



Le pair-à-pair

Fabien Mathieu

Plan du cours

1. Introduction
2. Les tables de hachage distribuées
3. Dimensionnement en bande passante
4. Streaming, problèmes de délai

Slides en ligne

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

Partie I

Le pair-à-pair, c'est quoi ?

Sommaire

Définitions

- Approches classiques

- Limites

- Autres possibilités

Principaux concepts

- Fonctionnement

- Objectifs

Une brève histoire du pair-à-pair

- Avant

- Après

Définition par énumération

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

Partage de fichier BitTorrent, eMule,...

Diffusion TVants, PPLive,...

Voix sur IP Skype

Définition par énumération

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 énumération

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 :

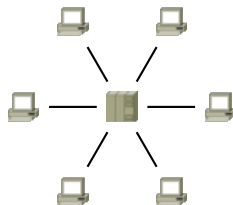
Calcul distribué BOINC project (Seti@home, Folding@home...)

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.

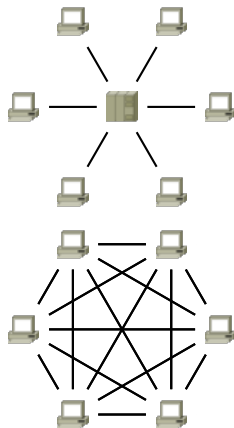


Définition réseau

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 → casser la séparation client/serveur : tous les pairs peuvent être aussi bien clients que serveurs.

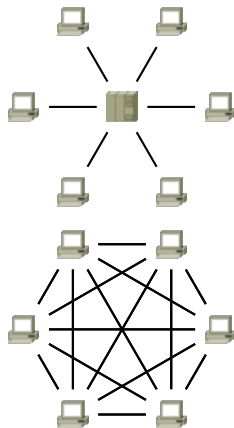


Définition réseau

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 → 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 applications P2P les plus connues. Pourquoi Skype est P2P ?

- Interaction de pair à pair ?



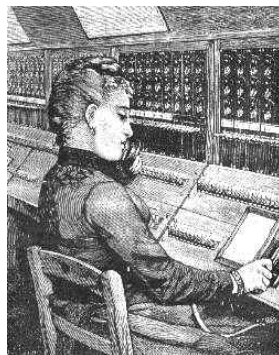
Existe depuis, très, très longtemps

Définition réseau : et Skype ?

Skype est l'une des applications 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 applications 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 applications 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».

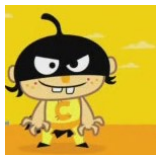
Interlude : une définition légale ?



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».

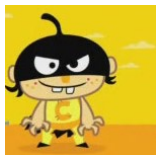
- ▶ Peut recouvrir le partage P2P (data et multimedia)
- ▶ Ne recouvre pas (trop) Skype
- ▶ Couvre la plupart des applications réseaux (incluant l'approche client/serveur) : Apache, SSH...

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.

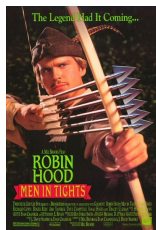
Interlude : une définition légale ?



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

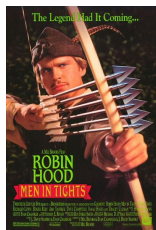
- ▶ Surveillance de contenu cibles (pots de miel)
- ▶ Preuve d'un véritable téléchargement par personne physique impossible à grande échelle
- ▶ Mise en place du délit de négligence caractérisée
- ▶ Échec technique, succès politique ?

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

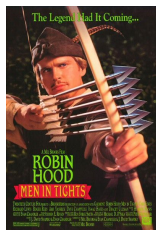


Le P2P utilise les ressources disponibles auquel il a accès (CPU, disque, bande passante) où qu'elles soient, et les redistribue à ceux qui en ont besoin.

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

Définitions

Approches classiques

Limites

Autres possibilités

Principaux concepts

Fonctionnement

Objectifs

Une brève histoire du pair-à-pair

Avant

Après

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 → 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 → DoS

En P2P

- ▶ La capacité de traitement augmente avec le nombre de clients
- ▶ → 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)

→ 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 requis | 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

Définitions

Approches classiques

Limites

Autres possibilités

Principaux concepts

Fonctionnement

Objectifs

Une brève histoire du pair-à-pair

Avant

Après

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

REFER YOUR FRIENDS AND GET UNLIMITED SPACE FREE!!!

YOUR FREE DISK SPACE

- 300 MB Free Virtual Hard Drive
- Private and secure
- Share your files
- Collaborate on files
- 24/7 access



The graphic features a globe with the text '300Megas' in large blue letters. Below it, 'Your Global Hard Drive' is written. A red 'X' icon is next to 'Get more info'. At the bottom, a blue button says 'Sign Up Now'.

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
- ▶ → 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

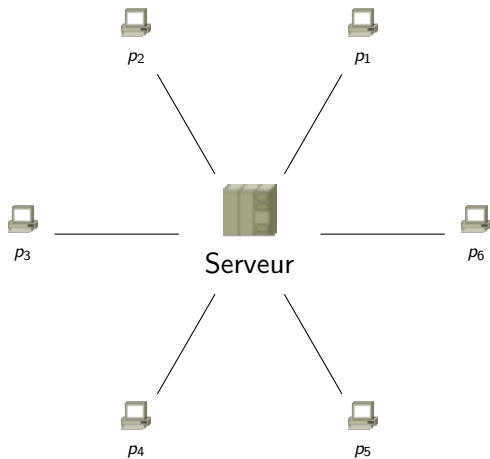
2006 Fin de Razorback 2.0 : émergence des DHTs

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

Napster

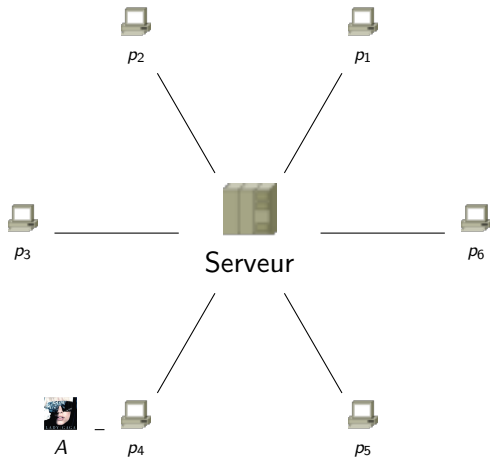
- ▶ Inventé en 1998 by Shawn "Napster" Fanning (The Italian Job) et Sean Parker (appears in The social network)
- ▶ Procès en 1999 (MPAA) - fermeture en juillet 2001
- ▶ Fonctionne en trois étapes :
 1. Indexation : les utilisateurs envoient leur index de MP3 (catalogue) à un serveur, `napster.com`
 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 : téléchargement auprès d'un client de la liste

