Le pair-à-pair

Fabien Mathieu

M2 Informatique
Marne-la-Vallée

29 janvier 2018

Plan du cours

1. Introduction
2. Les tables de hachage distribuées
3. Dimensionnement en bande passante
4. Streaming, problèmes de délai
5. Bonus
   - Un TP (Jupyter)
   - Exam de l'an dernier

Slides en ligne

https://github.com/balouf/UPEM-P2P

Partie I

Le pair-à-pair, c'est quoi ?

Sommaire

Définition par énumeration

Le pair-à-pair, «on» sait ce que c'est :

Partage de fichier BitTorrent, eMule,...

Diffusion TVants, PPLive,...

Voix sur IP Skype

Définition par énumeration

Le pair-à-pair, «on» sait ce que c'est :

Partage de fichier BitTorrent, eMule,...

Diffusion TVants, PPLive,...

Voix sur IP Skype

Ce qui n'est pas pair-à-pair aussi :

Non-réseau Office, Photoshop,...

Réseau Apache, SSH, browsers...

Définition par énumeration

Le pair-à-pair, «on» sait ce que c'est :

Partage de fichier BitTorrent, eMule,...

Diffusion TVants, PPLive,...

Voix sur IP Skype

Ce qui n'est pas pair-à-pair aussi :

Non-réseau Office, Photoshop,...

Réseau Apache, SSH, browsers...

Parfois, c'est flou :

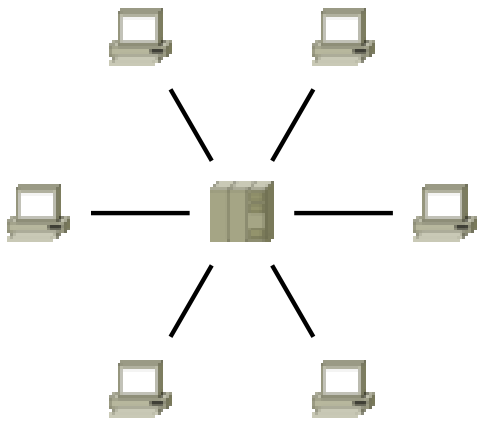
Calcul distribué BOINC project (Seti@home, Folding@home...)

Nuages Amazon, Wuala, ...

Définition réseau

La plupart des applications réseaux fonctionnent en client/serveur :

- D'un côté, le(s) serveur(s) fourni(t/ssent) des ressources,
- De l'autre, les clients consomment ces ressources.

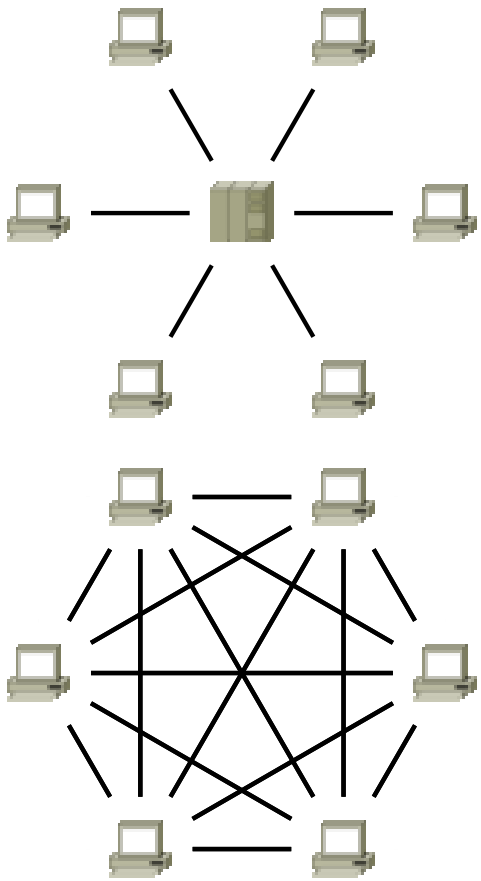


Définition réseau

La plupart des applications réseaux fonctionnent en client/serveur :

- D'un côté, le(s) serveur(s) fourni(t/ssent) des ressources,
- De l'autre, les clients consomment ces ressources.

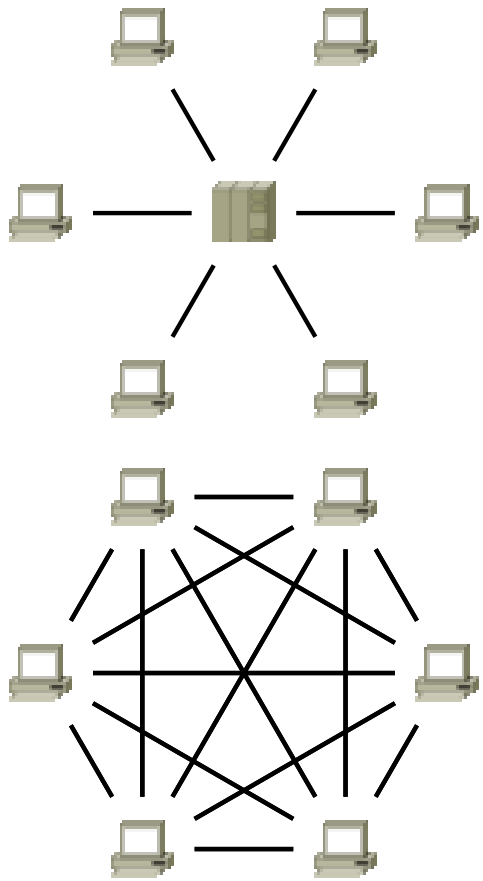
Définition réseau $\rightarrow$ casser la séparation client/serveur : tous les pairs peuvent être aussi bien clients que serveurs.



Définition réseau

La plupart des applications réseaux fonctionnent en client/serveur :

- D'un côté, le(s) serveur(s) fourni(t/ssent) des ressources,
- De l'autre, les clients consomment ces ressources.



Définition réseau $\rightarrow$ casser la séparation client/serveur : tous les pairs peuvent être aussi bien clients que serveurs.

Marche avec la plupart des exemples précédents ?

Définition réseau : et Skype ?

Skype est l'une des application P2P les plus connues. Pourquoi Skype est P2P ?

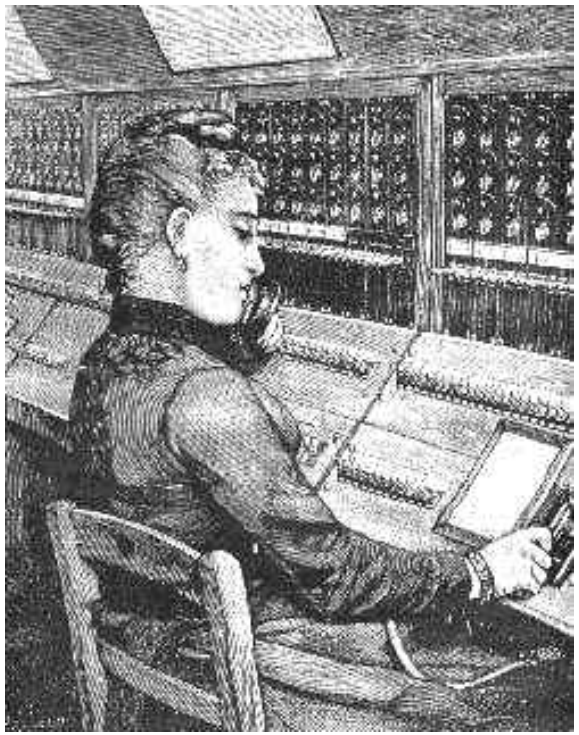
- Interaction de pair à pair ?

Existe depuis, très, très longtemps

Définition réseau : et Skype ?

Skype est l'une des application P2P les plus connues. Pourquoi Skype est P2P ?

- Interaction de pair à pair ?
- Interactions entre n'importe quelle paire de pairs ?



Existe depuis longtemps

Définition réseau : et Skype ?

Skype est l'une des application P2P les plus connues. Pourquoi Skype est P2P ?

- Interaction de pair à pair ?
- Interactions entre n'importe quelle paire de pairs ?
- Par les créateurs de KaZaA et Joost (cf plus tard) ?

Retour à l'énumération

Définition réseau : et Skype ?

Skype est l'une des application P2P les plus connues. Pourquoi Skype est P2P ?

- Interaction de pair à pair ?
- Interactions entre n'importe quelle paire de pairs ?
- Par les créateurs de KaZaA et Joost (cf plus tard) ?
- Peut-être que la définition réseau a ses limites ? ?

Définitions alternatives : méthodologie, structure

Interlude : une définition légale ?

Le P2P peut être utilisé à des fins non légales. DADVSI tente d'interdire tout «logiciel manifestement destiné à la mise à disposition du public non autorisée d'œuvres ou d'objets protégés».

Interlude : une définition légale ?

Le P2P peut être utilisé à des fins non légales. DADVSI tente d'interdire tout «logiciel manifestement destiné à la mise à disposition du public non autorisée d'œuvres ou d'objets protégés».

- Peut recouvrir le partage P2P (data et multimedia)
- Ne recouvre pas (trop) Skype
- Couvre la plupart des applications réseaux (incluant l'approche client/serveur) : Apache, SSH...

Interlude : une définition légale ?

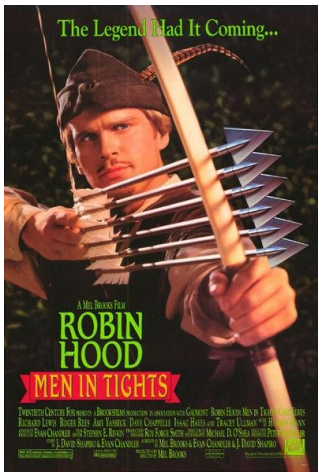
DADVSI n'est pas vraiment applicable. Prise de conscience de la difficulté de définir (légalement) un usage P2P illégal. Passage à Hadopi.

Interlude : une définition légale ?

DADVSI n'est pas vraiment applicable. Prise de conscience de la difficulté de définir (légalement) un usage P2P illégal. Passage à Hadopi.

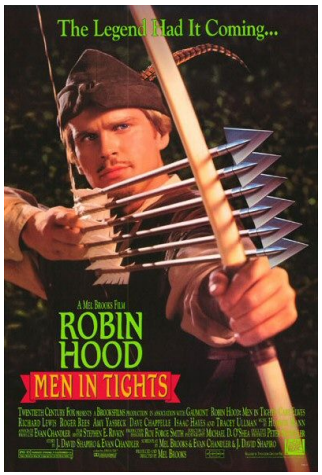
- Surveillance de contenu cibles (pots de miel)
- Preuve d'un véritable téléchargement par personne physique impossible à grande échelle
- Mise en place du délit de négligence caractérisée
- Échec technique, succès politique ?

Définition méthodologique



Le P2P utilise les ressources disponibles auquel il a accès (CPU, disque, bande passante) où qu'elles soient, et les redistribue à ceux qui en ont besoin.

Définition méthodologique

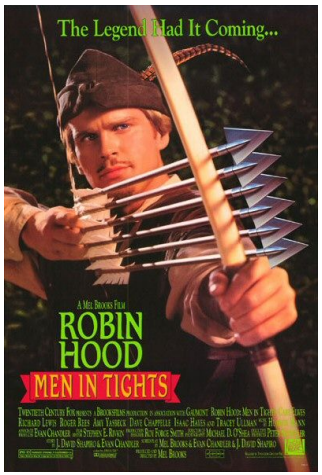


Le P2P utilise les ressources disponibles auquel il a accès (CPU, disque, bande passante) où qu'elles soient, et les redistribue à ceux qui en ont besoin.

Deux variantes suivant ce que recouvre «où» :

- Chez n'importe quel utilisateur (inclus BOINC)
- N'importe où dans les réseaux (CCN, . . .)

Définition méthodologique



Le P2P utilise les ressources disponibles auquel il a accès (CPU, disque, bande passante) où qu'elles soient, et les redistribue à ceux qui en ont besoin.

Deux variantes suivant ce que recouvre «où» :

- Chez n'importe quel utilisateur (inclus BOINC)
- N'importe où dans les réseaux (CCN, . . .)

Skype : borderline

Définition structurelle

Structure : comment les éléments d'un système sont agencés, et par extension, interagissent.
En réseau, on peut définir par structure la réponse à la question

Qui donne quoi à qui ?

Clients/serveurs, téléphone,. . . ont généralement des structures triviales.
Définition structurelle : le P2P est caractérisé par des structures réseaux non triviales.
Exclut les interactions triviales (Skype ?)

Définitions : récapitulatif

Il existe beaucoup de définitions possibles du P2P

Énumération On sait bien ce que c'est

Légal Confond technique et légalité de l'usage

Réseau N'est pas client/serveur

Méthodologique Recycle des ressources inutilisées

Structurelle Interactions non triviales

Recommandations

- Il n'y a pas de meilleure réponse (il y en a de moins bonnes)
- Utilisez votre tête !
- Est-ce que Skype est P2P ? C'est comme vous (et Skype) voulez !

Définition pour le cours

- Ce cours est centré sur la distribution (illégale ?) de contenu
- La plupart des définitions marchent
- Étude non-exhaustive

Sommaire

Overlay

Le modèle OSI

Le réseau est souvent découpé en 7 couches :

- Couche 7 (Application)
- Couche 4 (Connectivité point-à-point robuste)
- Couche 3 (Routage)

P2P

- Créer un réseau virtuel (overlay) en surcouche du réseau physique
- Overlay $\rightarrow$ graphe
- Émuler les couches routage / connectivité / . . . sur la topologie virtuelle
- Un voisin physique n'est pas forcément un voisin virtuel (et réciproquement !)

Overlay : deux grandes familles

Graphes structurés

Maintenir une certaine forme avec de bonnes propriétés

- Performances garanties par la structure du graphe
- Maintenance faible si réseau stable
- Fragile, maintenance forte si réseau très dynamique
- Peu adapté aux réseaux hétérogènes

Graphes non-structurés

Basés sur le hasard, l'auto-adaptation

- Robuste
- Adapté à l'hétérogénéité
- Maintenance constante (élevée si réseau stable)
- Pas de garantie immédiatement déduite du graphe

Passage à l'échelle

En client/serveur

- la capacité de traitement est fixe
- trop de clients $\rightarrow$ DoS

En P2P

- La capacité de traitement augmente avec le nombre de clients
- $\rightarrow$ le P2P peut gérer un nombre arbitraire de clients : Passage à l'échelle / scalabilité

Plutôt réelle, la scalabilité n'est cependant pas une propriété absolue et inébranlable

- Réseau physique sous-jacent
- Contraintes de QoS
- La taille compte en pratique (maintenance, effet logarithmique)

Notion d'équité (fairness)

Dans l'idéal

En P2P, chaque pair peut servir et être servi par n'importe qui : le P2P est isotropique.

En pratique

- Il n'est pas toujours possible de servir n'importe quoi à n'importe qui
- "All peers are equal, but some peers are more equal than others" (Orwell, the Server Farm)

$\rightarrow$ le P2P est isotropique a priori au moment d'intégrer le système

Ressources et objectifs

Ressources

- CPU
- Disque vide
- Disque rempli (contenu)
- Bande passante

Objectifs

- Calcul distribué
- Distribution de contenu
- Téléphonie (Skype)
- Gaming

Ressources et objectifs

Ressources

- CPU
- Disque vide
- **Disque rempli (contenu)**
- **Bande passante**

Objectifs

- Calcul distribué
- **Distribution de contenu**
- Téléphonie (Skype)
- Gaming

Distribution de contenu

Trois grandes familles

Partage de fichiers (filesharing)
Distribution générique

- Objectif principal : récupérer le contenu
- Objectifs secondaires : disponibilité, temps de téléchargement

Distribution de contenu

Trois grandes familles

À-la-Demande N'importe quoi n'importe quand

- Catalogue existant a priori (comme pour le filesharing)
- Minimiser l'initialisation (start-up delay)

Distribution de contenu

Trois grandes familles

Live Direct / Programme

- Tout le monde veut la même chose en même temps
- Minimiser le décalage (play-out-delay)

Distribution de contenu

Trois grandes familles

Famille	Partage simple	À-la-Demande	Live
Streaming ?	Non	Oui	Oui
Qualité requise	Minimale	N'importe quoi, N'importe quand	Small delay
Avantage technique	Trafic élastique	Contenu connu	Comportement homogène
Difficulté technique	Minimale	Comportement hétérogène	Anticipation impossible

Indexation/Distribution

Quel soit le système (P2P) de distribution de contenu, il y a deux questions à résoudre :

Qui possède ce que je veux ?

- Construire / maintenir un index
- Résoudre à la volée
- Utiliser une solution centralisée

Comment je récupère ce que je veux ?

- Sans réfléchir
- Intelligemment

Sommaire

Pré-histoire du P2P : communications

Quelques dates-clés

-200000 ?	Parole
-30000 ?	Peinture
-7000 ?	Écriture
-600 → -100	Longue distance (chevaux/pigeons/sémaphores)
1454	Gutenberg
1876	Téléphone
1969	Arpanet
1993	MP3
1999	ADSL (en France), DivX 3.11

Pré-histoire du P2P : communications

Quelques dates-clés

-200000 ?	Parole
-30000 ?	Peinture
-7000 ?	Écriture
-600 → -100	Longue distance (chevaux/pigeons/sémaphores)
1454	Gutenberg
1876	Téléphone
1969	Arpanet
1993	**MP3**
1999	**ADSL (en France), DivX 3.11**

Seuil technologique

- Vers 2000, on peut récupérer du contenu en temps raisonnable

	Main	Impr.	Modem	DSL 512k	DSL 20M
Livre	2-3j	$\frac{1}{2}$h	1m	6s	0.15s
Tube (WAV)	$\frac{1}{2}$a	1-2j	1h	6m	10s
Film (DVD)	100a	1a	10j	1j	<1h
Tube (MP3)	20j	3h	6m	30s	1s
Film (DivX)	12a	40j	1j	3h	5m

Seuil technologique

- Vers 2000, on peut récupérer du contenu en temps raisonnable
- Une autre métrique intéressante : peut-on «streamer» ?

	Main	Impr.	Modem	DSL 512k	DSL 20M
Livre	Non	Oui	Oui	Oui	Oui
Tube (WAV)	Non	Non	Non	Presque	Oui
Film (DVD)	Non	Non	Non	Non	Oui
Tube (MP3)	Non	Non	Presque	Oui	Oui
Film (DivX)	Non	Non	Non	Presque	Oui

Seuil technologique

Vers 2000

- Les contenus numériques deviennent la norme
  - CDs, MP3
  - DVDs, DeCSS, DivX
- Le «haut débit» rend la réception par Internet réaliste
- Pas de YouTube, MegaUpload, . . .
- Quelques solutions marginales
  - Newsgroups (`alt.binaries.*`)
  - FTPz
  - IRC
  - Myspace

Seuil technologique

Vers 2000

- Les contenus numériques deviennent la norme
  - CDs, MP3
  - DVDs, DeCSS, DivX
- Le «haut débit» rend la réception par Internet réaliste
- Pas de YouTube, MegaUpload, . . .
- Quelques solutions marginales
  - Newsgroups (`alt.binaries.*`)
  - FTPz
  - IRC
  - **Myspace**

Interlude : Myspace, un MegaUpload (trop ?) précoce

Succès

- MySpace offrait 300MB «dans le nuage»
- Création illimitée de comptes
- Pas de Captcha
- → Programmes automatiques
  - Découpage du contenu, création de compte, envoi
  - Infos de reconstructions disséminées dans un petit fichier

Interlude : Myspace, un MegaUpload (trop ?) précoce

Regrettably, Myspace is discontinuing its free consumer service. We will be bringing the service back online so that you can get your files. The service will be available from 12:00PM PST Tuesday May 29 until 5:00 PM PST Friday May 31 inclusive. THIS WILL BE THE ONLY PERIOD OF TIME YOU WILL BE ABLE TO RETRIEVE YOUR FILES! After this date, Myspace free consumer site will be closed. Myspace customers will not be able to access their accounts, and, to ensure your privacy all stored files WILL BE DELETED.

During this period, you will be able to either download your files or use Myspace's CD Burning service and have a CD with your files mailed to you. The CD Burning option is only available to customers in North America.

Thank you for using Myspace. We appreciate all our customers' support and hope that you enjoyed using the service.

La Chute

- Revenus :
  - Pub : peu de revenus via Internet
  - Modèle Freemium : pas dans les mœurs
- Dépenses :
  - Stockage : très cher
  - Bande passante : extrêmement cher
  - Démultipliées par l'automatisation
- $\rightarrow$ situation financièrement impossible

Interlude : Myspace, un MegaUpload (trop ?) précoce

Quelles leçons peut-on tirer de MySpace ?

- Certains internautes veulent télécharger du contenu (légal ?)
- Trop tôt pour qu'une solution commerciale soit intéressante (une décade)

Des conditions idéales

En résumé, vers 2000, tout est prêt pour faire du P2P

- Faisabilité : lien descendant suffisant, lien montant non utilisé
- Motivation : récupérer du contenu
- Opportunité : pas de réel compétiteur

P2P : Logiciels clés

1998	Napster : distribution de contenu P2P(MP3) distribution
2000	Gnutella : recherche décentralisée
2000	KaZaA/eDonkey : indexation hiérarchique
2003	BitTorrent : distribution performante
2006	Fin de Razorback 2.0 : émergence des DHTs
2006-2009	PPLive/UUSee/TVAnts/. . . : P2P streaming

P2P : Logiciels clés

1998 **Napster : distribution de contenu P2P(MP3) distribution**

2000 Gnutella : recherche décentralisée

2000 KaZaA/eDonkey : indexation hiérarchique

2003 BitTorrent : distribution performante

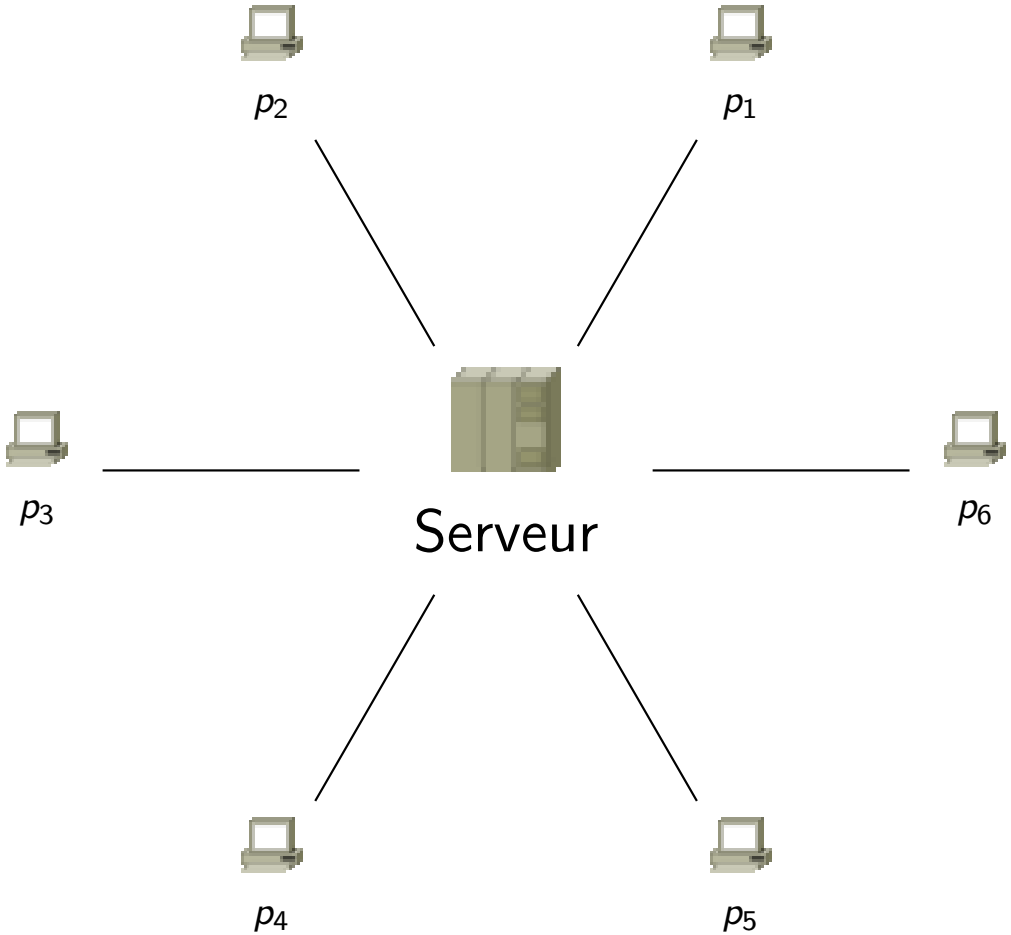
2006 Fin de Razorback 2.0 : émergence des DHTs

2006-2009 PPLive/UUSee/TVAnts/. . . : P2P streaming

Napster

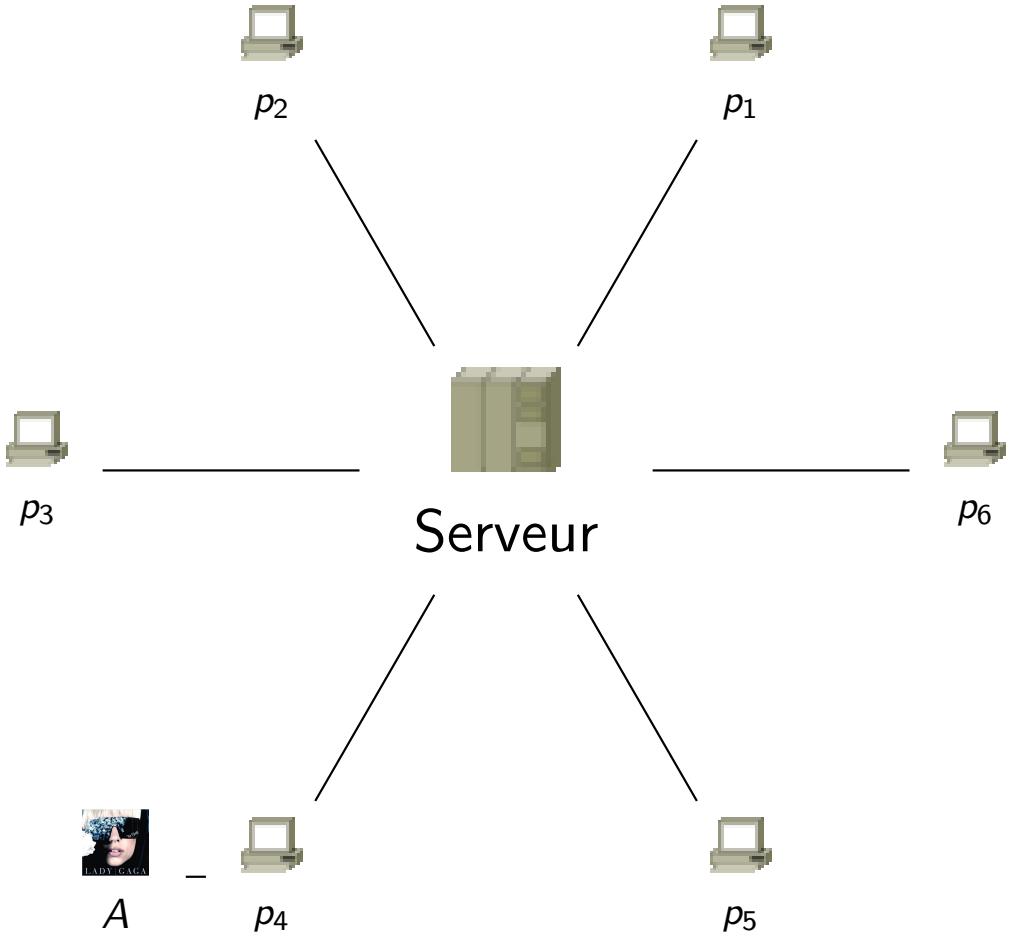
- Inventé en 1998 by Shawn "Napster" Fanning (The Italian Job) et Seam Parker (appears in The social network)
- Procès en 1999 (MPAA) - fermeture en juillet 2001
- Fonctionne en trois étapes :
  1. Indexation : les utilisateurs envoient leur index de MP3 (catalogue) à un serveur, `napster.com`
  2. Requête : demande d'un nom de fichier auprès du serveur ; retour d'un ensemble de clients (adresses) possédant un fichier compatible avec la requête
  3. Distribution : télechargement auprès d'un client de la liste

