

# Systemes Pair à Pair

## P2P : Introduction

- ▶ Définition historique : Un système distribué où un nœud peut être à la fois client ou serveur
  - ▶ Plus vraiment de distinction entre les deux rôles
- ▶ Définition actuelle : Un système distribué utilisant les ressources localisées aux frontières d'Internet
- ▶ Premier système pair à pair connu: Napster
  - ▶ Chaque nœud était à la fois producteur et consommateur de fichiers mp3
- ▶ Propriétés
  - ▶ Décentralisation (Decentralised)
  - ▶ Dynamicité (Dynamic)
  - ▶ Auto configuration (Self configurable)
  - ▶ Auto cicatrisation (Self-healing)

## P2P : Introduction

- ▶ Classification suivant les ressources partagées
  - ▶ P2P de calcul : seti@home, folding@home, distributed.net
  - ▶ P2P de fichiers : Napster, Gnutella, Freenet, edonkey, bittorrent
- ▶ Classification suivant l'organisation
  - ▶ Non structuré : pas d'ordre entre les pairs
  - ▶ Structuré : il existe une organisation des pairs

## Pair à Pair de calcul

## Principe

- ▶ Des calculs nécessitent d'énormes ressources de calcul
  - ▶ Pas d'ordinateur assez puissant
  - ▶ Pas de moyens financiers
- ▶ Des millions d'ordinateurs individuels sont sous utilisés
  - ▶ Un CPU travaille peu et se met souvent en économie d'énergie
  - ▶ La nuit, certaines entreprises laissent les ordinateurs allumés
- ▶ Idée : utiliser ces ordinateurs pour faire des calculs « utiles »

## Principe

- ▶ Basé sur le volontariat
- ▶ Un utilisateur voulant participer télécharge un logiciel et l'installe sur son ordinateur
- ▶ Le logiciel se connecte à un serveur et demande des tâches à traiter
  - ▶ Architecture très centralisée (maître – esclave)
  - ▶ Pas de communications entre les pairs
- ▶ Possibilité de paramétrer le client
  - ▶ Démarrage à heures fixes ou quand la machine est inutilisée
- ▶ En général, le client fait économiseur d'écran
  - ▶ Affiche au volontaire quelque chose...
  - ▶ Se déclenche automatiquement

## Bénéfices

- ▶ Pour les fournisseurs de calculs
  - ▶ Accès à des ressources énormes
  - ▶ Peu de frais (coût des serveurs et de la bande passante)
- ▶ Pour les utilisateurs
  - ▶ La satisfaction de participer à une grande cause
  - ▶ La gloire! (classements journalier)
  - ▶ L'argent parfois

## Seti@home

- ▶ Projet pionnier, démarré en
- ▶ Des radiotélescopes écoutent le ciel à la recherche de signaux extra terrestres
- ▶ Énorme quantité de données
- ▶ Analyse du signal par FFT
  - ▶ Répétitif
  - ▶ Pas très bien géré par les processeurs généraux (x86)
- ▶ Le logiciel client télécharge des données correspondant à une zone du ciel à une date donnée
- ▶ Recherche de signaux anormaux
- ▶ Résultats:
  - ▶ Très peu de signaux découverts
  - ▶ Chercheurs en colère : seti@home est plus sexy que d'autres projets, donc bénéficie de plus de crédits et de temps de radiotélescope
- ▶ Récompenses:
  - ▶ Classement régulier
  - ▶ La gloire

## Distributed.net

- ▶ Utilisé pour casser des chiffrements symétriques
- ▶ Explore l'espace des clés
- ▶ Un client télécharge un ensemble de clés à tester
  - ▶ Un client rapide testera plus de clés et aura donc plus de chances
- ▶ Financé par de grandes compagnies
  - ▶ RSA
- ▶ Récompense
  - ▶ Classement
  - ▶ La gloire
  - ▶ L'argent

## Problèmes

- ▶ Le logiciel de calcul peut contenir des bugs
- ▶ Comme tout système ayant une récompense (même symbolique), des gens trichent
- ▶ Un client n'effectue pas le calcul demandé
  - ▶ Renvoie de résultats anciens ou aléatoires
  - ▶ Permet de simuler une machine très rapide
- ▶ Un client renvoie un résultat faux
  - ▶ Bug dans le client
  - ▶ Tentative de sabotage

## Clients ne calculant pas

- ▶ Un client retourne toujours « pas la bonne clé » dans distributed.net
- ▶ Un client retourne « rien trouvé » dans seti@home
- ▶ Solution : Spot Checking
  - ▶ Faire faire à un client un calcul dont on est certain du resultat
- ▶ On fait faire par plusieurs clients différents le même calcul
  - ▶ On garde le résultat le plus couramment trouvé
- ▶ Problème: Comment choisir des clients différents?
  - ▶ Problèmes de clients collaborant pour tricher
  - ▶ Quelle est le % de tels clients qui fausseront le calcul?

