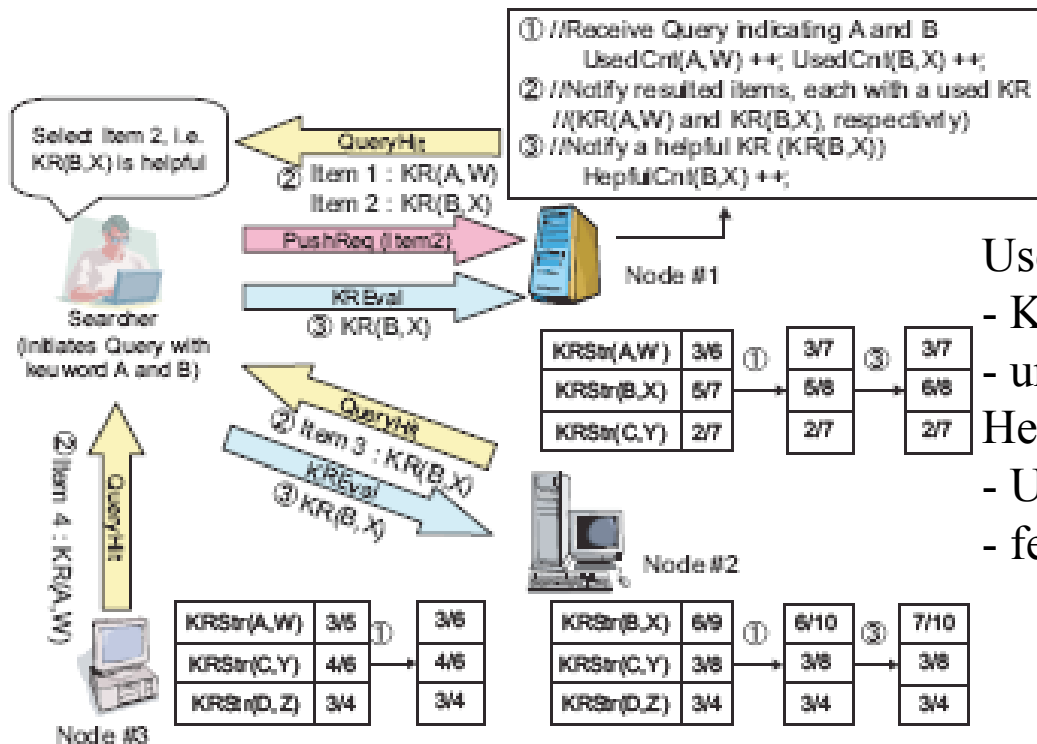
Figure 1. Creating Keyword Relationships from Local Data Items

- Dérivation de nouveau KR par transitivité (seuil)
- Valeur initiale à 0.5 de la force de l'association

Principes de maj des BC locales

- Besoin d'affiner la valeur des KRStr locaux en fonction de leur utilité dans la recherche : mécanisme de feedback
- Besoin d'augmenter la richesse des BC locales (plus de KR) : mécanisme de synchronisation

Mise à jour des BC locales (1)

