CENTRE D'INNOVATION
EN TELECOMMUNICATION
& INTEGRATION DE SERVICES

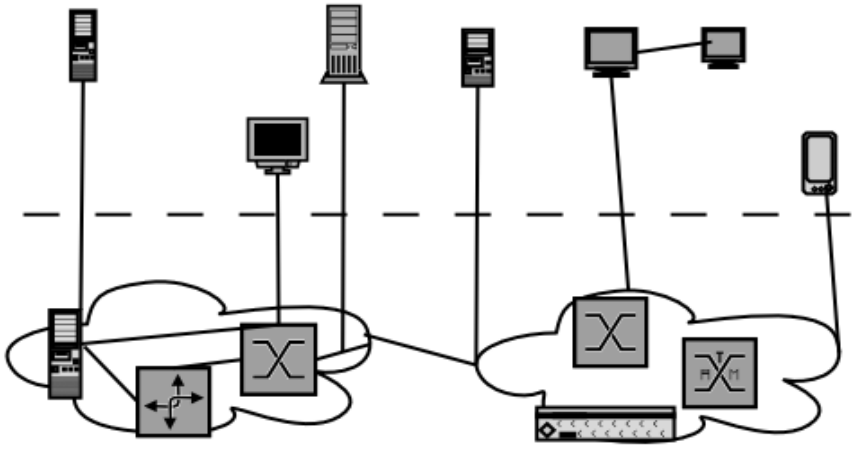
Réseaux P2P

Stéphane Frénot
stephane.frenot@insa-lyon.fr

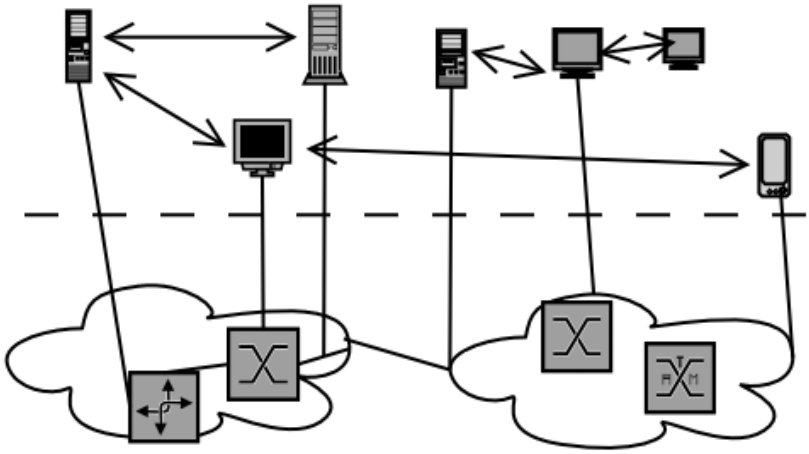
Materiel initial

- *Jon Crowcroft*, "Peer-peer and Application-Level Networking"*, Cambridge*
- *Karl Aberer, Manfred Hauswirth*, "Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems" EPFL-DSC, TU-Wien DSG
- *Dejan et al.,* "Peer-to-Peer Computing", HP Laboratories Palo Alto, 2002

Réseaux « classiques »

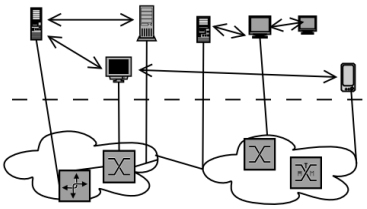


Réseaux Pair à Pair



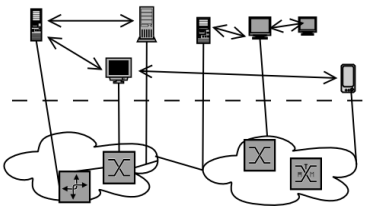
Réseaux Pair à Pair

- Applications
  - Naptser, Gnutella, Freenet : partage de fichiers
  - Réseaux Ad-Hoc
  - Les Overlays Multicast : distribution vidéo



Réseaux Pair à Pair

- Questions
  - Quels sont les nouveaux challenges techniques ?
  - Quels sont les nouveaux services/applications ?
  - Est-ce simplement mettre du réseau au niveau de l'application ?



Plan du cours

- Introduction
- Client/Serveur vs. P2P
- Etude de cas
  - Napster, Gnutella, Freenet, Bittorrent
- Algorithmes de localisation
  - Chord, Tapestry, CAN


Buts

- Partage/Réduction des coûts
  - Napster, SETI@home
- Amélioration du passage à l'échelle/tolérance
  - Pas d'administration centrale --> Algo plus robustes (ex DNS)
- Agrégation de ressources et Interopérabilité
  - cpu : SETI@Home, Distributed.net, Endeavours
  - filespace : gnutella, napster
- Augmentation de l'autonomie
  - napster, freenet
- Anonymats/zone privée
  - Freenet, Publius
- Dynamisme, apparition / disparition de nœuds
  - IM
- Communication ad-hoc et collaboration
  - Pas d'infrastructure fixe


Client Server vs. P2P

- Gérés
- Configurés
- Recherche
- Hiérarchique
- Statique
- Cycle de vie lié au serveur
- Centré IP
- Basé sur le DNS
- RPC/RMI
- Synchrone
- Asymétrique
- Axé sur des modèles de liaison et d'intégration du langage de programmation (stub IDL/XDR, compilateurs, etc…)
- Sécurité de type Kerberos : acl, crypto

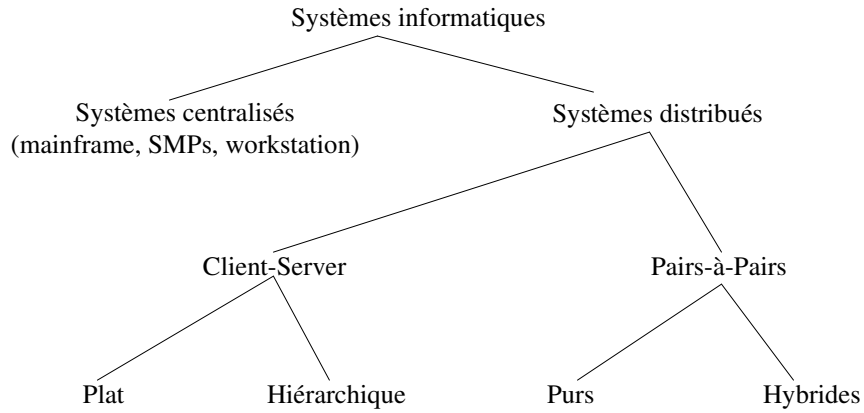
- Auto-Gérés
- Ad-hoc
- Découverte
- Maillage
- Mobiles
- Cycle de vie autonome
- Non restrictif à IP
- Nommage spécifique
- Messages
- Asynchrone
- Symétrique
- Axé sur la localisation de services, localisation du contenu, routage applicatif
- Anonyme, haute-disponible, intégrité
- Plus difficile à dominer


Le client/Server vs P2P

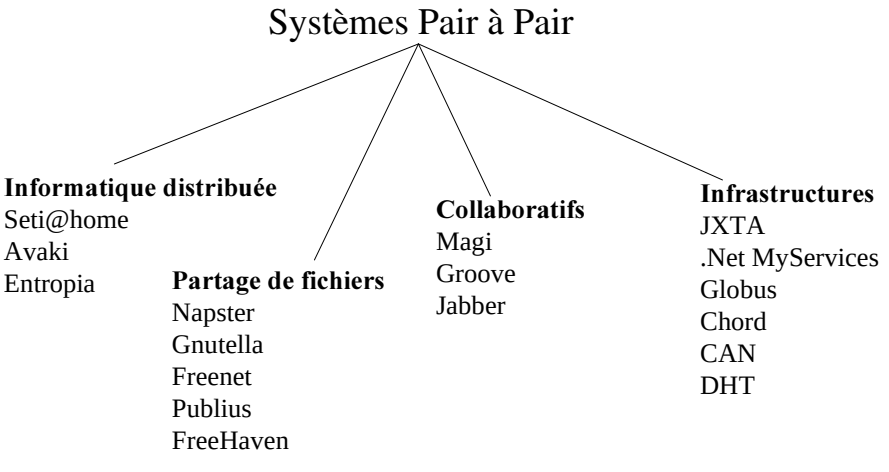
```
Srv_main_loop(){
  while(true){
    deque(call)
    switch(call.procid)
    case0:
      call.ret=proc1(call.args)
    case1:
      call.ret=proc2(call.args)
    …
    default:
      call.ret=exception
   }
}
```

```
Peer_main_loop(){
  while(true){
    attendre(event)
    switch(event.type)
    case timer_expire:
      faire une activité p2p();
      regénérer timer;
      break;
   case inbound message:
      gérermessage;
      répondre;
      break;
    default:
      ne rien faire
   }
}
```