## Clients ne calculant pas

- ▶ Autre attaque, beaucoup plus évoluée, utilisée contre seti@home
  - ▶ Un client reçoit un bloc de données à analyser
  - ▶ Il effectue 95% de l'analyse
  - ▶ Il envoie à ses amis l'analyse partielle ainsi que le bloc original
  - ▶ Ceux-ci peuvent finir l'analyse et être crédités du travail
- ▶ Comment est-ce possible?
  - ▶ Seti@home n'associe pas un client avec une tâche
- ▶ L'association a un coût élevé

## Résultats faux

- ▶ **Volontaire:**
  - ▶ Spot checking
- ▶ **Bug : très difficile à détecter**
  - ▶ Les clients différents donneront le même résultat faux
- ▶ **Aléatoire:**
  - ▶ Un résultat faux peut être le résultat de circonstances passagères (bug d'un autre logiciel, problème réseau...)
  - ▶ On ne veut pas se priver de participants trop rapidement
  - ▶ Mise en place de grey/black lists
  - ▶ Un client en liste grise sera testé plus souvent (période de probation)
  - ▶ Un client en liste noire sera interdit de travailler
- ▶ **Limitations:**
  - ▶ On ne sait pas bien identifier un client
  - ▶ Login : la création de comptes est gratuite et facile
  - ▶ IP : IP dynamiques ou utilisation de proxy

## 2 - Pair à Pair de fichiers

# Principe

- ▶ Le P2P de fichier est né du « besoin » de partager
  - ▶ Mettre à disposition des autres des fichiers
  - ▶ Extension du « cercle des amis »
- ▶ Chaque pair met à disposition des autres des fichiers
  - ▶ Les downloads se font de pair à pair
- ▶ Principalement utilisé à la limite (hors) de la légalité
- ▶ Mais fort potentiel
  - ▶ Distribution de fichiers sans serveur centralisé (exemple: patch dans WoW)
  - ▶ Récupération des espaces libres sur plusieurs machines dans un LAN
- ▶ Principaux problèmes
  - ▶ Recherche
  - ▶ Passage à l'échelle, tolérance aux pannes
  - ▶ Équilibre entre download et uploads
  - ▶ Pollution

# Problèmes

- ▶ Recherche
  - ▶ Mécanismes pour trouver le fichier recherché
  - ▶ Efficacité
    - ▶ Utilisation de la bande passante
    - ▶ Nombre de réponses
    - ▶ Temps
- ▶ Passage à l'échelle
  - ▶ Combien de pairs le réseau peut supporter en restant efficace
- ▶ Résistance aux pannes
  - ▶ Combien de pairs peuvent disparaître avant que le réseau ne soit plus efficace?
  - ▶ Liens avec la théorie de la percolation (étude à l'échelon global des effets d'actions locales)
- ▶ Équilibre down/up
  - ▶ Partage de fichiers, implique... partage :)
  - ▶ Comment inciter certains à partager?
- ▶ Pollution
  - ▶ Détecter les fichier corrompus
  - ▶ Les éliminer du réseau