Napster



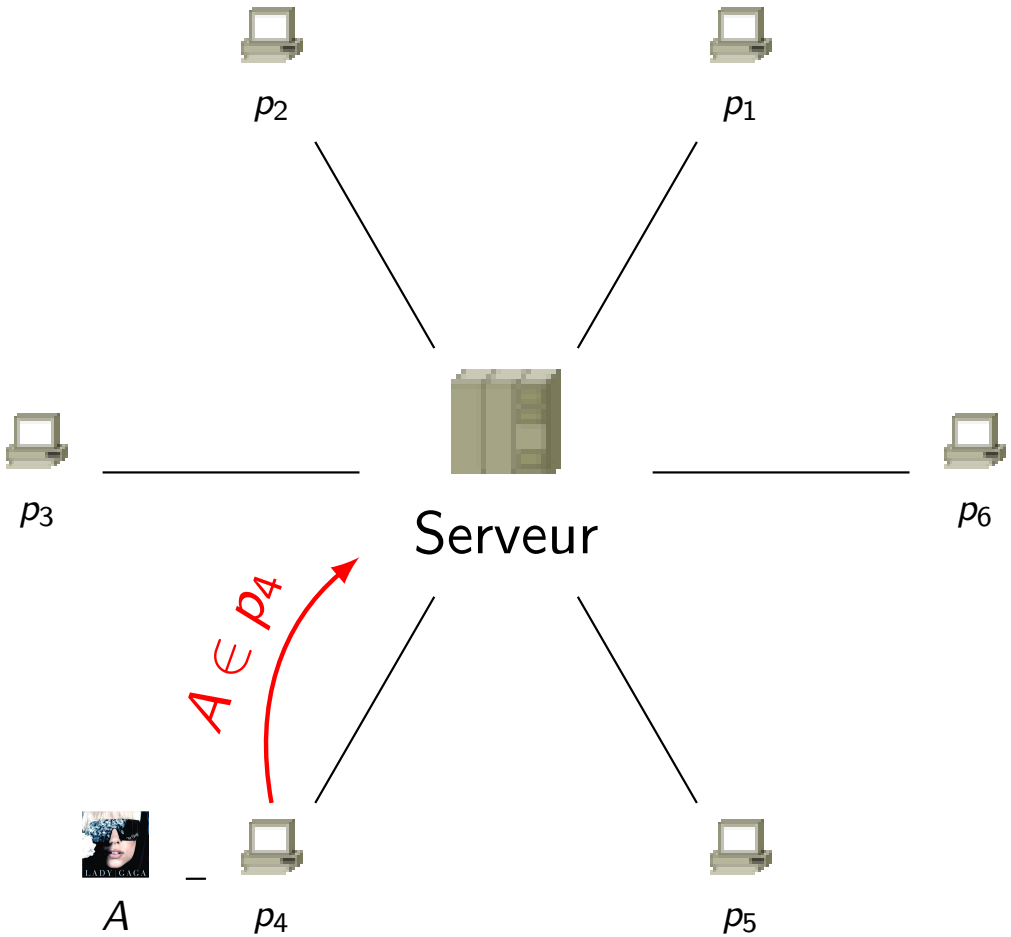
1. Indexation

Napster



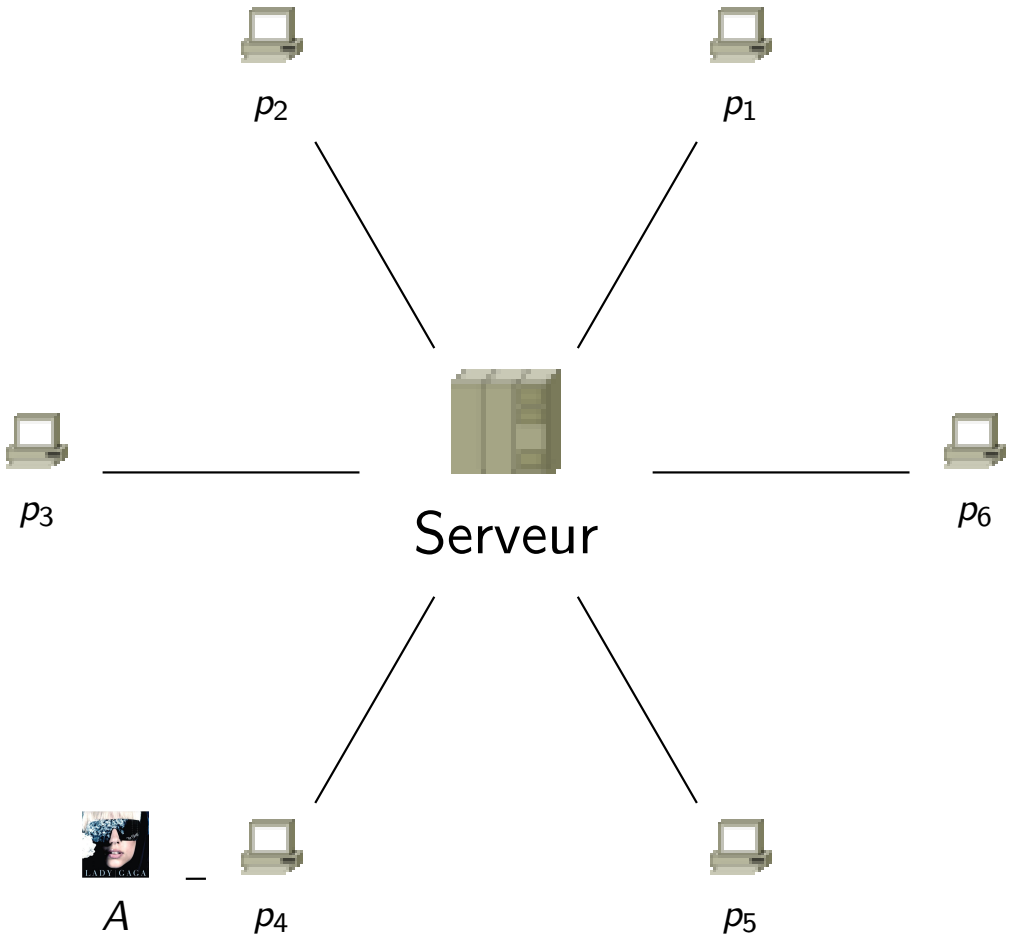
1. Indexation
   - p_4 possède A

Napster



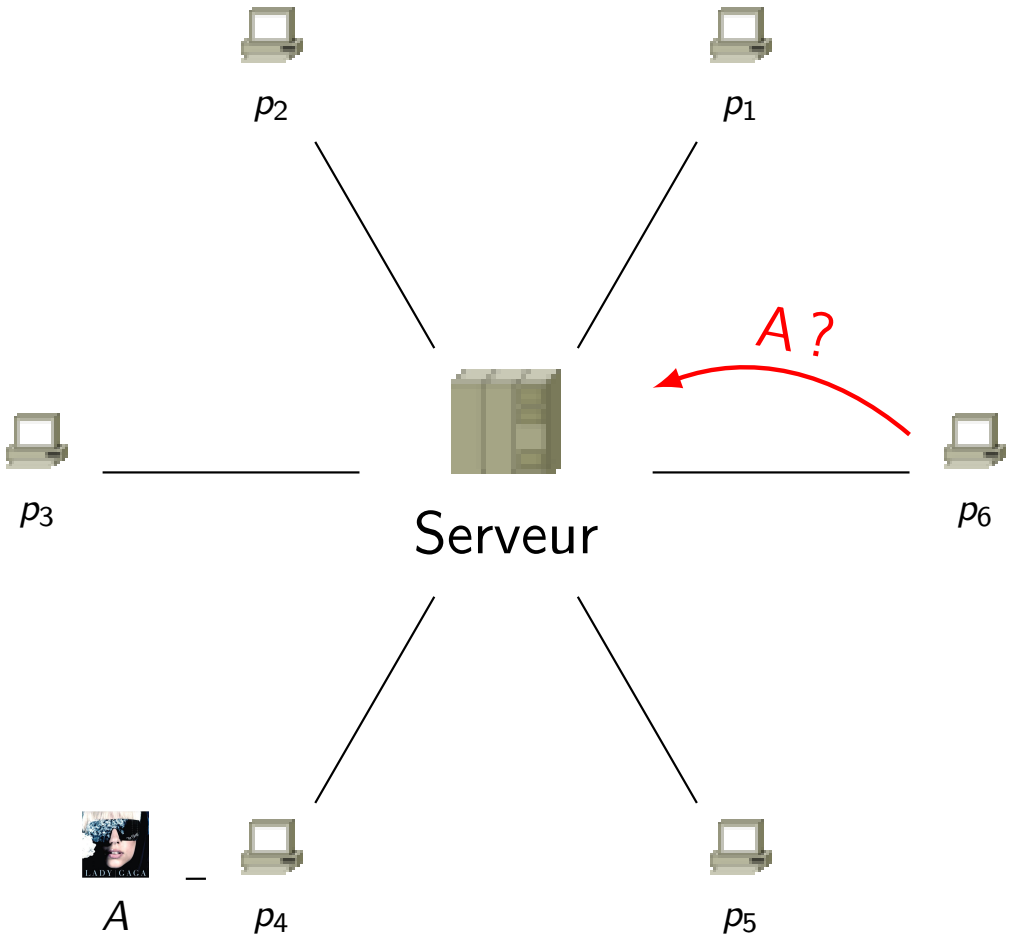
1. Indexation
   - p_4 possède A
   - p_4 informe s

Napster



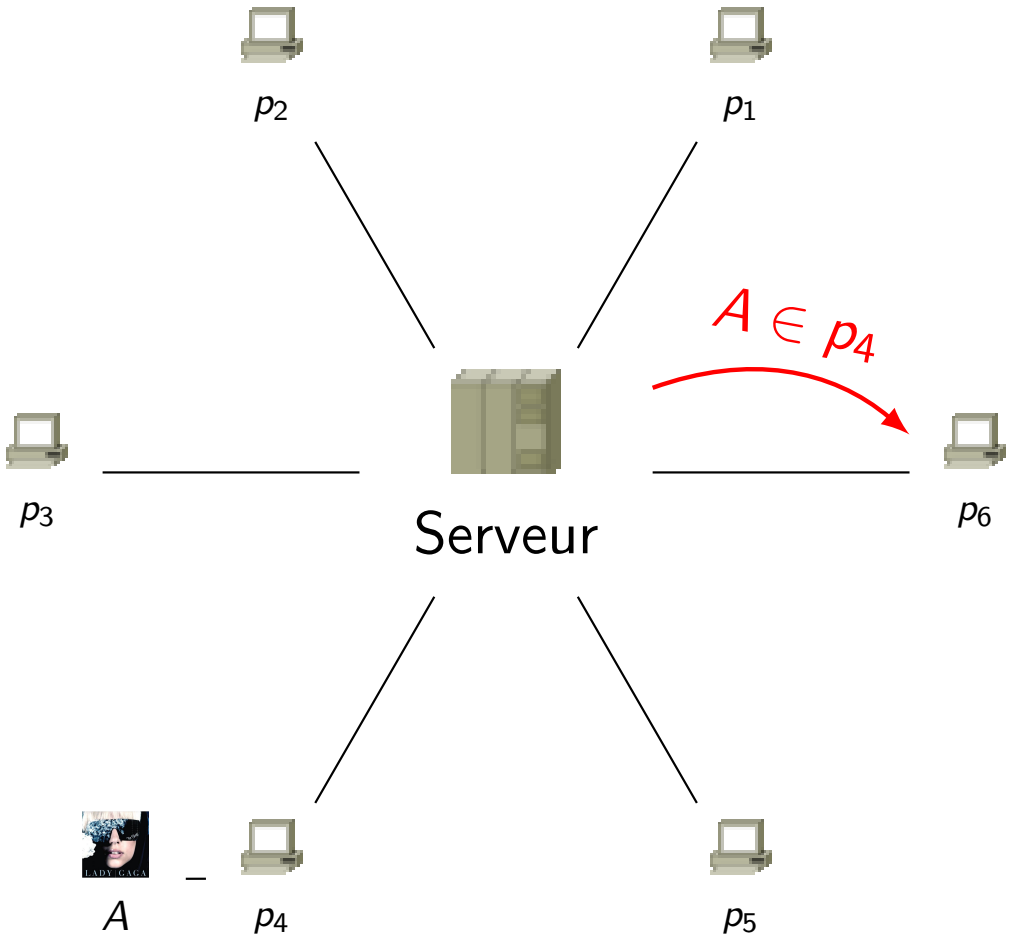
1. Indexation
   - p_4 possède A
   - p_4 informe s
2. Requête

Napster



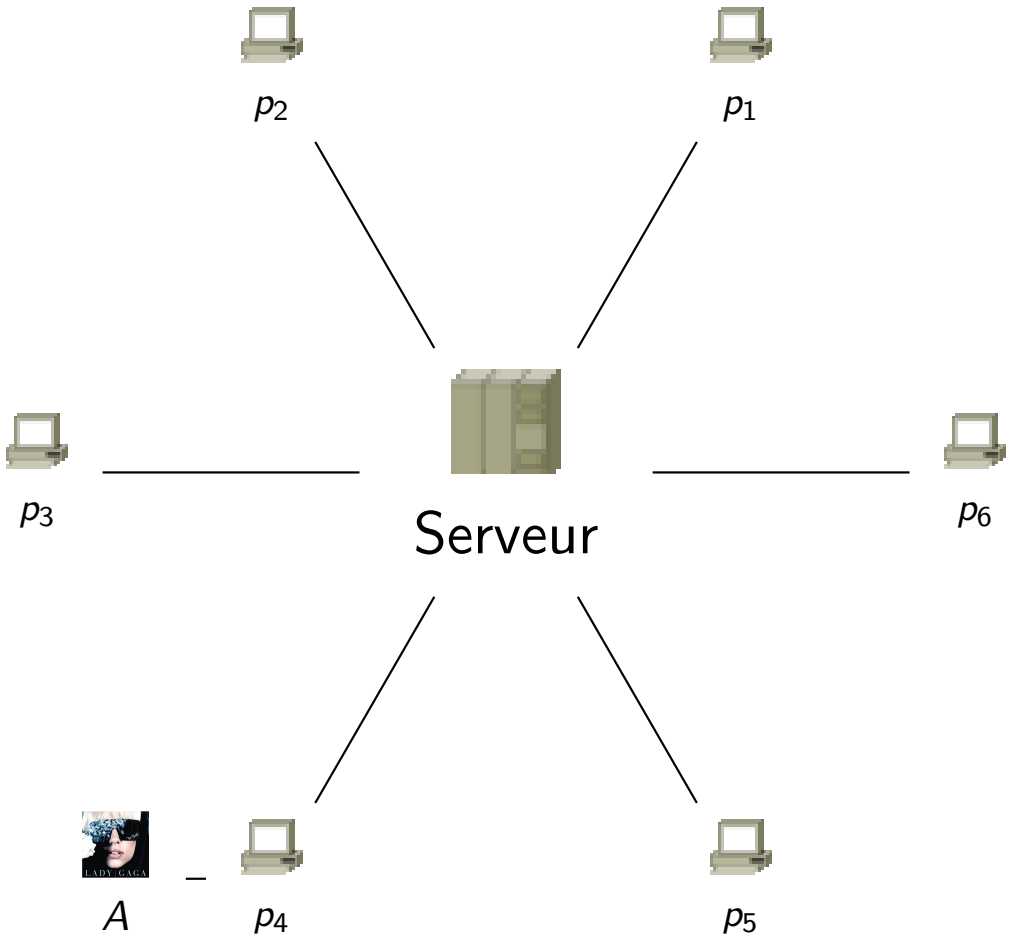
1. Indexation
   - p_4 possède A
   - p_4 informe s
2. Requête
   - p_6 demande à s qui a A

Napster



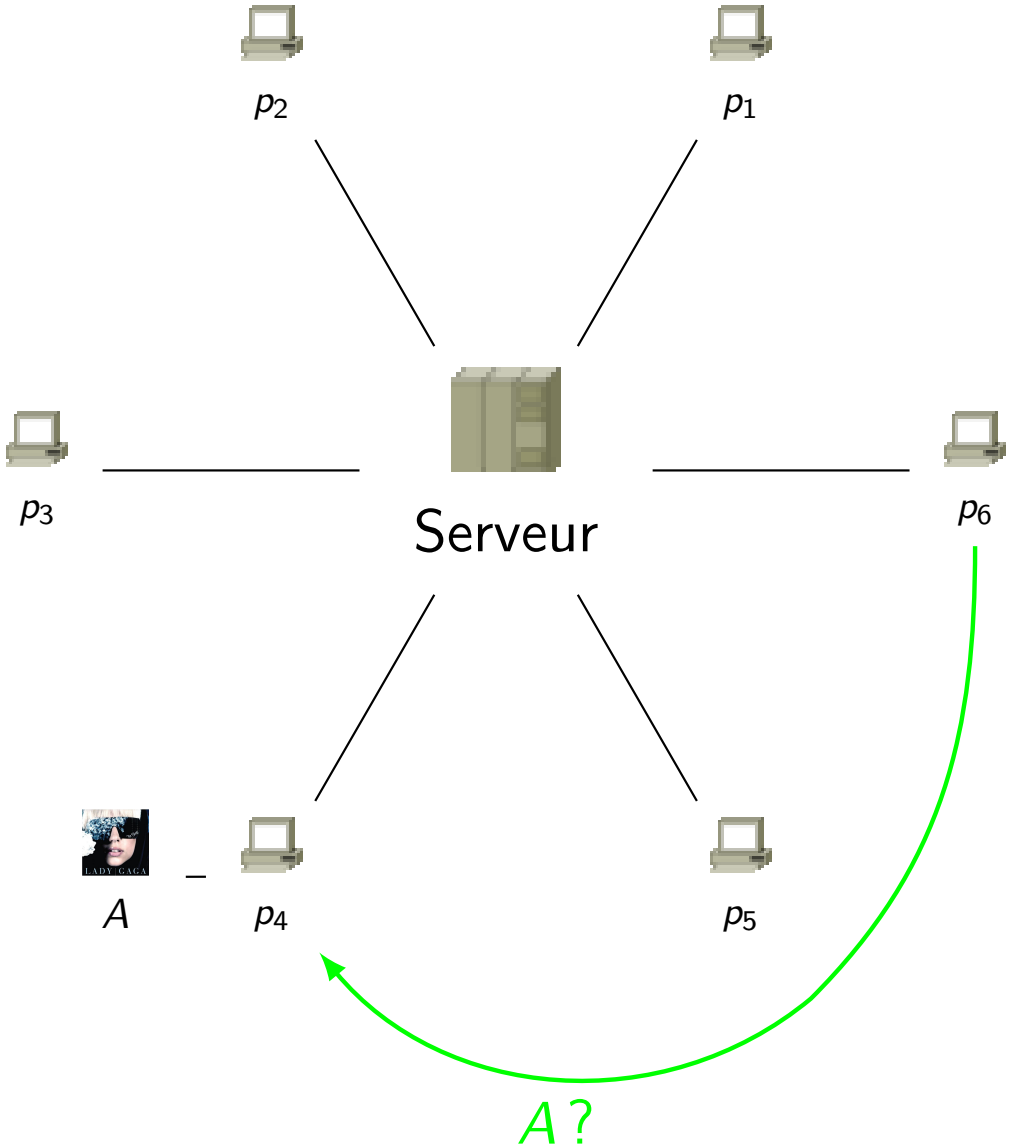
1. Indexation
   - p_4 possède A
   - p_4 informe s
2. Requête
   - p_6 demande à s qui a A
   - s répond p_4 (IP)

Napster



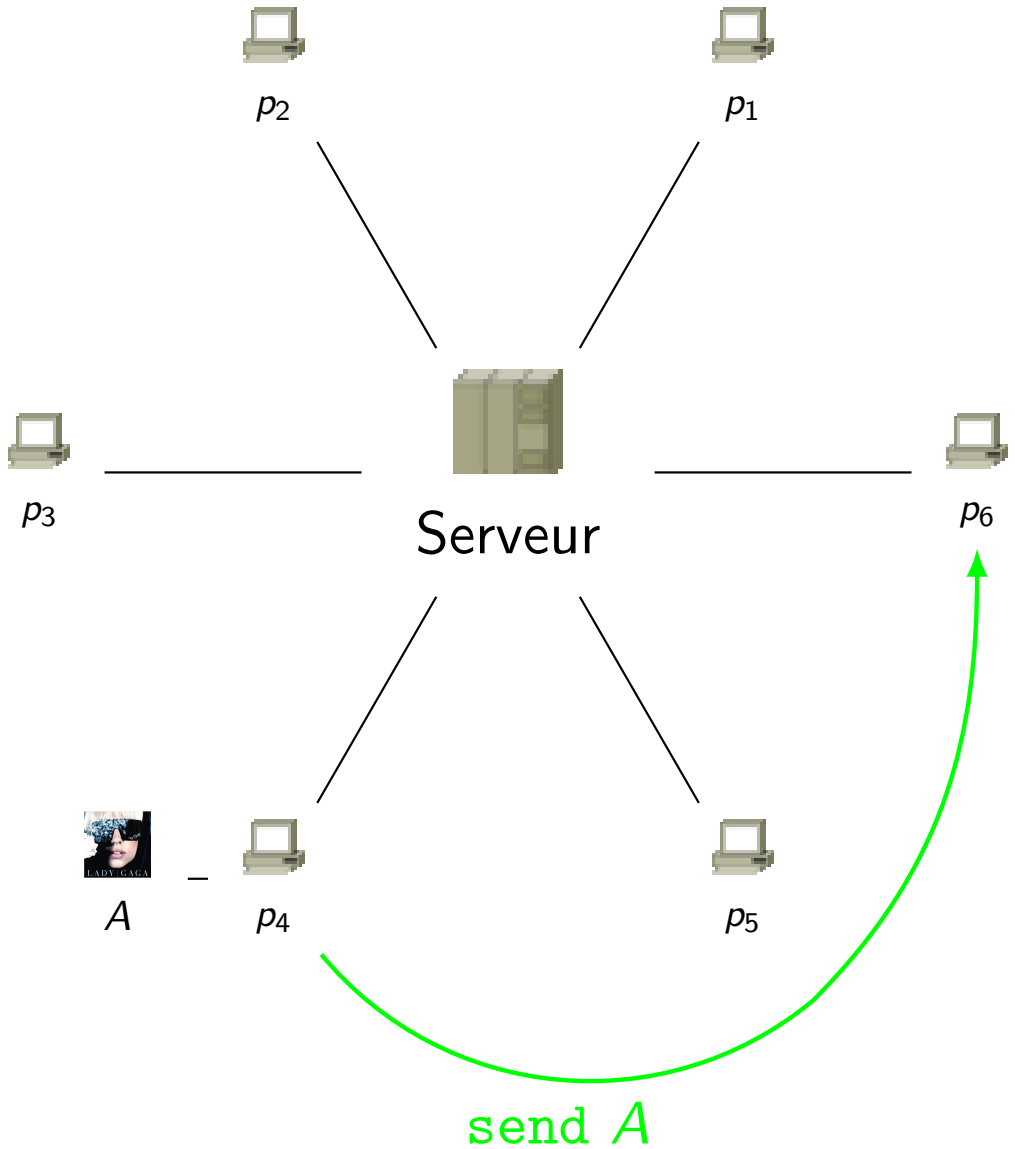
1. Indexation
   - p_4 possède A
   - p_4 informe s
2. Requête
   - p_6 demande à s qui a A
   - s répond p_4 (IP)
3. Distribution

Napster



1. Indexation
   - p_4 possède A
   - p_4 informe s
2. Requête
   - p_6 demande à s qui a A
   - s répond p_4 (IP)
3. Distribution
   - p_6 contacte p_4

Napster



1. Indexation
   - p_4 possède A
   - p_4 informe s
2. Requête
   - p_6 demande à s qui a A
   - s répond p_4 (IP)
3. Distribution
   - p_6 contacte p_4
   - p_4 envoie à p_6

Napster

Innovation

- Indexation centralisée → requêtes complexes (regexp) faisables
- Distribution d'un pair à un autre (littéralement)

Limites

- Indexation centralisée → point de vulnérabilité (d'ailleurs. . .)
- Distribution «stupide» (un vers un, fichier entier)

P2P : Logiciels clés

1998 Napster : distribution de contenu P2P(MP3) distribution

2000 **Gnutella : recherche décentralisée**

2000 KaZaA/eDonkey : indexation hiérarchique

2003 BitTorrent : distribution performante

2006 Fin de Razorback 2.0 : émergence des DHTs

2006-2009 PPLive/UUSee/TVAnts/. . . : P2P streaming

Gnutella

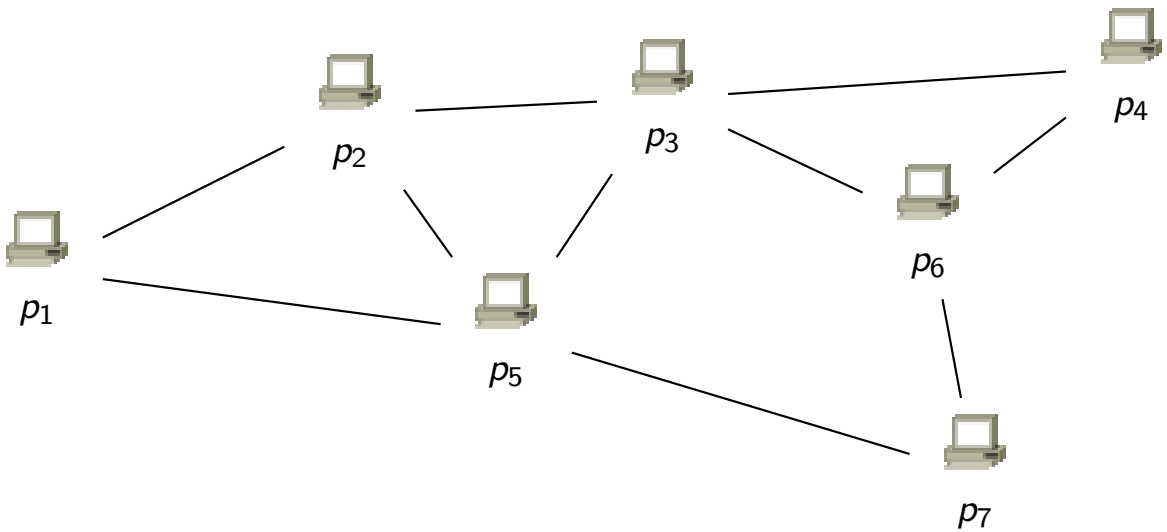
Histoire

- 1er client : 2000 (nullsoft-AOL). Retiré au bout de 1 jour (!)
- Code disponible → repris en open-source (LimeWire, Morpheus...)
- Plus gros trafic IP de l'année à son époque (quasi disparu aujourd'hui)

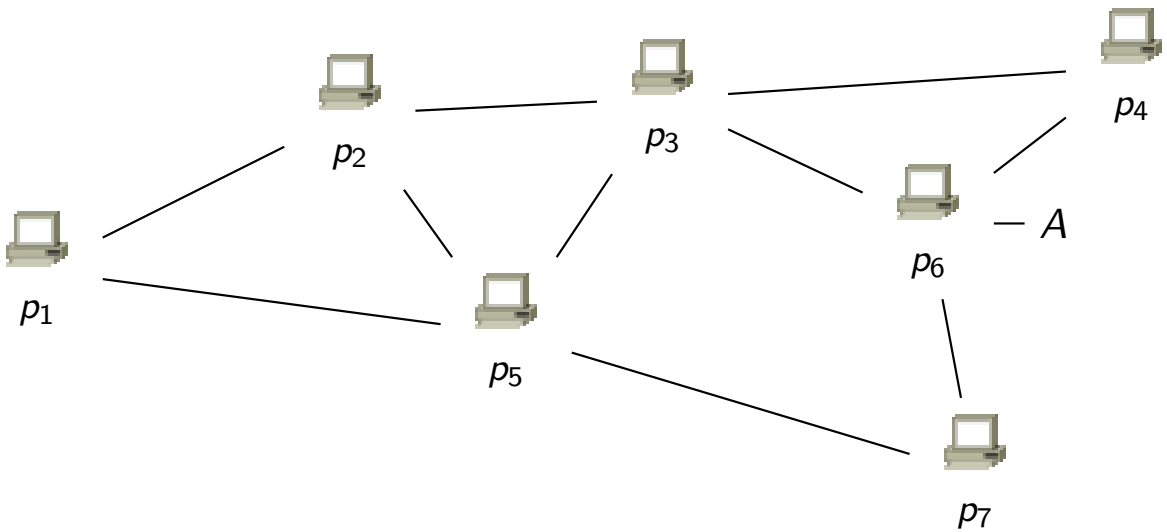
Idée du protocole initial (0.4)

- Totalement décentralisé : pas d'index central
- les recherches sont faites par **inondation**
- Améliorations ultérieures (hashage, ultrapeers, chunks...)