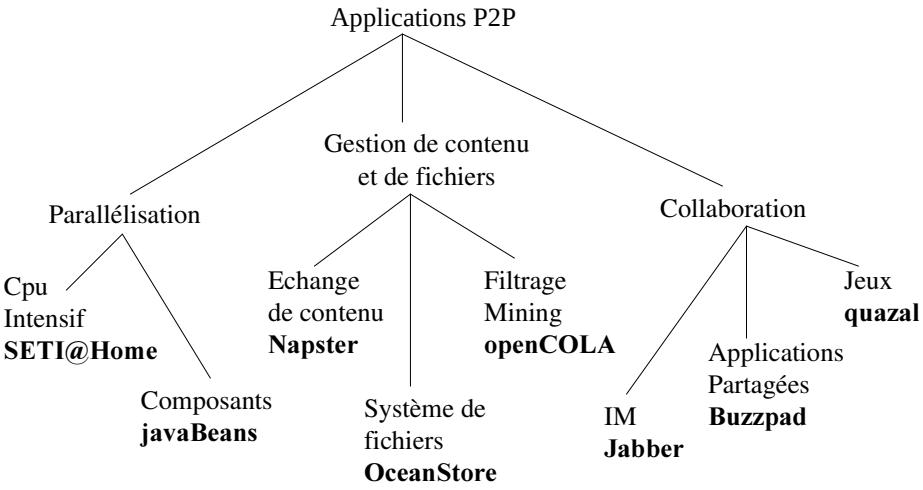

Taxonomie des Systèmes




Taxonomie des systèmes P2P




Taxonomie des applications P2P



Les routeurs IP sont p2p

- Ils découvrent une topologie et la maintiennent
- Ne sont ni client ni serveurs
- Dialoguent continuellement entre eux
- Sont tolérant aux pannes
- Sont autonomes

Réseau Ad Hoc et P2P

- Pas de connaissance a priori de la topologie
- Pas d'infrastructure de base
- Fabrication à partir d'un minimum d'information


Un système p2p c'est

- Un système qui n'a pas de rôle prédéfini
- Pas de point d'engorgement ou de panne
- Cependant, ils leur faut des algo distribués pour :
  - Découverte de service (nom, adresse, route, métrique, etc…)
  - Recherche de voisins
  - Routage de niveau applicatif
  - Rémanence, récupération sur faute de liaison ou d'exécution


Etudes de cas : partage de fichiers

- Les classiques
  - Napster -->Index Centralisé
  - Gnutella -->Innondation
  - Freenet -->Routage documentaire
  - Chord, Tapestry, Can -->Routage documentaire optimisé
  - BitTorrent --> Transport optimisé


Napster

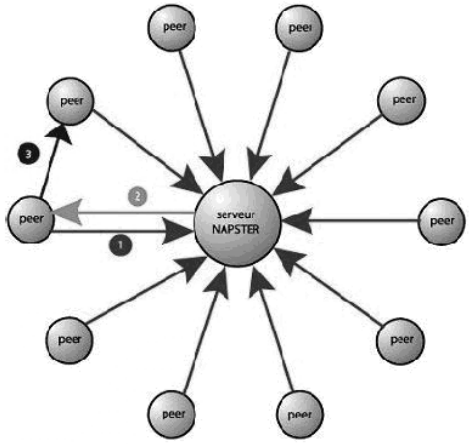
- Le plus connu
- Pas le premier (Eternity, Ross Anderson, Cambridge)
- Source d'inspiration, pour le bien et le moins bien
- Egalement un message politique, économique, légal...


Napster

- Programme de partage de fichiers sur Internet
- histoire :
  - 5/99 : Shawn Fanning, Boston (aka napster) fonde Napster Online music service
  - 12/99 : premier procès
  - 03/00 : 25% du traffic de l'université du Wisconsin
  - 07/01 : 160k utilisateurs de napster, 40k Gnutella


Napster principe

- Protocole client/serveur point à point au niveau applicatif !!
- Quatre étapes :
  - Connexion au serveur napster
  - Chargement de sa liste de fichiers sur le serveur (push)
  - Envoyer ses critères de recherche sur la liste principale
  - Sélectionner la meilleur réponse (pings)
  - Récupérer le fichier

Messages napster

- [chunkSize] [ChunkInfo] [data…]
- ChunkSize
  - Intel-endian 16bit integer
  - Size of [data] in bytes
- ChunkInfo: (hex)
  - Intel-endian 16-bit integer
  - 00 -- login rejected
  - 02 -- login requested
  - 03 -- login accepted
  - 0D -- challenge? (nuprin 17 15)
  - 2D -- added to hotlist
  - 2E -- browse error
  - 2F -- user offline

5b -- whois query
5c -- whois result
5d -- whois: user is offline!
69 -- list all channels
6a -- channel info
90 -- join channel
91 -- leave channel
...

Napster récupération de fichier

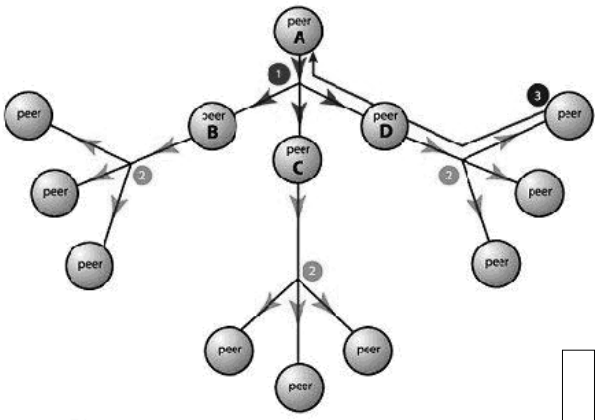
- Envoyé à napster (après connexion)
  2A 00 CB 00 toto "c:\mp3\Rammstein -- Du Hast.mp3"
- Reçu de napster
  5D 00 CC 00 toto
  2965119704 (forme ip = a.b.c.d)
  6699 (port)
  "c:\mp3\Rammstein -- Du Hast.mp3" (morceau)
  (32-byte checksum)
  (vitesse ligne)
  [connexion à a.b.c.d]
- Reçu du client
  31 00 00 00 00 00
- Envoyé au client
  GET
- Reçu du client
  00 00 00 00 00 00
- Envoyé au client
  toto
  " c:\mp3\Rammstein -- Du Hast.mp3"
  0 (port où se connecter)
- Reçu du client
  (taille en octets)
- Envoyé à napster
  00 00 DD 00 (notification d 'ok)
- Reçu du client
  (données)

Napster : commentaires

- Serveur centralisé
  - Point de panne
  - Peut équilibrer la charge sur la rotation DNS
  - Congestion
  - Contrôle de napster (fausse libertée)
- Pas de sécurité
  - Mots de passe en clair
  - Pas d'authentification
  - Pas d'annonymisation

Le P2P décentralisé

Un peer **A**, équipé d'un programme spécifique, se connecte à un peer **B**, lui aussi équipé de ce programme (point 1). **A** lui annonce ainsi sa présence sur le réseau. **B** relaie cette information à tous les peers auxquels il est connecté (point 2).



Ceux-ci signifient alors leur présence à **A** et relaient l'information à leur tour aux peers auxquels ils sont connectés, et ainsi de suite. Le fichier recherché est localisé et une réponse est envoyée à **A**. **A** peut alors directement télécharger le fichier via une connexion http (point 3).

Source: www.Zdnet.fr

Centralisé vs décentralisé

+ Utilisation d'un **index central** => localisation rapide

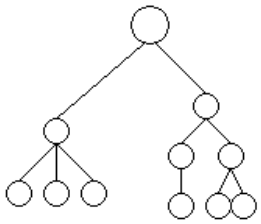
- Existence d'un **seul point d'entrée** sur le réseau => risque d'arrêt du système (défaillances techniques ou décisions judiciaires)

+ Aucune dépendance à un serveur => **robustesse** du système
+ Il tire partie de l'intermittence des connexions des nœuds => les requêtes sont poursuivies vers d'autres ordinateurs si l'un est défaillant.
- La bande passante nécessaire pour chaque requête **croît exponentiellement** quand le nombre de peers croît linéairement => risque d'inefficacité

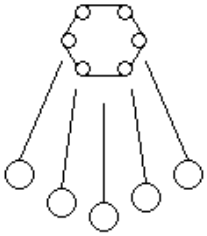
Architectures hybrides

- Des modèles hybrides apparaissent afin de tirer parti des points forts des deux modèles précédents, **robustesse et rapidité des requêtes**. Le nombre de combinaisons est assez vaste, en voici deux exemples:

Architecture hiérarchique



Architecture en anneau



L'effet Napster

Depuis Napster, une **multitude de programmes** d'échange de fichiers (certains généralistes et d'autres spécialisés dans certains types de fichiers) ont vu le jour:

- WinMX
- Blubster
- FileTopia
- KaZaa
- Shareaza
- Clustone
- XoloX
- Freewire
- Limewire
- BearShare
- Morpheus
- Neti
- e-mesh
- e-donkey
- ...

L'effet Napster (2)

Par crainte de fermeture, les logiciels audio les plus répandus sont actuellement ceux possédant une architecture décentralisée.

Nombre de téléchargements depuis leur création (au 7 janvier 2003):

1. KaZaa (P2P décentralisé)	**172'793'393**	
2. Morpheus (P2P décentralisé)	108'141'673	
3. iMesh (P2P centralisé)	**41'330'017**	
4. Audiogalaxy (P2P centralisé)	31'303'662	=> devenu payant
5. BearShare (P2P décentralisé)	17'939'547	

Source: download.com

L'effet Napster (2)

Par crainte de fermeture, les logiciels audio les plus répandus sont actuellement ceux possédant une architecture décentralisée.