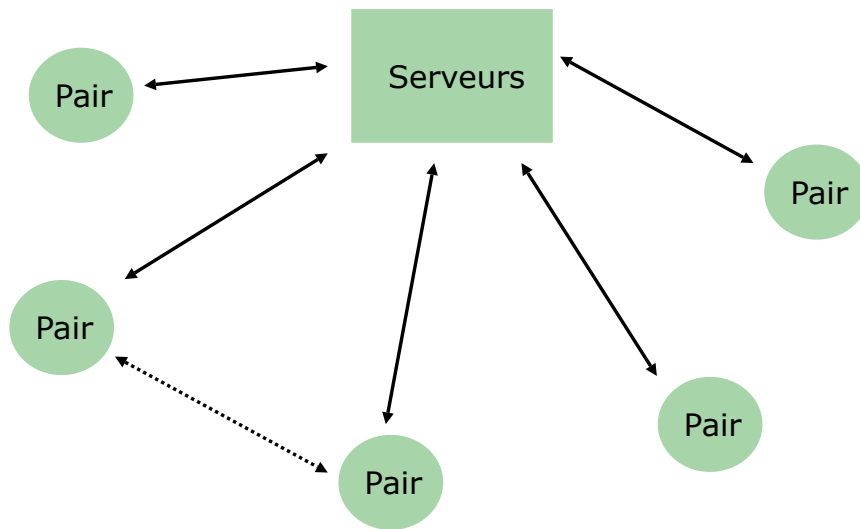
## 2 - Pair à Pair de fichiers

### 2.1 Napster

## Napster

- ▶ Premier systèmes P2P
  - ▶ Inventé par Shawn Fanning en 1999
  - ▶ Utilise des serveurs centralisés pour l'indexation
  - ▶ Transferts de fichiers en P2P
- ▶ Vision utilisateur
  - ▶ Les serveurs sont transparents, tout se passe comme si le partage se faisait de pair à pair
- ▶ Pas de tolérance aux pannes
  - ▶ Un fichier était chargé complètement depuis un autre pair
    - ▶ Si disparition du pair, disparition du fichier
  - ▶ Dépendant des serveurs
    - ▶ Les détenteurs de droits ont fait fermer les serveurs, tuant le réseau
- ▶ Passage à l'échelle
  - ▶ Utilisation de serveurs multiples

# Napster



↔ Offre/Recherche  
↔ Téléchargement

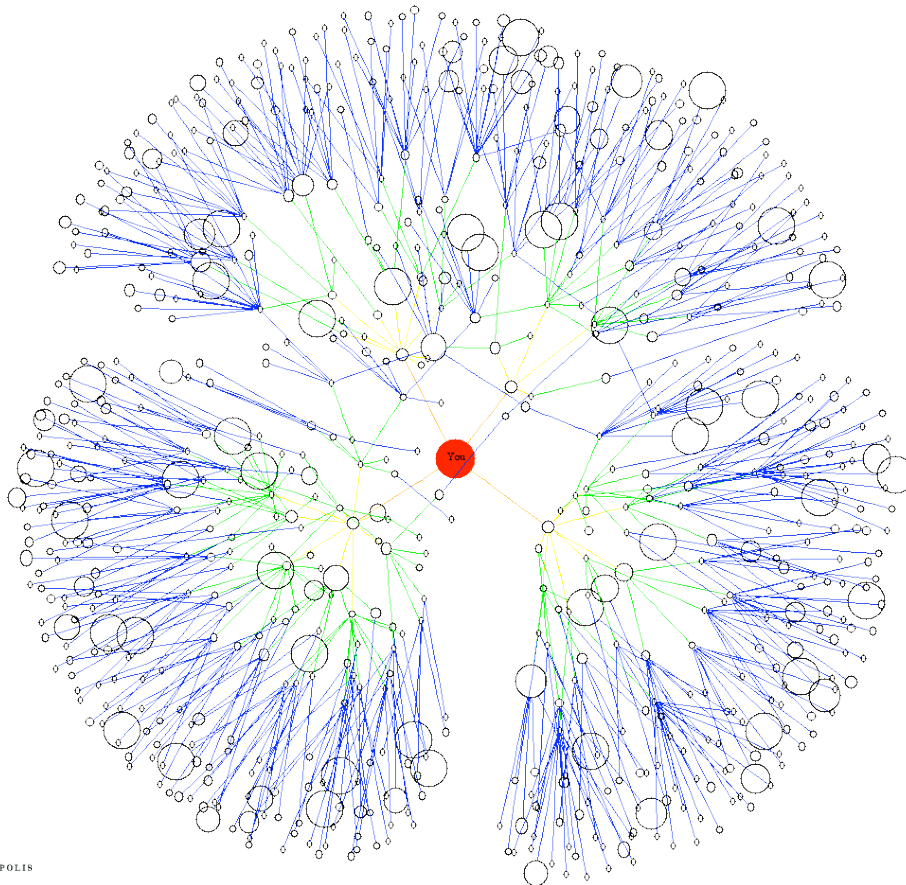
## 2 - Pair à Pair de fichiers 2.2 Gnutella

# Gnutella

- ▶ Système Pair à Pair totalement décentralisé
- ▶ Gnutella est un protocole
  - ▶ Pour partager des fichiers sur un réseau (Internet)
  - ▶ Construit comme remplace décentralisé de Napster
- ▶ Construit pour être tolérant aux pannes
  - ▶ “Gnutella can withstand a nuclear war and an army of lawyers”.
- ▶ Spécification ouvertes, pas de standardisations
  - ▶ Plein de clients plus ou moins compatibles
  - ▶ Ajout de fonctionnalités
- ▶ Les plus connus: BearShare, LimeWire, Gnucleus