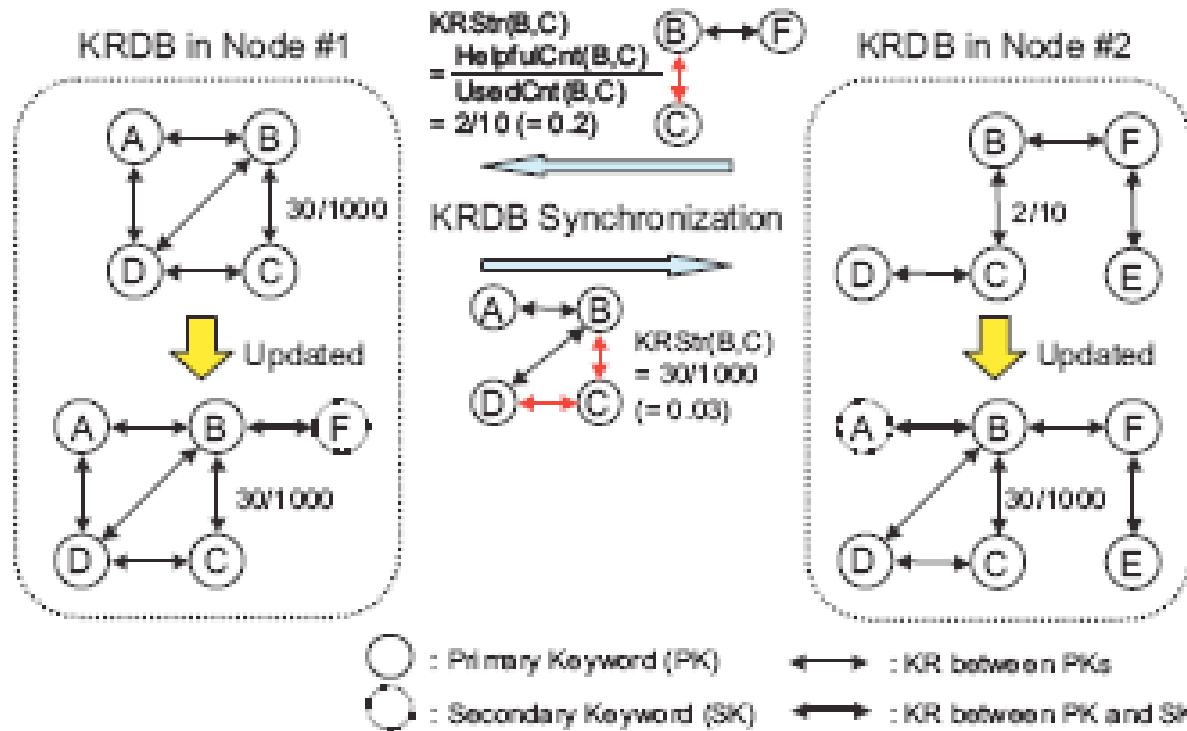


- UsedCnt(k_i, k_j) augmente si
- KR(k_i, k_j) utilisé dans l'expansion
 - une réponse contenant k_j a été trouvée
- HelpfulCnt(k_i, k_j) augmente si
- UsedCnt(k_i, k_j) augmente
 - feedback utilisation sur réponse contenant k

$$KRStr(k_i, k_j) = \frac{HelpfulCnt(k_i, k_j)}{UsedCnt(k_i, k_j)}$$

Figure 3. KRDB Update by Evaluation Feedback

Mise à jour des BC locales (2)



Ajoute (PK, PK), (PK, SK)
 (SK, PK) mais pas (SK, SK)
 Renforce poids de relation existante

Figure 4. KRDB Synchronization

PK = mots clés locaux
 SK = mots clés obtenus par synchronisation
 Deux nœuds sont synchronisés
 s'ils partagent suffisamment de PKs

Bilan

- Montre l'intégration de techniques évoluées de RI dans un système P2P
- Idée d'utilisation du feedback pour mettre à jour les BC locales

Gestion de la confiance dans les systèmes P2P

Fonctionnalités d'un système de confiance

Acquisition d'information	Valuation et classement	Réponse
Gestion de l'identité	Bon vs mauvais comportement	Incitation (+)
Sources d'info.	Quantité vs qualité	Punition (-)
Agrégation d'info.	Pondération temporelle	
Gestion des inconnus	Seuil de sélection	
	Sélection d'un pair	

Les menaces

- Selfish peer : utilise sans contribution équivalente
- Traitor peer : commence par bien se comporter avant de changer (eBay)
- Collusion entre pairs : pairs qui s'entendent pour « attaquer » le système
- Front peers : forme de collusion où les pairs coopèrent entre eux pour augmenter leur réputation et/ou baisser la réputation des bons pairs
- Whitewasher peer : pair changeant d'identité lorsque sa réputation devient mauvaise
- Denial of service : demande forte de ressources pour « écrouler » le système