Napster



1. Indexation

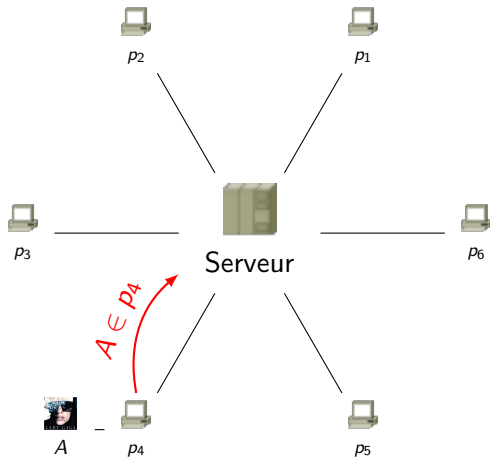
Napster



1. Indexation

- p_4 possède A

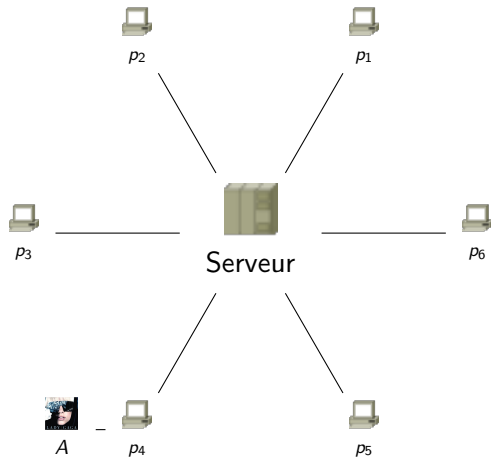
Napster



1. Indexation

- ▶ p_4 possède A
- ▶ p_4 informe s

Napster

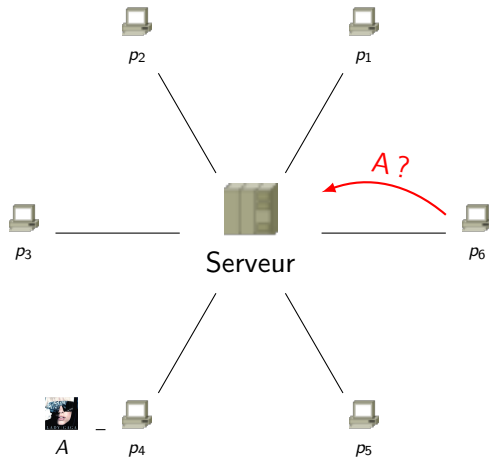


1. Indexation

- ▶ p_4 possède A
- ▶ p_4 informe s

2. Requête

Napster



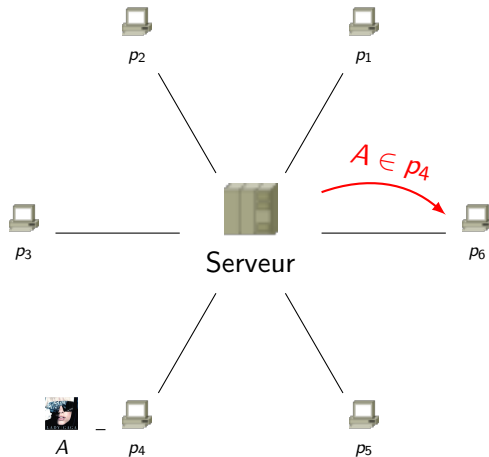
1. Indexation

- ▶ p_4 possède A
- ▶ p_4 informe s

2. Requête

- ▶ p_6 demande à s qui a A

Napster



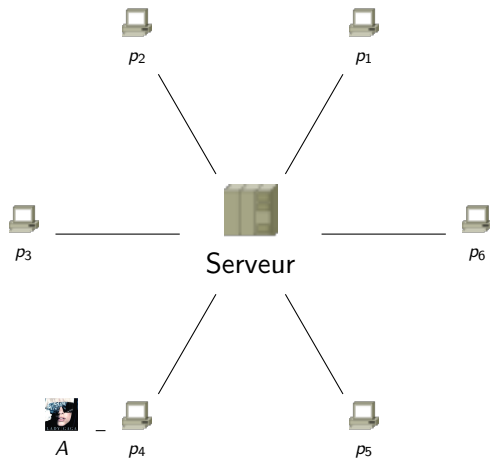
1. Indexation

- ▶ 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

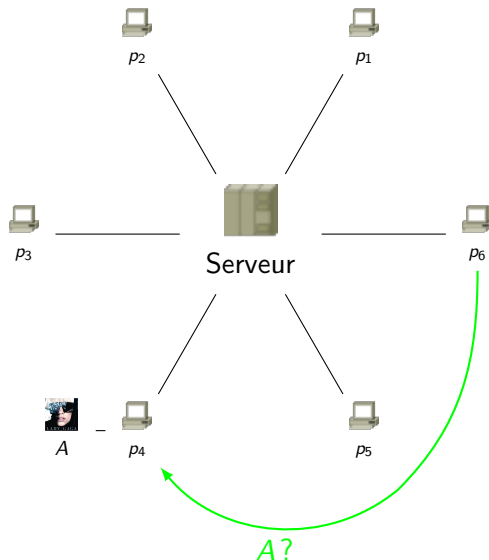
- ▶ p_4 possède A
- ▶ p_4 informe s

2. Requête

- ▶ p_6 demande à s qui a A
- ▶ s répond p_6 (IP)

3. Distribution

Napster



1. Indexation

- ▶ p_4 possède A
- ▶ p_4 informe s

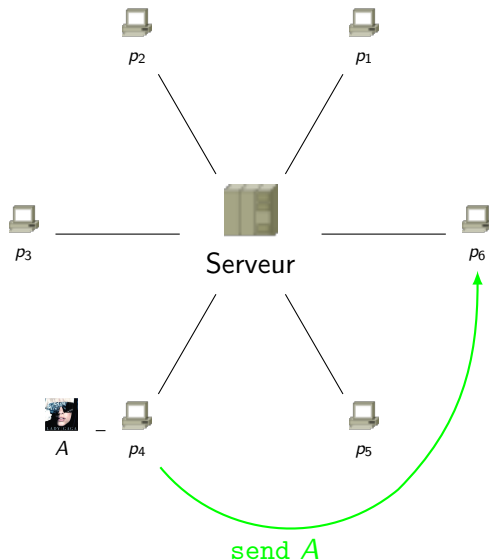
2. Requête

- ▶ 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

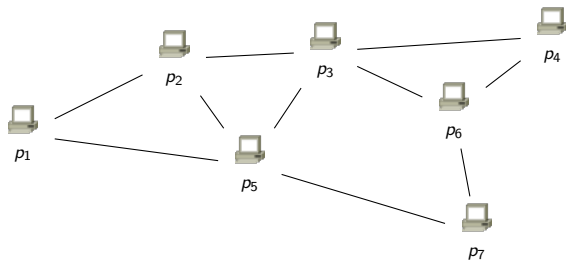
Histoire

- ▶ 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)

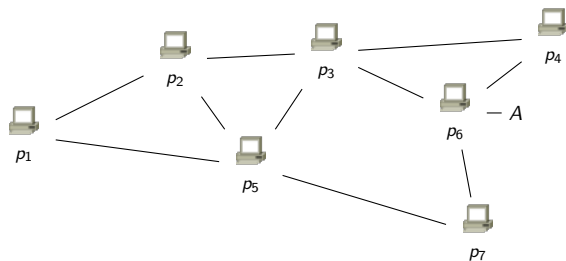
Idée du protocole initial (0.4)

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

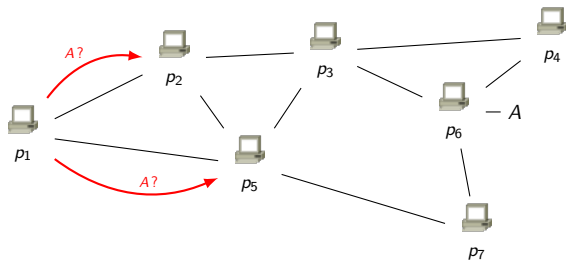
Gnutella



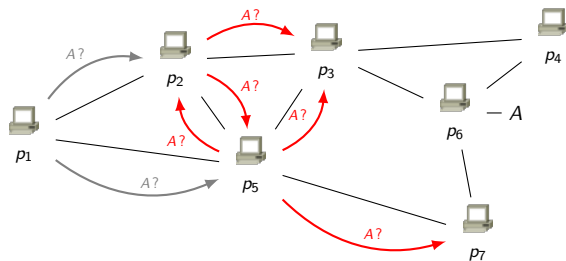
Gnutella



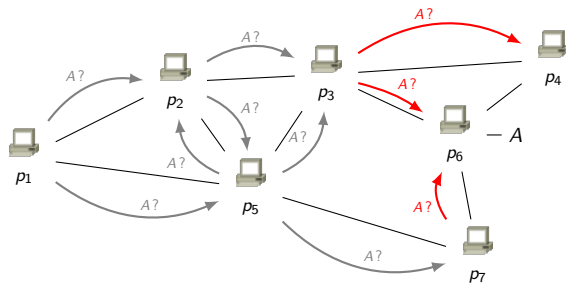
Gnutella



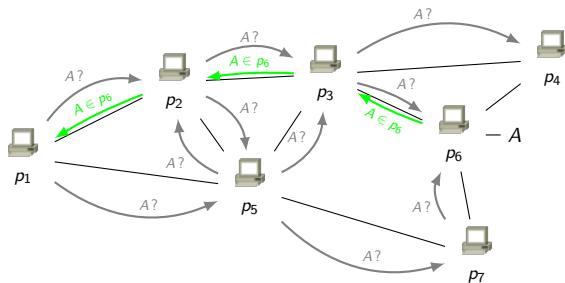
Gnutella



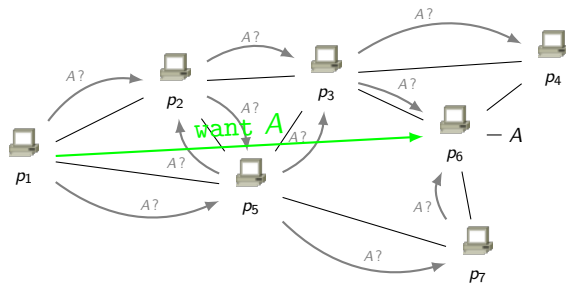
Gnutella



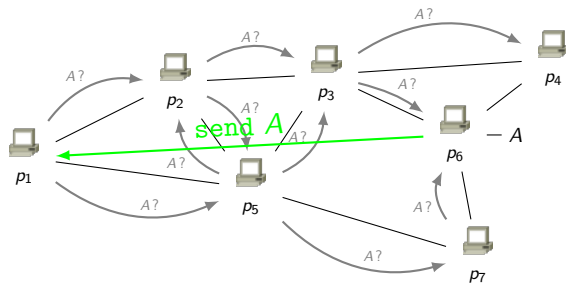
Gnutella



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

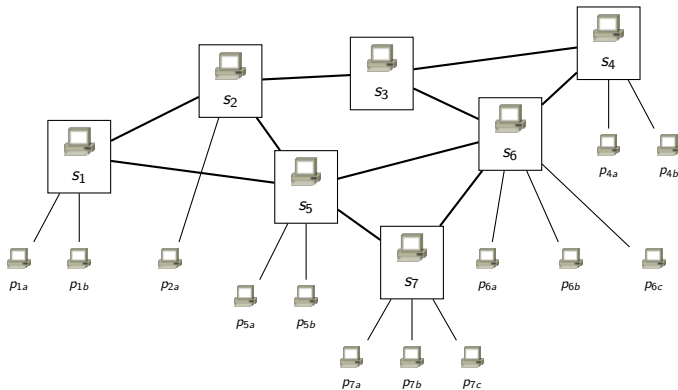
Histoire

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

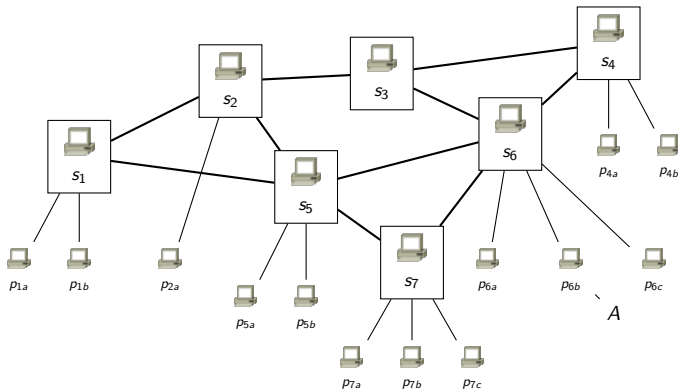
Principale idée : **indexation hiérarchique**