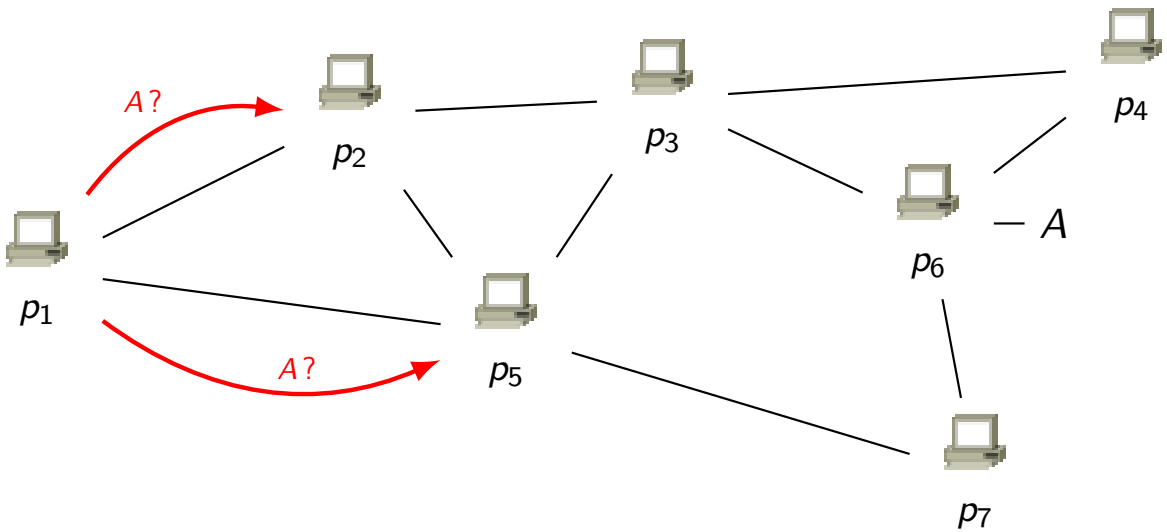
Gnutella



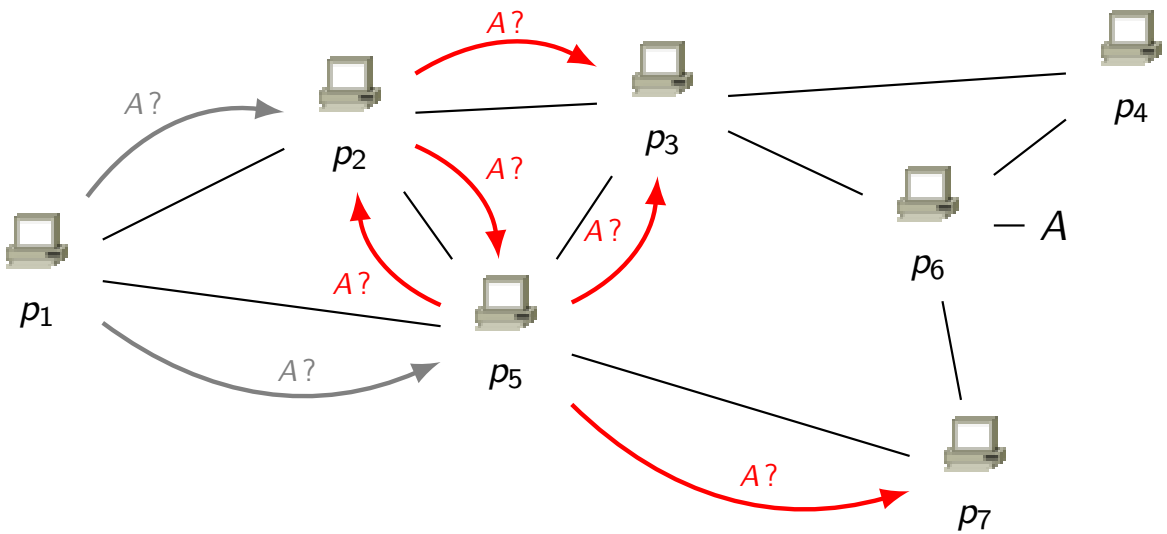
Gnutella



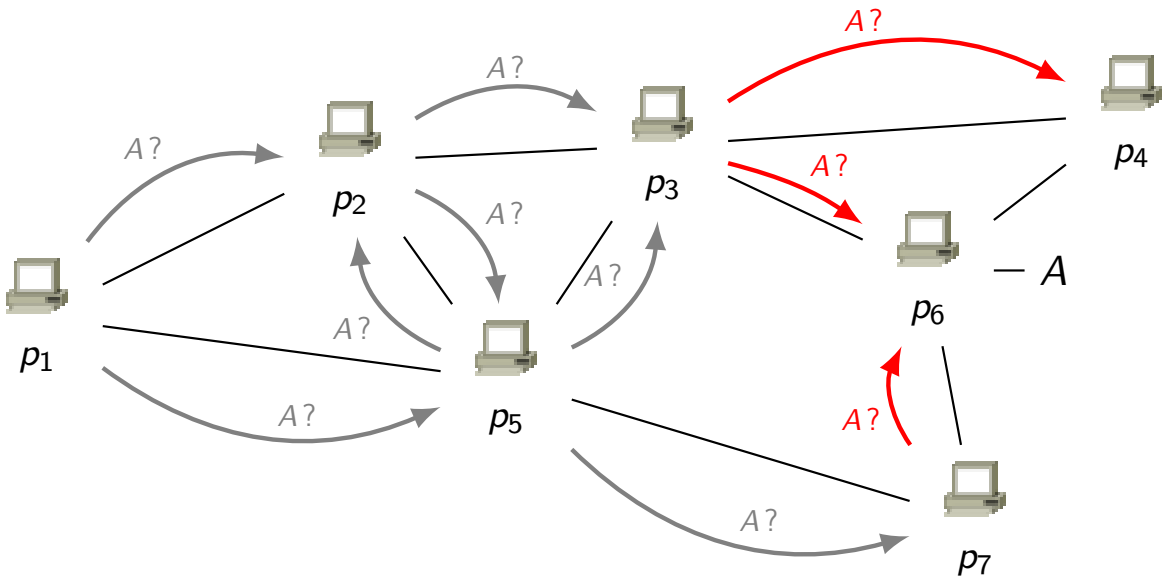
Gnutella



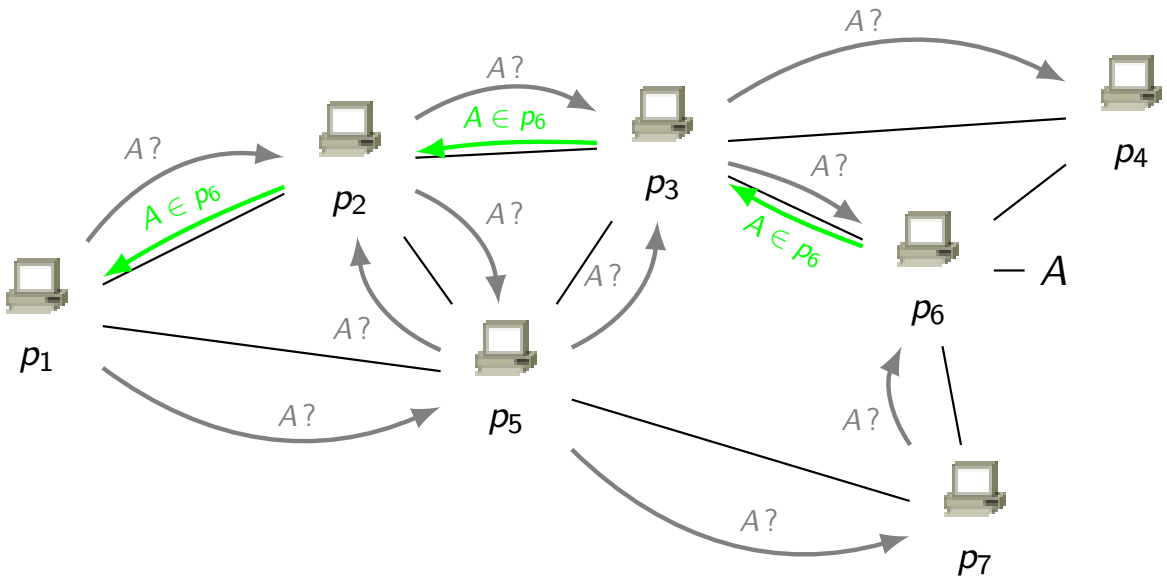
Gnutella



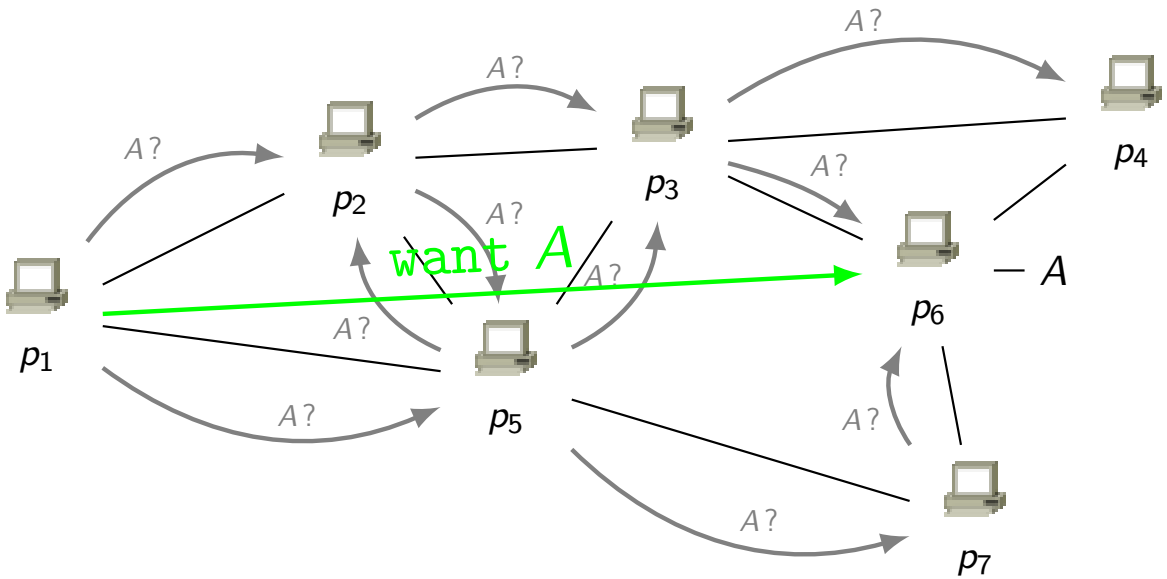
Gnutella



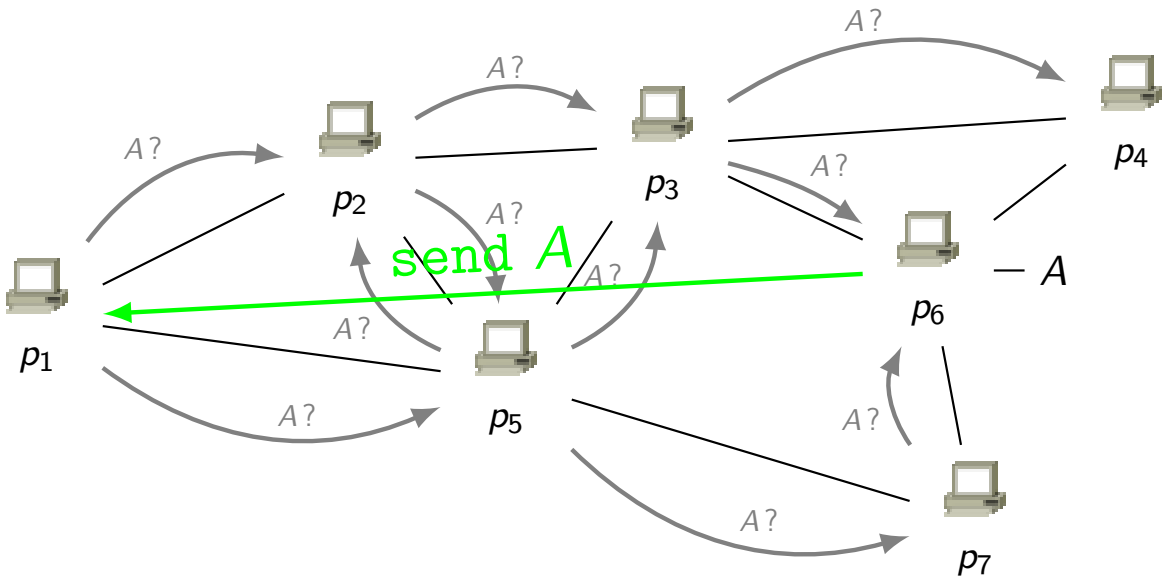
Gnutella



Gnutella



Gnutella



Gnutella

Innovation

pas d'index, regexp toujours possibles

Limites

- L'inondation, c'est MAL : à chaque requête, une seule une fraction des pairs reçoit le message
  - fraction trop petite : réponse incomplète
  - fraction trop grande : surcoût en message (proportion mesurée des messages : plus de la moitié du trafic)
- Distribution toujours «stupide»

P2P : Logiciels clés

1998 Napster : distribution de contenu P2P(MP3) distribution

2000 Gnutella : recherche décentralisée

2000 **KaZaA/eDonkey : indexation hiérarchique**

2003 BitTorrent : distribution performante

2006 Fin de Razorback 2.0 : émergence des DHTs

2006-2009 PPLive/UUSee/TVAnts/. . . : P2P streaming

KaZaA

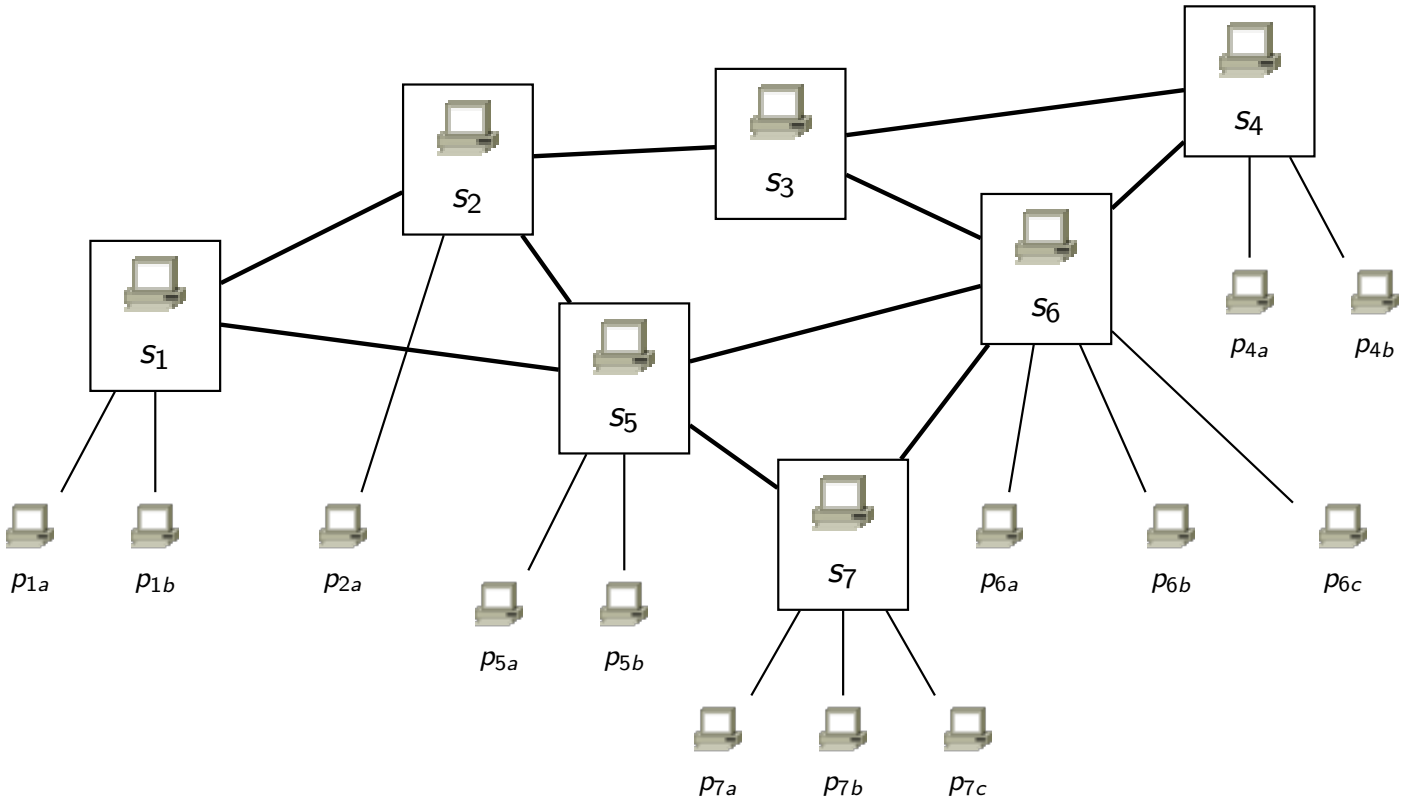
Histoire

- 2000, Niklas Zennström et Janus Friis (Skype, Joost)
- Repose sur le réseau FastTrack (propriétaire)
- Plus gros trafic de l'année, puis disparaît

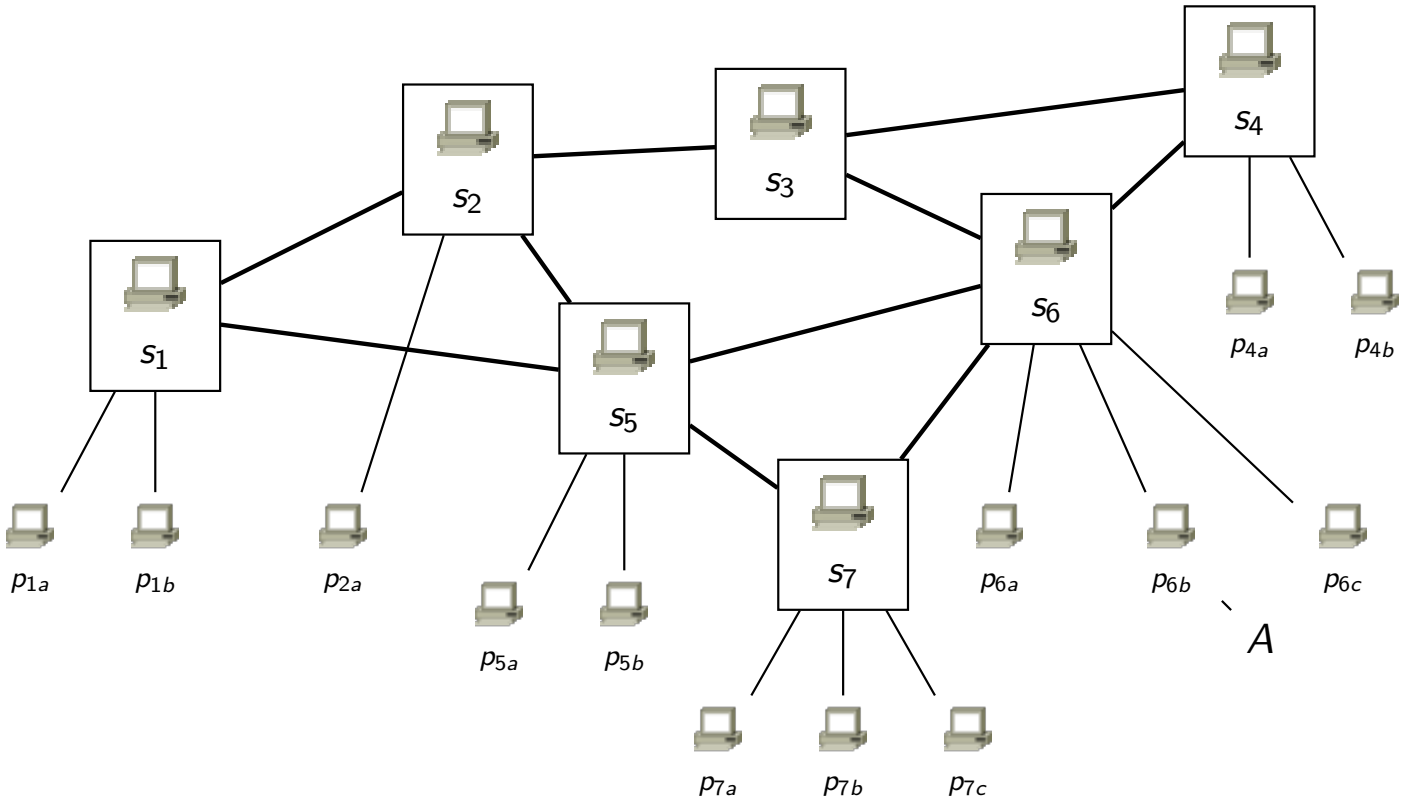
Principale idée : **indexation hiérarchique**

- Choix de supernodes : pairs avec grande bande passante et souvent disponibles
- Chaque pair ordinaire dépend (est relié) à un supernode (ratio $100 - 1000$)
- Les supernodes gèrent leurs pairs (indexation, gestion des requêtes) et communiquent par inondation (Gnutella-style)
- Également, distribution par chunks basée sur la réputation