Gestion des identités

- pas d'anonymat (IP sont visibles Gnutella, KaZaa),
- simples pseudonymes (KaZaa),
- identification par sa clé publique (besoin d'une autorité de confiance pour éviter les attaques MITM),
- identification non falsifiable (unforgeable) mais nécessite une autorité centrale de confiance (sinon trop coûteux et sans garantie)

Sources d'information

- Informations locales (rarement suffisant)
- Informations venant de sources sûres (amis, relations, ...) : également insuffisant
- Utiliser les sources voisines + la transitivité
- Utiliser une histoire globale

Validité de l'information

- Donner plus de poids à mon opinion vs opinion des sources sûres vs opinion des autres
- Valuer l'opinion d'un pair par sa réputation
- Avoir la preuve de l'existence des transactions (TrustMe) pour limiter la fraude
- Politique de gestion des étrangers (local ou global)
 - Optimiste, pessimiste, probabiliste (par rapport aux expériences récentes)

Valuation et classement

- Objectifs :
 - Décider si une transaction avec un pair x est trop risquée ou non
 - Choisir parmi plusieurs pairs remplissant le même service, celui (ceux) les plus dignes de confiance
- Entrées de la fonction de valuation
 - Nombre de succès et d'échecs (généralement pas connu surtout pour les selfish peers)
 - La qualité est aussi importante que la quantité (échec sur une transaction à 100^E plus grave que 3 échecs à 1^E)
 - La distribution dans le temps des succès et des échecs est aussi importante

Valuation et classement (2)

- Sorties
 - Intéressant de maintenir différentes statistiques sur un pair (e.g reliability and credibility)
- Sélection d'un pair
 - Définition d'un seuil

Actions prises

■ Incitations

- ❑ Améliorer la vitesse de téléchargement
- ❑ améliorer la qualité (vidéo par exemple)
- ❑ augmenter la quantité
- ❑ diminuer le coût