- ▶ 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

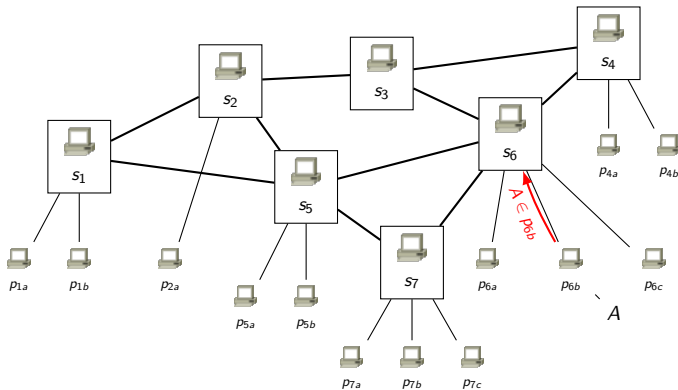
KaZaA



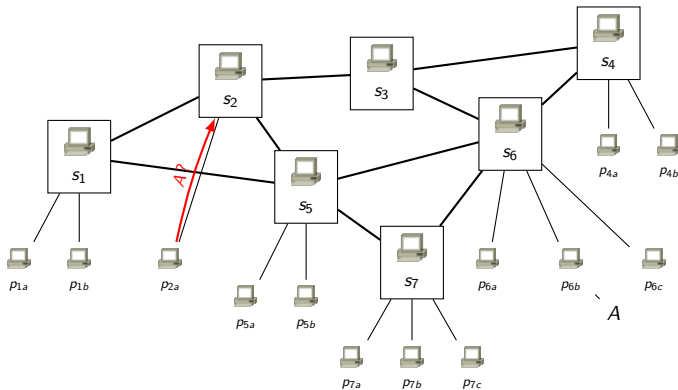
KaZaA



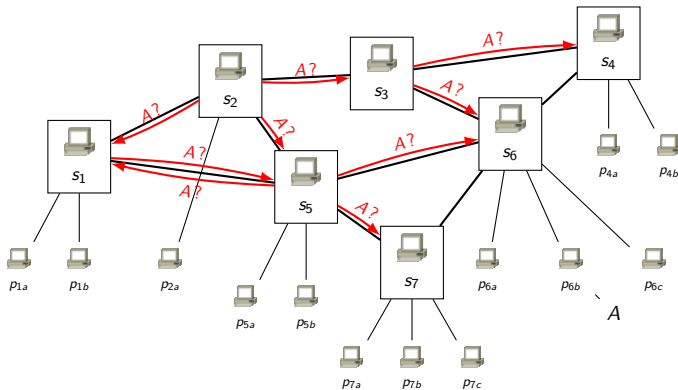
KaZaA



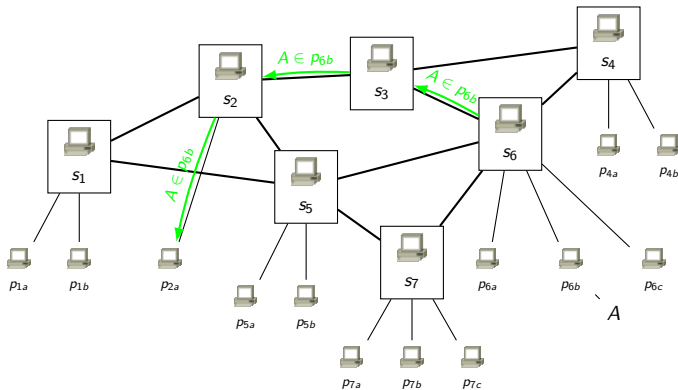
KaZaA



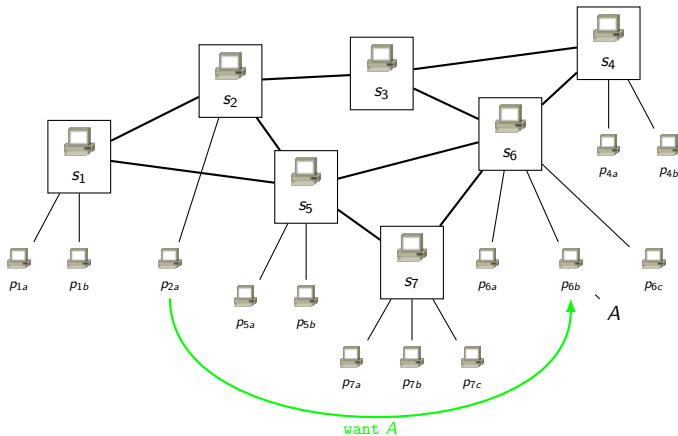
KaZaA



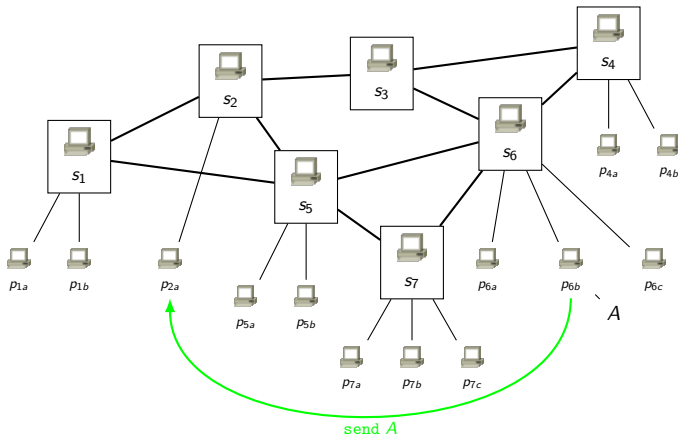
KaZaA



KaZaA



KaZaA



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, μ torrent. . .)
- ▶ Plus gros trafic (de loin) pendant plusieurs années. Toujours le plus grand trafic P2P

Idée principale : se concentrer sur le contenu et sa distribution

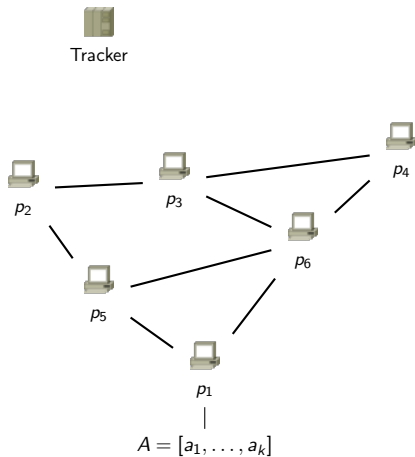
- ▶ À chaque contenu est associé un overlay : l'essaim (swarm)
- ▶ Un essaim est indexé par un tracker
- ▶ Politique d'incitation au partage : donnant-donnant (Tit-for-Tat)
- ▶ + 2/3 autres idées simples et efficaces