Nombre de téléchargements depuis leur création (au 7 janvier 2003):

1. KaZaa (P2P décentralisé)	**172'793'393**	
2. Morpheus (P2P décentralisé)	108'141'673	
3. iMesh (P2P centralisé)	**41'330'017**	
4. Audiogalaxy (P2P centralisé)	31'303'662	=> devenu payant
5. BearShare (P2P décentralisé)	17'939'547	

Source: download.com

Gnutella

- Résolution des problèmes de napster
- OpenSource
- Entièrement Distribué
- Très-Très politique


Gnutella

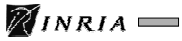
- Réseau p2p :
  - les pairs se connectent aux pairs
- Objectif :
  - méthode décentralisée de recherche de fichiers
- Chaque pair (instance applicative) est utilisé pour :
  - stocker des fichiers spécifiques
  - router les requêtes (recherche de fichiers) de et vers ses pairs voisins
  - répondre aux requêtes (servir le fichier) si le fichier est stocké localement
- Historique
  - 14 jours de développement, NullSoft (winamp)
  - Initialement pour l'échange de recettes
  - publié en GNU gpl
  - lancé par AOL (propriétaire de NullSoft), immédiatement retiré
  - trop tard !!!! : 100k utilisateurs
  - de nombreux bug initiaux … Perte d'utilisateurs


Gnutella : comment ça marche !

- Pas de serveur central
  - Pas de poursuite judiciaire (napster)
- Broadcast forcé (Inondation)
  - Chaque pair envoie les paquets qu'il reçoit à tous ses pairs (typiquement 4)
  - Durée de vie d'un paquet limité par un TTL (typiquement 7)
  - Les paquets ont un id unique pour détecter les boucles (le paquet est quand même reçu deux fois)
- Pour rentrer dans un réseau gnutella, il faut connaître au moins 1 hôte gnutella
  - gnutellahosts.com:6346


Gnutella : format du message

- Message Id : 16 bytes (oui octets)
- Function Id : 1 byte
  - 00 ping : recherche d'hôtes gnutella
  - 01 pong : réponse de ping, nb de fichiers
  - 80 query : recherche chaîne, b/w minimale
  - 81 query hit : correspondance 80:query, myIP, adresse/port, b/w
- Remaining TTL : décrémenté à chaque pair pour éviter les inondations par TTL
- HopsTaken : Nombre de pairs visités par ce message
- DataLength : taille du champ data

Gnutella

- Transfert de fichiers par http lite
- Push descripteur pour passer les firewalls

- Les propriétés « Small-world » sont vérifiés (tout est à proximité)
- Back-bone + extensions


Gnutella : problèmes initiaux

- Chargement libre (free riding)
  - Connexion sur gnutella sans apports de données, ni routage de requêtes
- Arrêts prématurés de transferts :
  - Temps de chargement sur modems
    - Les utilisateurs sont intermittents, de plus les autres utilisateurs sont bloqués
    - Solution : busy code
    - 2000 : 10% de transferts réussis, 2001: 25%


Gnutella : problèmes initiaux (suite)

- 2000 : 400-800 hôtes
  - modems: trous noirs de routage
- ==> Hiérarchie de pairs en fonction des capacité de transport
  - avant : tous les pairs identiques
  - Préférence de connexion
    - Préférence de route vers les pairs « bien-connectés »
    - Favoriser les réponses vers des pairs fournissant beaucoup de fichiers (éviter le freechargement)
  - Apparition des passerelle limewire
    - cf napster pour la recherche

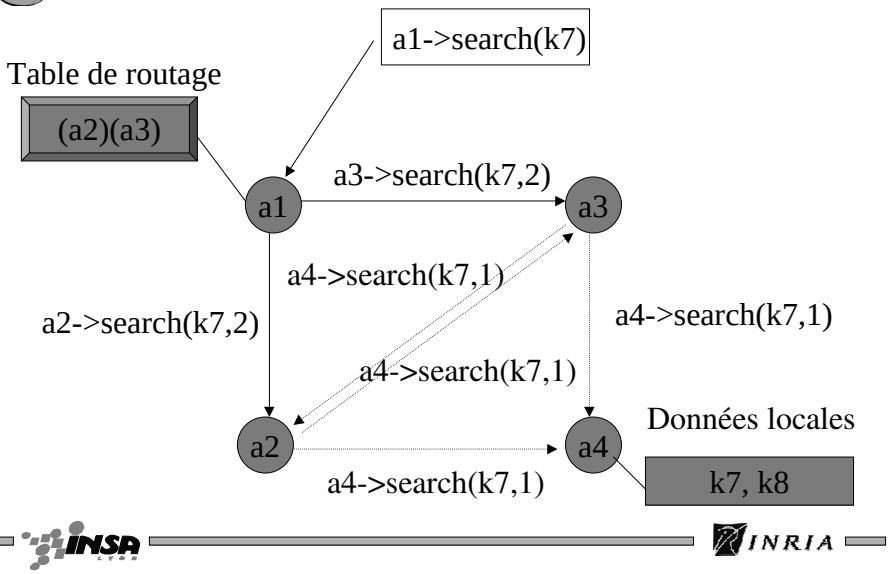

Gnutella conclusion

- Complètement décentralisé
- Les 'hits' sont élevés
- Haute tolérance de pannes
- S'adapte très bien et dynamiquement au changement de population des pairs
- Le protocole et très gourmand (3,5 Mbps)
  - 4 connexions C / pair, TTL=7
  - 1 ping peut générer : $2*\Sigma_{(0,TTL)}C*(C-1)^i$=26240 packets
- Pas de borne sur la durée d'une requête
- Pas d'estimation sur le résultat de la requête
- La topologie est inconnue => pas d'exploitation algorithmique
- FreeRiding est un problème
- Pas de répudiation des pairs
- Simple, robuste, passe à l'échelle (pour le moment)


Gnutella Recherche

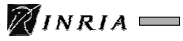
- Chaque pair connaît un nombre fixe **d'autres pairs** (càd 4)
- Les pairs se découvrent par des messages ping
- Les requêtes de recherches sont propagées vers ces pairs dans la limite de 7 TTL
- Les pairs peuvent répondre à la requête s'ils stockent le fichier correspondant

Gnutella Recherche



Gnutella Discussion

- Type de recherche
  - Texte libre
- Mise à l'échelle
  - Recherche pauvre d'un point de vue global
  - Temps de recherche en O(Log n) (small-world)
  - Mise à jour excellent : rien à faire
  - Information de routage : coût faible
- Robustesse
  - Elevée, plusieurs chemins sont évalués
  - Exploite les propriétés small-world
- Autonomie
  - Stockage : pas de restriction, les pairs stockent les clés de leurs fichiers
  - Routage : les pairs sont les cibles de n'importe quelles requêtes (pas d'autonomie)
- Connaissance globale
  - Non

Gnutella refs

[Adar00] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. Technical Report. Xerox PARC, 9, Septembre 2000. http://www.parc.xerox.com/istl/groups/iea/papers/gnutella/Gnutella.pdf

[Clip01] Clip2. The Gnutella Protocol Specification V0.4 (Document Revision 1.2) June 15, 2001, http://www.clip2.com/GnutellaProtocol04.pdf

[Gnutella01] Gnutella homepage, 2001. http://gnutella.wego.com/

[Jovanovic01] M.A. Jovanovic, F.S. Annexstein, and K.A. Berman. Scalability Issues in Large Peer-to-Peer Networks - A case study of Gnutella. University of Cincinnati, Laboratory for Network and Applied Graph Theory, 2001. http://www.ececs.uc.edu/~mjovanov/Research/Paper.ps

[Sripanidkuchai01] Kunwadee Sripanidkulchai. The popularity of Gnutella queries and its implication on scalability. February 2001. http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html


Gnutella : Non Anonyme

- La personne qui fourni le fichier connaît son destinataire
- Eternity
- Freenet


P2p : le point !

- Modèle centralisé
  - Napster
  - Index global maintenu par une autorité centrale
  - Relation directe demandeur/fournisseur
- Modèle décentralisé
  - Freenet, Gnutella, Chord
  - Pas d'index central - Connaissance locale uniquement (réponse approximative)
  - Les mise en relation sont indirectes : ce sont des chaînes d'intermédiaires


Freenet Historique

- PFE : Ian Clarke, Edimbourg, 1999
- Sourceforge : http://www.sourceforge.net
- V.0.1 (Mars 2000)
- V.0.4 (Sept. 2001)


C’est quoi et pourquoi ?

- La paranoïa
- Système de fichiers partagés, distribué, p2p
- Complètement anonyme pour l'émetteur et le consommateur de l'info
  - Impossible de déterminer l'origine ou la destination d'une donnée
  - Difficulté pour un nœud de savoir ce qu'il stocke (les fichiers sont envoyés et stockés cryptés)

  ==> Pas de poursuite possible
- Les requêtes sont routées vers la localisation physique la plus probable
  - Pas de serveur central
  - Pas de broadcast contraint (cf gnutella)
- Les fichiers sont identifiés indépendamment de leurs localisation physique
- Réplication dynamique des données
- Résistant aux attaques dos…de tiers


Freenet : comment ça marche

- Structure de données
- Gestion de clés
- Problèmes
  - Comment un nœud connaît les autres
  - Comment peut-il récupérer des données
  - Comment ajouter de nouveaux nœuds
  - Comment freenet gère ses données
- Détails du protocole
  - Information de l'entête


Freenet : Structure de données

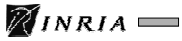
- Table de routage
  - Liste fixe d'autres pairs
  - ip,tcp : clé (1 clé par pair)
- Répertoire de données
  - Prérequis
    - Trouver rapidement un document connaissant sa clé
    - Trouver rapidement une clé proche
    - Gérer la popularité d'un document et savoir quels documents supprimer en cas de congestion...