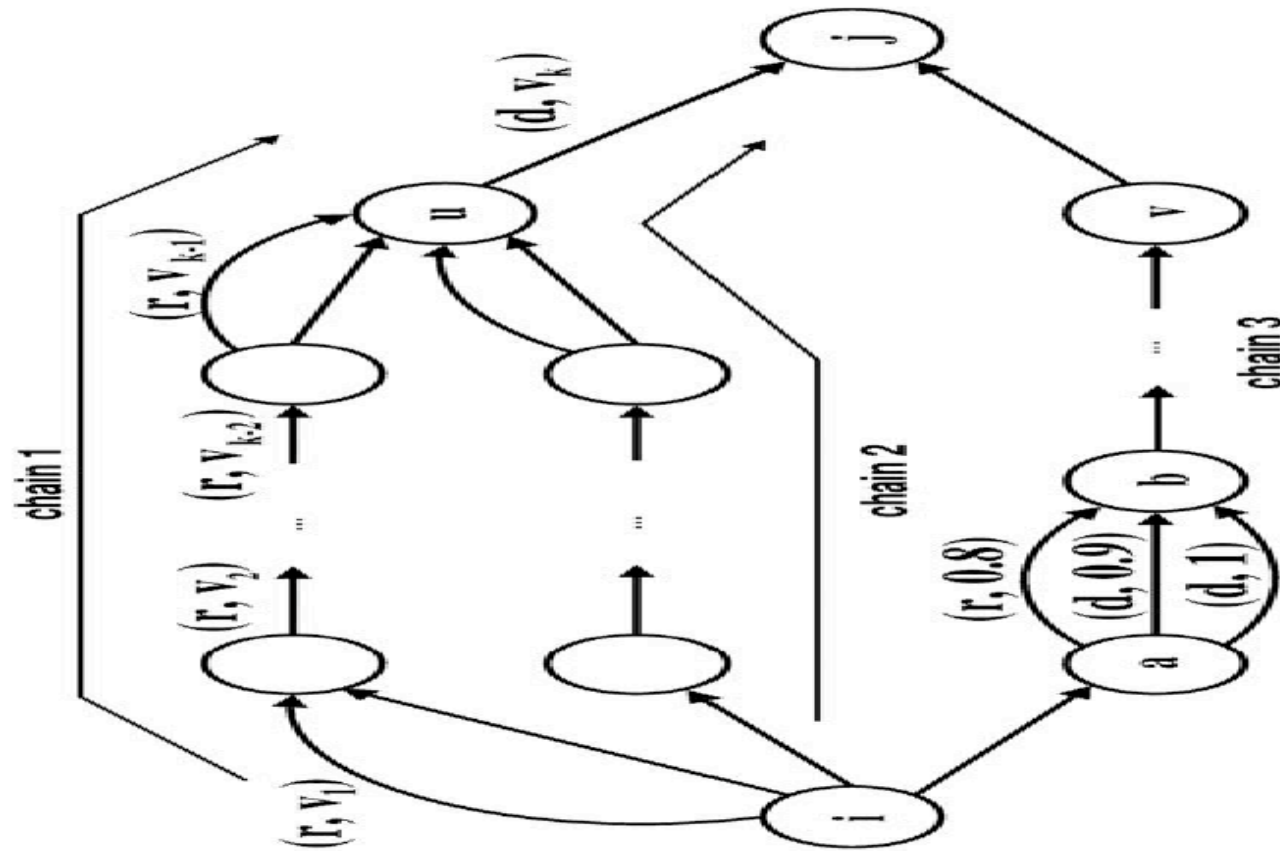
■ Punitions

- ❑ Sortir le « mauvais » pair du réseau

Définition générale d'un système de réputation

- Un système P2P de réputation est un quadruplet (G, W, A, T)
 - G est un multigraphe orienté et pondéré (P, V)
 - P est un ensemble de pairs
 - V est un ensemble d'arcs pondérés selon une valeur prise dans W
 - A est un algorithme qui opère sur G et fournit une valeur t prise dans T pour chaque p pris dans P

Multi-graphe de confiance



Définition du problème

- Un système de réputation doit
 - ❑ Définir une notion de feedback émis sur les partenaires à la suite d'une interaction (W)
 - ❑ Définir un algorithme d'agrégation des feedbacks
 - ❑ Produire en sortie une valeur de réputation encourageant le bon comportement des pairs
- Points cruciaux
 - ❑ Définir la sémantique de la confiance
 - ❑ Coût d'implémentation

Modèles de confiance

- Approche réseaux sociaux
- Approche probabiliste
- Approche théorie des jeux

Conclusion sur gestion de la confiance

- Thématique importante mais encore jeune
- Modèles économiques semblent prometteurs
- Intégration avec sécurité