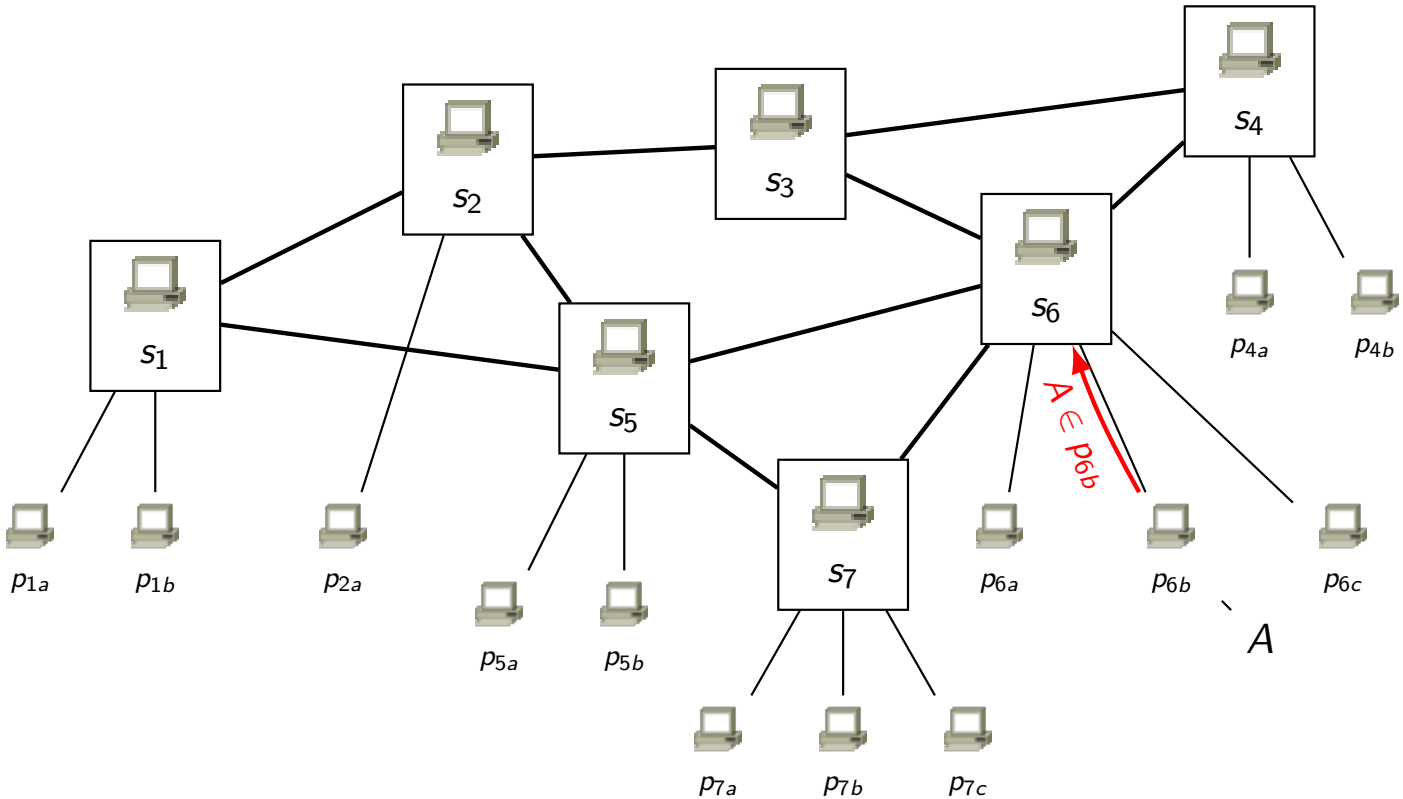
KaZaA


s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
p_{7a}
p_{7b}
p_{7c}

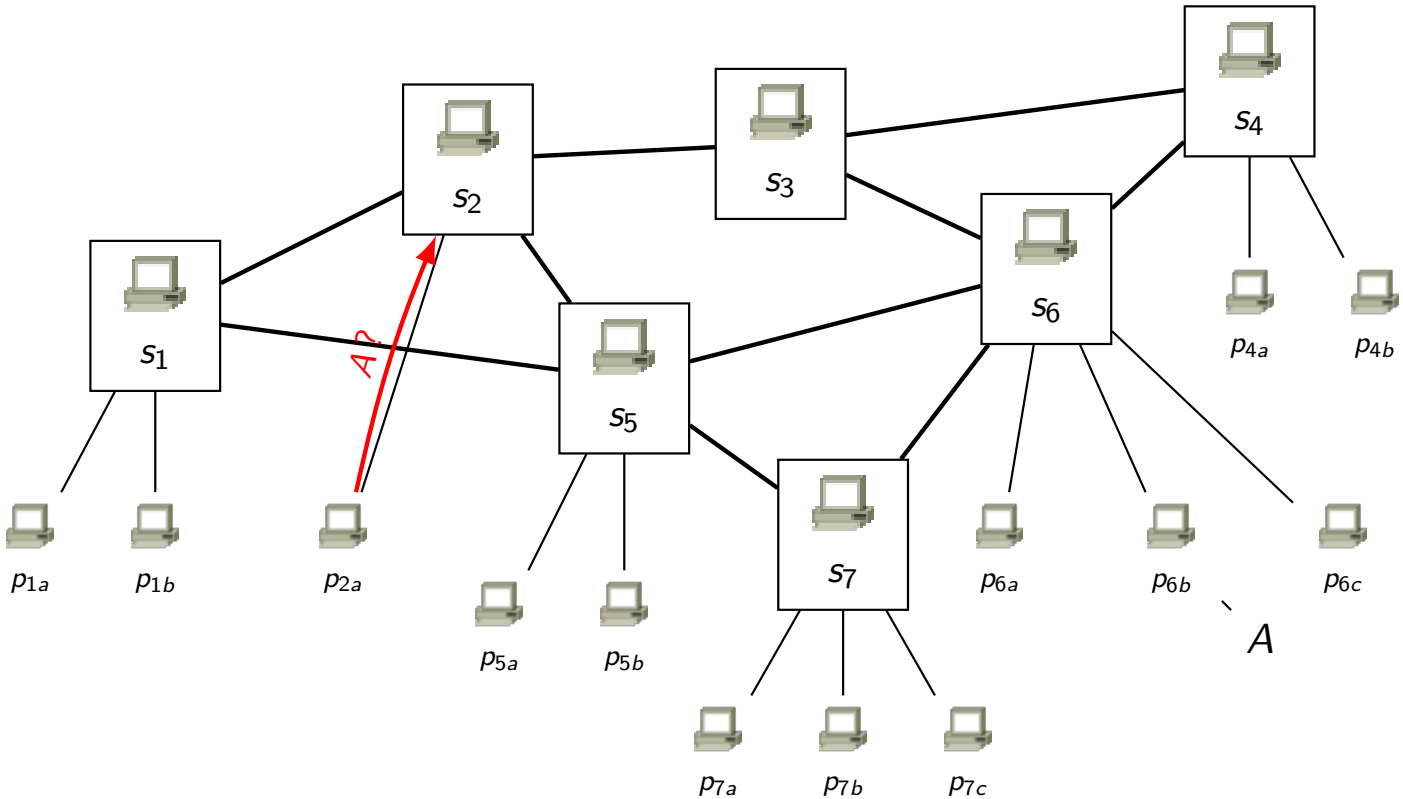
KaZaA


s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
p_{7a}
p_{7b}
p_{7c}
A

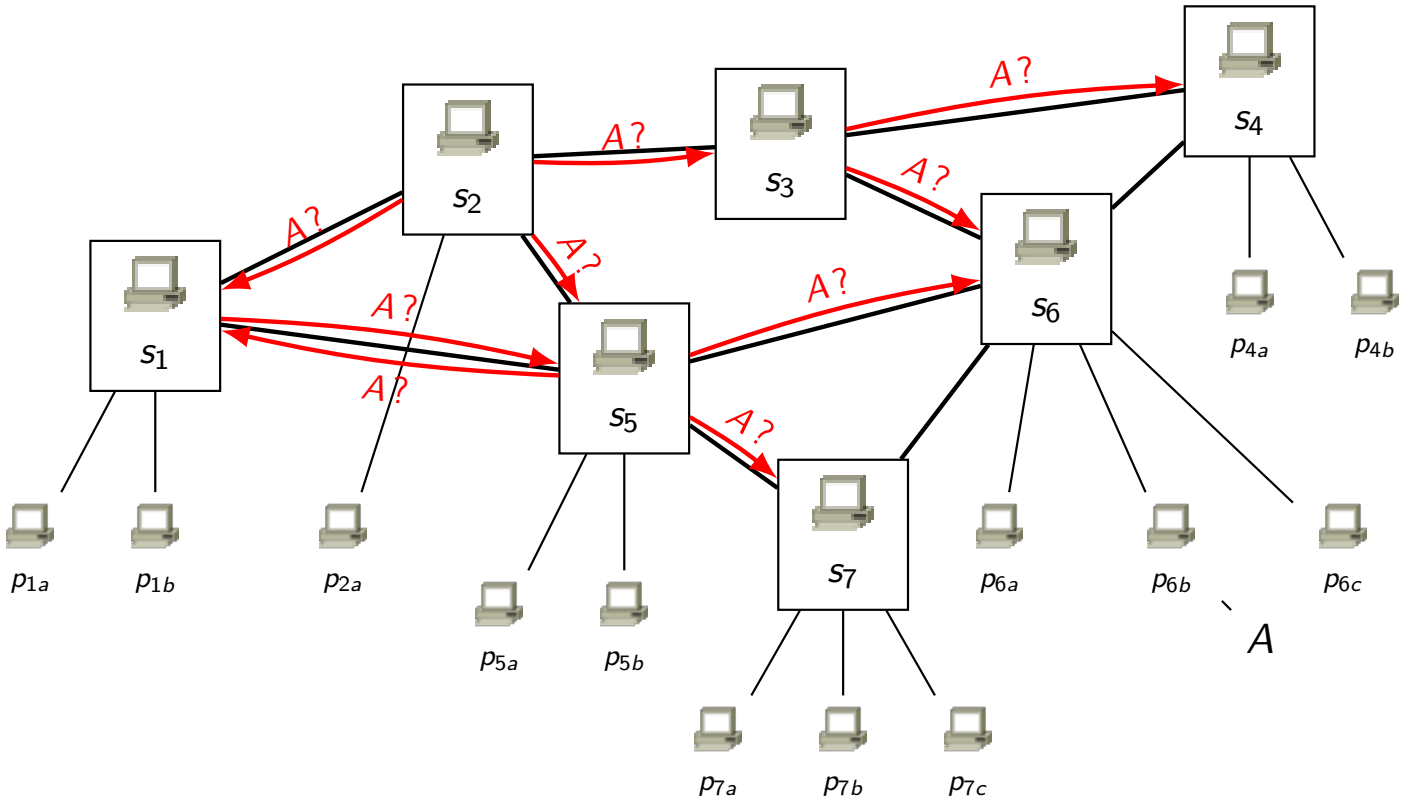
KaZaA


s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
p_{7a}
p_{7b}
p_{7c}
$A \in p_{6b}$
A

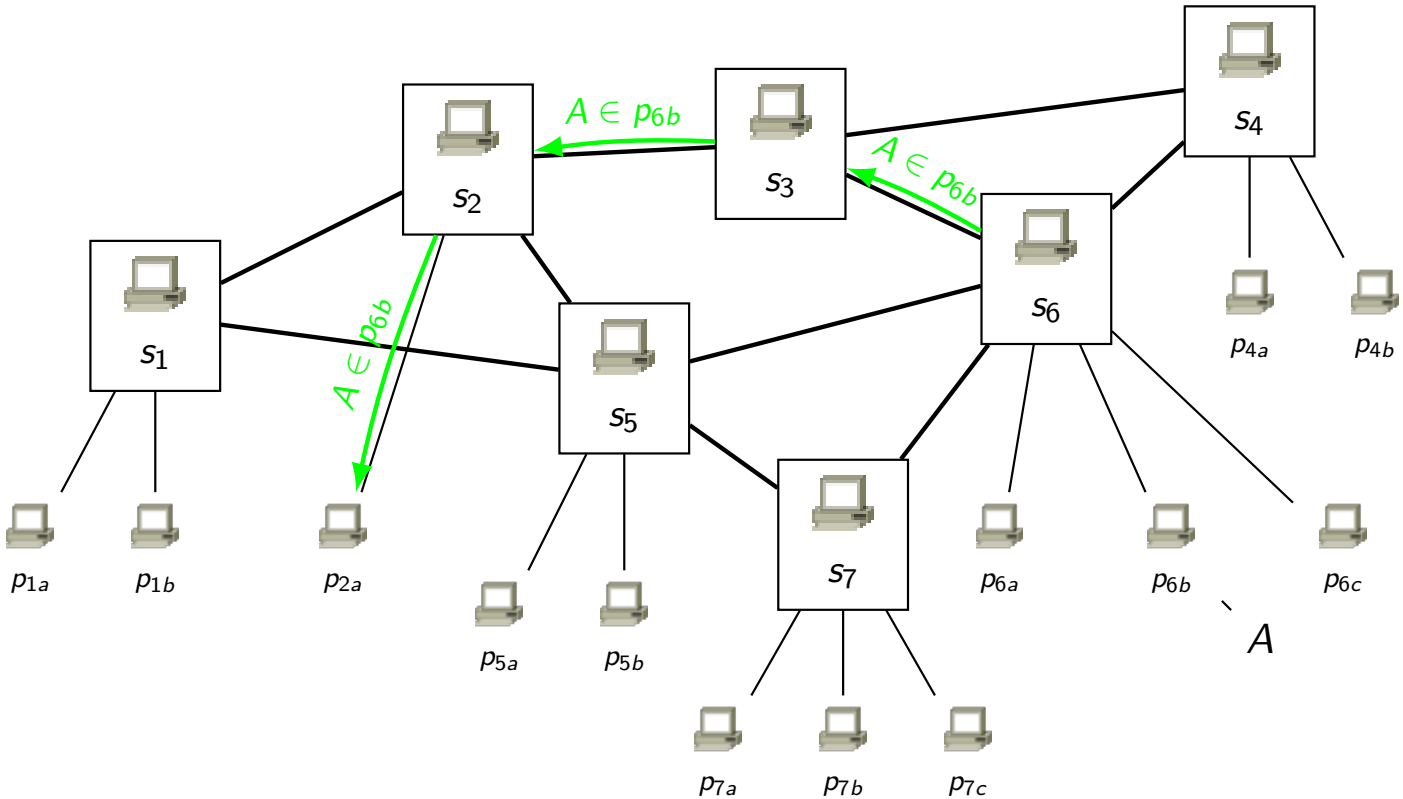
KaZaA


s_1
s_2
s_3
s_4
s_5
s_6
s_7
A?
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
A
p_{7a}
p_{7b}
p_{7c}

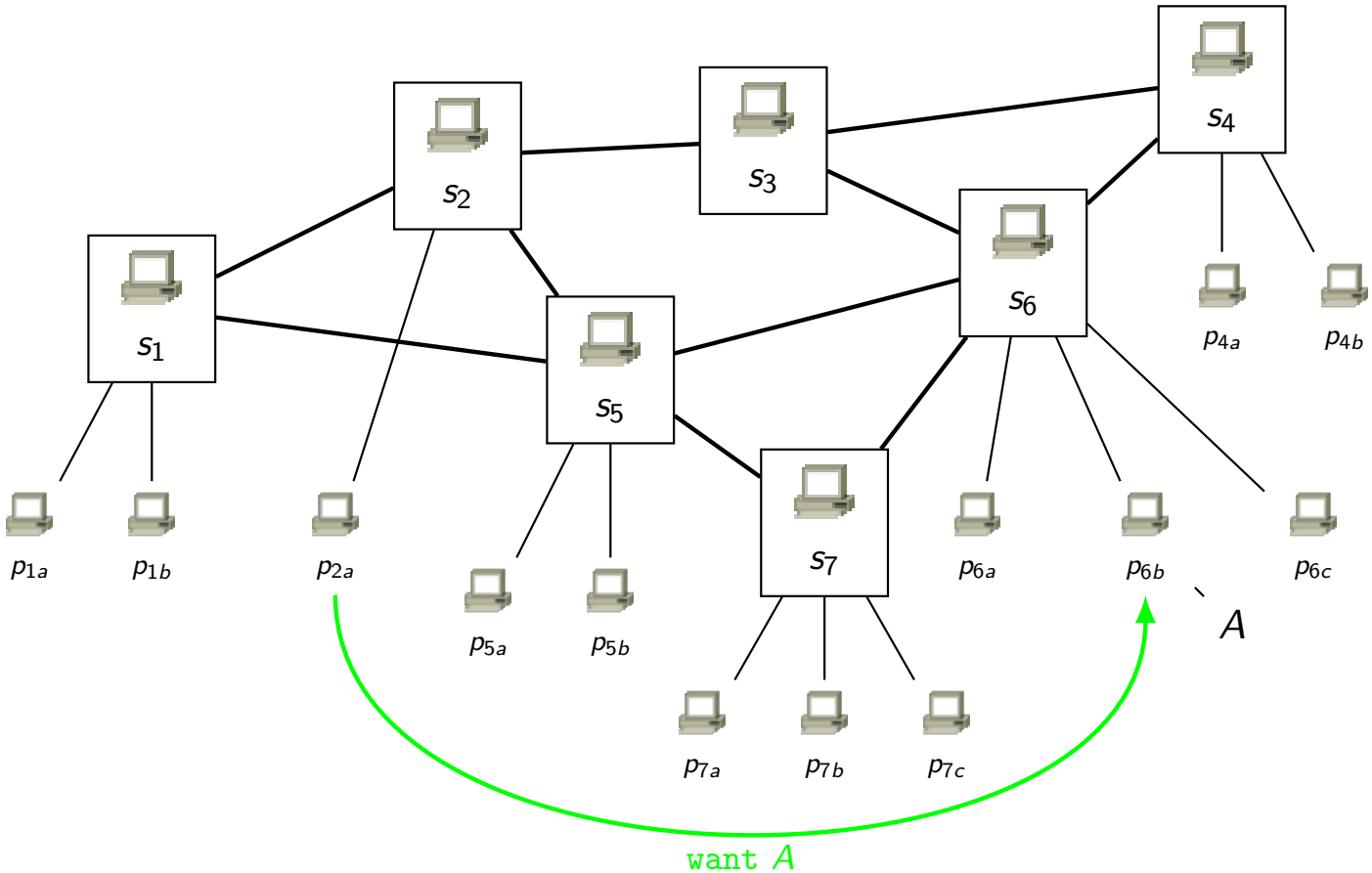
KaZaA


A?
A?
A?
A?
A?
A?
A?
A?
A?
A?
s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
p_{7a}
p_{7b}
p_{7c}
A

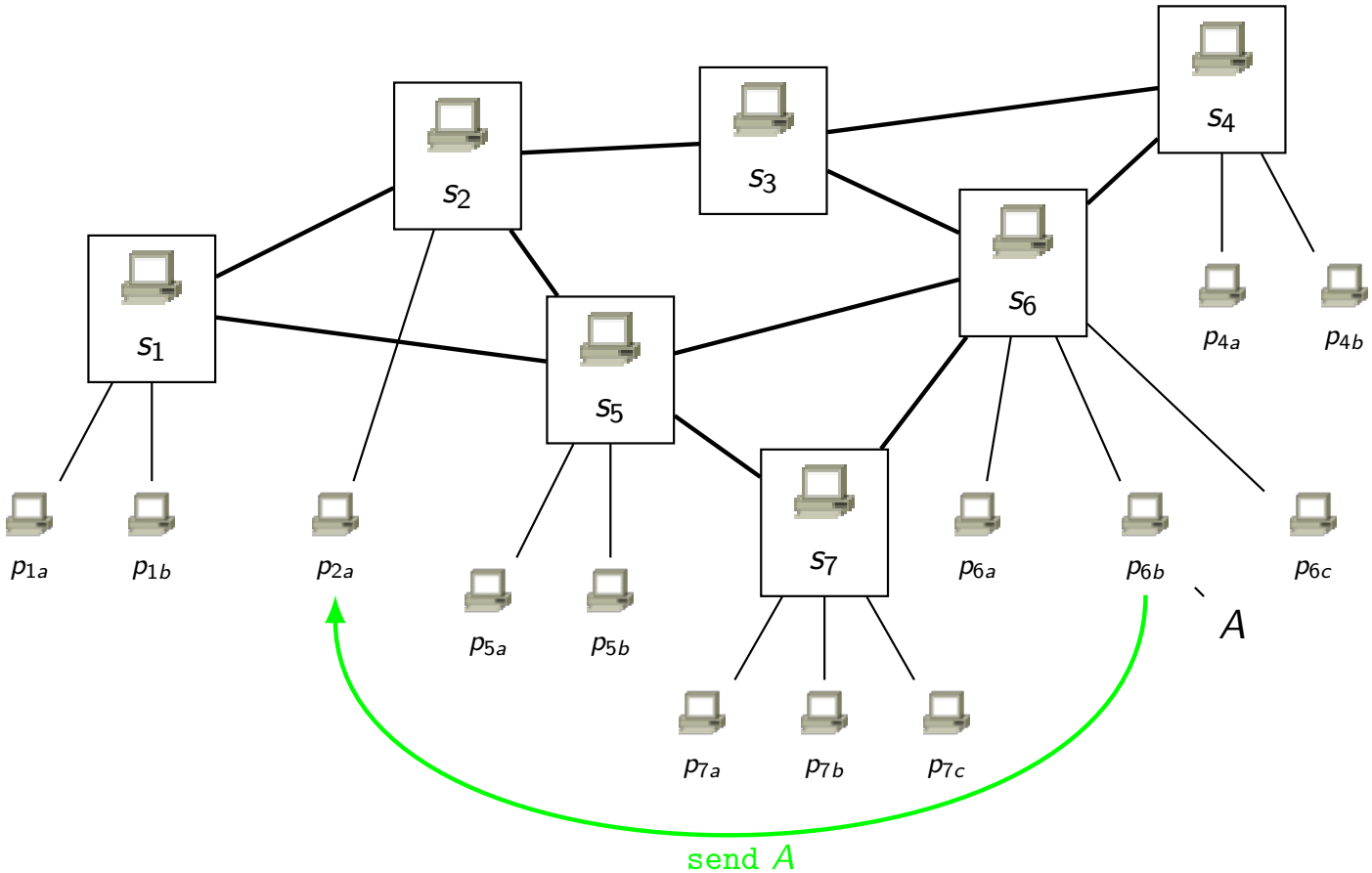
KaZaA


$A \in p_{6b}$
$A \in p_{6b}$
$A \in p_{6b}$
s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
A
p_{7a}
p_{7b}
p_{7c}

KaZaA


s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
p_{7a}
p_{7b}
p_{7c}
A
want A

KaZaA


s_1
s_2
s_3
s_4
s_5
s_6
s_7
p_{1a}
p_{1b}
p_{2a}
p_{4a}
p_{4b}
p_{5a}
p_{5b}
p_{6a}
p_{6b}
p_{6c}
p_{7a}
p_{7b}
p_{7c}
A
send A

eMule - eDonkey

Histoire

- Le client eDonkey2000 sort en. . . 2000
- Société MetaMachine, fermeture en 2005 (RIAA)
- Protocole documenté → clones Open-source (eMule)
- plus grand trafic. . . (toujours 2e en 2012)

Principe

- Serveurs eDonkey : serveurs dédiés à l'indexation (supernodes «professionnels»)
- Distribution par chunks avec files d'attente

KaZaA/eDonkey

Innovation

- Indexation hiérarchique
- Concept de chunk («paquet» de l'overlay)

Limites

- KaZaA : ordonnanceur de chunks séquentiel → «chunk manquant»
- KaZaA Lite K++ : fausse réputation, toujours supernode
- eDonkey : files d'attente saturées → délai très grand
- Serveurs/ supernodes → points faibles

P2P : Logiciels clés

1998 Napster : distribution de contenu P2P(MP3) distribution

2000 Gnutella : recherche décentralisée

2000 KaZaA/eDonkey : indexation hiérarchique

2003 BitTorrent : distribution performante

2006 **Fin de Razorback 2.0 : émergence des DHTs**

2006-2009 PPLive/UUSee/TVAnts/. . . : P2P streaming

KAD vs Razorback 2.0

Kad

RazorBack 2.0

Le cycle de la vie

KAD vs Razorback 2.0

Kad

- Objectif : éviter l'index (semi)-centralisé ET l'inondation
- Solution : les tables de hachage distribuées (DHTs), introduites dès 2001 (papiers académique)
- L'indexation est répartie sur tous les pairs
- Une des implantations, Kad, est incorporée assez vite à eMule

RazorBack 2.0

Le cycle de la vie

KAD vs Razorback 2.0

Kad

RazorBack 2.0

- Une course au meilleur serveur eDonkey s'engage au début des années 2000
- Un vainqueur incontestable : Razorback 2.0
- Les autres serveurs dépérissent (spam, catalogue et réactivité faibles)

Le cycle de la vie

KAD vs Razorback 2.0

Kad

RazorBack 2.0

Le cycle de la vie

- Au début, Kad très peu populaire (R2 est plus rapide)
- 21 février 2006, fermeture brutale de Razorback2
- Beaucoup d'utilisateurs basculent vers Kad, seule véritable alternative. . .
- . . . et **ça marche !** (résiste à la montée en charge)

Les tables de hachage distribuées

Innovation

Index totalement décentralisé

Limites

- Seulement des combinaisons simples de requêtes exactes (~~regexp~~)
- Prix à payer logarithmique (délai et maintenance)

P2P : Logiciels clés

1998 Napster : distribution de contenu P2P(MP3) distribution

2000 Gnutella : recherche décentralisée

2000 KaZaA/eDonkey : indexation hiérarchique

2003 **BitTorrent : distribution performante**

2006 Fin de Razorback 2.0 : émergence des DHTs

2006-2009 PPLive/UUSee/TVAnts/. . . : P2P streaming

BitTorrent

Histoire

- Papier fondateur et client proposés en 2003 par Bram Cohen
- Protocole documenté
- Grand nombre de clients (Générique, Azureus/Vuze, μtorrent...)
- Plus gros trafic (de loin) pendant plusieurs années. Toujours le plus grand trafic P2P

Idée principale : se concentrer sur le contenu et sa distribution

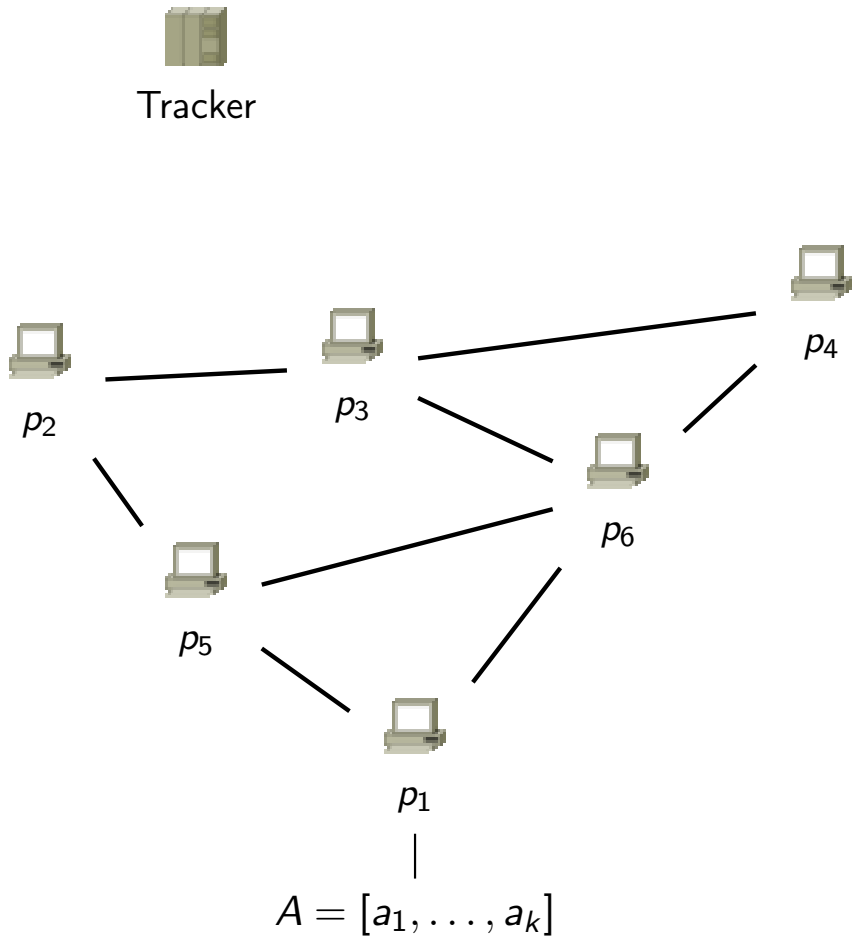
- À chaque contenu est associé un overlay : l'essaim (swarm)
- Un essaim est indexé par un tracker
- Politique d'incitation au partage : donnant-donnant (Tit-for-Tat)
- + 2/3 autres idées simples et efficaces

BitTorrent

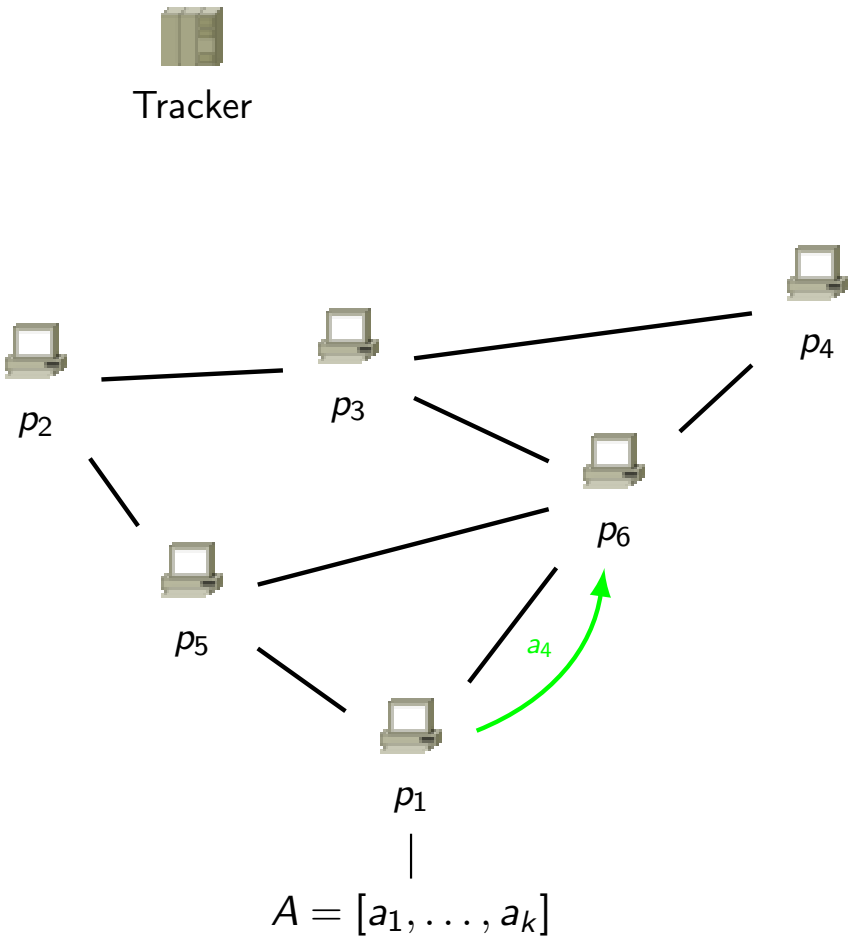
Une architecture minimale

- Un tracker gère l'ensemble des pairs intéressé
- Quand je suis intéressé, je récupère des voisins
- Échange de chunks jusqu'à la fin du téléchargement