Freenet : Gestion de clé

- Les clés sont représentées par des URI :
  - freenet:TypeDeCle@data
- Avec TypeDeCle
  - KSK : Keyword Signed Keys, elle représente un document
  - SVK : Signature Verification Key
  - SSK : SVK Subspace key, sous-arbre de nomage
  - CHK : Content Hash Keys, signature de documents
- Les clés peuvent être utilisées pour des indirections
  - KSK ---> CHK


Freenet : Keyword-Signed Key (KSK)

- Basé sur une petite chaîne de description, classiquement un ensemble de mots clés décrivant le document
  - freenet:KSK@cours/INSA/telecom/sfrenot/p2p
- Deux améliorations :
  - Espace de nomage global (SVK, SSK)
  - Encodage du fichiers (CHK)

Freenet : Gestion de clé (paranoïa)

- Signed-subspace Key (SSK)
  - Ajout d'informations concernant l'émetteur afin d'éviter les conflits de nomage (sous-espace)
  - Clé privé pour signer l'espace/clé publique pour vérifier
    - freenet:SVK@HDOKWIUn10291jqd097euojhd01
    - freenet:SSK@1093808JQWIOEh8923Iah10/text/book/1984.html
- Content-hash Key (CHK)
  - Algorithme de signature de message (MD5), calcul de hash du document


Freenet : Insertion d'une clé

- La clé est calculée (KSK, CHK,…)
- Un message d'insertion avec la clé et un nombre de sauts aléatoire (hop) est envoyé sur le réseau
- Chaque pair contrôle si la clé est dans son système de stockage local
  - oui ==> la clé doit être regénérée
  - non ==> on route vers le nœud suivant hop --(pour choisir un nœud suivant, il y a un algorithme de recherche de clés proche)
  - On continue jusqu'à hop=0
- Si hop==0 et pas de collision, la clé est insérée sur tout le chemin de routage


Eléments de sécurité et d'authentification

- Anonymisation
  - Les nœuds mentent aléatoirement sur les requêtes et s'annoncent comme étant à l'origine ou à la destination d'une requête
  - Les Hop-to-live sont aléatoires
  - Il est impossible de remonter au nœud source d'un document
  - Il est impossible de connaître le nœud qui a inséré le document

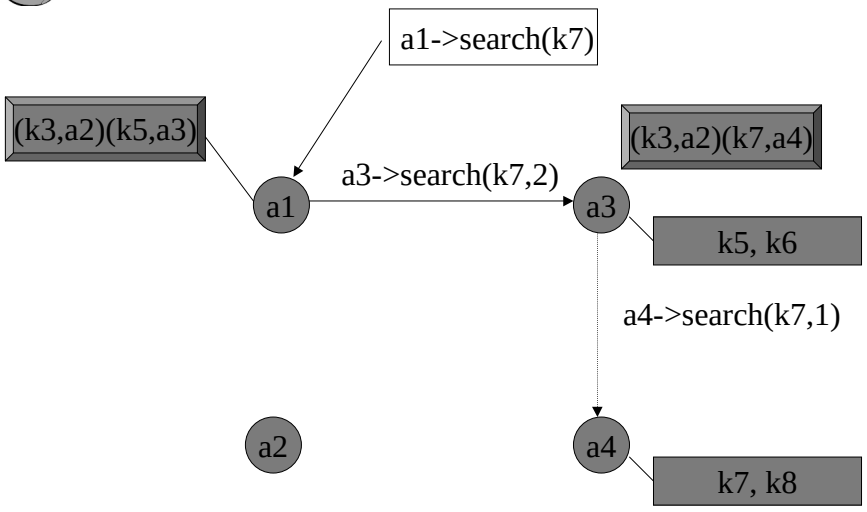

Freenet : Résumé

- Entièrement décentralisé
- Forte tolérance aux pannes
- Robuste et passe à l'échelle
- Réplication automatique du contenu
- S'adapte parfaitement et dynamiquement au changement de population de pairs
- Le spam peut être limité (sous-espace)
- Le routage adaptatif respecte la bw
- Pas d'estimation de la durée d'une requête
- Pas d'évaluation sur la réussite d'une requête
- Pas de topologie=pas d'algorithme
- Le routage circonscrit les « free-riders »
- Répudiation de pairs n'est pas pris en compte
- Prend en compte l'anonymité du producteur et du lecteur


Recherche dans freenet

- Chaque pair connaît un nombre fixe d'autres pairs et **une clé** que le pair stocke
- Les recherches sont routées vers le pair avec la clé la plus similaire
  - Si insuccés, la clé similaire suivante est essayée
  - Distance lexicographique (autre distance possible)
- Les requêtes de recherche sont limitées dans le temps (500 sauts)
- Un pair peut répondre s'il stocke la clé
- Quand la réponse est renvoyée les pairs intermédiaires peuvent mettre à jour leur information de routage

Freenet Recherche



Freenet Dicussion

- Type de recherche
  - Uniquement égalité
  - Cependant si les clés n'étaient pas hashées, une similarité sémantique pourrait être utilisé
- Passage à l'échelle
  - Bonne recherche O(Log nà
  - Mise à jour excellente, pas de surcharge
  - Information de routage : une phase de prédiction est nécessaire
- Robustesse
  - Bonne, car les chemins alternatifs sont explorés
- Autonomie
  - Pas de restriction sur le stockage
  - Routage : dépendance entre les clés stockées et les requêtes reçues
- Connaissance globale
  - Clé de hash


Freenet

- [Clarke01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed Anonymous Information Storage and Retreival System. Designing Privacy Enhancing Technologies: International Workshop on Design Issue in anonymity and Unobservability, LLNCS 2009, Springer Verlag 2001, http://www.freeenetproject.org/index.php?page=icsi-revised
- [Freenet01] Freenet project. Protocol Specs. 2001. http://www.freenetproject.org/doc/book.html
- [Langley01] Adam Langley. The Freenet Protocol 2001. http://www.freenetproject.org/index.php?page=protocol
- [hong01] Theodore Hong. Performance in Decentralized FileSharing Networks. Presentation given at the O'Reilly Peer-to-Peer Conference, San Fransisco. February 14-16, 2001. http://www.freenetproject.org/p2p-theo.ppt


p2p applicatif --> p2p optimisé

- Exploitation des principes de « small-worlds »
- Extension à de plus larges échelles, systèmes basés sur les principes de hash
  - Chord
  - Tapestry
  - CAN


Chord

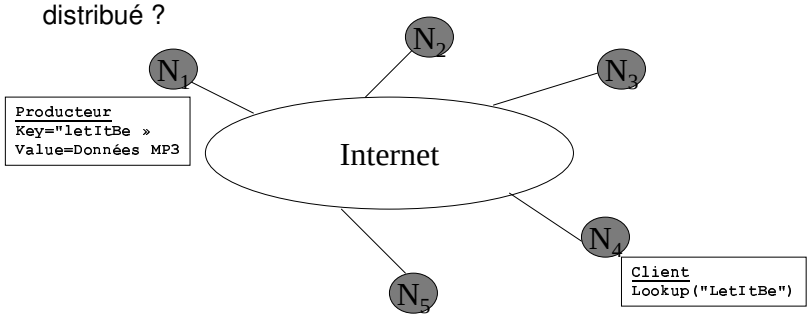
Service de recherche P2Pextensible pour applications Internet

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan

MIT & Berkeley


Motivation

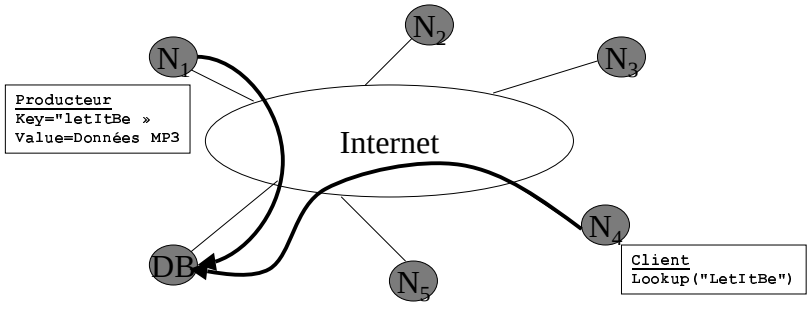
- Comment retrouver une donnée dans un système de fichiers partagé distribué ?



- La recherche est le point dur !

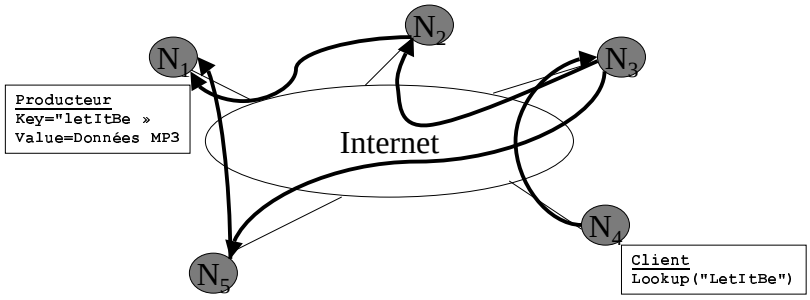
Solution centralisée

- Serveur central (Napster)



Solution distribuée (1)

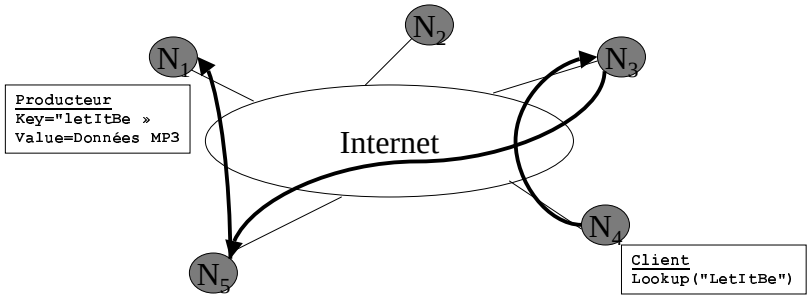
- Inondation (flooding) : Gnutella, Morpheus, KaZaa



- Pire cas : O(N) messages par requêtes

Solution distribuée (2)

- Routage de messages (Freenet, Tapestry, Chord, CAN)



- Recherche Exacte !