BitTorrent

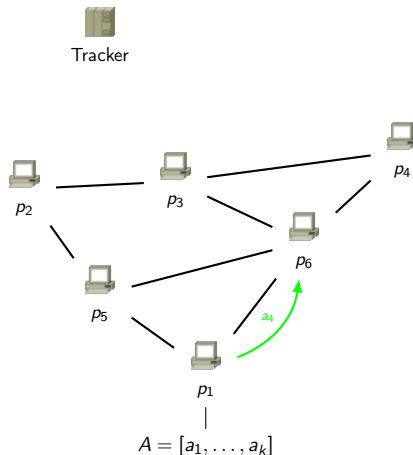
Une architecture minimale

- ▶ 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

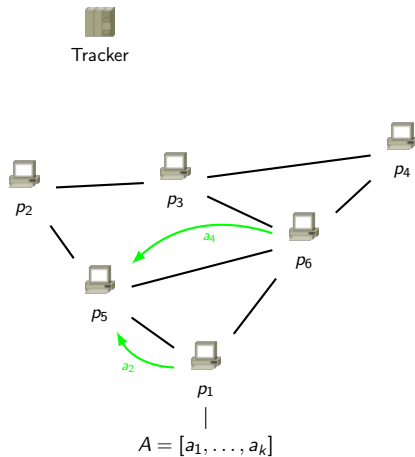
BitTorrent



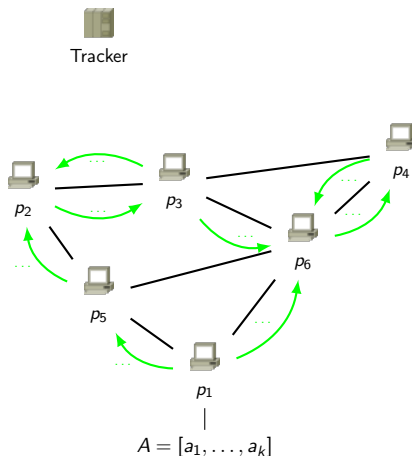
BitTorrent



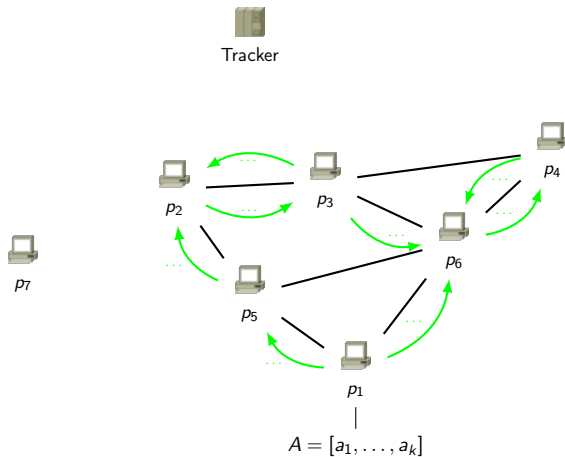
BitTorrent



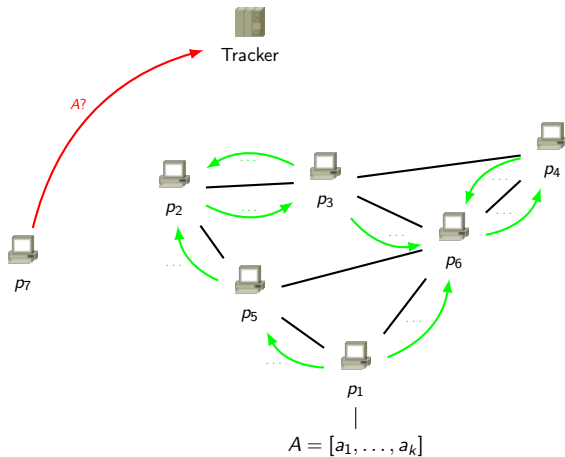
BitTorrent



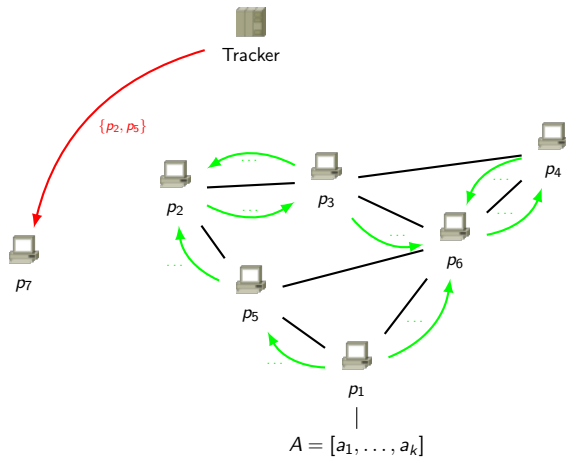
BitTorrent



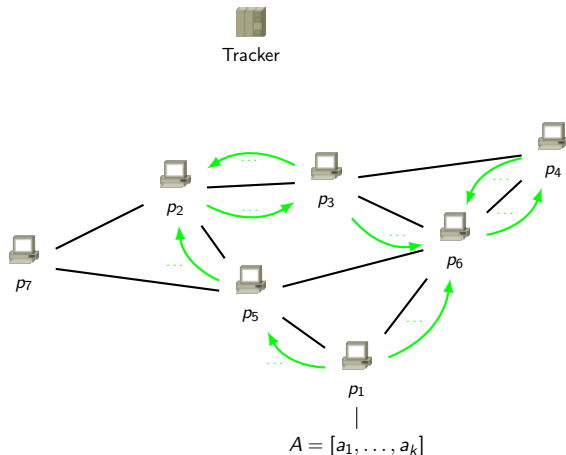
BitTorrent



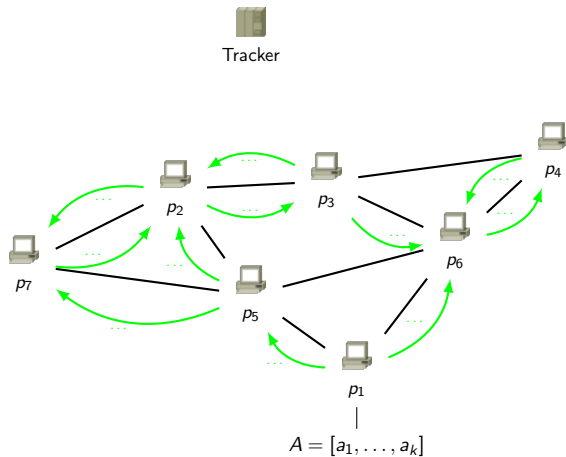
BitTorrent



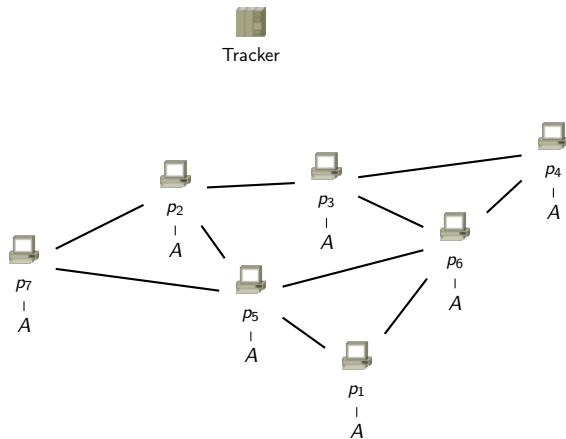
BitTorrent



BitTorrent



BitTorrent



BitTorrent

Innovation

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

Limites

- ▶ Trackers → vulnérabilité
- ▶ Durée de vie des essais 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 : événements 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

Principe

CAN

Chord

Viceroy

Topologie : hypercube

Zoom sur Kademlia

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 variantes 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 !

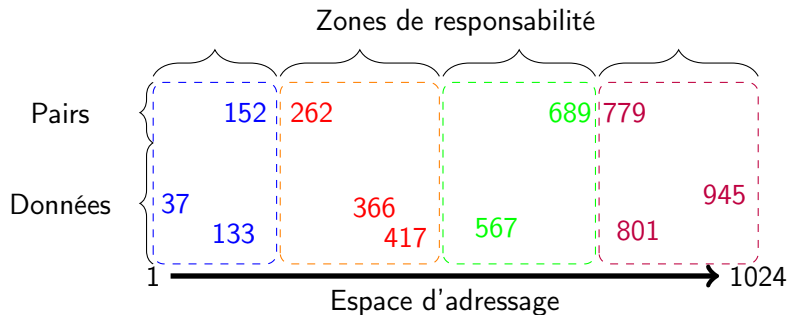
Une chose à retenir

Gérer une clé ne veut pas dire posséder le contenu !

La bonne réponse est savoir :

- ▶ quels contenus sont associés à 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. $h(\text{“Game”}) = 1234$
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. $h(\text{“Thrones”}) = 5678$
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 indexée par le pair Q
- ▶ P connaît 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 à 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

Principe

CAN

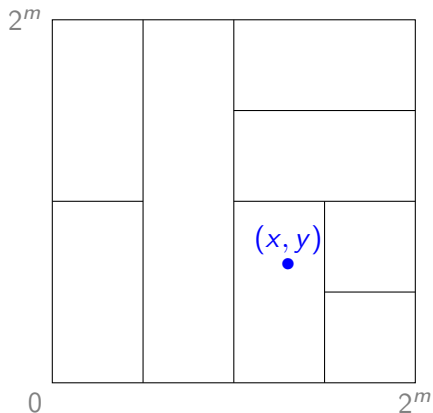
Chord

Viceroy

Topologie : hypercube

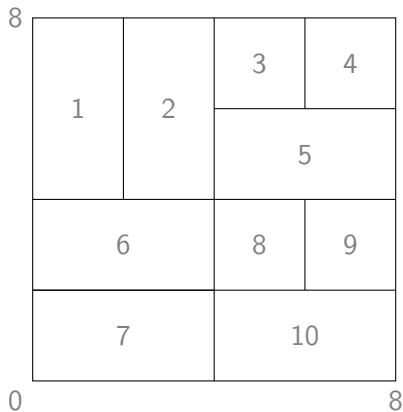
Zoom sur Kademlia

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 hashes
- ▶ Routage vers le voisin de la zone la plus proche pour la distance de Manhattan

Exemple



Exemple

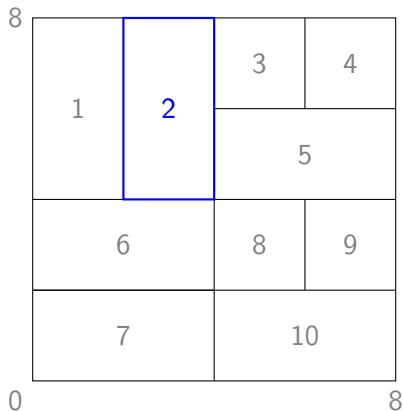


Table de routage de 2

| 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

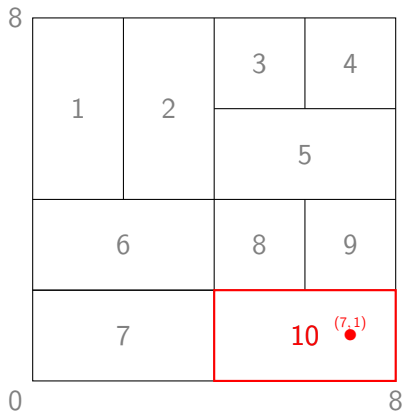


Table de routage de 2

| 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

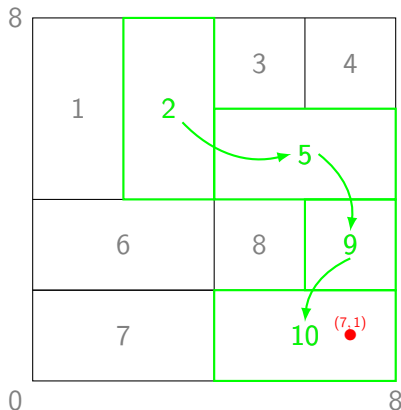


Table de routage de 2

| 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) : $2 \rightarrow 5 \rightarrow 9 \rightarrow 10$.