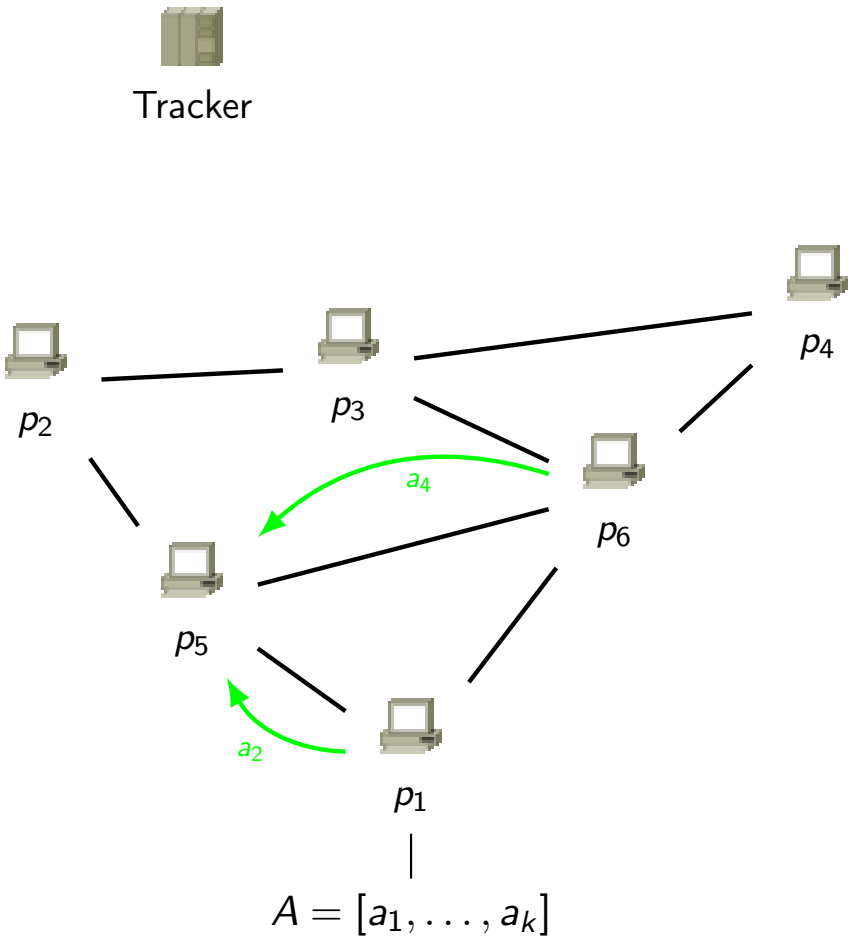
BitTorrent



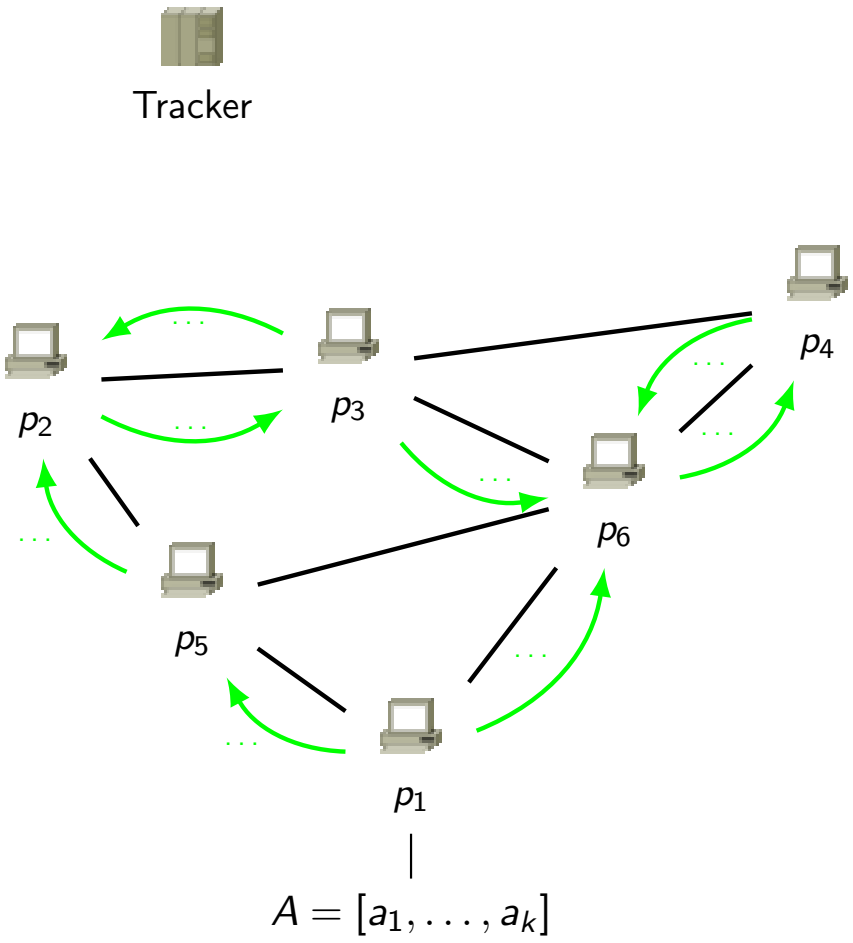
BitTorrent



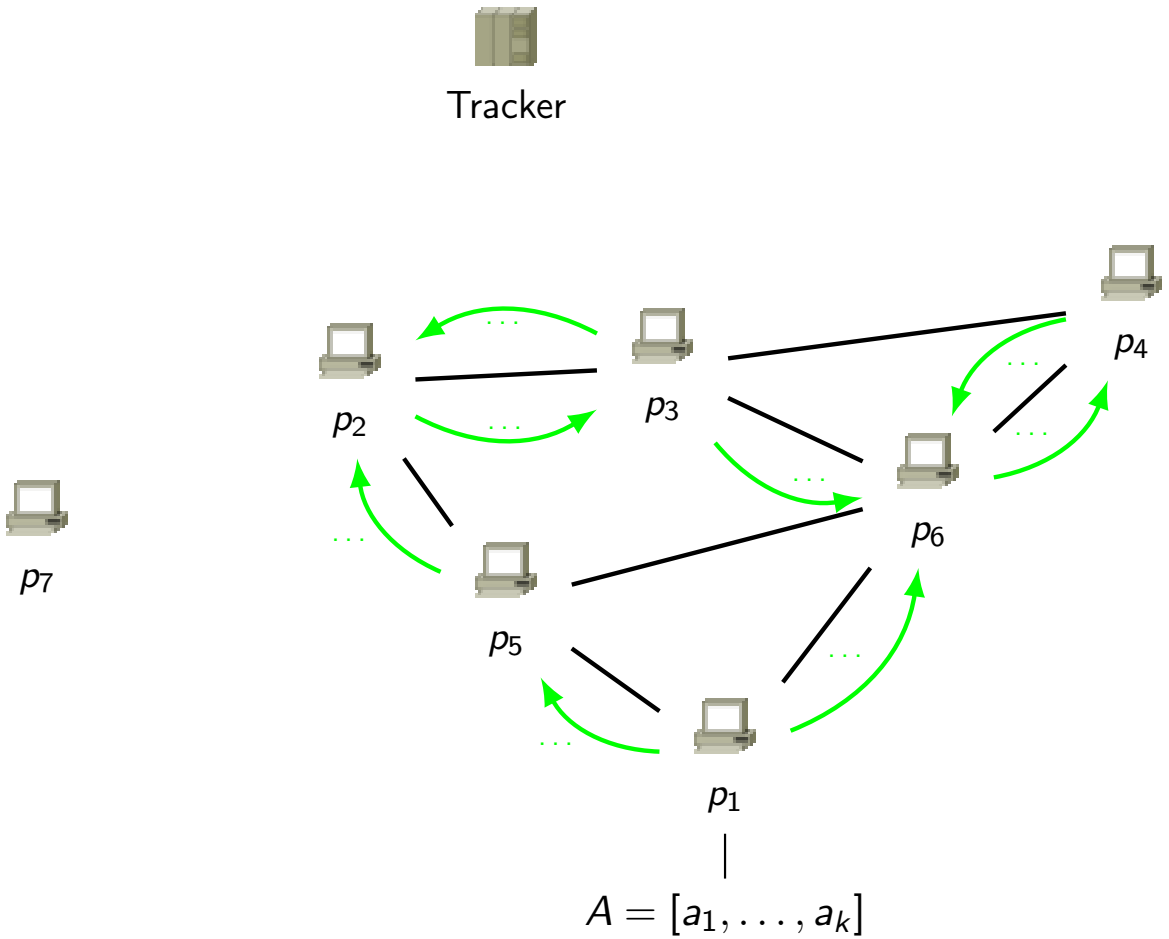
BitTorrent



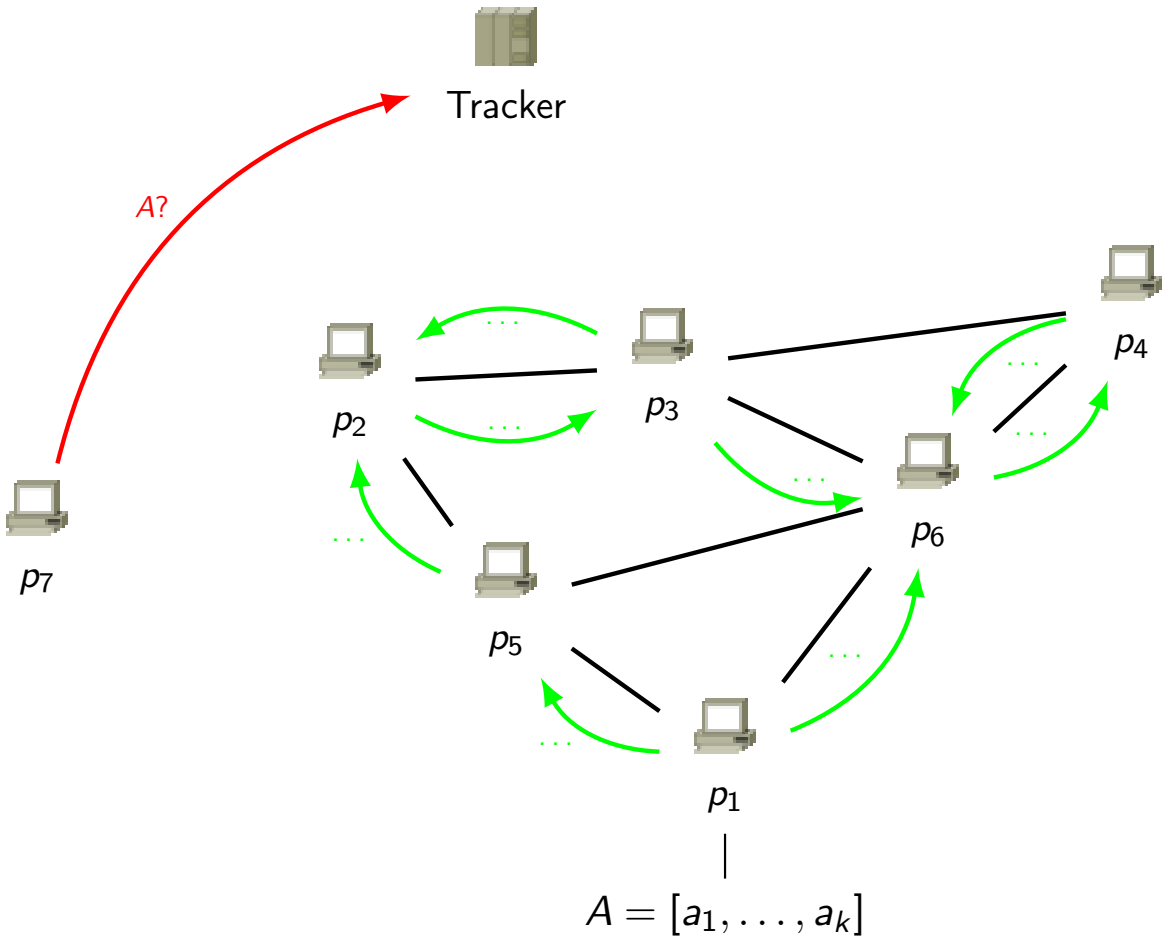
BitTorrent



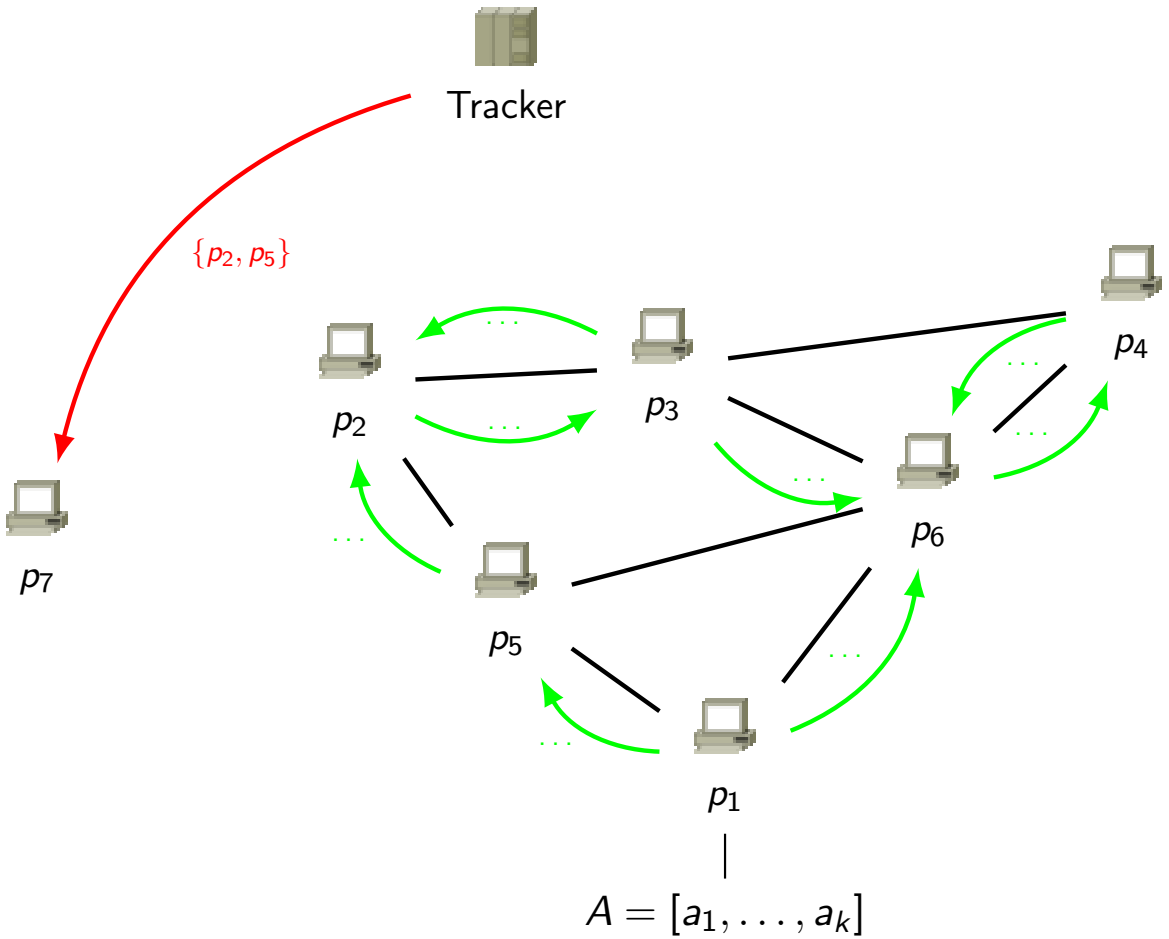
BitTorrent



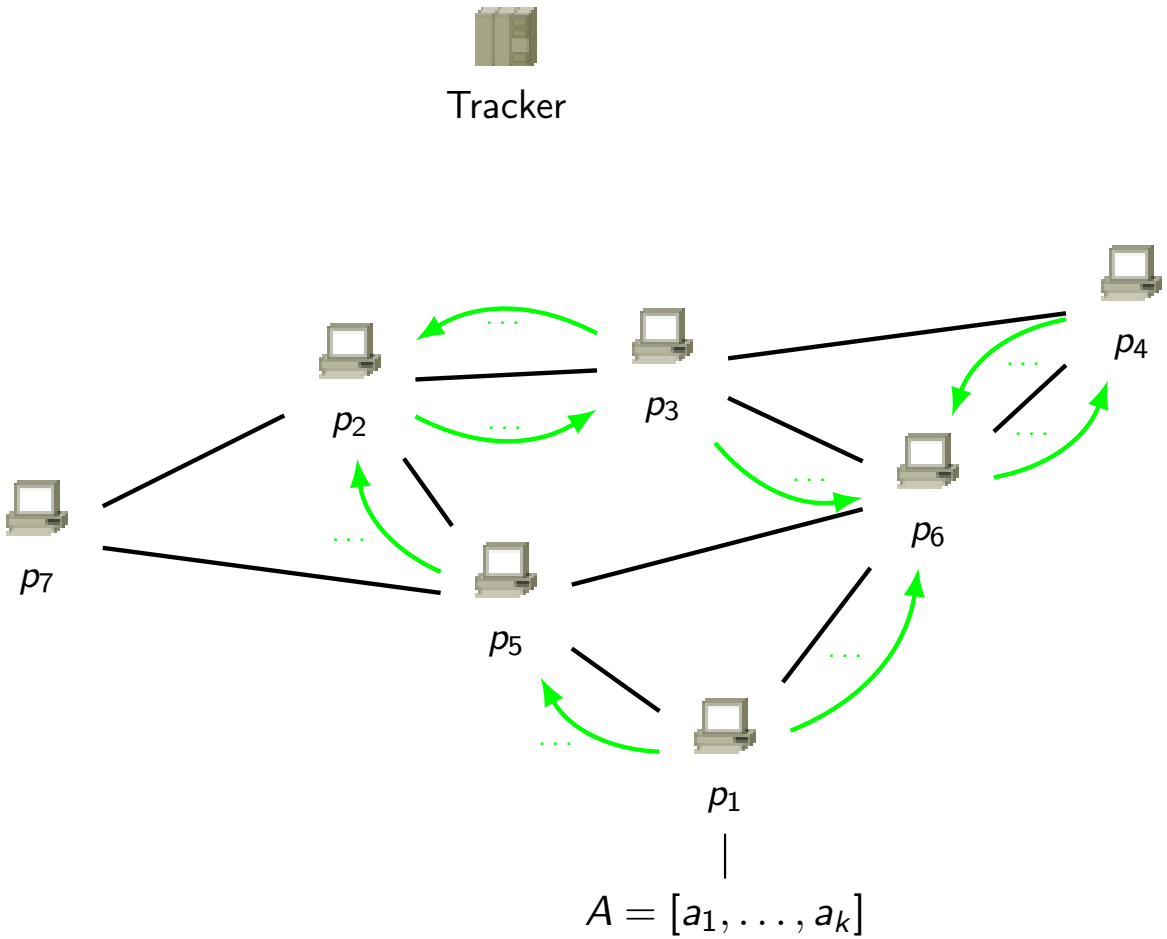
BitTorrent



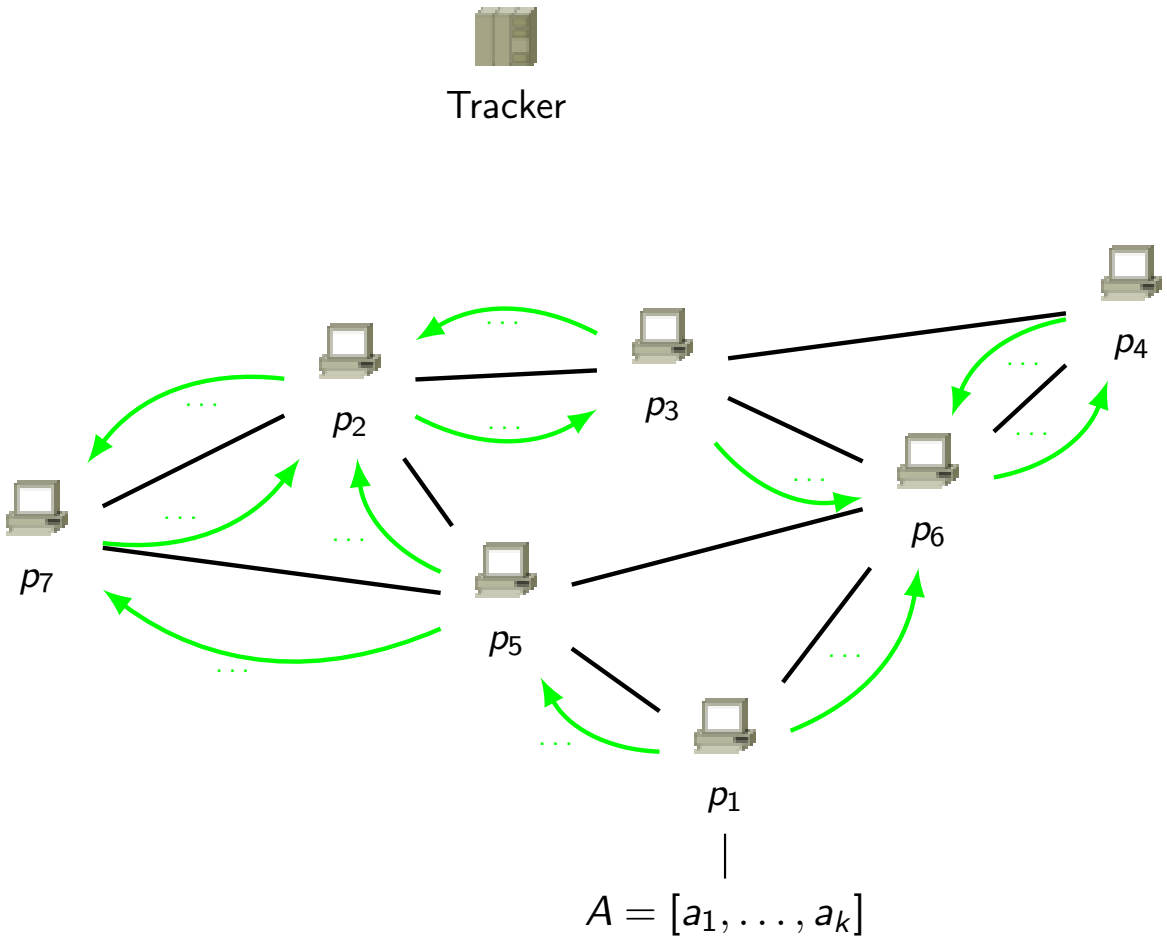
BitTorrent



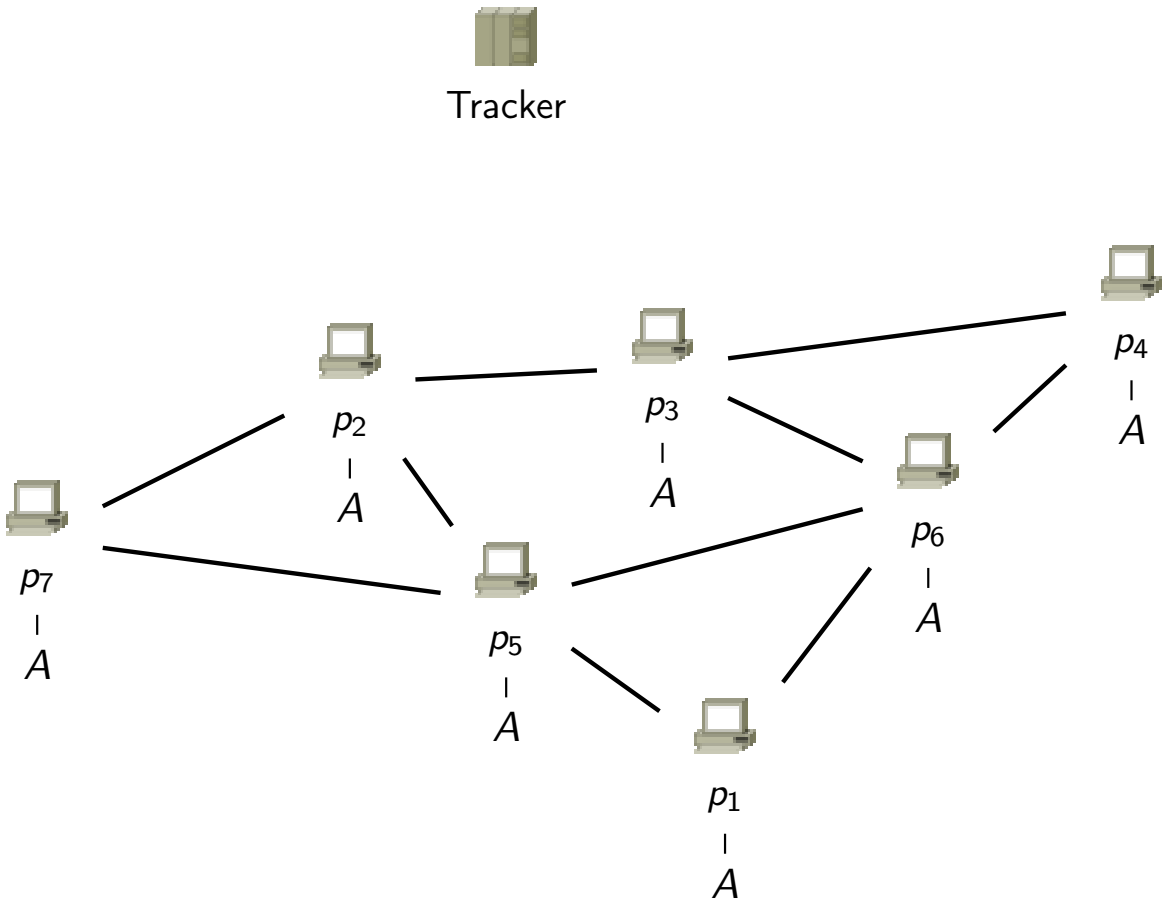
BitTorrent



BitTorrent



BitTorrent



BitTorrent

Innovation

- Une architecture orientée contenu plutôt qu'utilisateur
- Véritable optimisation de la partie distribution

Limites

- Trackers $\rightarrow$ vulnérabilité
- Durée de vie des essaims faible / rétention eMule

P2P : Logiciels clés

1998	Napster : distribution de contenu P2P(MP3) distribution
2000	Gnutella : recherche décentralisée
2000	KaZaA/eDonkey : indexation hiérarchique
2003	BitTorrent : distribution performante
2006	Fin de Razorback 2.0 : émergence des DHTs
2006-2009	**PPLive/UUSee/TVAnts/... : P2P streaming**

P2PTV

Histoire

- Origine : évènements sportifs
- Une flopée de clients propriétaires incompatibles (UUSee, PPLive, TVAnts,. . .)
- Principalement utilisé en Asie, et chez des communautés «expats»

Idée : réduire le délai

- À-la-demande, commencer sans tout récupérer d'abord
- Optimiser la diffusion des chunks
- Booster le système à coups de serveurs

P2PTV

Innovation

Algorithmes pour optimiser le délai

Limites

- La barrière logarithmique
- Youtube et les autres

Partie II

Les Tables de Hachage Distribuées (DHTs)

Sommaire

Tables de Hachage Distribuées

Motivation

Localiser un contenu à moindre coût

Problèmes à résoudre

Décentralisation le travail doit être réparti entre les pairs

Scalabilité la recherche doit marcher même sur les très grands systèmes

Dynamicité les pairs vont et viennent, gentiment ou pas

Tables de Hachage Distribuées

Fonction de hachage

fonction $h : E \mapsto F$

E = tout ! (ensemble de fichiers, noms de fichiers, adresses IP...)