Les défis du routage

- Définir une bonne métrique de proximité de clé
- Conserver le nombre de sauts petit
- Conserver des tables de routage raisonnable
- Rester robuste même en cas de changement rapide de communauté de pairs

- Chord : se focalise sur l'efficacité et la simplicité

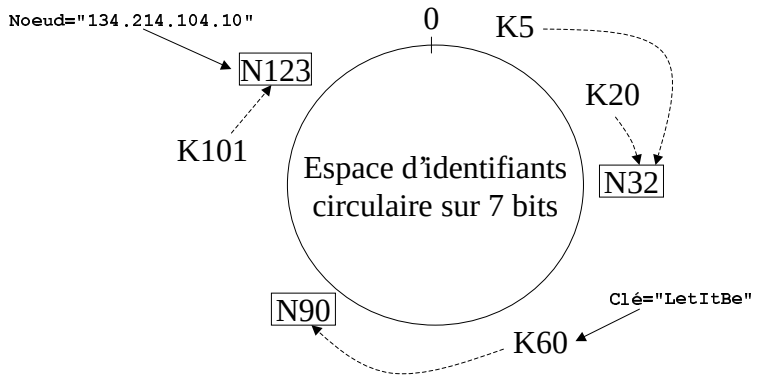

Les identifiants de Chord

- Identification sur m bits pour les clés et les nœuds
- Identification de Clé: SHA-1(key)

  `Key="LetItBe" ---SHA-1----> ID=60`
- Identification de Noeud : SHA-1(@ip)

  `Node="134.214.104.10" ---SHA-1----> ID=123`
- Les deux sont uniformément distribués
- Comment mapper les ids de clé sur les ids de noeud


Hashage consistent [Karger 97]

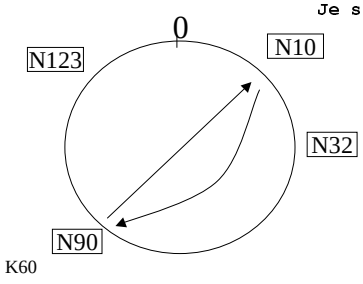


- Une clé est stockée sur son nœud successeur : le nœud avec le plus gros ID suivant

Hashage consistent

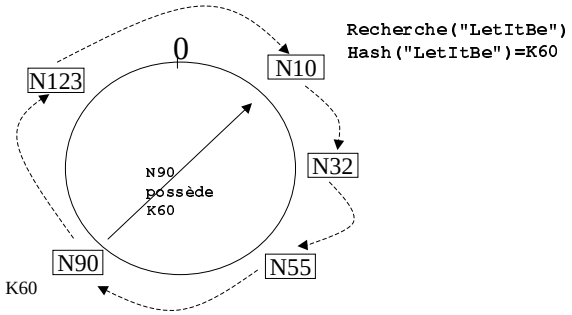
- Chaque nœud connaît tous les autres
  - Connaissance globale (! p2p)
- Les tables de routages sont larges
  - O(N)
- Recherche rapide O(1)

```
Recherche("LetItBe")
Hash("LetItBe")=K60
Je sais que N90=>K60
```



Chord : recherche de base

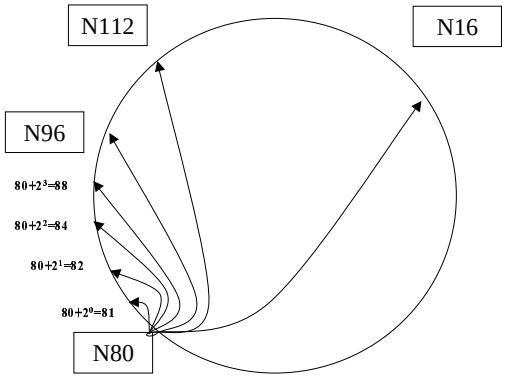
- Chaque nœud connaît sont successeur dans l'anneau



- Recherche O(N) sauts

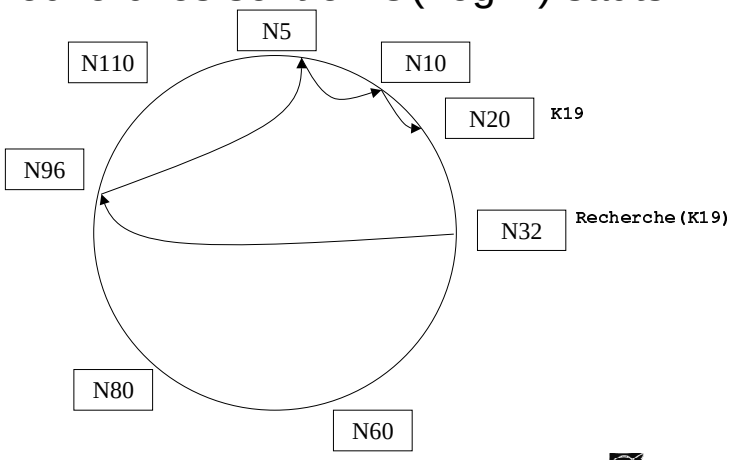
"Finger table" tables d'identification

- Chaque nœud connaît m autres nœuds de l'anneau
- En augmentant la "distance" exponentiellement



Les recherches sont plus rapides

- Les recherches sont en O(Log N) sauts

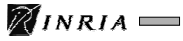


Rejoindre l'anneau

- Trois étapes
  - Initialiser les fingers des nouveaux nœuds
  - Mise à jour des fingers des autres nœuds
  - Transférer les clés des successeurs vers le nouveau nœud
- Mécanisme moins agressif (mise à jour paresseuse des fingers)
  - N'initialiser que le finger pour le nœud suivant
  - Vérifier périodiquement le successeur et le prédécesseur immédiat
  - Rafraîchir périodiquement les entrées de la table finger

Gestion des pannes

- Une panne de nœud génère une recherche incorrecte
- Un nœud ne connaît pas ses "bons" voisins
  - Chaque nœud connaît r successeurs directs
  - Après une panne, on connaît le premier succ. vivant
  - Des successeurs corrects garantissent une recherche correcte
- La garantie sous une certaine probabilité
  - Choisir r, afin d'avoir un taux de panne suffisamment faible

Evaluation

- Recherche rapide dans de grands systèmes
- Variation faible dans les coûts de recherche
- Robuste même en cas de panne massive

- Fondé sur un modèle théorique
- Performance démontrée
- Robuste


Faiblesse

- Pas si simple (cf CAN)
- Ajout d'un membre est complexe
  - Mécanisme agressif ==> bcp de messages et de mise à jour
  - Pas d'analyse de la convergence dans le mode paresseux
- Mécanisme de gestion des clés partagé entre les couches applicatives et chord
  - Les couche hautes font les insertions et la gestion des pannes de nœuds
  - Chord transfert les clés quand les nœuds arrivent (pas de mécanisme pour quitter !)
- Les tables de routages augmentent avec le nombre de membre du groupe
- Le pire cas de recherche peut être lent


Chord discussion

- Type de recherche
  - Exacte
- Passage à l'échelle
  - Recherche O(Log n)
  - Mise à jour nécessite une recherche
  - Fabrication : O(Log^2 n) si un nouveau nœud arrive
- Résistance (Robustesse)
  - La réplication peut être utilisée en déposant les réplicats sur les nœuds suivants
- Autonomie
  - Stockage et Routage : aucun
  - Les nœuds ont, de part leur adresse un rôle spécifique
- Connaissance globale
  - Mappage adresse / clé


Chord

- [Karger97]


Tapestry

Routage et Localisation décentralisés

Ben Y. Zhao
CS division, U. C. Berkeley


Motivations

- Les systèmes de stockage partagés nécessitent un mécanisme de routage et de localisation des données
  - Trouver un pair, tout en passant à l'échelle est un problème difficile
  - Mécanisme d'insertion et de recherche de données dans un très grand système
- Solutions actuelles
  - Centralisée : Coûteux à étendre, moins tolérant aux pannes, vulnérable aux attaques type DoS (napster, dns)
  - Inondation : Pas de passage à l'échelle (gnutella)


Clé : localisation et routage

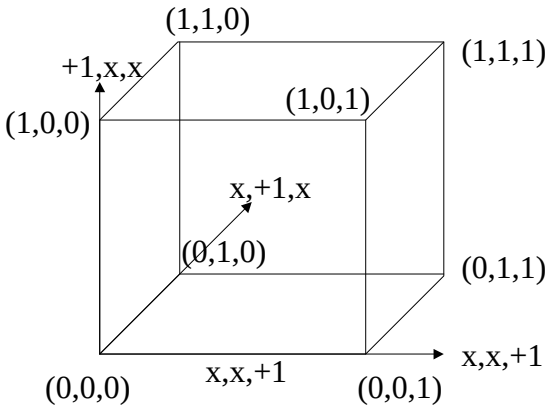
- Problème difficile
  - Localisation et envoi de message vers les ressources et les données
- Approche : infrastructure d'overlays large échelle
  - Passe à l'échelle, Dynamique, Tolérant aux pannes, Equilibrage de charge
- Hiérarchie de localisation décentralisée
  - Un objet est stocké dans une hiérarchie de nœuds
  - La hiérarchie débute à un nœud racine qui identifie de manière unique l'objet