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

Principe

CAN

Chord

Viceroy

Topologie : hypercube

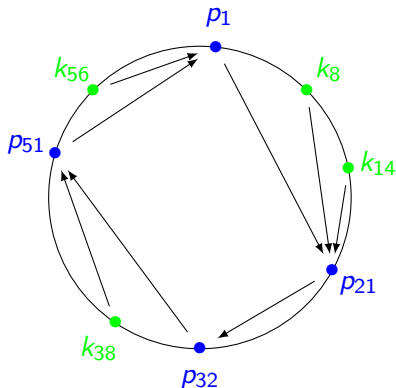
Zoom sur Kademlia

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

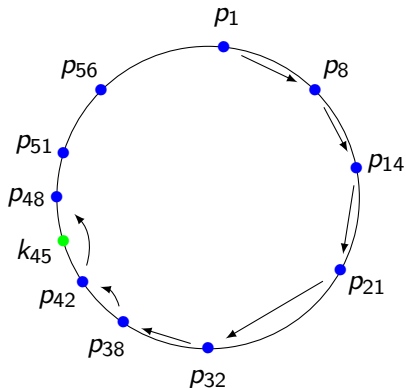


Un théorème

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
- ▶ $C = O(\ln(N))$

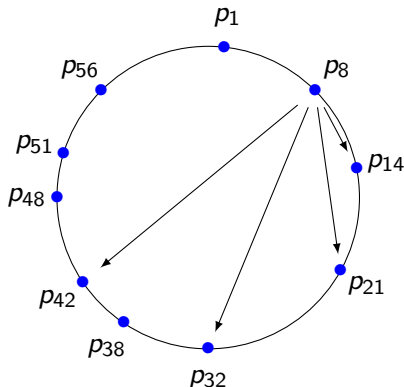
Routage par successeur



► Exemple : 1 cherche
45 (géré par 48)

► ☹️☹️☹️

Solution : finger table

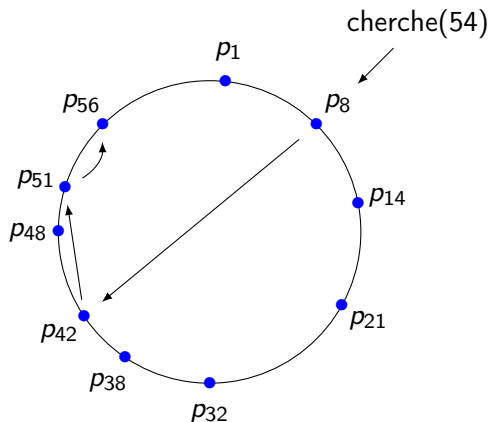


- $\text{finger}(k)$: successeur de $n + 2^{k-1}$ modulo 2^m

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

Principe

CAN

Chord

Viceroy

Topologie : hypercube

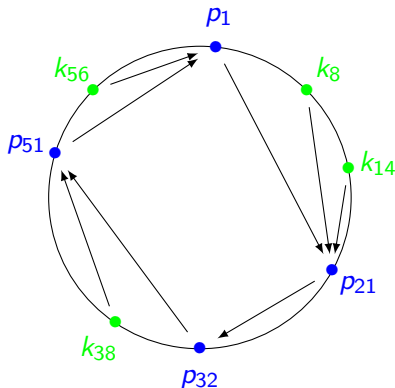
Zoom sur Kademlia

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 : successeur(k) et prédécesseur(k)

Successeurs

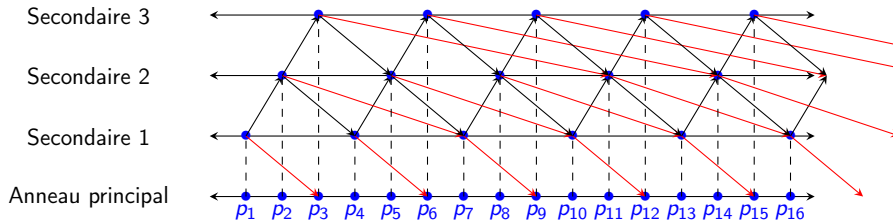
Un pair gère l'ensemble des clés dont il est le successeur



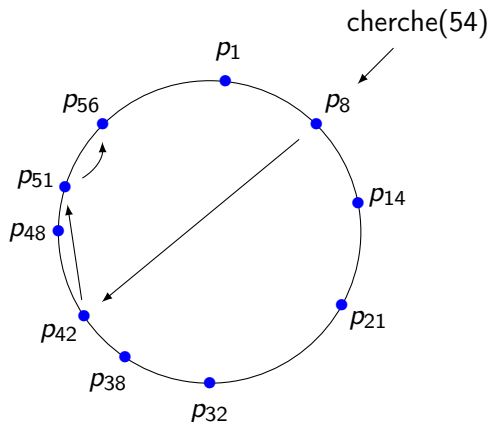
Solution Viceroy : graphe papillon

- ▶ 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 $\text{succ}(x + 2^p)$ dans l'anneau du dessous.

Graphe «papillon»



Rappel : routage Chord

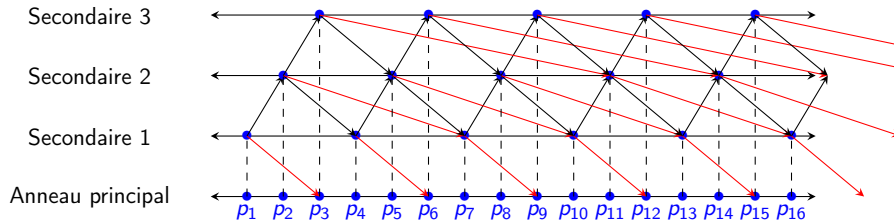


Routage papillon

Idée : imiter Chord

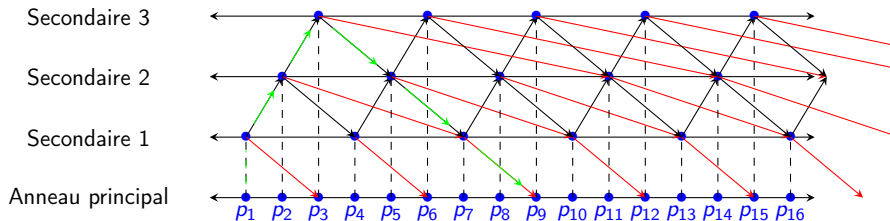
- ▶ Toujours : si on risque de dépasser, on route sur l'anneau principal
- ▶ On commence par monter au niveau le plus puissant ;
- ▶ Puis on descend les niveaux un par un, en utilisant
 - ▶ Le lien long si possible ;
 - ▶ L'ascenseur normal sinon.
- ▶ Coût total : quelques $\log(n)$.