# Principes

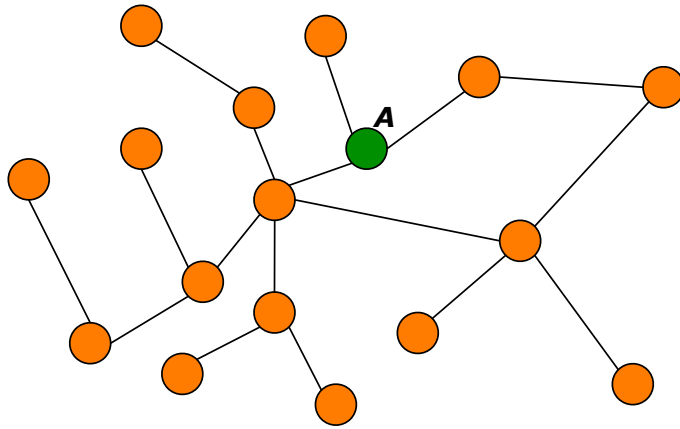
- ▶ Tous les nœuds sont égaux et ont 3 rôles
  - ▶ Producteur: partage de fichiers avec les autres nœuds
  - ▶ Consommateur : téléchargement de fichiers
  - ▶ Routeur: Maintient de l’infrastructure du réseau
- ▶ Chaque nœud a un ensemble (limité) de voisins avec lesquels il peut communiquer
  - ▶ Formation d’un réseau logique (overlay network)
  - ▶ Pas forcément adapté au réseau physique
- ▶ Les messages envoyés servent à
  - ▶ Maintenir l’infrastructure
  - ▶ Chercher/annoncer des fichiers
- ▶ Un nœud est obligatoirement un routeur, même si aucun fichier n’est partagé
- ▶ Le transfert de fichiers se fait entre deux nœuds via http
- ▶ Routage = inondation



## Inondation

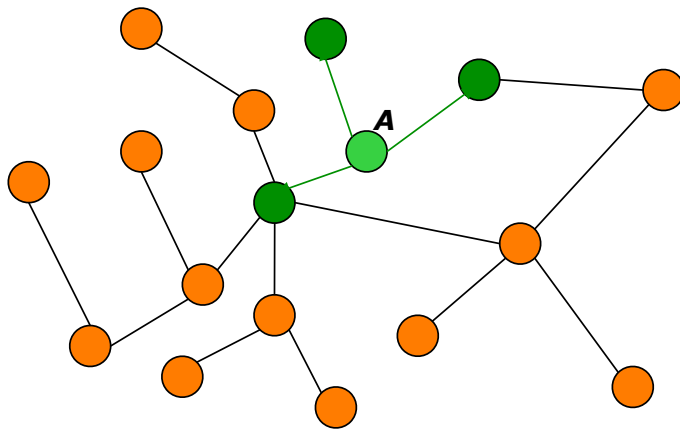
- ▶ **Aucun vision centralisée**
  - ▶ Les nœuds ne connaissent que leurs voisins directs
  - ▶ Aucun serveur d'indexation
- ▶ **Pour communiquer avec des nœuds autres que ses voisins, on inonde**
  - ▶ On envoie le message à tout ses voisins
  - ▶ Qui l'envoient à leurs voisins...
- ▶ **Difficultés**
  - ▶ Un nœud risque de recevoir plusieurs fois le même message (cycle dans le réseau)
  - ▶ Si pas de limitations, risque de surcharger le réseau
- ▶ **Chaque nœud se souvient des messages qu'il a fait transiter**
  - ▶ Cache limité dans le temps
  - ▶ Casse les cycles
- ▶ **On fixe un nombre, le TTL (Time To Live)**
  - ▶ Un message est envoyé avec une valeur bien choisie
  - ▶ Quand un nœud reçoit un message, il décrémente le TTL
  - ▶ Si  $TTL > 0$ , le message est envoyé aux voisins, sinon la retransmission s'arrête
- ▶ **Cela limite la visibilité**
  - ▶ Un nœud ne peut atteindre que les nœuds qui sont à TTL sauts au maximum
  - ▶ C'est l'horizon