F = espace des clés (cercle $[0, ..2^n[$, carré, hypercube, ...)

Idée de base d'une DHT (Distributed Hash Table)

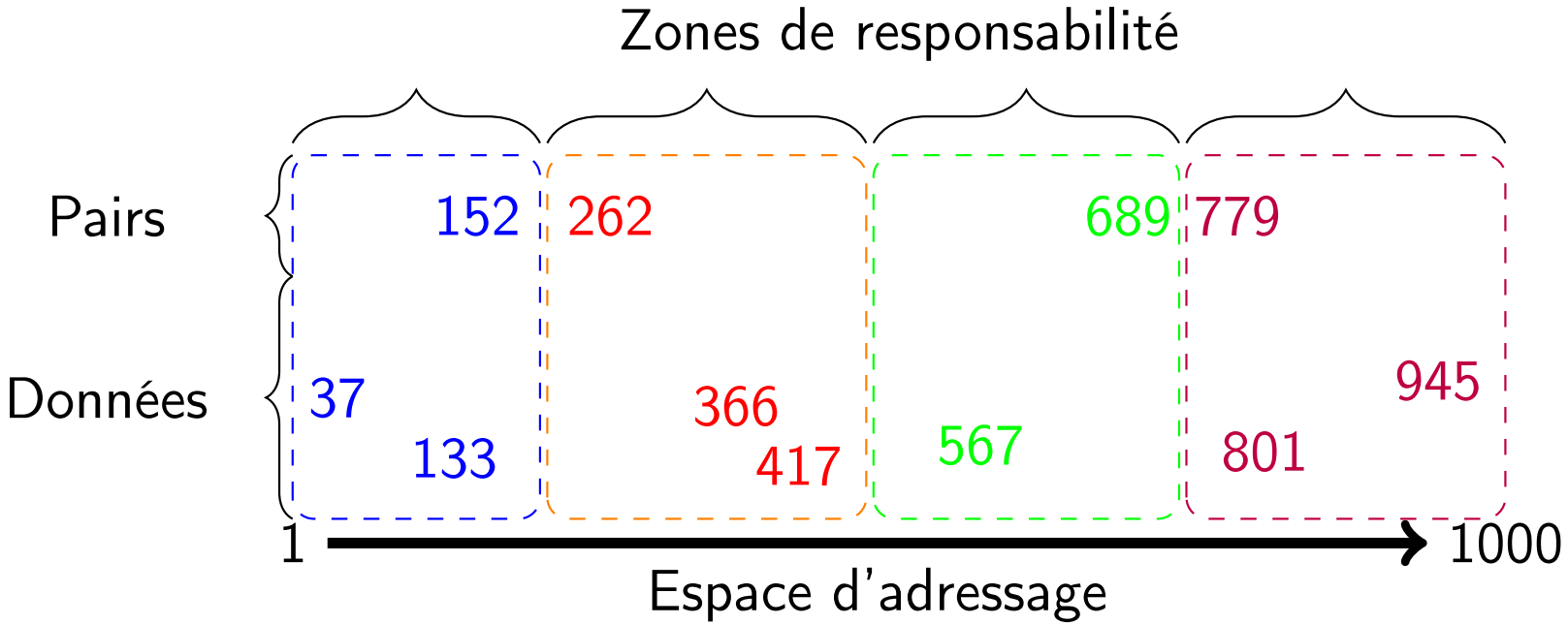
- Chaque pair a un numéro (clé ou hash)
- Chaque donnée a un numéro (clé ou hash)
- Un pair gère une zone de F (les données dont la clé est dans la zone)
- Trouver une clé $\rightsquigarrow$ trouver une route vers le pair qui gère la clé
- Beaucoup de variante suivant F et le mécanisme de routage
  - CAN, Pastry/Tapestry, Chord,...

Une chose à retenir

Gérer une clé ne veut pas dire posséder le contenu !
(la bonne réponse est : savoir

- quels contenus ont la clé
- qui les possèdent)

Exemple : indexation par plus proche



Le pair de hashcode 152 gère/indexe les données de hashcode 37 et 133...

Exemple de requête : “Game of Thrones”

1. h(“Game”) = 1234
2. recherche du pair de hash le plus proche de 1234
3. C’est le pair de hash 1100
4. On lui demande une liste de contenus
5. h(“Thrones”) = 5678
6. rech. plus proche : 5600 $\rightarrow$ requete $\rightarrow$ contenus
7. Intersection des listes de contenus
8. Affichage
9. Choix du contenu de hash 6666 par l’utilisateur
10. recherche du pair de hash le plus proche de 6666 : 6667
11. demande d’une liste d’IPs a 6667
12. téléchargement depuis ces IPs

Routage par la clé

Algorithme de recherche générique

- Le pair P recherche une donnée de clé c
- Cette donnée est indéxée par le pair Q
- P connait quelques voisins
- Il transmet la requête au voisin le «plus proche» de c
- qui transmet, etc.
- finalement on arrive à Q

Analyse

- On va essayer de le faire en $O(\log n)$ demandes (sauts)
- Ressemble a la recherche dichotomique
- Plus d'inondation !

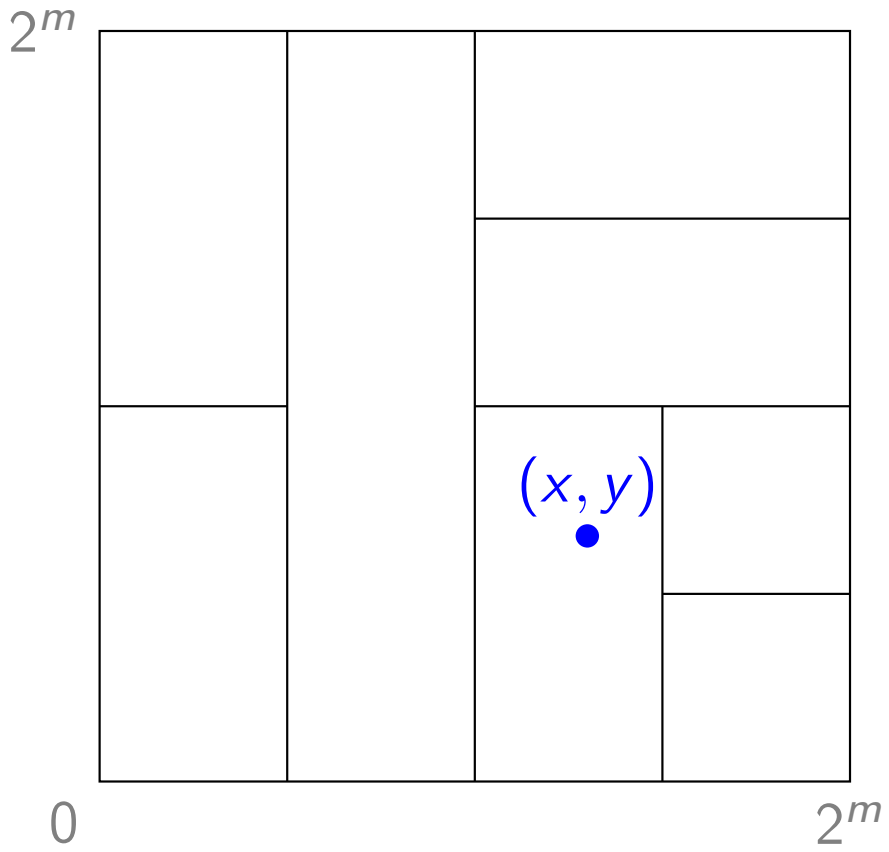
Routage par la clé

Comment faire ?

- Le réseau des connaissances essaie d'avoir une certaine **topologie** :
  - CAN : tore multidimensionnel
  - Chord : cercle
  - Viceroy : papillon
  - Tapestry/Pastry/Kademlia : hypercube
  - D2B : de Bruijn
- On fait une marche dans ce graphe
  - de voisin en voisin
  - à chaque étape, on progresse vers le but
  - Si le diamètre du graphe est $O(\log n)$...
- On arrive donc sur le voisin recherché en $O(\log n)$ étapes

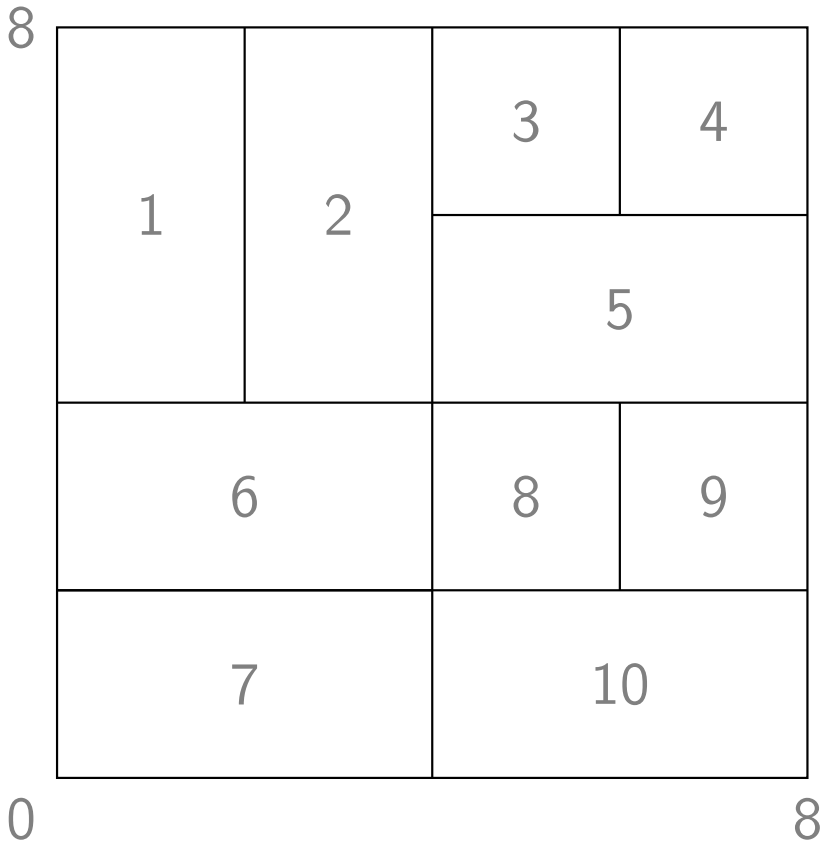
Sommaire

Content-Addressable Network (CAN)



- F : hypercube à d dimensions ($d = 2$)
- Chaque pair gère un rectangle
- d fonctions de hachage $\rightsquigarrow$ les clés correspondent à un d-uplet (une paire) de hashs
- Routage vers le voisin de la zone la plus proche pour la distance de Manhattan

Exemple



Exemple

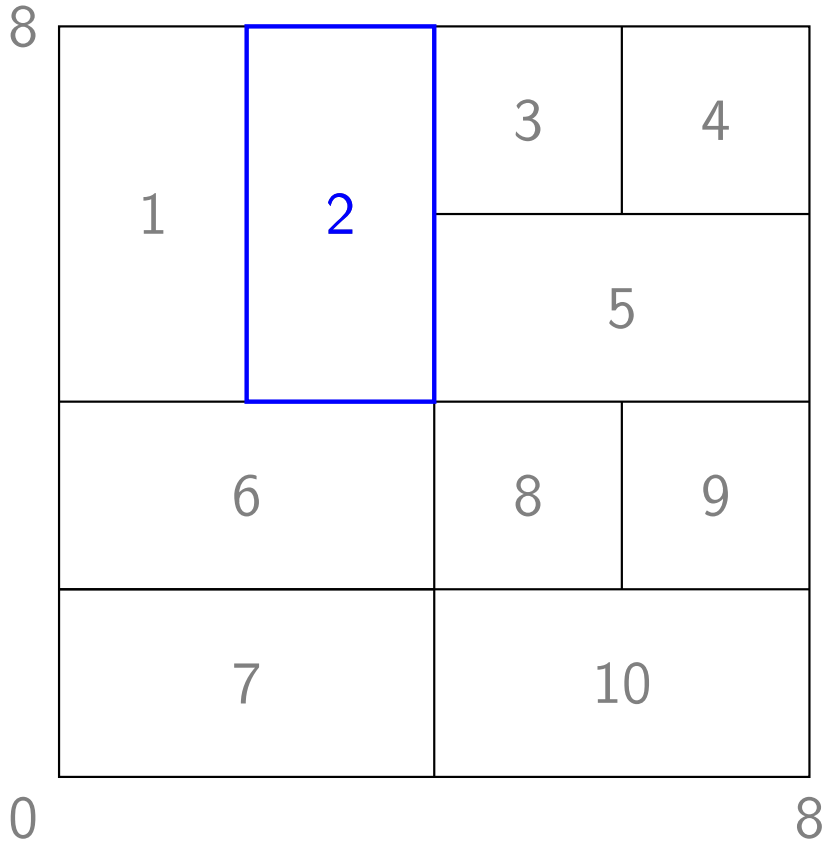


Table de routage de 2

Pair	Zone
2	$(2,4)$ à $(4,8)$
1	$(0,4)$ à $(2,8)$
3	$(4,6)$ à $(6,8)$
5	$(4,4)$ à $(8,6)$
6	$(0,2)$ à $(4,4)$

Exemple

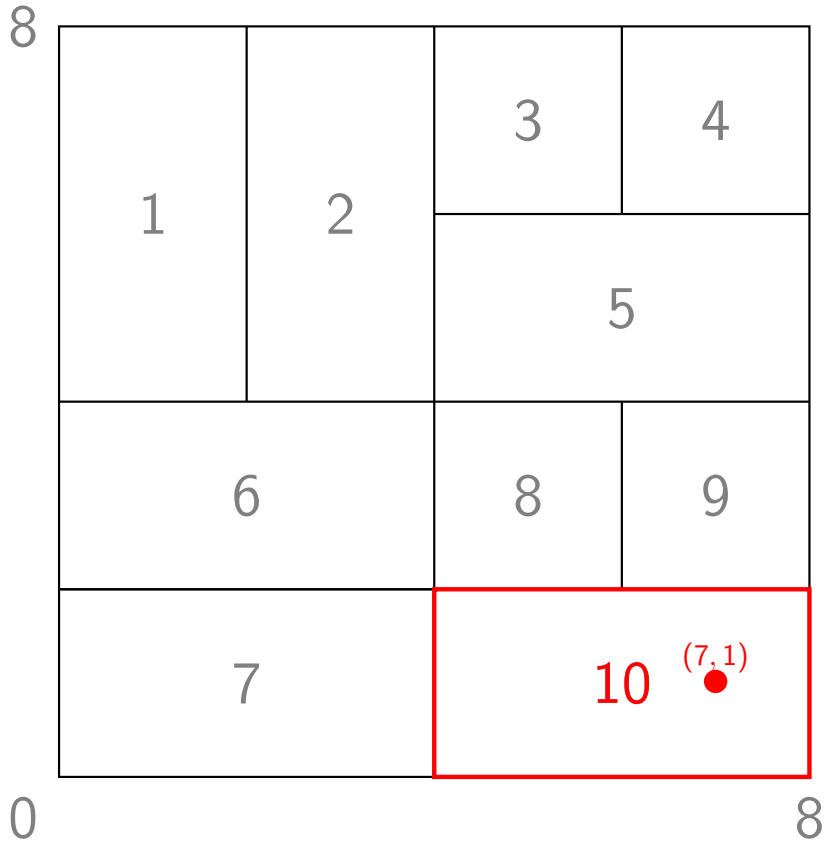


Table de routage de 2

Pair	Zone
2	$(2,4)$ à $(4,8)$
1	$(0,4)$ à $(2,8)$
3	$(4,6)$ à $(6,8)$
5	$(4,4)$ à $(8,6)$
6	$(0,2)$ à $(4,4)$

- Recherche clé $(7,1)$ possédée par le pair 10.

Exemple

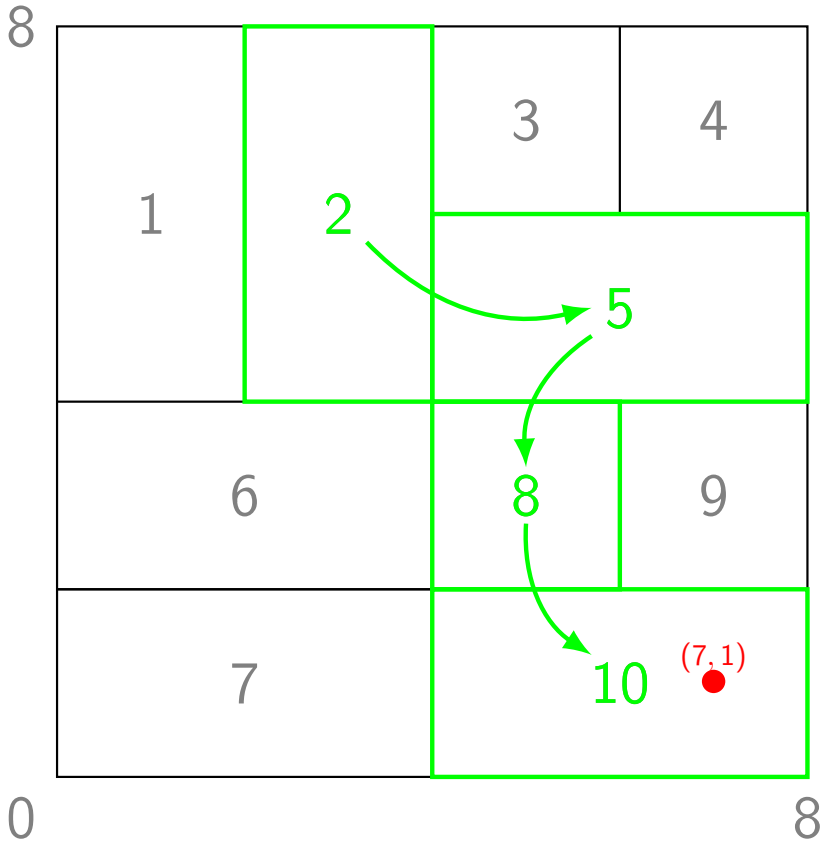


Table de routage de 2

Pair	Zone
2	$(2,4)$ à $(4,8)$
1	$(0,4)$ à $(2,8)$
3	$(4,6)$ à $(6,8)$
5	$(4,4)$ à $(8,6)$
6	$(0,2)$ à $(4,4)$

- Recherche clé $(7,1)$ possédée par le pair 10.
- Route de 2 vers $(7,1)$: $2 \rightarrow 5 \rightarrow 8 \rightarrow 10$.

Départs et arrivées

- Arrivée de A : A prend une clé (X, Y) et contacte un pair B quelconque (bootstrap)
  - A contacte par B le pair C responsable de (X, Y)
  - C partage sa zone avec A (les tables de routage sont ajustées)
  - A et C contactent leurs voisins pour mise à jour des tables
- Départ d'un pair A : A contacte ses voisins et détermine qui doit prendre le relais
  - Envoie des mise à jour aux voisins
  - la zone peut revenir à plusieurs voisins

Propriétés

- Facile à comprendre, facile à étendre
  - Chevauchements des zones pour plus de robustesse
  - Dimension d quelconque
  - Ré-agencement dynamique des zones pour rééquilibrer la charge
- Taille de la table : $O(d)$ ☺
- Nombre de sauts : $O(dn^{1/d})$ ☹
- Bilan : bof (coûteux en messages et lent)

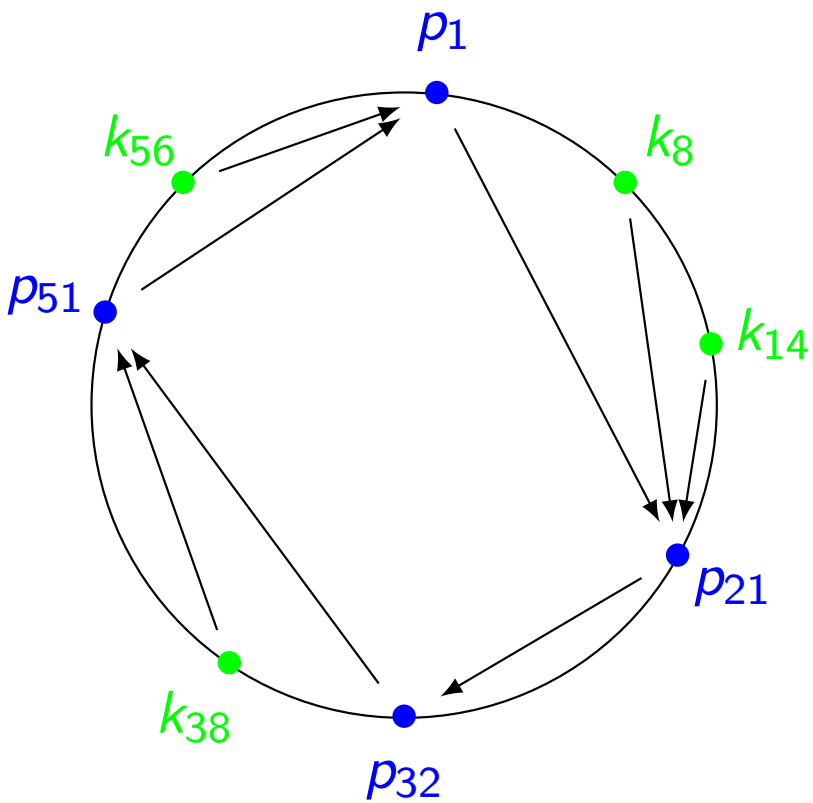
Sommaire

Protocole Chord

- Toujours le même principe ...
- Fonction de hachage : SHA-1
- $F = [0, ..2^m[(m = 160)$ vu comme un cercle modulo 2^m : l'anneau Chord
- Base du routage : successeur(k) = premier pair dont la clé est strictement supérieure à k modulo 2^m

Successeurs

Un pair gère l'ensemble des clés dont il est le successeur

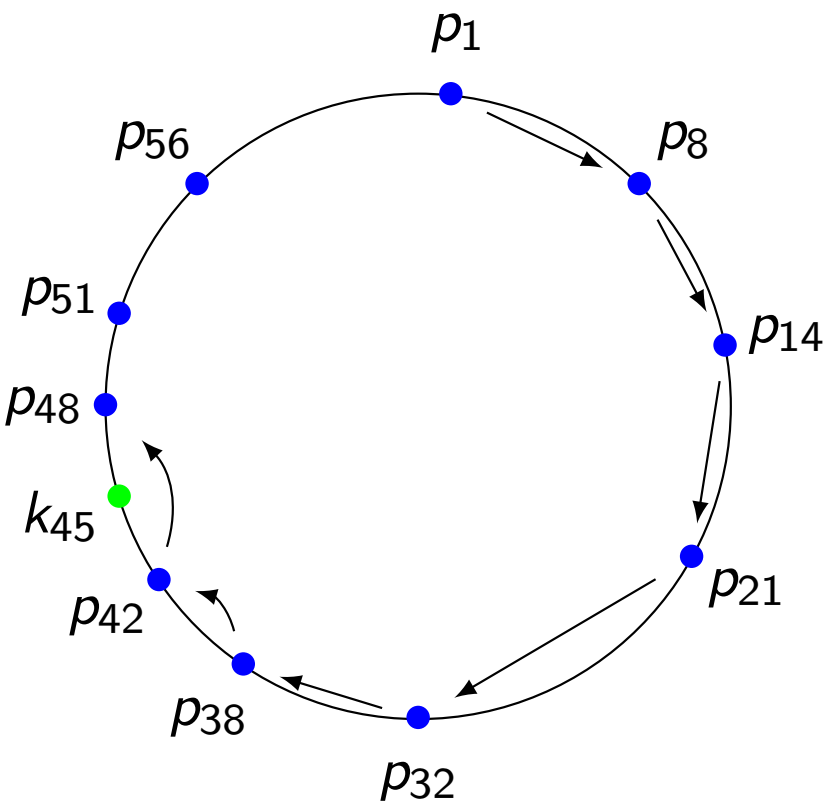


Un théorème

Avec forte probabilité, pour N pairs et K clés :

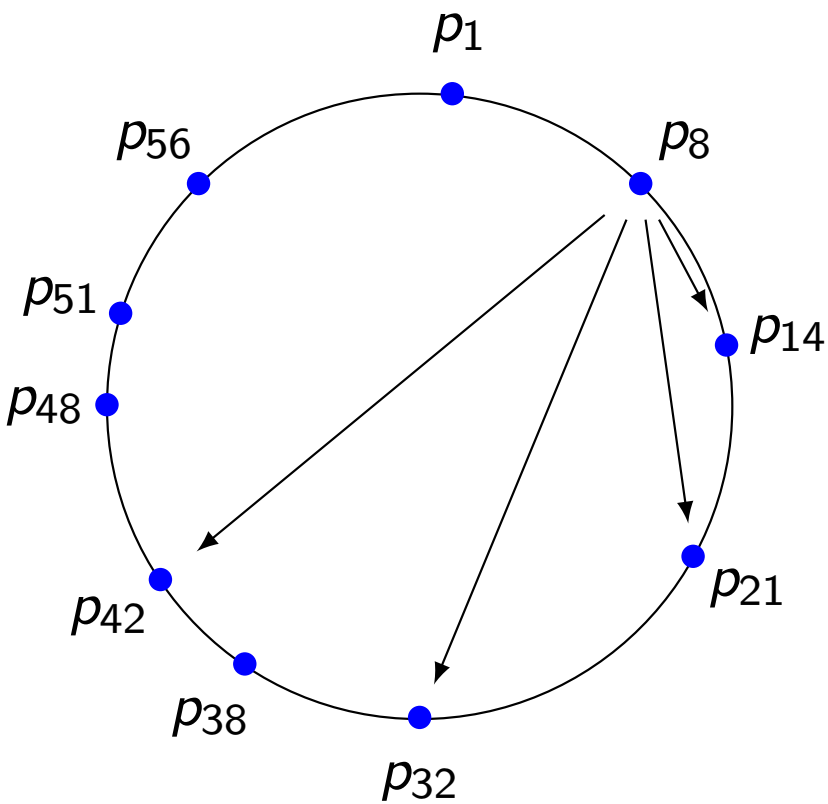
- Chaque pair gère au plus $\frac{(1+C)K}{N}$ clés
- Si un pair arrive ou part, moins de $O(\frac{K}{N})$ clés changent de propriétaire
- $C = O(\ln(N))$