Routage papillon



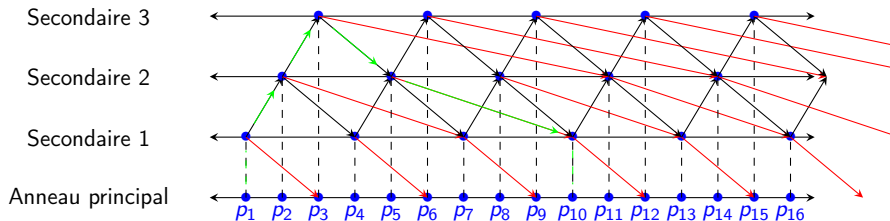
Routage papillon

$p_1 \rightarrow p_9$



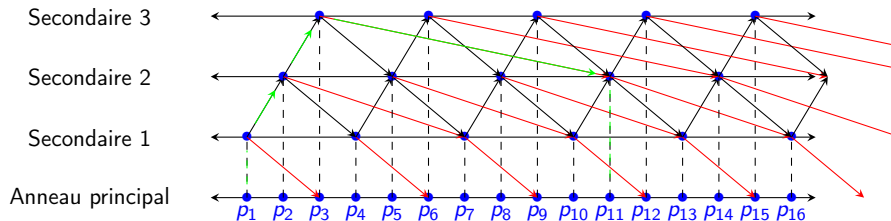
Routage papillon

$p_1 \rightarrow p_{10}$



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

Principe

CAN

Chord

Viceroy

Topologie : hypercube

Zoom sur Kademlia

Topologie : l'hypercube

Idée présente dans Tapestry, Pastry et Kademia

Idée de base

- ▶ Chaque pair P a un identifiant (hashcode) $H(P) = h_1 h_2 \dots h_m$ (typiquement $m = 160$: SHA-1)
- ▶ Chaque pair P essaie de connaître un voisin par préfixe :
 - ▶ $\overline{h_1}$
 - ▶ $h_1 \overline{h_2}$
 - ▶ $h_1 h_2 \overline{h_3}$
 - ▶ ...
 - ▶ $h_1 h_2 \dots h_{v-1} \overline{h_v}$
- ▶ 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

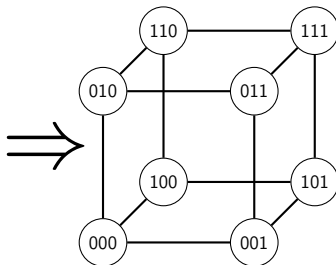
Exemple

- ▶ Le pair d'indentifiant 0100010010001101011010010101...01 a comme voisins :
 - ▶ 1100100100111011010101010...10
 - ▶ 0000100111101001101010011...11
 - ▶ 0111010001010100111100100...11
 - ▶ ...
 - ▶ 0100010101001001001000101000...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'hypercube de dimension v
 v a la même valeur pour tout le monde

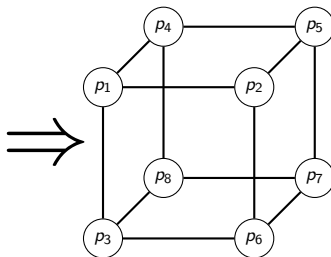
- ▶ $p_1 = 010101001001$
- ▶ $p_2 = 011011101010$
- ▶ $p_3 = 000000110101$
- ▶ $p_4 = 110110110010$
- ▶ $p_5 = 111001111000$
- ▶ $p_6 = 001010101001$
- ▶ $p_7 = 101110010100$
- ▶ $p_8 = 100011001110$



Topologie : l'hypercube

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

- ▶ $p_1 = 010101001001$
- ▶ $p_2 = 011011101010$
- ▶ $p_3 = 000000110101$
- ▶ $p_4 = 110110110010$
- ▶ $p_5 = 111001111000$
- ▶ $p_6 = 001010101001$
- ▶ $p_7 = 101110010100$
- ▶ $p_8 = 100011001110$



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

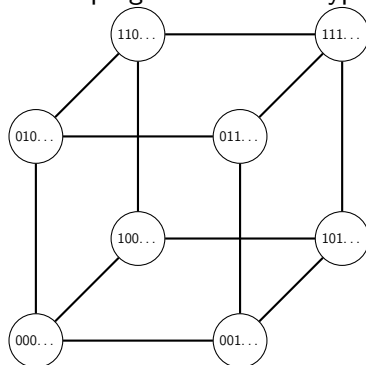
- ▶ Intuition : on «compose» l'adresse de destination
- ▶ On fait une recherche incrémentale, chiffre par chiffre
- ▶ À chaque coup, on se rapproche de la destination

Algorithme de recherche

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

$H(P) \text{ XOR } H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.

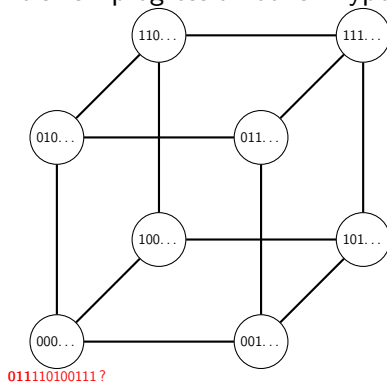


Algorithme de recherche

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

$H(P) \text{ XOR } H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.

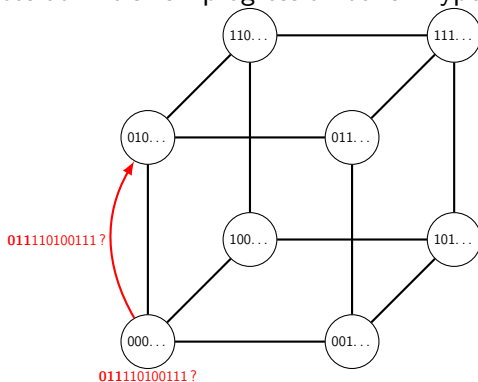


Algorithme de recherche

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

$H(P) \text{ XOR } H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.

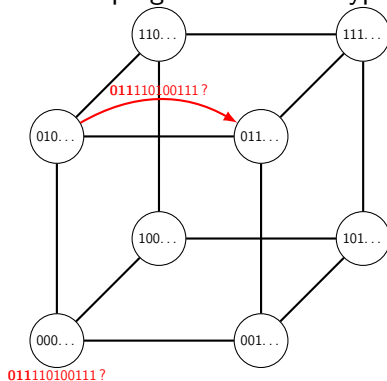


Algorithme de recherche

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

$H(P) \text{ XOR } H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.

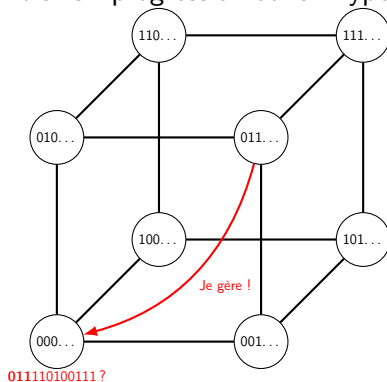


Algorithme de recherche

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

$H(P) \text{ XOR } H(D)$ dit où commencer la requête

Ensuite requêtes aux voisins : progression dans l'hypercube.



Algorithme de recherche

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

$H(P) \text{ XOR } H(D)$ 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

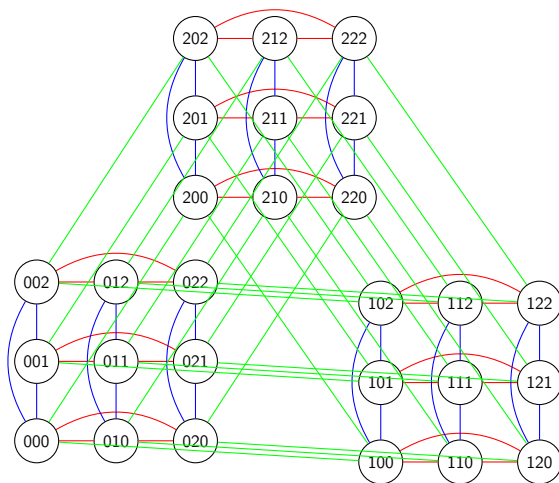
Base plus grande

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)

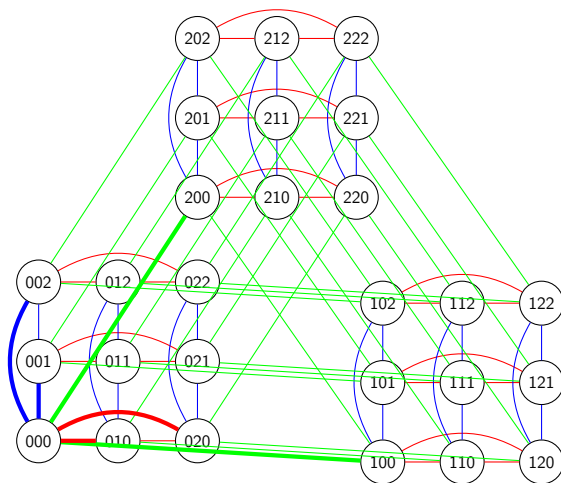
Améliorations de la vitesse de recherche

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×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 :

$$\frac{\ln 2^{21}}{\ln 2^3} * (2^3 - 1) * 20 = 7 * 7 * 20 = 980 \text{ voisins. Faisable.}$$

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 récupérer 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

Principe

CAN

Chord

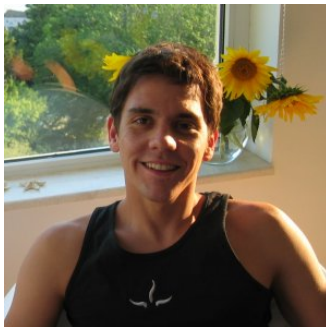
Viceroy

Topologie : hypercube

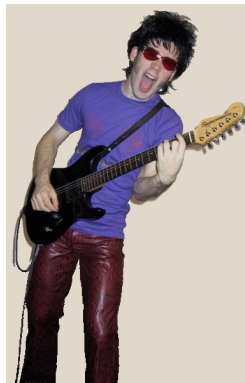
Zoom sur Kademlia

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 $X + Y \geq X \oplus Y$)
- ▶ **unidirectionnelle** : 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

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)

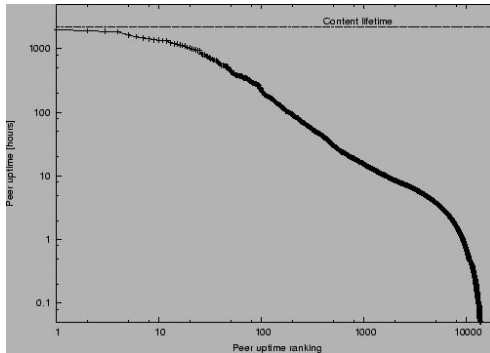
Maintenance du tiroir : LRU amélioré

- ▶ Un tiroir contient au plus k adresses de pairs
- ▶ Remplissage du tiroir (connaître de nouveaux pairs)
 - ▶ réponses aux requêtes
 - ▶ requêtes transmises
- ▶ 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
- ▶ Kademia [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 → pas de requêtes complexes !
- ▶ Plus lent qu'un serveur
- ▶ Protocoles plus complexes et bugs plus durs à corriger

Partie III

Loi de la bande passante

Sommaire

Introduction

- Motivation

- Modèle

Conservation de la bande passante

- Formule générale

- Exemple du streaming

- Théorème de surprovisionnement

Zoom sur BitTorrent

- Flashcrowd, régime stationnaire, mort

- Coopérer ou pas : théorie des jeux

P2P = Mauvaises performances ?