Tentative de Synthèse

Choix P2P, problèmes ouverts

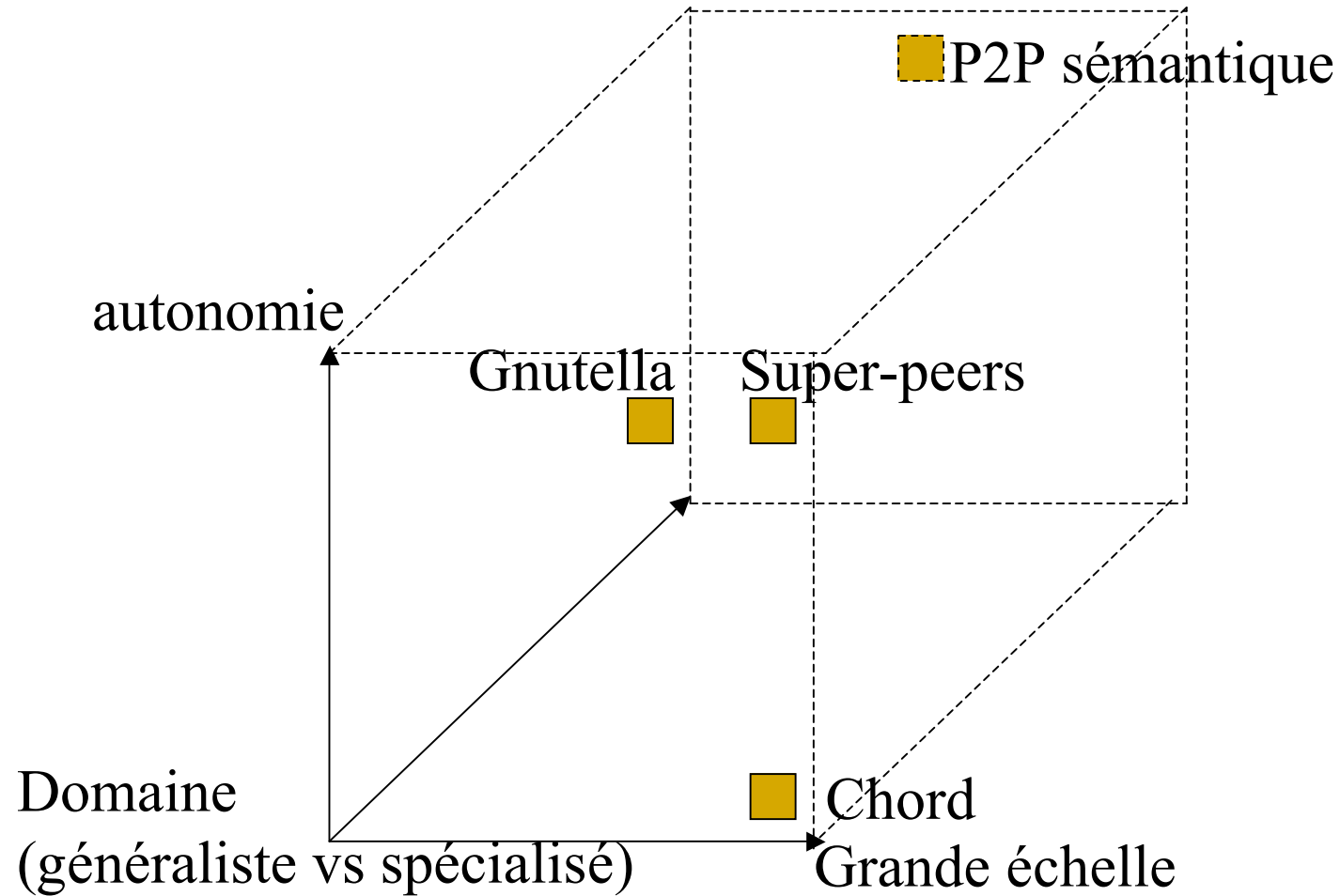
Besoins applicatifs

	Type recherche	résultats	Garantie	Autonomie stockage	Autonomie routage
Gnutella	filtre	Tous?	Non	Oui	Oui
Chord	égalité	Un	oui	non	non
P-Grid	préfixe	Un	oui	Faible	non
Freenet	égalité	Un	Non	Oui	O/N
Super-peer	filtre	K meilleurs	non	Oui	Oui pour SP, non pour autres
sémantique	filtre	K meilleurs	non	Oui	oui

Comparatif performances

	Modèle recherche	Coût recherche (messages)	ajout nœud	MAJ table routage
Gnutella	largeur	$2 * \sum_{i=0}^{TTL} C * (C-1)^i$	Init. routage	Périodique (ping)
Chord	Arbre binaire recherche	$O(\text{Log}(n))$	Maj routage, échange ressources	Périodique (stabilize)
P-Grid	Arbre binaire préfixe	$O(\text{Log}-n)$	Maj routage, échange ressources	rencontres
Freenet	profondeur	$O(\text{Log}(n)) ?$	Init. routage	Pendant recherche et insertion