Tapestry

- Infrastructure de localisation et de routage, décentralisée, passant à l'échelle, tolérante aux fautes et adaptative
- Couche réseau du système de stockage de masse OceanStore
- Routage hypercube basé sur le suffixe
  - Plaxton Algorithm (Plaxton, Rajamaran, Richa (PRR) SPPAA'97)
- Core API
  - publishObject(ObjectID, [ServerId])
  - sendmsgToObject(ObjectId)
  - sendmsgToNode(NodeId)

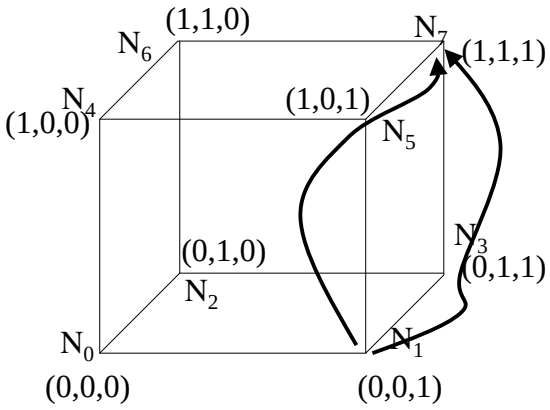
Routage hyper-cube

- Hyper-cube de dimension 3 :



Routage hyper-cube

- Hyper-cube de dimension 3 :



Routage

N1 (0,0,1) -> N7(1,1,1)

==> 2 sauts, car 2 bits de différence

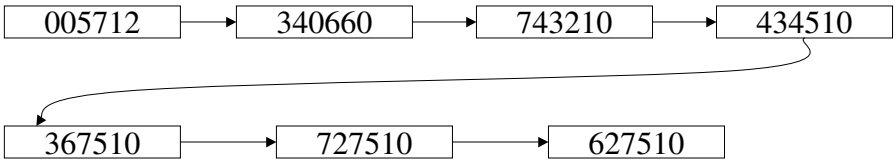
N1 --> N5 --> N7

ou

N1 --> N3 --> N7

Routage de base

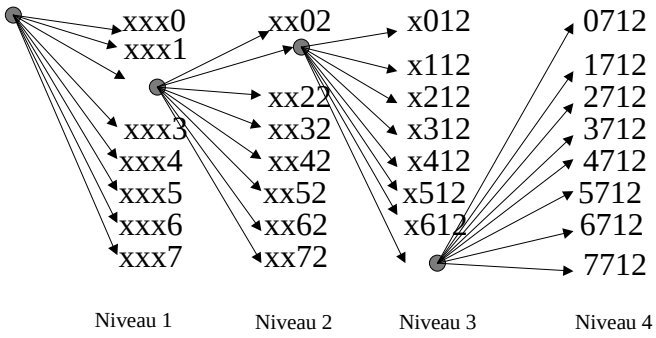
- Exemple : valeur octale, nommage 2^{18}, 005712-> 627510
- Le routage se fait par rapprochement successif vers l'objectif



==> Le nombre de sauts est 6 ?

Chaque nœud possède une table de routage hiérarchique

- Ex : table pour de routage pour 5712



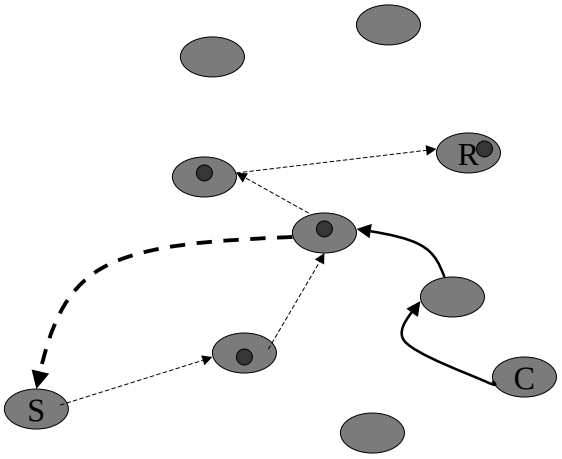
Insertion d'un objet

- Espace de nommage commun (nœud et objets)
  - Suffisamment grand pour éviter les collisions (2^{160})
  - (taille N dans $Log_2(N)$ bits)
- Insertion
  - Hash de l'objet dans E ==> ObjectID
  - For (i=0, i< $Log_2(N)$, i+j){ //On définit la hiérachie de l'obj
    - j est la base de la taille des valeurs utilisée (j=4, hex)
    - Insérer l'objet dans le nœud le plus proche qui correspond au dernier i bits
    - Quand aucune correspondance n'est trouvée, prendre le nœud qui correspond à (i-n) bits avec le plus grand ID, break

Routage vers un objet

- Recherche d'un objet
  - Traverser le réseau comme pour une insertion, mais en plus rechercher à chaque nœud l'objet
  - For (i=0; i<Log_2(N),i+n)
    - Aller au nœud et rechercher si l'objet y est stocké
  - Chaque objet est mappé sur la hiérarchie définie par un nœud racine f(ObjectId)=RootId)
  - Publier / Rechercher les deux remontent de manière incrémentale vers la racine

Schéma



Le modèle est un peu plus complexe

- Il y a une indirection
  - Pour déposer un objet, le client calcule un identifiant de nœud racine, qui connaît la localisation exacte de l'objet
  - La hiérarchie accède en fait à l'identification du nœud racine.

store (Object, ObjectLocation)

store (ObjectLocation, Root(Object))

pub(Root(Object), Hash(ObjectId))


Commentaires

- Points forts
  - Modèle théorique (Algo de Plaxton)
  - Entièrement décentralisé et passe à l'échelle pour la localisation et le routage deterministe
- Points faibles
  - Compliqué
    - Insertion dynamique de nœud est complexe
    - Ajout de nombreux nœuds simultanéement
  - Comment choisir le nœud racine ?
    - Panne


CAN

Un réseau d'adressage
de contenu qui passe à l'échelle

Sylvia Ratnasamy, Paul Francis, Mark Handley,
Richard Karp, Scott Shenker

ACIRI, UC. Berkeley, Tahoe Networks

Transparents : (Sigcomm 2001)

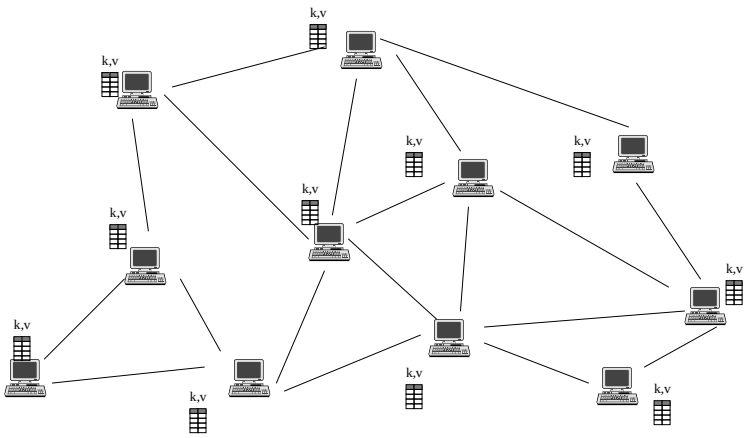
Hash tables à l'échelle d'internet

- Hash tables
  - Structure de données essentielles de tous les logiciels
- CAN : Internet-Scale Hash tables
  - Aussi intéressant pour des systèmes distribués large échelle
  - insert (clé, valeur)
  - valeur=accède(clé)
- Objectif
  - Passage à l'échelle
  - Simple à mettre en œuvre/utiliser
  - Bonne performance

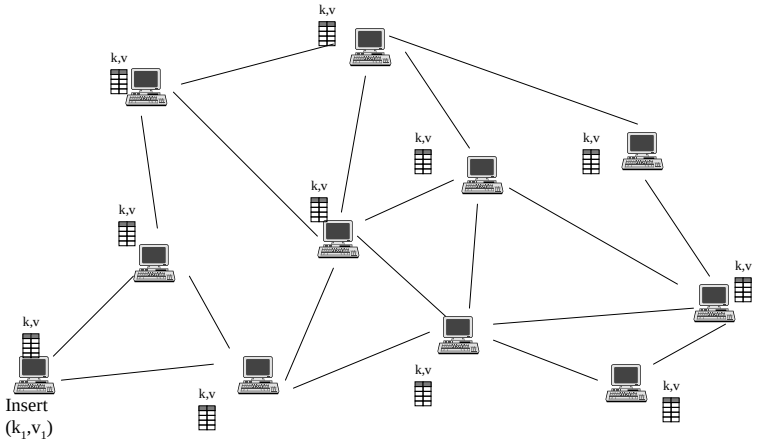
Etendue du problème

- Concevoir un système qui fournit une interface
  - qui passe l'échelle
  - robuste
  - performante
  - sécurisée
- Au niveau applicatif
  - recherche par mot-clé
  - contenu changeant
  - anonyme