- ▶ Le P2P est considéré comme low cost

P2P = Mauvaises performances ?

- ▶ Le P2P est considéré comme low cost
- ▶ Il faut donc toujours préférer une solution centralisée ?

| Paquets / Réseaux | Nœud / Goutte | Statut | Progression |
|--|---------------|---------------------------|---------------------|
| 📁 📄 House.S08E01.VOSTFR.HDTV... [megaupload.com] | [1/1] | ETA 06m:56s @ 590.00 KB/s | 108.09 MB/347.96 MB |

P2P = Mauvaises performances ?

- ▶ Le P2P est considéré comme low cost
- ▶ Il faut donc toujours préférer une solution centralisée ?

| Requêtes / Réponses | Nouvel upload | Status | Progression |
|--|---------------|---------------------------------|---------------------|
| House.S08E01.VOSTFR.HDTV... [megaupload.com] | | [1/1] ETA 06m:56s @ 590.00 KB/s | 108.09 MB/347.96 MB |

- ▶ Vraiment ?

| | | | | | | | | |
|---|---------|---------|----------|----------|--------|----------|----------|--------|
| History Great Battles Medieval v1.1.zip | 2.00 MB | 100.00% | Seeding | 0 (0) | 0 (0) | | | |
| House.S08E01.FASTSUB.VOSTFR.HDTV-PNG... | 350 MB | 90.2% | Downl... | 69 (958) | 7 (71) | 1.4 MB/s | 3.5 k... | 3m 16s |
| Modern Combat 2 Black Dragon HD v3.0.4.0... | 3.81 MB | 100.00% | Seeding | 0 (0) | 0 (0) | | | |

P2P = Mauvaises performances ?

- ▶ Le P2P est considéré comme low cost
- ▶ Il faut donc toujours préférer une solution centralisée ?

| Requêtes / Fichiers | Nouvel upload | Status | Progression |
|--|---------------|---------------------------------|---------------------|
| House.S08E01.VOSTFR.HDTV... [megaupload.com] | | [1/1] ETA 06m:56s @ 590.00 KB/s | 108.09 MB/347.96 MB |

- ▶ Vraiment ?

| | | | | | | | | |
|---|---------|--------|----------|----------|--------|----------|----------|--------|
| History Great Battles Medieval v1.1.zip | 2.00 MB | 100.0% | Seeding | 0 (0) | 0 (0) | | | |
| House.S08E01.FASTSUB.VOSTFR.HDTV-PNG... | 350 MB | 90.2% | Downl... | 69 (958) | 7 (71) | 1.4 MB/s | 3.5 k... | 3m 16s |
| Modern Combat 2 Black Dragon HD v3.0.4.0... | 3.81 MB | 100.0% | Seeding | 0 (0) | 0 (0) | | | |

- ▶ Peut-on comprendre d'où vient la performance ?

P2P = Mauvaises performances ?

- ▶ Le P2P est considéré comme low cost
- ▶ Il faut donc toujours préférer une solution centralisée ?

| Requêtes / Réponses | Heure de début | Statut | Progression |
|--|---------------------------------|--------|---------------------|
| House.S08E01.VOSTFR.HDTV... [megaupload.com] | [1/1] ETA 06m:56s @ 590.00 KB/s | | 108.09 MB/347.96 MB |

- ▶ Vraiment ?

| | | | | | | | | |
|--|---------|--------|----------|----------|--------|----------|----------|--------|
| History Great Battles Medieval v1.1.zip | 2.00 MB | 100.0% | Seeding | 0 (46) | 0 (0) | | | |
| House.S08E01.FASTSUB.VOSTFR.HDTV-PNG... | 350 MB | 90.2% | Downl... | 69 (958) | 7 (71) | 1.4 MB/s | 3.5 k... | 3m 16s |
| Modern Combat 2 Black Dragon HD v3.0.4.zip | 3.81 MB | 100.0% | Seeding | 0 (0) | 0 (0) | | | |

- ▶ Peut-on comprendre d'où vient la performance ?
- ▶ Étude sur une distribution orientée contenu (BitTorrent)

Modèle complet (hybride)

On considère souvent trois types de «pairs» :

Serveurs centraux (C) Composante non-P2P du système

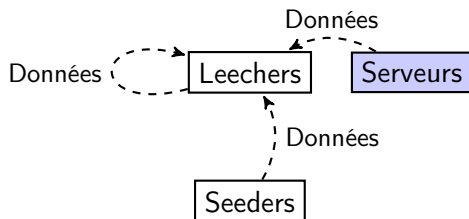
Leechers (L) Veulent du contenu ; peuvent en fournir à d'autres leechers

Seeders (S) Possèdent du contenu et peuvent le fournir aux leechers

Deux variantes du modèle

- ▶ Fermé
- ▶ Ouvert

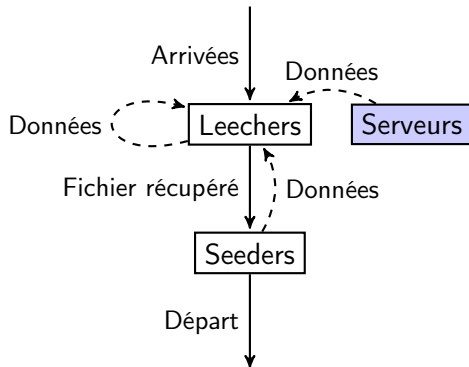
Modèle fermé



Les leechers reçoivent du contenu de tout le monde

- ▶ Autres leechers
- ▶ Seeders
- ▶ Serveurs

Modèle ouvert



Cycle de vie :

- ▶ Un pair arrive en temps que leecher
- ▶ Une fois le téléchargement terminé, il devient seeder
- ▶ Au bout d'un certain temps, il part
- ▶ Les serveurs sont statiques

Notation

- ▶ Serveurs/leechers/seeders : $C, s \in S, l \in L$
- ▶ Nombre de pairs : N_S, N_L
- ▶ Upload : U_S, U_C, u_l, \bar{u}_S
- ▶ Download : d_l, d_{\max}

Sommaire

Introduction

Motivation

Modèle

Conservation de la bande passante

Formule générale

Exemple du streaming

Théorème de surprovisionnement

Zoom sur BitTorrent

Flashcrowd, régime stationnaire, mort

Coopérer ou pas : théorie des jeux

Conservation de la bande passante

- ▶ Rien ne se perd, rien ne se crée, tout se télécharge

$$\sum_{I \in L} d_I \leq U_L + U_S + U_C$$

- ▶ Par exemple, si download infini,

$$\sum_{I \in L} d_I = \eta_L U_L + \eta_S U_S + \eta_C U_C$$

- ▶ η : coefficient d'efficacité
- ▶ Systèmes idéaux : $\eta = 1$

Conservation de la bande passante

- ▶ Rien ne se perd, rien ne se crée, tout se télécharge

$$\sum_{I \in L} d_I \leq U_L + U_S + U_C$$

- ▶ Par exemple, si download infini,

$$\sum_{I \in L} d_I = \eta_L U_L + \eta_S U_S + \eta_C U_C$$

- ▶ η : coefficient d'efficacité
- ▶ **Systemes idéaux** : $\eta = 1$

Conservation de la bande passante

- ▶ Si partage uniforme de la bande passante

$$d = \min \left(d_{\max}, \bar{u}_L + \beta \bar{u}_S + \frac{U_C}{N_L} \right), \text{ avec } \beta = \frac{N_S}{N_L}$$

- ▶ Marche sur systèmes ouverts et fermés
- ▶ Marche en trafic contraint (streaming) et en élastique (filesharing)
- ▶ Beaucoup d'améliorations possibles (η , d_{\max} , partage non-uniforme. . .)