## Inondation (étape 1)



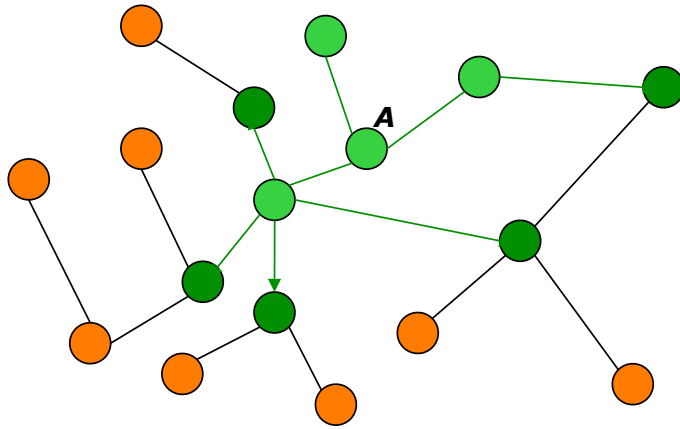
Le nœud A envoie un message de recherche, avec un TTL de 3

## Inondation (étape 2)



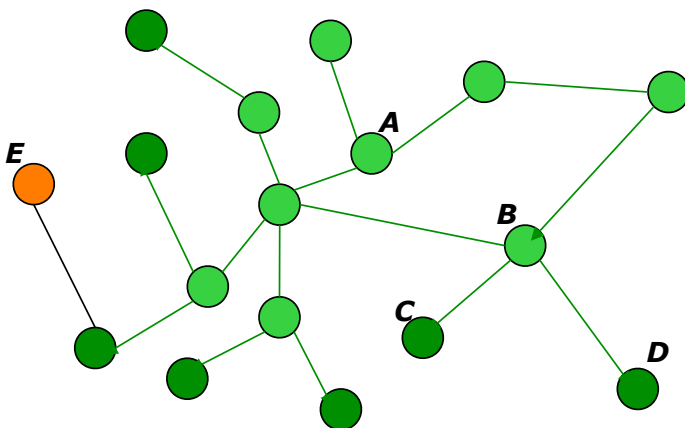
Le TTL est décrémenté par les autres clients, il vaut maintenant 2

## Inondation (étape 3)



Le TTL est à nouveau décrémenté, il vaut 1

## Inondation (étape 4)



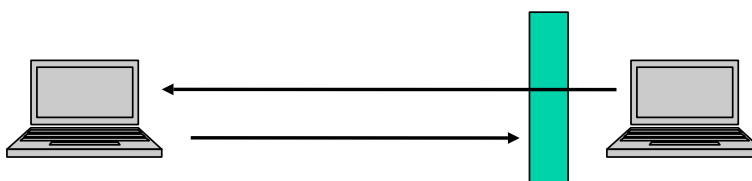
Le nœud B reçoit le message pour la seconde fois, il ne le retransmet pas  
Le TTL vaut 0, la transmission s'arrête, le nœud E ne reçoit pas le message

## Description du protocole

- ▶ 5 types de messages
  - ▶ Ping: un nœud prévient de son existence et demande à ses voisins de s'identifier
    - ▶ Transmis par inondation
  - ▶ Pong: réponse à un Ping
    - ▶ Seulement envoyé dans la direction du ping
  - ▶ Query: recherche de fichiers répondant à des critères
    - ▶ Transmis par inondation
  - ▶ QueryHit: réponse à une query. Contient la liste des URLs où le fichier peut être téléchargé
    - ▶ Envoyé seulement dans la direction de la query
  - ▶ Push: Demande au serveur d'initier le transfert de fichier pour limiter les problèmes de pare-feu.

## Gestion des pare-feu

- ▶ Un pare feu est un programme qui effectue un filtrage au niveau réseau
- ▶ Filtrage en entrée et en sortie
  - ▶ Entrée : autorise une machine distante à contacter un programme local
  - ▶ Sortie : autorise un programme local à contacter une machine distante
- ▶ Souvent, aucun filtrage en sortie, mais blocage en entrée
  - ▶ Une machine distante ne peut contacter un programme local
  - ▶ Il faut que ça soit le programme qui initie la connexion
  - ▶ Le push est transmis sur une connexion existante et permet de downloader un fichier sur une machine derrière un pare-feu