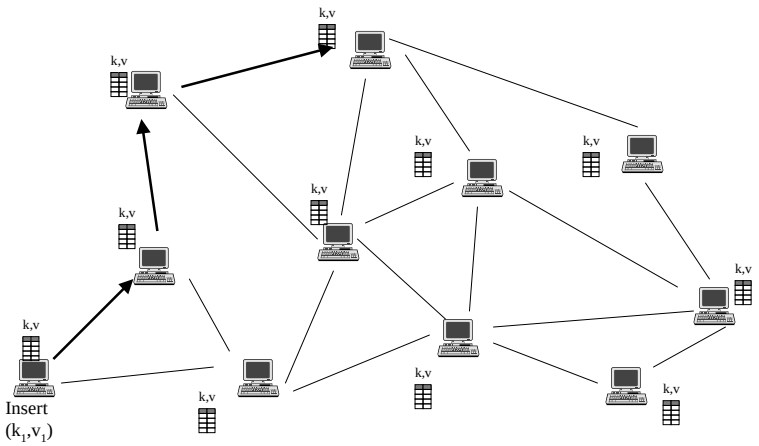
Can idée de base



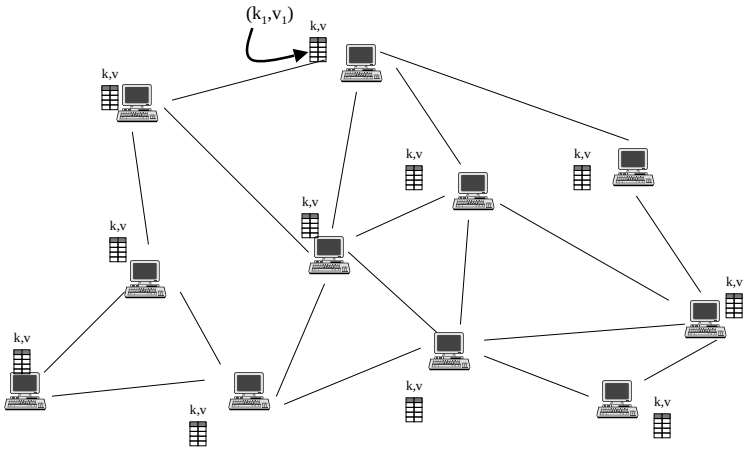
Can idée de base



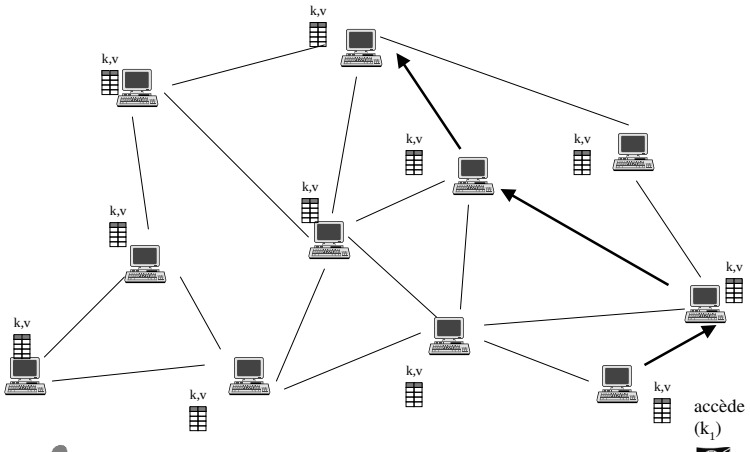
Can idée de base



Can idée de base



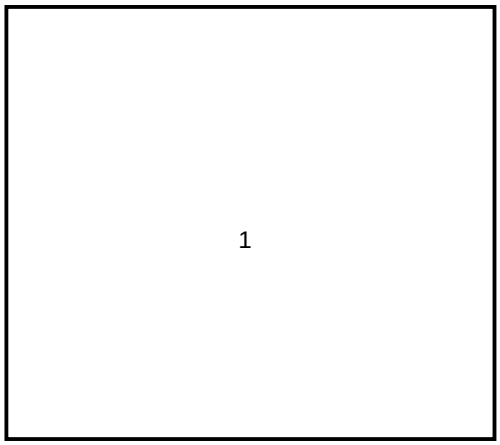
Can idée de base



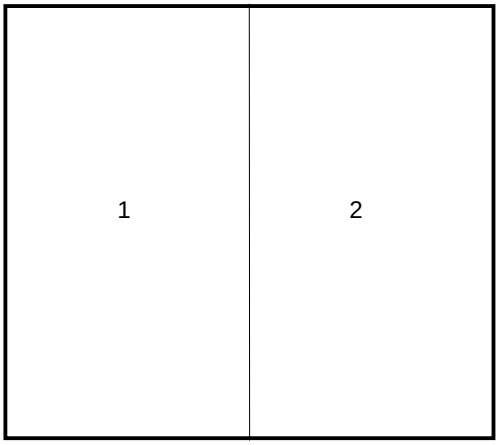
CAN : Solution

- Coordonnées cartésiennes virtuelles de l'espace
- Tous l'espace est partitionné par rapport aux différents nœuds
  - Chaque nœud « possède » une partie de l'espace
- Abstraction
  - On stoque des données sur des « points » de l'espace
  - On peut router d'un point vers un autre
- Point = nœud qui contient toute la zone de coordonnées

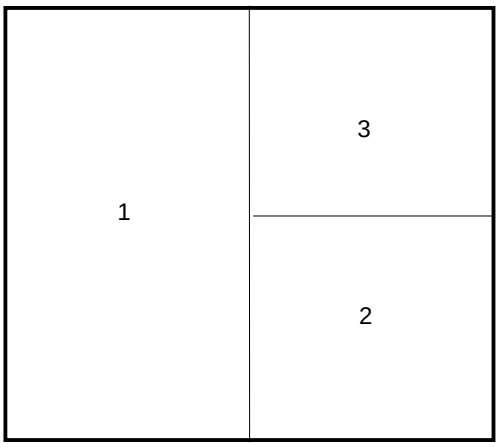

CAN: example de base



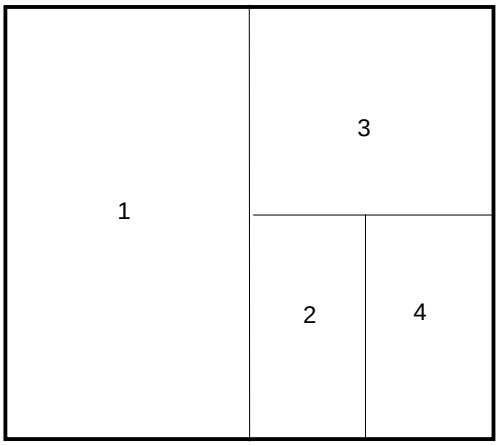

CAN: example de base



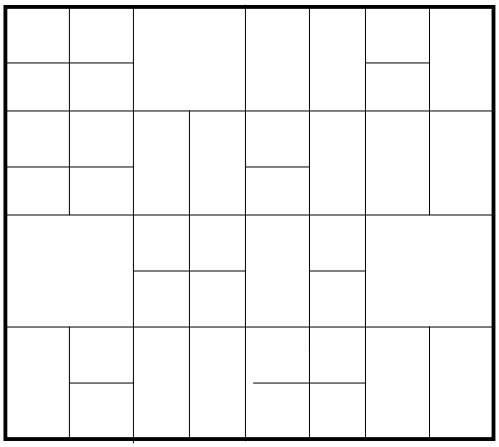

CAN: example de base



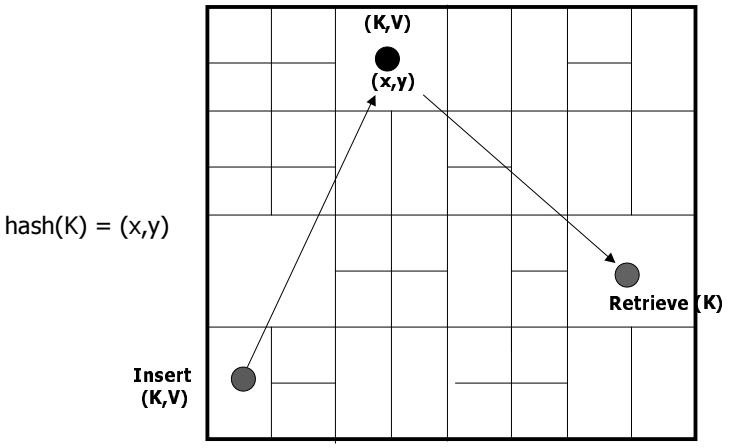

CAN: example de base




CAN: example de base




CAN: example de base

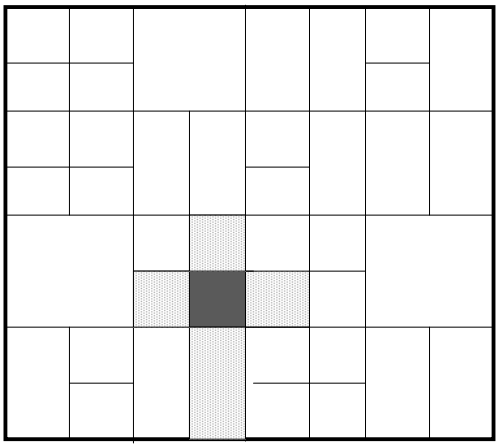


Conception de CAN

- Routage dans CAN
- Construction plan
  - Insertion de nœuds
- Maintenance de CAN
  - Suppression de nœud
  - Récupération de panne
  - Relocalisation