Routage par successeur



- Exemple : 1 cherche 45 (géré par 48)
- ☹☹☹

Solution : finger table

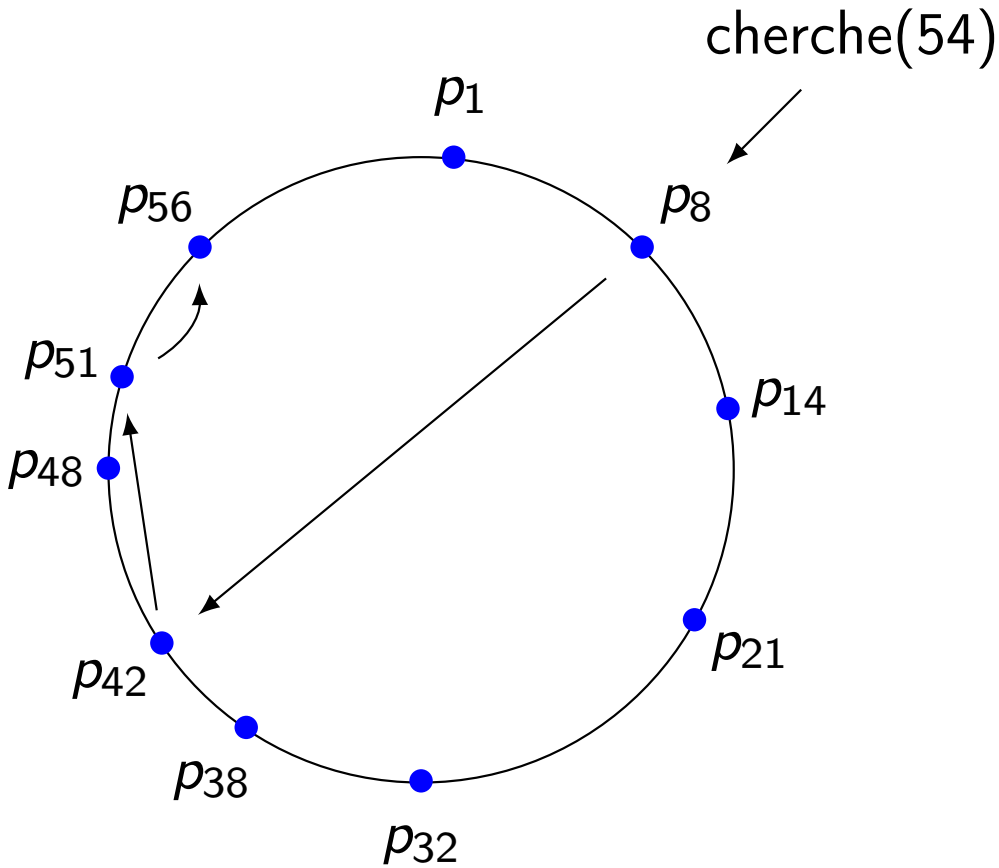


- finger(k) : successeur de $n + 2^{k-1}$ modulo 2^m

Finger table de p_8

$S(8 + \{2^0, 2^1, 2^2\})$	p_{14}
$S(8 + \{2^3\})$	p_{21}
$S(8 + \{2^4\})$	p_{32}
$S(8 + \{2^5\})$	p_{42}

Routage avec Finger table



Routage amélioré

- Chaque saut raccourcit la distance de moitié (à peu près)
- Théorème : avec forte probabilité, le nombre de saut nécessaires pour trouver le responsable d'une clé est $O(\ln(N))$

Départs et arrivées

- En arrivant, un pair N contacte un point d'entrée A
  - A cherche B, prédécesseur de N
  - N devient le successeur de B
  - N contacte B, récupère sa table
  - N ajuste les fingers (y compris des autres)
  - N contacte son successeur et partage les clés
- En partant, un pair contacte ses prédécesseurs (finger inclus) pour signaler son départ. Les clés sont envoyées au successeur et le successeur est signalé au prédécesseur.

Propriétés de Chord

- Correct même en cas de finger corrompues (il reste la route des successeurs)
- Redondance possible pour éviter la perte de données
- Possibilité de mettre en cache des résultats pour accélérer des requêtes futures
- Assez simple à comprendre (si, si !)

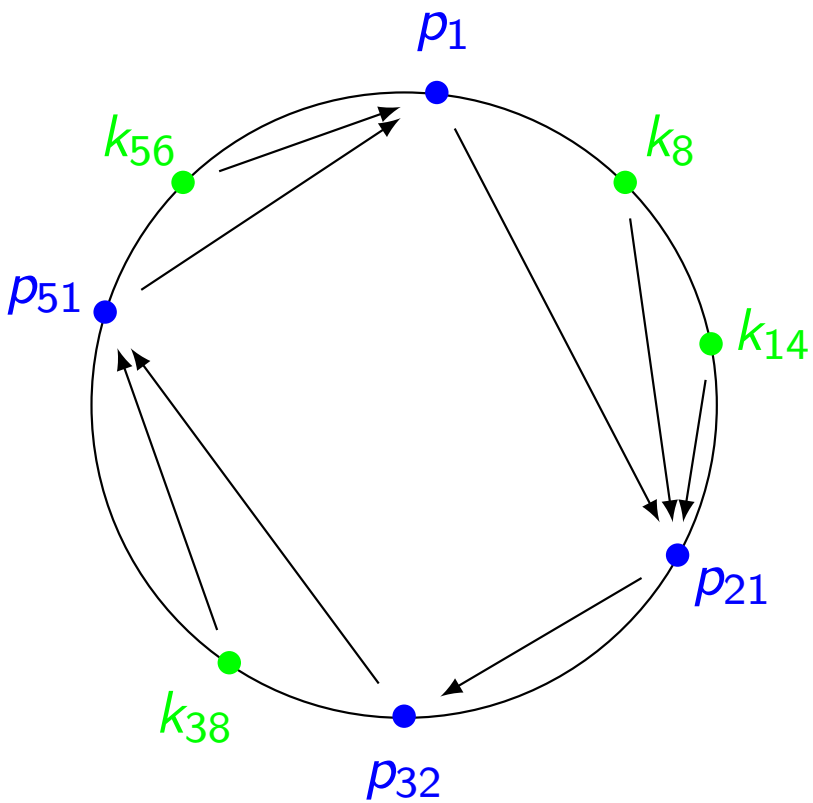
Sommaire

Viceroy

- But du jeu : faire du chord à degré constant
- $F = [0, ..2^m[$ vu comme un cercle modulo 2^m, comme Chord
- Base du routage : $\underline{\text{successeur}(k)}$ et $\underline{\text{prédécesseur}(k)}$

Successeurs

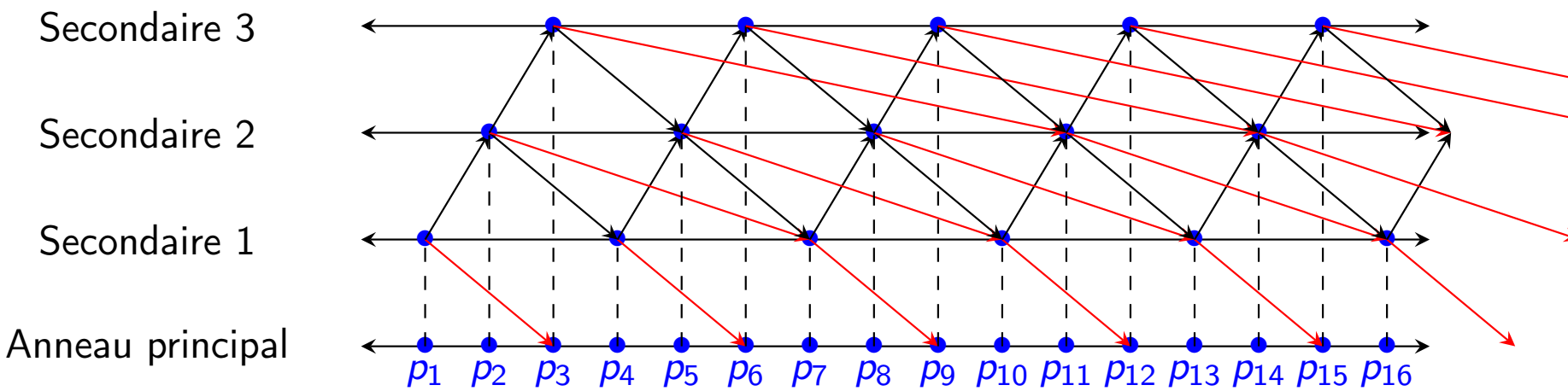
Un pair gère l'ensemble des clés dont il est le successeur



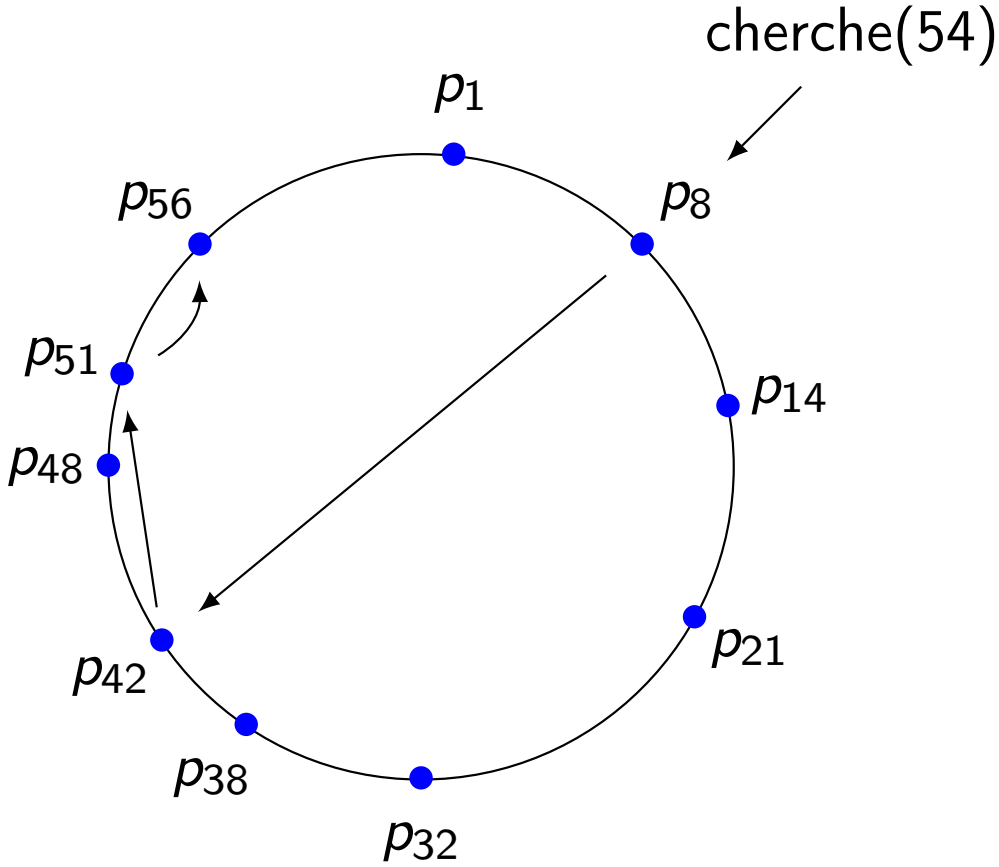
Solution Viceroy : graphe papillon

- Chaque nœud appartient, en plus de l'anneau principal, à un des $\log(n)$ anneaux secondaires
- Un nœud de niveau p possède 7 voisins :
  - 2 voisins dans l'anneau principal ;
  - 2 voisins dans l'anneau secondaire ;
  - Deux ascenseurs ;
  - Un lien long vers $succ(x + 2^p)$ dans l'anneau du dessous.

Graphe «papillon»



Rappel : routage Chord

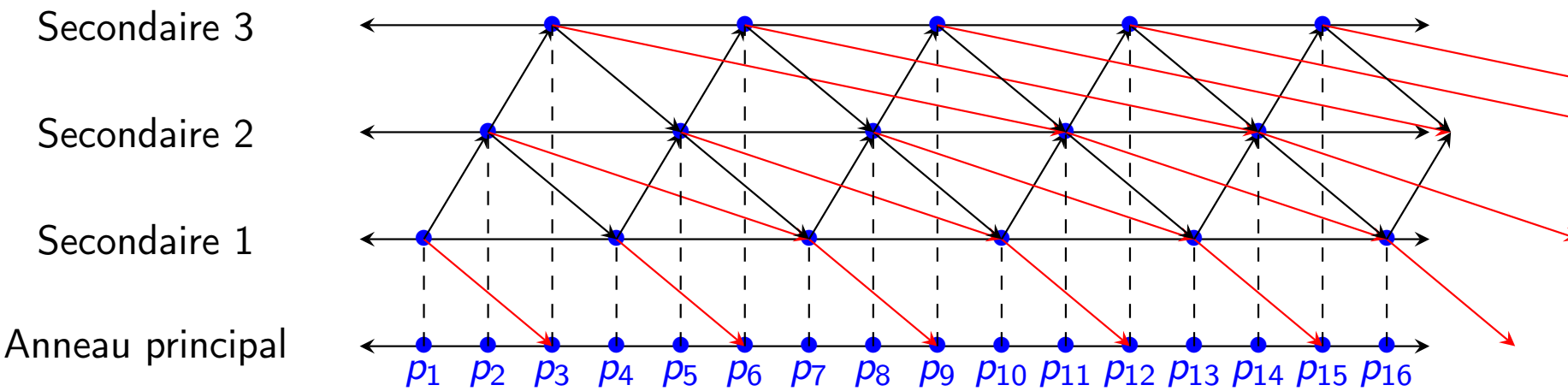


Routage papillon

Idée : imiter Chord

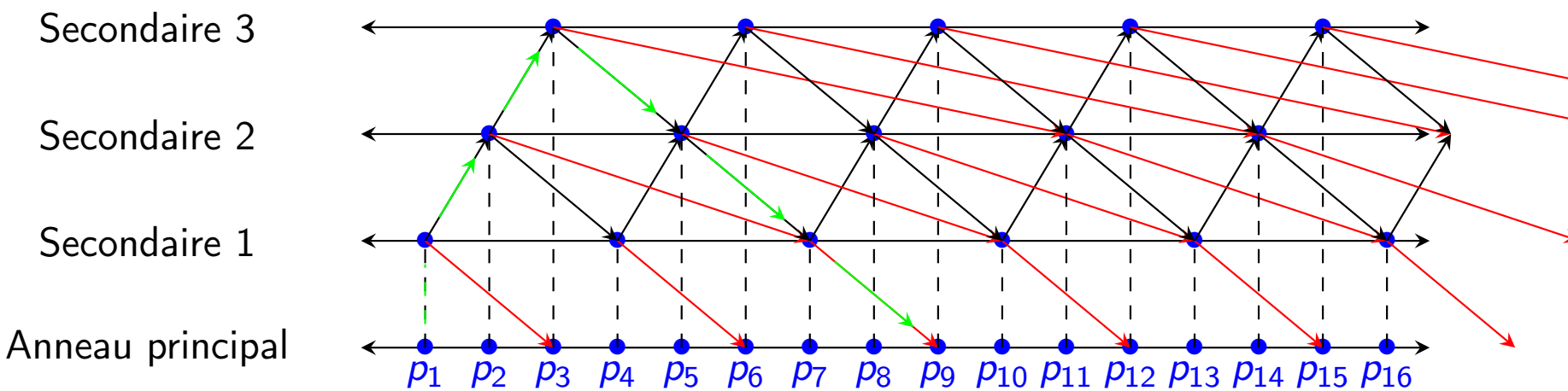
- Toujours : si on risque de dépasser, on route sur l'anneau principal
- On commence par monter au niveau le plus puissant ;
- Puis on descend les niveaux un par un, en utilisant
  - Le lien long si possible ;
  - L'ascenseur normal sinon.
- Coût total : quelques $\log(n)$.

Routage papillon



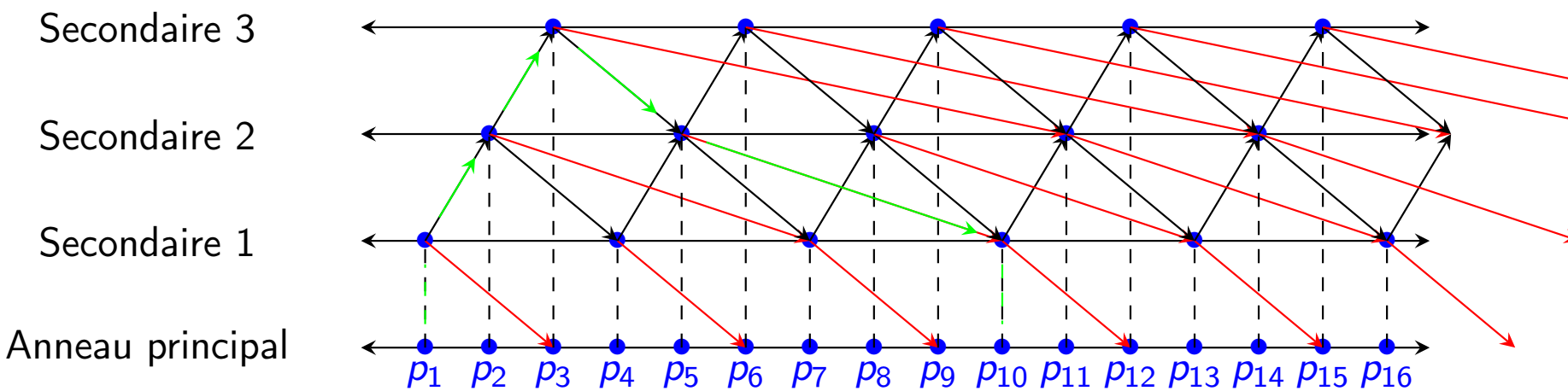
Routage papillon

$p_1 \rightarrow p_9$



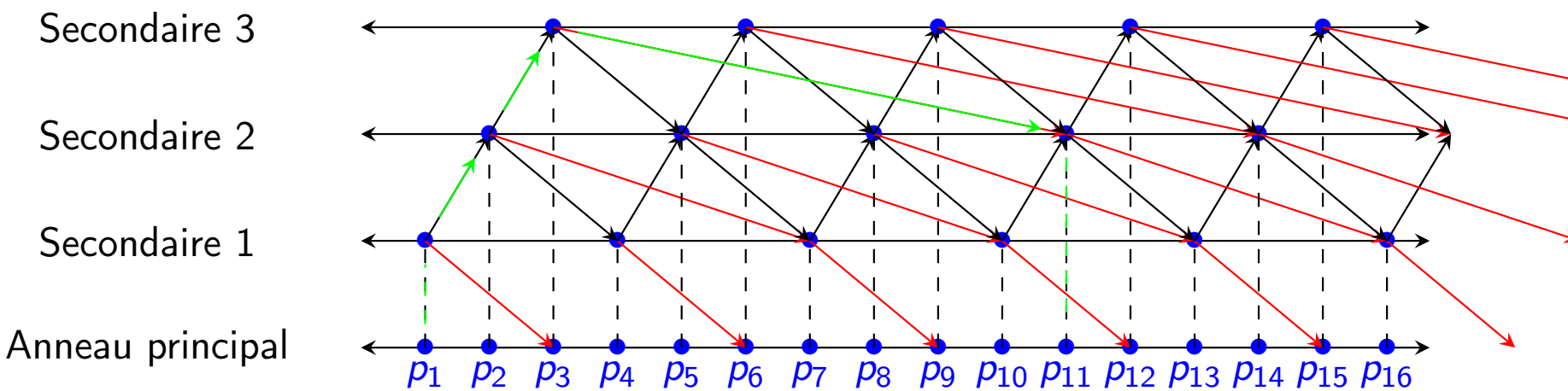
Routage papillon

$p_1 \rightarrow p_{10}$



Routage papillon

$p_1 \rightarrow p_{11}$



Propriétés

- Facile à comprendre... si on a bien compris Chord
- Taille de la table : $O(1)$ ☺☺
- Nombre de sauts : $O(\log(n))$ ☺
- Bilan : super... en théorie

Sommaire

Topologie : l'hypercube

Idée présente dans Tapestry, Pastry et Kademlia

Idée de base

- Chaque pair P a un identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ (typiquement $m = 160$: SHA-1)
- Chaque pair P essaie de connaître un voisin par préfixe :
  - $\overline{h_1}$
  - $h_1 \overline{h_2}$
  - $h_1 h_2 \overline{h_3}$
  - ...
  - $h_1 h_2 ... h_{v-1} \overline{h_v}$
- v : plus long préfixe commun avec P
- $v = O(\log N)$ en moyenne (car fonction de hachage uniforme)

Topologie : l'hypercube

Idée présente dans Tapestry, Pastry et Kademlia

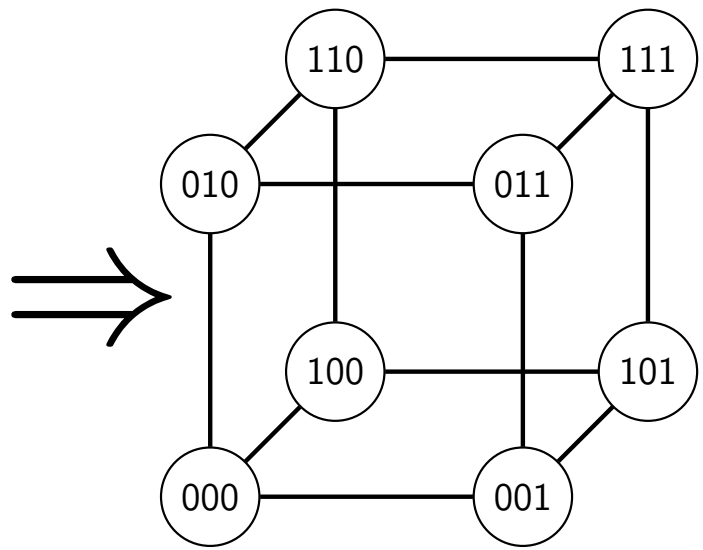
Exemple

- Le pair d'indentifiant 0100010010001101011010010101...01 a comme voisins :
  - **1**1001001001110110101010010...10
  - **00**0010011110100110101001 1...11
  - **011**0100010101001111001 00...11
  - ...
  - **01000101**0100100100010100 0...00
- On $v = 8$ donc 7 bits en commun avec son plus proche voisin :
- de l'ordre de 2^8 pairs dans le système

Topologie : l'hypercube

Exemple d'application de "un voisin par préfixe" :
le graphe des connaissances est l'hypercude de dimension v
v a la même valeur pour tout le monde

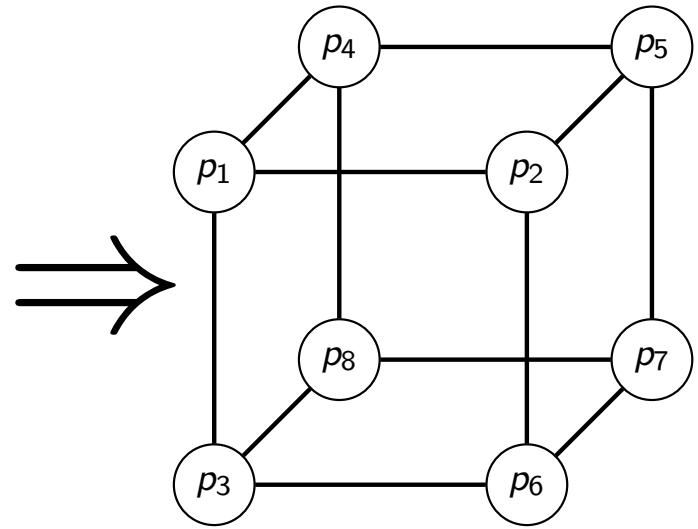
- $p_1 = \mathbf{010}101001001$
- $p_2 = \mathbf{011}011101010$
- $p_3 = \mathbf{000}000110101$
- $p_4 = \mathbf{110}110110010$
- $p_5 = \mathbf{111}001111000$
- $p_6 = \mathbf{001}010101001$
- $p_7 = \mathbf{101}110010100$
- $p_8 = \mathbf{100}011001110$



Topologie : l'hypercube

Exemple d'application de "un voisin par préfixe" :
le graphe des connaissances est l'hypercude de dimension v
v a la même valeur pour tout le monde

- $p_1 = \mathbf{010}101001001$
- $p_2 = \mathbf{011}011101010$
- $p_3 = \mathbf{000}000110101$
- $p_4 = \mathbf{110}110110010$
- $p_5 = \mathbf{111}001111000$
- $p_6 = \mathbf{001}010101001$
- $p_7 = \mathbf{101}110010100$
- $p_8 = \mathbf{100}011001110$



Dimension du cube ?

v n'a pas la même valeur pour tout le monde !

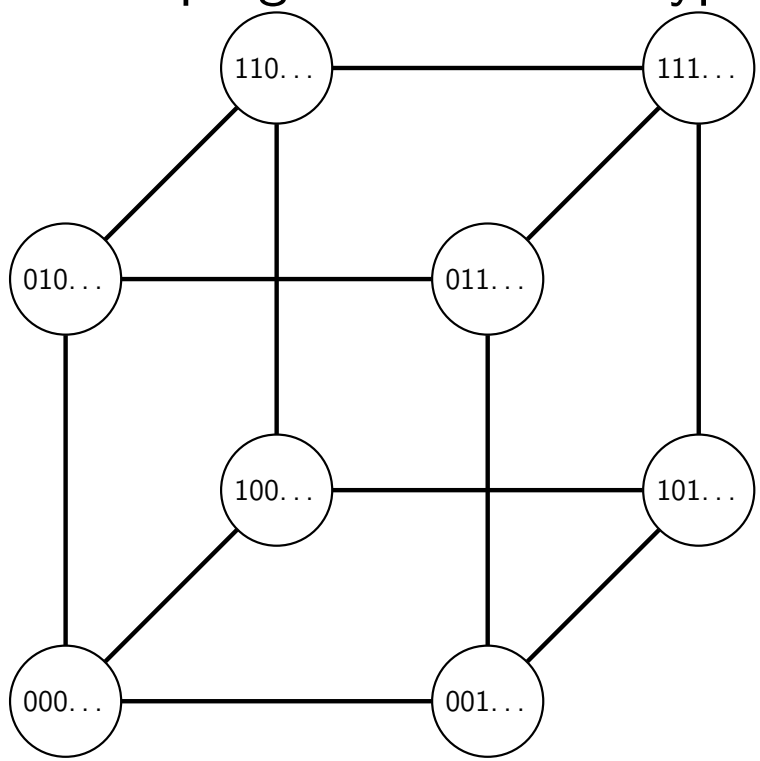
- en moyenne : $v_{moy} = \log N$
- au pire : $v_{max} = 2 \log N$

Algorithme de recherche : Plaxton Mesh

- Intuition : on «compose» l'adresse de destination
- On fait une recherche incrémentale, chiffre par chiffre
- À chaque coup, on se rapproche de la destination

Algorithme de recherche

Le pair P d'identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ recherche la donnée D d'identifiant (hashcode) $H(D) = d_1 d_2 ... d_m$
$H(P)$ XOR $H(D)$ dit où commencer la requête
Ensuite requêtes aux voisins : progression dans l'hypercube.