## Connexion à un réseau gnutella

- ▶ Comment se connecter à un réseau Gnutella?
- ▶ Il faut pouvoir se connecter à quelqu'un, mais comment le trouver?
  - ▶ Des nœuds sont « bien connus » et servent d'entremetteurs
  - ▶ Ex: gnutellahost.com:6346, router.limewire.com:6346
- ▶ Un client se connecte à un nœud gnutella
- ▶ Il envoie ensuite un ping
  - ▶ Celui-ci se propage par inondation
- ▶ Les pairs recevant le ping peuvent répondre un pong
  - ▶ Ces pongs contiennent une référence vers eux
  - ▶ Ces nœuds deviendront nos voisins directs dans le réseau
- ▶ En général, les entremetteurs ferment la connexion après le retour des pongs, ils ne participent pas au transfert de fichiers
- ▶ Plus un nœud reste longtemps dans le réseau, plus il devient connecté

## Problèmes

- ▶ L'inondation ne passe pas à l'échelle
  - ▶ Quelle que soit la taille du réseau, un nœud est limité par son horizon (en général TTL=7)
- ▶ La recherche de fichiers se fait par nom
  - ▶ Deux fichiers aux noms différents peuvent avoir le même contenu
  - ▶ Très sensible à la pollution
- ▶ Les messages d'architecture constituent une part importante du trafic de chaque nœud
- ▶ Les nœuds sont identifiés par IP et port
  - ▶ Impossible de retrouver un nœud qui a subi une panne temporaire
- ▶ Protocole de download primitif (http)
  - ▶ Source unique
  - ▶ Pas de download en parallèle

## Quelques faits

- ▶ La popularité des recherches suit la même loi que les pages webs
  - ▶ Les technologies de cache ont de bons résultats
  - ▶ Mettre en cache les résultats de recherches pendant 30 min réduit le trafic de 50%
- ▶ Free riding (leechers)
  - ▶ 66% des nœuds ne partagent aucun fichier
  - ▶ 73% des nœuds partagent moins de 10 fichiers
  - ▶ 37% des fichiers sont partagés par 1% des nœuds
  - ▶ Encore très proche du client/serveur
- ▶ Certains nœuds prennent une grande importance dans le réseau
  - ▶ Ils partagent beaucoup de fichiers
  - ▶ Ils ont beaucoup de connexions
- ▶ Rejoint une structuration à la Napster

## Le monde est petit

- ▶ En 1967, Stanley Milgram effectue une expérience
  - ▶ Il écrit 160 lettres qu'il donne au hasard à des gens d'Omaha, Nebraska
  - ▶ Il leur demande de les faire parvenir à un courtier de New-York
  - ▶ Interdiction d'utiliser la poste, juste passer la lettre de main en main, à des connaissances bien choisies pour se rapprocher du courtier
- ▶ À la fin, 42 lettres sont arrivées à destination
- ▶ En moyenne, les lettres sont passées entre les mains de seulement 5.5 personnes
- ▶ Cela montre que le réseau social des États-Unis (200 millions de personnes) a un diamètre d'environ 6 personnes
  - ▶ Certaines personnes ont beaucoup de relations, ce sont des ponts entre les communautés
  - ▶ Dans l'expérience de Milgram, 25% des lettres sont passées par le même marchand, 50% par les mêmes trois personnes
- ▶ Un petit nombre de ponts suffit à réduire énormément le nombre de sauts: small world effect

## 2 - Pair à Pair de fichiers

### 2.3 Edonkey

## Edonkey

- ▶ Similaire à Napster
  - ▶ Serveurs utilisés pour l'indexation
  - ▶ Transfert direct entre les clients
- ▶ Données identifiées par des valeurs de hachage
- ▶ Clients sont identifiés par des IDs
  - ▶ HighID : Hachage réversible de l'adresse IP si pas de pare-feu
  - ▶ LowID : Nombre aléatoire autrement
- ▶ Chaque client est connecté à un seul serveur en TCP/IP
  - ▶ Connexions multiples en UDP

## Edonkey

- ▶ Le contenu à partager est divisé en morceaux (chunk)
- ▶ Un client peut télécharger des morceaux en parallèle de plusieurs clients
- ▶ Une fois téléchargé, l'intégrité d'un morceau est vérifiée
  - ▶ Utilisation des informations de hachage
- ▶ Chaque morceau téléchargé est automatiquement mis en partage
  - ▶ Obligatoire par le logiciel, rien dans le protocole

## Téléchargement

- ▶ Le client A envoie une requête à un serveur
- ▶ Le serveur répond avec une liste d'ID de clients
  - ▶ Si highID, connexion directe possible
  - ▶ Si lowID, la connexion est établie par l'intermédiaire du serveur
    - ▶ A envoie une requête au serveur
    - ▶ Le serveur fait suivre cette requête au pair
    - ▶ Le pair ouvre d'ouvrir une connexion avec A
    - ▶ Le transfert a lieu
- ▶ Problèmes:
  - ▶ lowIDs augmentent la charge du serveur, et sont donc souvent rejetés
  - ▶ Si les 2 pairs sont en lowID, pas de connexion possible