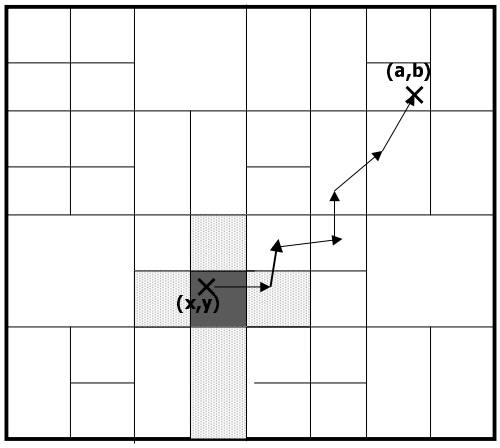
CAN: routage

- Voisins
  - 4 voisins pour un plan 2d

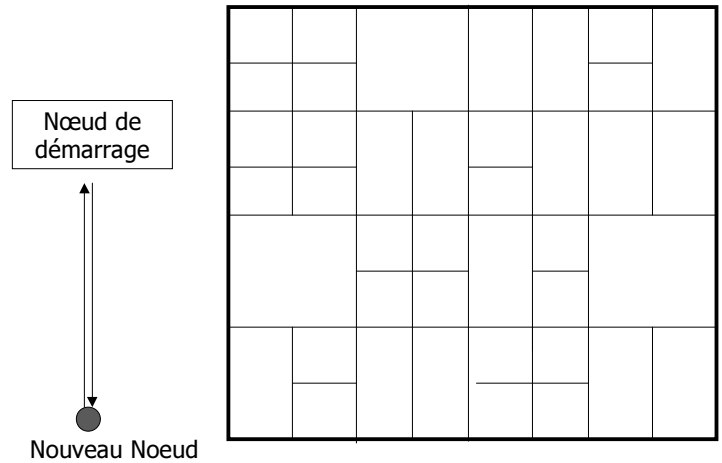


CAN: routage

- Espace dimension d
- n zones
- Taille moyenne de la route
  - $(d/4)(n^{1/d})$
- Un nœud ne maintient la route que sur ses 4 voisins

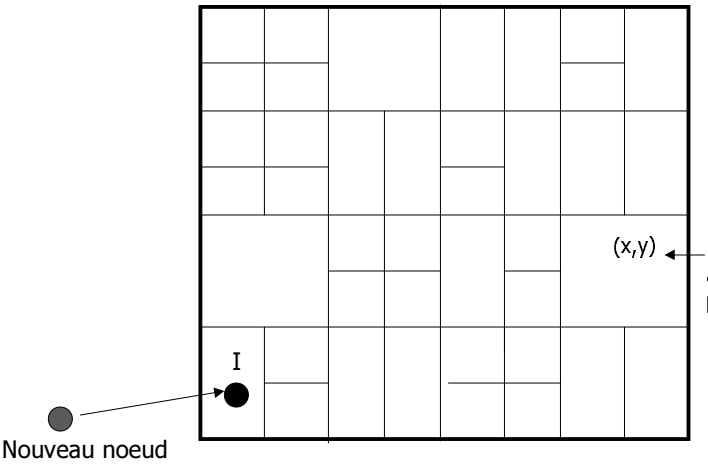


CAN: construction



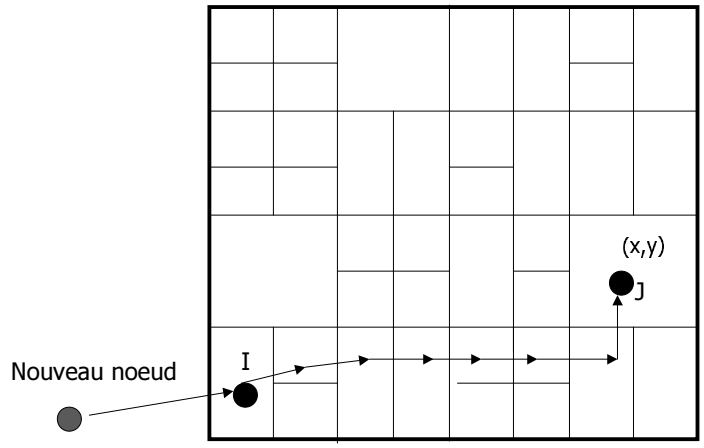
1) Trouver un nœud "I" déjà dans CAN

CAN: construction



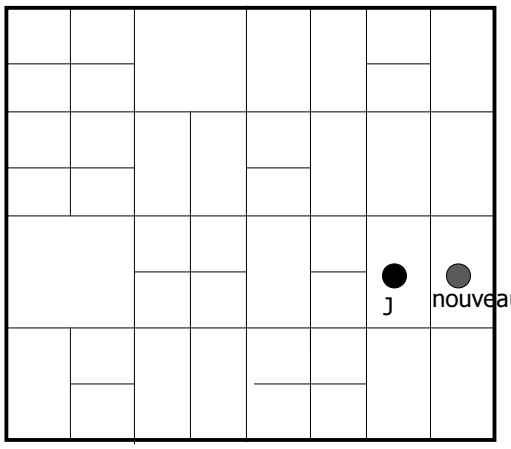
2) Prendre un point au hazard dans le plan

CAN: construction



3) I route vers (x,y) et tombe sur J

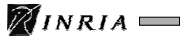
CAN: construction



4) Séparer la zone de J en deux moitiées, le nouveau possède une des moitiées

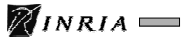
CAN insertion

- L'insertion d'un nouveau nœud n'affecte qu'un seul autre voisin et ses voisins immédiats

CAN : reprise sur panne

- Panne simple
  - On connaît les voisins des voisins
  - Quand un nœud tombe, un de ses voisins prend possession de la zone
- Modèle de panne plus complexe
  - Panne simultanée de plusieurs nœuds adjacents
    - Inondation partielle pour trouver les voisins
    - Rare
- Normalement seul le nœud en panne et ses voisins sont concernés

CAN : résumé de la conception

- CAN de base
  - Complètement distribué
  - Auto-organisé
  - Les nœuds ne maintiennent des états que pour les voisins immédiat
- De plus
  - Espaces indépendant multiples (réalités)
  - Algorithme d'equilibrage de charge en tâche de fond
  - Heuristiques simple pour améliorer les performances


CAN : Forces & Faiblesses

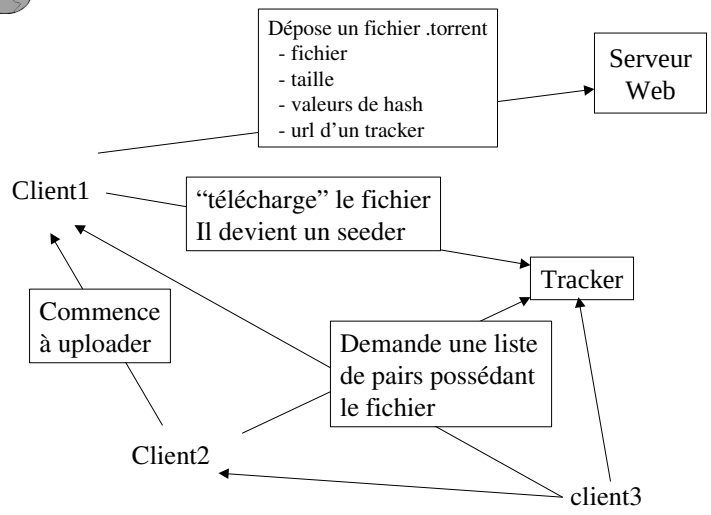
- Forces
  - Plus absorbant que des réseau d'inondation
  - Efficace pour trouver une info
  - Haute dispo des nœuds et des données
  - Possibilité de gérer les tables de routage et le trafic réseau
- Faiblesses
  - Pas de recherche floues
  - Activité Malicieuses possibles (Dos)
  - Doit maintenir une cohérence globale (Surcharge réseau, Distribution efficace)
  - Latence du routage applicatif
  - Performances faibles par rapport aux améliorations


Bittorrent

- Optimisation du transport de gros volumes
  - Pas de recherche,
  - Protocole de téléchargement p2p
  - Repose sur un algorithme de type :
    - ‘Un prêté pour un rendu’ (tit-for-tat)
    - 1/1 upload-download
  - Interface simple (save As)
  - Les uploaders décident vers qui envoyer les fichiers

Bittorrent
Bram Cohen
bram@bitconjurer.org
2003

Protocole



Un fichier torrent

- Segmenté en morceaux de 256k
  - Chaque morceau possède un code de hash
  - Une fois qu'un morceau est téléchargé il est mise à disposition des autres
- Chaque morceau est également décomposé en 16k
  - 5 requêtes de sous-morceaux sont mises en file d'attente
- ==> Un objectif tout ce qui rendre doit sortir
- ==> Bittorrent cherche à optimiser le téléchargement

Algo de Selection des pièces

- Priorité stricte
  - Si un sous-morceau a été demandé, le sous-morceaux suivant doivent appartenir au même morceau
  - On charge un morceau le plus rapidement possible
  - Un morceau n'est mis en dispo que si sa clé de hash est valide
- Le plus rare en premier
  - On télécharge le morceau le moins disponible
  - Le seed répartit plus rapidement les morceaux
  - Si le seed initial disparaît c'est plus robuste
- Le premier est choisi au hasard
  - Un morceau rare sera plus long à charger
  - Aucune connaissance a priori des capacités des pairs
- Derniers bits
  - Tous les sous-morceaux du dernier morceau sont demandés simultanément à tous les pairs
  - On évite que le dernier download soit ralentit parce qu'on tombe sur un pair lent


Algorithmes de suffocation (choke)

- Les nœuds doivent optimiser leur chargement sur un mode « un prêté pour un rendu ».
- 4 pairs sont régulièrement déchoké… Lesquels
  - On cherche à mettre des pairs directement en contact
  - On déchoke les pair dont le download est le plus élevé
    - Echantillon sur 10s, et attente sur 10s de plus (Pourquoi ?)
  - Déchokage optimiste aléatoire à chaque itération un des 4 est choisi aléatoirement (30s) (seule manière de trouver des pairs plus rapides)
  - Anti-snobisme : si un pair ne reçoit rien d 'un autre pair pendant 1 minute, il coupe sont upload vers lui.
  - Chargement uniquement : dans ce cas le choix se fait sur les pairs qui ont un meilleur taux d'upload


- FIN...