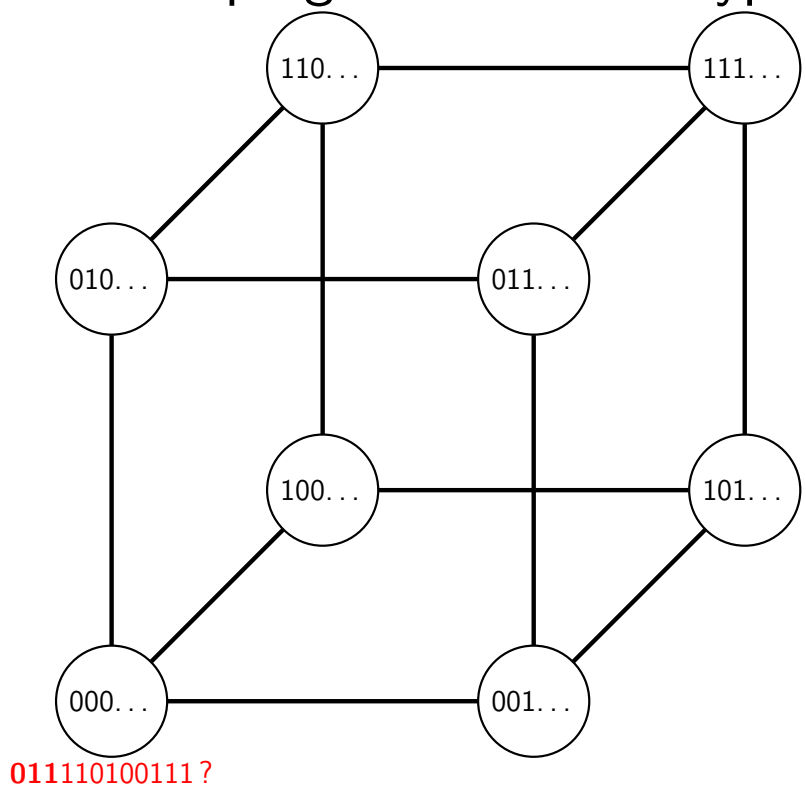


Algorithme de recherche

Le pair P d'identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ recherche la donnée D d'identifiant (hashcode) $H(D) = d_1 d_2 ... d_m$

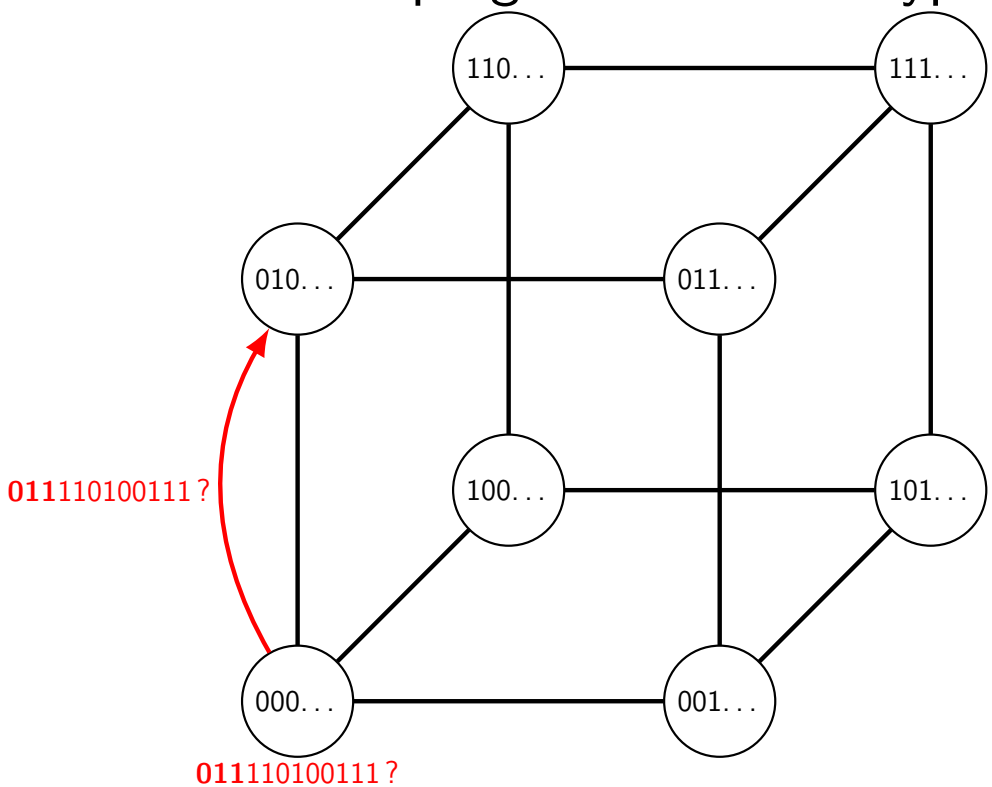
$H(P)$ XOR $H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.



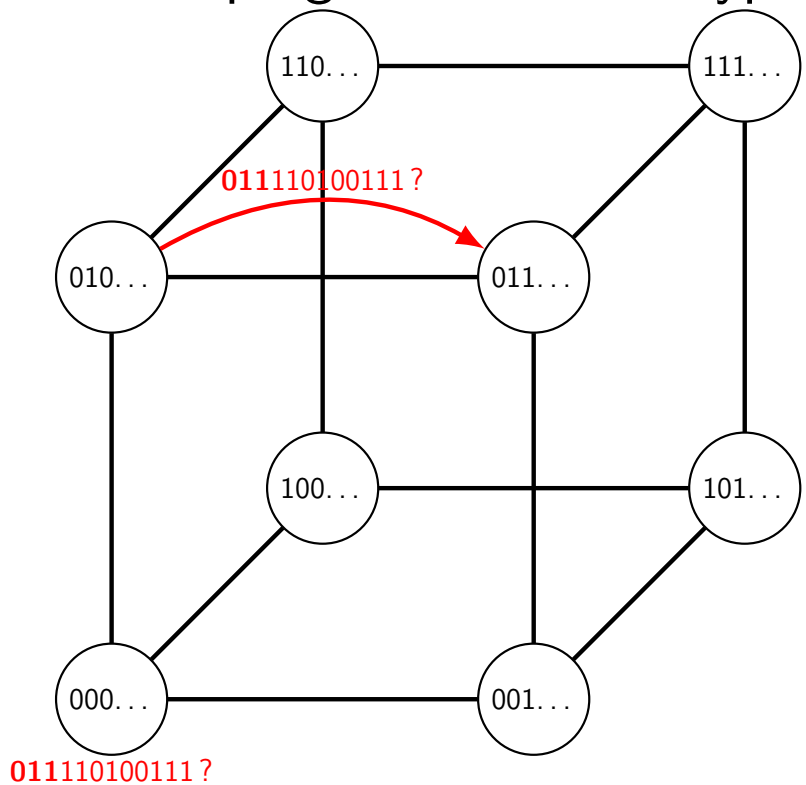
Algorithme de recherche

Le pair P d'identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ recherche la donnée D d'identifiant (hashcode) $H(D) = d_1 d_2 ... d_m$
$H(P)$ XOR $H(D)$ dit où commencer la requête
Ensuite requêtes aux voisins : progression dans l'hypercube.



Algorithme de recherche

Le pair P d'identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ recherche la donnée D d'identifiant (hashcode) $H(D) = d_1 d_2 ... d_m$
$H(P)$ XOR $H(D)$ dit où commencer la requête
Ensuite requêtes aux voisins : progression dans l'hypercube.

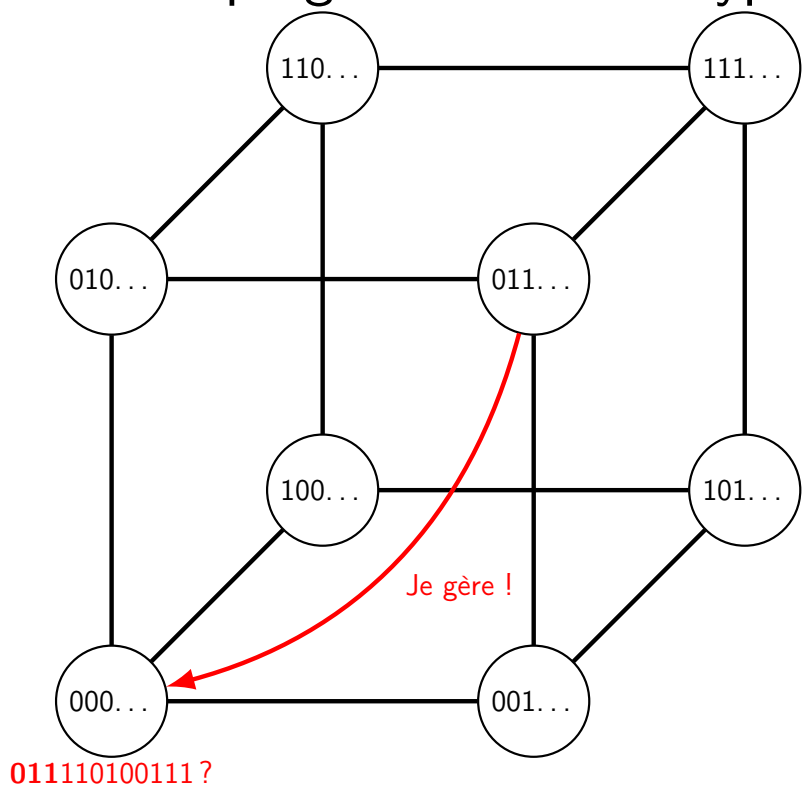


Algorithme de recherche

Le pair P d'identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ recherche la donnée D d'identifiant (hashcode) $H(D) = d_1 d_2 ... d_m$

$H(P)$ XOR $H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.



Algorithme de recherche

Le pair P d'identifiant (hashcode) $H(P) = h_1 h_2 ... h_m$ recherche la donnée D d'identifiant (hashcode) $H(D) = d_1 d_2 ... d_m$
$H(P)$ XOR $H(D)$ dit où commencer la requête
Ensuite requêtes aux voisins : progression dans l'hypercube.

donc :

pour N pairs

- Table de routage en $O(\log N)$ pairs
- Recherche en $O(\log N)$ requêtes

Améliorations de la vitesse de recherche

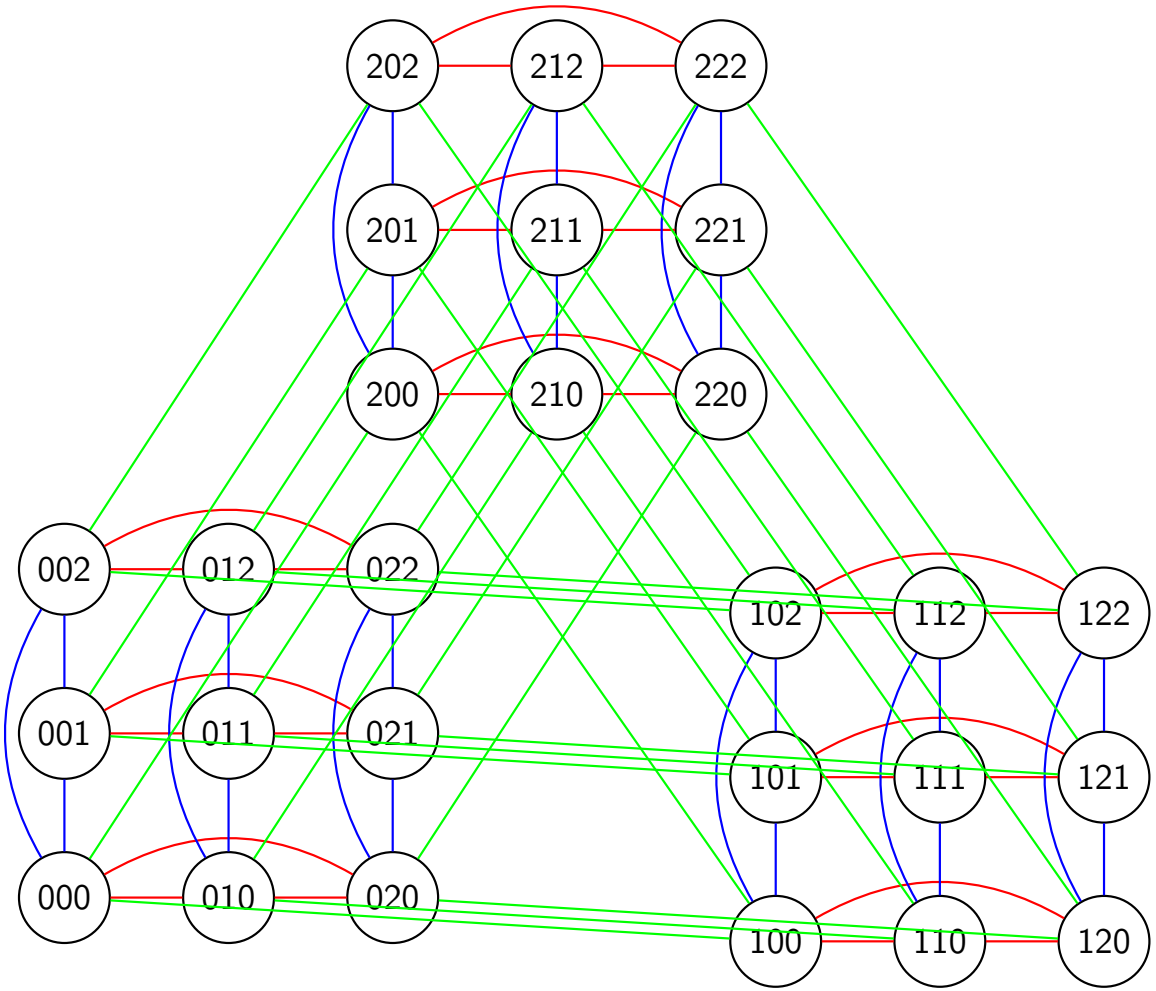
Base plus grande

Au lieu de se placer en base 2 on peut se placer en base b

- Avantage : requête en $\log_b(N)$ et non $\log_2(N)$ on gagne $\frac{\ln b}{\ln 2}$ temps
- Inconvénient : augmente d'un facteur b la taille des tables de routage ($b-1$ et non plus 1 voisins par dimension)

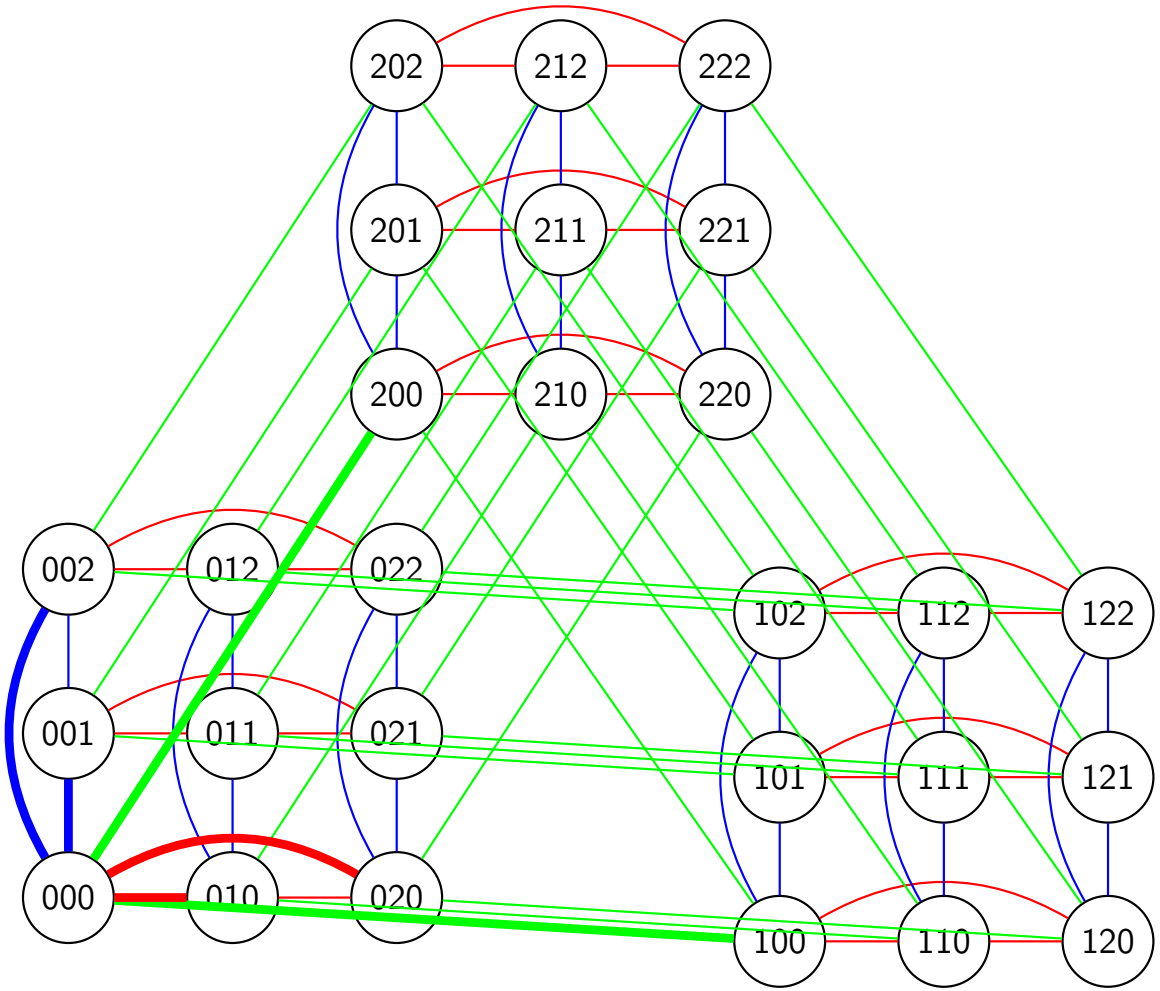
Améliorations de la vitesse de recherche

En base 3, dimension 3 :



Améliorations de la vitesse de recherche

En base 3, dimension 3 :



Mieux que $\log N$?

Briser la borne de $O(\log N)$

Si on se place en base $b = \ln N$

- Requêtes en temps $O(\frac{\ln N}{\ln \ln N})$ (5,2 pour $N = 1000000$)
- $k \times O(\frac{(\ln N)^2}{\ln \ln N})$ contacts par pair ($72\times$k pour $N = 1000000$)

Améliorations de la robustesse

Nombre de voisins

Au lieu de garder un seul voisin par dimension, on en garde k

- Avantage : gère la dynamique
- Inconvénient : augmente d'un facteur k la taille des tables de routage

Publication multiple

La donnée est stockée chez les k plus proches pairs et non chez LE plus proche

Taille des tables

Pour $N = 2\ 000\ 000$ pairs, $b = 8$ (3 bits), $k = 20$ copies :
$\frac{\ln 2^{21}}{\ln 2^3} * (2^3 - 1) * 20 = 7 * 7 * 20 = 980$ voisins. Faisable.

Insertion dans le réseau ?

Algorithme d'insertion

1. Tirer son identifiant H dans $[0..2^m[$
2. Rechercher H dans la table...
3. ...donne ses nouveaux voisins !
4. Se signaler auprès d'eux pour créer les liens
5. et recupérer les copies des données indexées par eux

Bootstrap

Toujours le même problème !
On suppose par exemple le pair d'identifiant 000..0 connu

Analyse

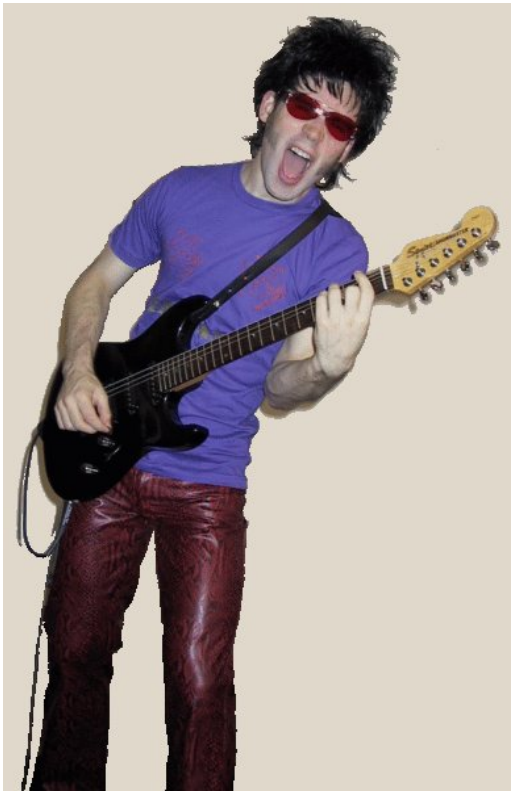
Même temps qu'une recherche : $O(\log_b(N))$

Sommaire

Kademlia [Maymounkov Mazières 2002]

Implantation complète de l'hypercube : Overnet, Kad.

Petar Maymounkov
(MIT)



David Mazières
(Stanford)

Distance XOR

A XOR B noté $A \oplus B$ est interprété comme un nombre.

La distance XOR

- **distance :** $(A \oplus B) + (B \oplus C) \geq (A \oplus C)$
  (car $(A \oplus B) \oplus (B \oplus C) = A \oplus C$ et $X + Y \geq X \oplus Y$)
- **unidirectionelle** : La sphere de centre A et de rayon r contient **un seul** point $B = A \oplus r$

Un tiroir (bucket)

Ensemble de k points a distance $[2^i, 2^{i+1}[\simeq$ une arête de l'hypercube

Table de routage

Table de routage

Table de routage

Chaque pair possède 160 tiroirs. Ceux de numéro $\geq \log N$ sont vides en général (et ceux $\geq 2 \log N + cstre$ vides a.f.p)
Contenu d'une entrée : triplet (IP, port UDP, hashcode du pair)

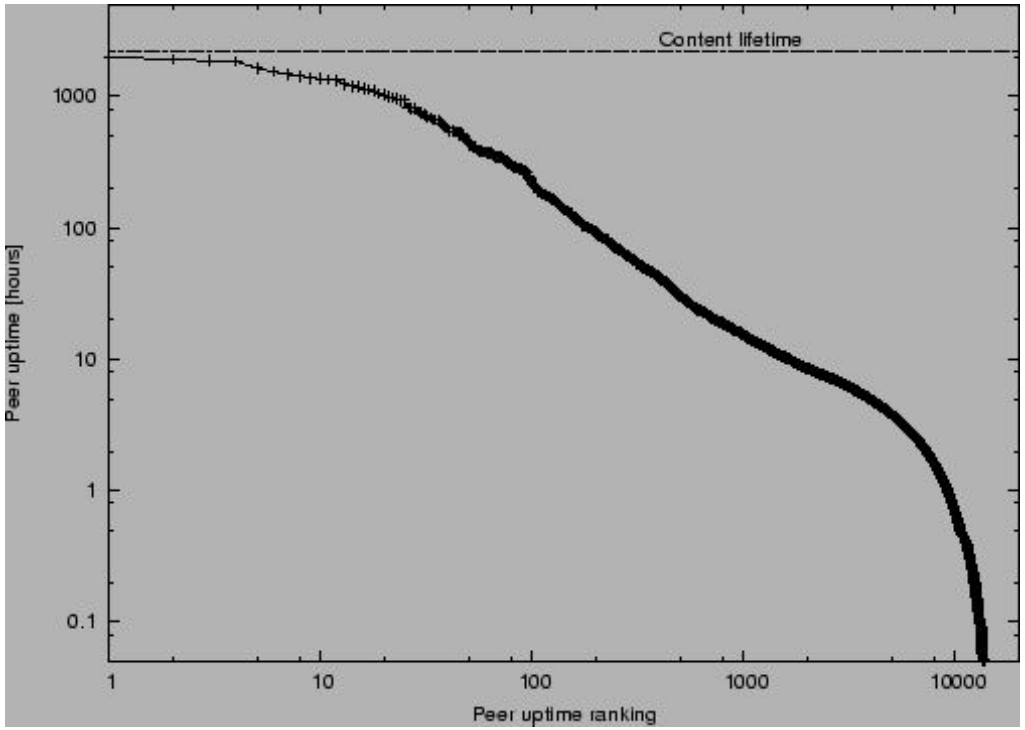
Maintenance du tiroir : LRU amélioré

- Un tiroir contient au plus k adresses de pairs
- Remplissage du tiroir (connaître de nouveaux pairs)
  - réponses aux requêtes
  - requêtes transmise
- Vidange du tiroir
  - Garder les anciens pairs plutôt que les récents
  - Toutefois enlever ceux qui ne répondent plus
  - Un ping par heure

Pourquoi garder les anciens pairs ?

Loi de décroissance

La proba. de rester un temps T suit une loi de puissance



uptime de 53833 pairs sur le torrent "Beyond Good and Evil"
[Pouwelse 2004]

Protocole

Paquets : 4 types (UDP)

1. ping
2. store : ordre de stocker (clé, valeur)
3. recherche de pair : doit retourner k plus proches connus
4. recherche de clé : retourne la clé sinon k plus proches connus

Requêtes en parallèle

- Première tentative : 3 parmi les 20 de k, en parallèle
- Si echec : les 17 suivants en parallèle
- Continuer à la première réponse (sur 3 ou sur 17)

Autres points

Mise en cache des requêtes

- Avantage : réponse plus rapide aux requêtes (qui suivent la même route)
- Inconvénient : le cache grossit !

Solution : temps d'expiration exponentiel en la distance (garder 2 fois plus longtemps ce qui est 1 bit plus près)

Republications

Pour éviter les données périmées :

- republication par le responsable chaque heure (ping aux voisins)
- republication par le propriétaire chaque 24 heures

Conclusion sur Kademlia

Prouvable

La persistance est prouvable

Résistance aux attaques

- par inondation de faux pairs grâce au LRU amélioré
- par fausse requête : écho d'un nombre aléatoire / requête

Forces du protocole

- Métrique XOR : un algo de routage dans l'hypercube plus naturel et cache plus efficace
- requêtes concurrentes (3 puis 17)
- utilise efficacement la distribution du temps de présence

Conclusion sur DHT

Topologies existantes

- Chord [Stoica & alii, 2001] : cercle
- CAN [Ratnasamy et al., 2001] : tore multidimensionnel
- Pastry [Rowstron et Druschel 2001] : hypercube
- Viceroy [Malkhi et al., 2002] : papillon
- Kademlia [Maymounkov Mazières 2002] : hypercube
- D2B [Fraigniaud et Gauron, 2003] : de Bruijn

Conclusion sur DHT

Avantages des DHTs

- Routage rapide : $O(\log N)$ et même en temps $O(\frac{\ln N}{\ln \ln N})$ on se place en base $b = \ln N$ sur l'hypercube
- Tables de routage de taille raisonnable : $O(\log N)$, $O(\frac{(\ln N)^2}{\ln \ln N})$, voire constante.
- Attaques plus dures que sur un serveur
- Réellement pair-à-pair, passe à l'échelle à coût nul

Inconvénients

- Table de hachage seulement $\rightarrow$ pas de requêtes complexes !
- Plus lent qu'un serveur
- Protocoles plus complexes et bugs plus durs à corriger

Remerciements

Ce cours doit beaucoup à Laurent Viennot et à Fabien de Montgolfier.