## Limitations

- ▶ Un client ne voit que les fichiers disponibles sur le serveur auquel il est connecté
  - ▶ Requêtes possibles vers d'autres serveurs en UDP
  - ▶ Aucune garantie de réponse, et augmente la charge des serveurs
- ▶ Toujours des free-riders
  - ▶ Le ratio down/up est géré par le client
  - ▶ Très facile à contourner avec un client « maison » (mldonkey) ou du traffic-shapping
- ▶ Les pairs sont très faciles à identifier
  - ▶ Transformation highID vers IP possible
  - ▶ Liste des partages de chaque client disponible (mais peut être interdit)

## 2 - Pair à Pair de fichiers

### 2.4 Bittorrent

# Bittorrent

- ▶ Système P2P avec serveurs (trackers)
- ▶ Les trackers ne font que coordonner les clients
  - ▶ Aucune connaissance des données distribuées
  - ▶ Peu gourmand en ressources
- ▶ Les données distribuées sont décrites dans des fichiers .torrent contenant
  - ▶ Le nom du fichier
  - ▶ Sa taille
  - ▶ Des informations de hachage
  - ▶ L'URL du tracker
- ▶ Pas de mécanismes de recherche
- ▶ Le protocole essaie de forcer le partage
  - ▶ Probablement le plus réussi actuellement

# Bittorrent

- ▶ Pour télécharger, un client donne au tracker
  - ▶ Des informations sur ce qu'il recherche
  - ▶ Ses ports disponibles
- ▶ Le tracker retourne une liste de pairs ayant le même intérêt
- ▶ Les clients peuvent alors se connecter entre eux
- ▶ Les données sont divisées en morceaux (typiquement 250KB)
- ▶ Les morceaux peuvent être téléchargés chez n'importe quel pair
- ▶ Pour démarrer, un pair spécial ayant le contenu complet est nécessaire : la graine (seed)

# Bittorrent

- ▶ Plusieurs algorithmes pour sélectionner le prochain morceau à télécharger
- ▶ Priorité stricte (Strict Priority) : dès qu'un sous morceau est téléchargé, les autres sous morceaux de la même pièce le seront ensuite
- ▶ Plus rare (Rarest First): demande du morceau le plus rare en premier
  - ▶ Limite la charge sur la graine
  - ▶ Augmente la probabilité d'avoir le fichier complet
- ▶ Aléatoire(Random):
  - ▶ Au démarrage, rien à uploader, les pièces rares ne sont que sur quelques pairs, donc choix d'une pièce au hasard
- ▶ Fin de partie (EndGame):
  - ▶ Quand seulement quelques sous morceaux manquants, on les demande à tous les pairs
  - ▶ Une fois qu'un téléchargement commence depuis un pair, on annule notre demande aux autres

# Connexions

- ▶ Les connexions entre pairs ont plusieurs états
  - ▶ interested : indique qu'on veut downloader quelque chose
  - ▶ choked : indique qu'on refuse d'uploader
- ▶ Sert à maintenir des informations entre les pairs
- ▶ Utilisé pour décider du sens d'écoulement des informations
- ▶ Un pair download ssi
  - ▶ Il a indiqué interested
  - ▶ Il n'est pas choked par l'autre pair
- ▶ Un pair upload ssi
  - ▶ Il n'a pas choked l'autre pair
  - ▶ Si l'autre pair est interested
- ▶ Il est impératif de maintenir ces 2 états à jour
  - ▶ Un pair choked indiquera quand même qu'il est interested

## Connexions

- ▶ Un connexion est marquée interested si le pair a des pièces qui nous intéressent
- ▶ Un pair peut temporairement refuser d'uploader vers un autre pair: il l'étouffe (choke)
  - ▶ Sert à indiquer qu'on ne veut pas envoyer de données, si par exemple on ne download rien
  - ▶ Un client choke toujours une seed (la seed ne veut pas recevoir de données)
  - ▶ Utile pour maintenir des connexions ouvertes sans rien envoyer
- ▶ Les décisions d'étouffement sont prises toutes les 10 secondes
  - ▶ Évite de basculer trop rapidement entre choked/unchoked
- ▶ On dé étouffe les 4 pairs qui ont le meilleur upload (i.e envoient des fichiers vers vous avec le meilleur débit) et sont interested
  - ▶ Permet de maximiser le download
- ▶ Les pairs qui ont un meilleur upload mais ne sont pas interested sont dé étouffés
  - ▶ Si ils deviennent interested ils remplacent le(s) plus mauvais des 4 précédents