Taxonomie



Problèmes ouverts

- Sécurité
- réputation
- Contrôler le « free riding »
- P2P sémantique
 - On en est au début
 - Mixer différents types de connaissances
 - Connaissances dynamiques (apprentissage)

Sécurité

- Disponibilité (résistance aux attaques DoS)
- Authentification des ressources (si plusieurs réponses différentes pour la même ressource)
 - Plus vieille ressource
 - Réputation (« experts »)
 - Vote
- Anonymat
 - Auteur : quels utilisateurs ont créé quels documents?
 - Document : sur quels nœuds est stocké un document donné?
 - Lecteur : quels utilisateurs accèdent quels documents?
 - Serveur : quels documents sont stockés sur un nœud donné?

Bibliographie (générale)

- K. Aberer, M. Hauswirth. Peer-to-Peer Information Systems: Concepts and Models, state-of-the-art, and Future Systems, Tutorial IEEE ICDE, 2002
- E. Adar, B. Huberman. Free Riding on Gnutella, Technical Report, Xerox PARC, septembre 2000
- I. Clarke et al. Freenet: A Distributed Anonymous Information Storage and Retrieval System, LNCS 2009, Springer Verlag, 2001
- L. Gong. JXTA: A Network Programming Environment, IEEE Internet Computing, 5(3), mai-juin 2001
- Gnutella : <http://www.gnutella.com>
- M.A. Jovanovic et al Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella, Research report, Univ. Cincinnati, 2001
- J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective, Technical Report 99-1776, Cornell Univ., 1999
- J. Kubiawicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage, Proc. ASPLOS, 2000
- S. Saroiu, P. Gummadi, S. Gribble. A Measurement Study of Peer-to-Peer Sharing Systems, Proc. Multimedia Computing and Networking, 2002
- K. Sripanidkulchai. The Popularity of Gnutella Queries and its Implications on Scalability, février 2001
- B. Yang, H. Garcia-Molina. Improving Search in Peer-to-Peer Systems, Proc. 28th Conf. On Distributed Computing Systems, 2002

Bibliographie (DHT et super-pairs)

- K. Aberer et al. Improving Data Access in P2P Systems, IEEE Internet Computing, January 2002
- K. Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems, Proc. COOPIS, 2001
- A. Rowstron, P. Dreschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, Proc. IFIP/ACM Conf. On Distributed Systems Platforms, 2001
- I. Stoica et al. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, Proc. ACM SIGCOMM 2001
- S. Ratsanamy et al. A Scalable Content Addressable Network, Proc. ACM SIGCOMM 2001
- B. Yang, H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems, Proc. VLDB Conference, 2002
- B. Yang, H. Garcia-Molina. Designing a Super-Peer Network, Proc. ICDE Conf., 2003

Bibliographie (P2P et RI)

- A. Crespo, H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems, Proc. ICDCS 2002
- F.M Cuenca Acuna, C. Peery, R.P Martin, T.D Nguyen. PlanetP: Using Gossiping to Build Content Addressable P2P Information Sharing Communities, Proc. 12th IEEE Int. Symp. On HPDC, 2003
- P. Haase, R. Siebes. Peer Selection in Peer-to-Peer Networks with Semantic Topologies, Proc. WWW Conf., 2004
- K. Nakauchi et al. Peer-to-Peer Keyword search using Keyword RelationShip, Proc. Third International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC 2003), Tokyo, Japan, May 2003
- H. T Shen, B. Yu. Efficient Semantic-Based Content Search in P2P Network, IEEE TKDE 16(7), 2004
- C. Tang, Z. Xu, S. Dwardakas. P2P Information Retrieval Using Self-Organizing Semantic Overlay Networks, Proc. ACM SIGCOMM, 2003