## Bittorent

- ▶ Recherche:
  - ▶ Utilisation de serveurs web recensant des fichiers torrent
  - ▶ Modération, élimination de la pollution
- ▶ Certains trackers forcent un ratio download/upload
  - ▶ Si en dessous du ratio, alors un pair ne peut plus downloader
- ▶ Rend moins facile la collecte d'informations
  - ▶ Seuls le tracker et le serveur web ont des informations globales
  - ▶ Il faut avoir le fichier .torrent et donc avoir une cible
  - ▶ Mais étant donné un tracker, très facile d'avoir la liste des pairs
- ▶ Modèle économique
  - ▶ Chaque pièce à une valeur dans le protocole
  - ▶ Une pièce rare sera demandée par beaucoup de clients, et donc permettra d'en downloader d'autres
  - ▶ Une pièce très répandue ne sera jamais needed

## 2 - Pair à Pair de fichiers

### 2.5 Freenet

## Freenet

- ▶ Système construit pour être tolérant en panne et respecter la confidentialité
- ▶ Ian Clarke à l'université d'Edinburgh (1999)
- ▶ Les documents sont chiffrés, découpés en petits morceaux, mis en caches et répliqués sur certains nœuds
- ▶ Les utilisateurs fournissent de l'espace disque et de la bande passante mais ne savent pas
  - ▶ Quels documents sont sur leurs machines
  - ▶ Où leurs documents sont stockés
- ▶ Les systèmes de chiffrement utilisés sont extrêmement robustes (cryptographie asymétrique)
- ▶ Freenet se concentre sur le stockage de fichiers, et non sur la recherche

## Freenet

- ▶ Le routage dans Freenet est auto-organisé
- ▶ Chaque fichier a une clé unique associée
- ▶ Chaque nœud garde localement une table de routage qui contient les clés des fichiers stockés chez leurs voisins
- ▶ Ajout d'un fichier
  - ▶ Une nouvelle clé est calculée
  - ▶ On recherche le nœud qui contient déjà des clés proches
  - ▶ Limitation par TTL
- ▶ Recherche de fichier
  - ▶ Principe similaire à l'ajout : passage d'une requête de nœuds en nœuds dans la bonne direction
  - ▶ Une recherche fructueuse provoque une mise à jour des tables de routage : adaptation
- ▶ Un document très demandé sera trouvé plus rapidement
- ▶ Si une recherche échoue, il est possible d'augmenter le TTL

## Freenet

- ▶ Un pair peut aussi être candidat pour stocker une partie d'un fichier (réplication)
- ▶ Avec ces mécanismes, Freenet converge vers une architecture à la Gnutella
  - ▶ Un petit nombre de nœuds a d'énormes tables de routage
- ▶ La connexion à un réseau Freenet se fait de manière similaire à Gnutella

## 3 -Pair à Pair structuré

## P2P structurés

- ▶ Idées : structurer les réseaux P2P
  - ▶ Éviter la centralisation à la Napster
  - ▶ Éviter la jungle de gnutella
- ▶ Solution : Assigner un contenu donné à un nœud donné
- ▶ La recherche est simplifiée : on sait à quel(s) pair(s) demander
- ▶ Extension des concepts derrière Freenet

## Tables de hachage distribuées

### ▶ Rappels:

- ▶ Une table de hachage est une structure de données qui associe une clé à une valeur
- ▶ Connaissant la clé, on peut retrouver la valeur en temps constant
- ▶ Même principe que le tableau, mais avec un espace de clés restreint
- ▶ Plusieurs façon de gérer les collisions (chaînage ou table ouverte)

## Tables de hachage distribuées

- ▶ Une table de hachage associe un élément et une clé
- ▶ L'élément est stocké dans la table à un indice correspondant à la valeur de hachage de la clé
- ▶ Si deux éléments ont la même valeur de hachage, alors ils sont placés dans une chaîne
- ▶ Dans une DHT, les cases des tables correspondent à des nœuds différents
- ▶ Dans un système P2P à base de DHT, le nombre de nœuds varie, donc le nombre de cases
- ▶ Solution: utilisation d'un hachage consistant
  - ▶ Le nombre de valeurs possibles ne dépend pas du nombre de noeuds
  - ▶ Un élément va sur le nœud dont la valeur de hachage est la plus proche