Exemple du streaming (système fermé)

- ▶ Pour un débit cible r , on pose $\alpha_X = \frac{\bar{u}_X}{r}$ et $N_C = \frac{U_C}{r}$
- ▶ Pour diffuser, il faut $d \geq r$, sinon...
- ▶ $\alpha_L + \beta\alpha_S$ est la capacité normalisée du système P2P
- ▶ Cas 1 : $\alpha_L + \beta\alpha_S \geq 1$
 - ▶ Le système passe à l'échelle (indépendamment de N_C)
 - ▶ Les serveurs ne sont pas nécessaires pour la bande passante (mais bootstrap/delai/securité)
- ▶ Cas 2 : $\alpha_L + \beta\alpha_S < 1$
 - ▶ N_L ne peut dépasser $\frac{N_C}{1-(\alpha_L+\beta\alpha_S)}$
 - ▶ → Effet de levier : la capacité serveur est multipliée par $\frac{1}{1-(\alpha_L+\beta\alpha_S)}$

Exemple du streaming (système fermé)

Preuve

- ▶ On part de la condition à vérifier :

$$U_C + U_S + U_L \geq N_L r$$

$$U_C + N_L \bar{u}_L + N_S \bar{u}_S \geq N_L r$$

$$U_C + N_L r (\alpha_L + \beta \alpha_S) \geq N_L r$$

$$\frac{N_C}{N_L} + (\alpha_L + \beta \alpha_S) \geq 1$$

- ▶ $\alpha_L + \beta \alpha_S \geq 1 \rightarrow$ toujours vérifié
- ▶ $\alpha_L + \beta \alpha_S < 1 \rightarrow N_L \leq \frac{N_C}{1 - (\alpha_L + \beta \alpha_S)}$

Interlude : Joost

Histoire d'un fiasco monumental

- ▶ Après KaZaA (2000), Niklas Zennström et Janus Friis veulent révolutionner le monde (du P2P).
- ▶ Le monde n'est pas prêt, donc ils font un petit projet en attendant : Skype
- ▶ 2005 : vente de Skype à eBay. Retour au grand projet (avec des sous).
- ▶ 2007 : Lancement de Joost, le programme P2P ultime (fait aussi le café)
- ▶ 2008 : abandon de la partie P2P de Joost ; fonctionnement sur navigateur (type Youtube)...
- ▶ 2009 : Clé sous la porte ; fin de l'histoire

Interlude : Joost

- ▶ Joost n'était pas ouvert
- ▶ En mesurant le rapport upload/download ratio, $\alpha = \frac{1}{6}$
- ▶ Si $\beta = 0$, l'effet de levier P2P est de 20%
- ▶ Si $\beta = 1$, on passe à 50% (most likely behavior)
- ▶ Si $\beta = 2$, 100%
- ▶ ...
- ▶ Le passage à l'échelle nécessite $\beta \geq 5$ (même en heure de pointe)
- ▶ Conclusion : Joost n'utilisait le P2P que comme levier

Modèle de système ouvert

- ▶ On fixe un fichier de taille k comme objectif
- ▶ Les pairs arrivent à un taux λ (arrivées par unité de temps)
- ▶ Un pair l télécharge en $T_L(l)$, puis il seede
- ▶ Après $T_S(s)$, le seeder s quitte le système
- ▶ Système ouvert, donc dynamique. . .
- ▶ . . . on ne regarde que le régime stationnaire

Régime stationnaire

- ▶ Les pairs arrivent différents uploads u .
- ▶ $p(u)$: probabilité qu'un pair qui arrive ait upload u .
- ▶ Calcul de la moyenne, 2 écritures :

- ▶ Version discrète :

$$\bar{u} = \sum_k u_k p(u_k),$$

où les u_k sont les bandes passantes possibles.

- ▶ En pratique, toutes les bandes passantes sont possibles, on utilise donc plutôt une notation continue :

$$\bar{u} = \int_{\mathbb{R}^+} u p(u) du$$

- ▶ C'est la vitesse moyenne des leechers et des seeders SI le comportement des pairs ne dépend pas de u .

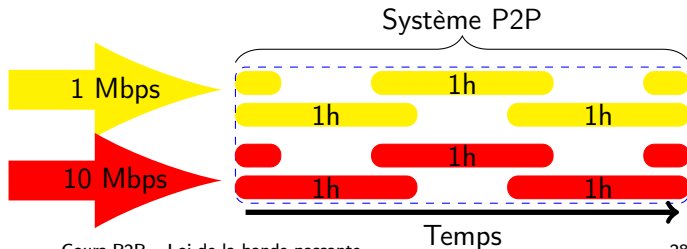
Distortion de probabilité

La vitesse moyenne est un concept subtil. Exemple sur les arrivées 50% de pairs avec $u_1 = 1Mbps$, 50% avec $u_2 = 10Mbps$

Distorsion de probabilité

La vitesse moyenne est un concept subtil. Exemple sur les arrivées 50% de pairs avec $u_1 = 1Mbps$, 50% avec $u_2 = 10Mbps$

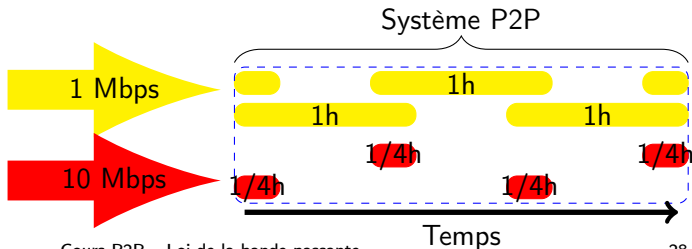
- Scenario 1 : Le temps de séjour est le même pour tout le monde (e.g. 1 heure)
- Alors $\bar{u} = \frac{1+10}{2} = 5,5Mbps$



Distorsion de probabilité

La vitesse moyenne est un concept subtil. Exemple sur les arrivées 50% de pairs avec $u_1 = 1Mbps$, 50% avec $u_2 = 10Mbps$

- Scenario 1 : Le temps de séjour est le même pour tout le monde (e.g. 1 heure)
- Alors $\bar{u} = \frac{1+10}{2} = 5,5Mbps$
- Scenario 2 : les u_1 restent 1 heure, les u_2 1/4 d'heure.
- Alors ?



Distorsion de probabilité

- ▶ Si λ , p_1 , $T(u_1)$, alors il y a $\lambda p_1 T(u_1)$ «pauvres» en moyenne dans le système (loi de Little) ;
- ▶ De même, $\lambda p_2 T(u_2)$ «riches» et donc $\lambda p_1 T(u_1) + \lambda p_2 T(u_2) = \lambda \bar{T}$ en tout ;
- ▶ Proportions dans le système : $\frac{p_1 T(u_1)}{\bar{T}}$ «pauvres» et $\frac{p_2 T(u_2)}{\bar{T}}$ «riches».

Distortion de probabilité

- ▶ Si λ , p_1 , $T(u_1)$, alors il y a $\lambda p_1 T(u_1)$ «pauvres» en moyenne dans le système (loi de Little) ;
- ▶ De même, $\lambda p_2 T(u_2)$ «riches» et donc $\lambda p_1 T(u_1) + \lambda p_2 T(u_2) = \lambda \bar{T}$ en tout ;
- ▶ Proportions dans le système : $\frac{p_1 T(u_1)}{\bar{T}}$ «pauvres» et $\frac{p_2 T(u_2)}{\bar{T}}$ «riches».
- ▶ Et donc, bande passante du système dans le scénario 2 ?

Distortion de probabilité

- ▶ Si λ , p_1 , $T(u_1)$, alors il y a $\lambda p_1 T(u_1)$ «pauvres» en moyenne dans le système (loi de Little) ;
- ▶ De même, $\lambda p_2 T(u_2)$ «riches» et donc $\lambda p_1 T(u_1) + \lambda p_2 T(u_2) = \lambda \bar{T}$ en tout ;
- ▶ Proportions dans le système : $\frac{p_1 T(u_1)}{\bar{T}}$ «pauvres» et $\frac{p_2 T(u_2)}{\bar{T}}$ «riches».
- ▶ Et donc, bande passante du système dans le scénario 2 ?
- ▶ 2,8 Mbps (à comparer avec 5.5 Mbps).

Distortion de probabilité (formules générales)

- ▶ Un pair télécharge pendant $T_L(l) = \frac{k}{d(l)}$. Si d lié à u , on a alors $T_L(u)$.
- ▶ Calcul du temps de téléchargement moyen :

$$\bar{T}_L = \sum T_L(u)p(u)$$

- ▶ Plus un pair télécharge longtemps, plus on le voit. La probabilité d'un leecher d'upload u est donc

$$p_L(u) = \frac{T_L(u)p(u)}{\bar{T}_L}.$$

- ▶ La vitesse moyenne des leechers est alors modifiée :

$$\bar{u}_L = \sum up_L(u).$$

Distortion de probabilité (formules générales)

- ▶ Résultats similaires pour les seeders : si $T_S(u)$...
- ▶ Calcul du temps de téléchargement moyen :

$$\bar{T}_S = \sum T_S(u)p(u)$$

- ▶ Plus un pair seede longtemps, plus on le voit. La probabilité d'un seeder d'upload u est donc

$$p_S(u) = \frac{T_S(u)p(u)}{\bar{T}_S}.$$

- ▶ La vitesse moyenne des seeders est alors modifiée :

$$\bar{u}_S = \sum up_S(u).$$

Théorème de surprovisionnement

- Donne une condition suffisante pour qu'un système optimal atteigne d_{\max} dans la limite fluide du régime stationnaire :

$$\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} > k \left(1 - \frac{\bar{u}}{d_{\max}} \right)$$

Théorème de surprovisionnement

- ▶ Donne une condition suffisante pour qu'un système optimal atteigne d_{\max} dans la limite fluide du régime stationnaire :

$$\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} > k \left(1 - \frac{\bar{u}}{d_{\max}} \right)$$

- ▶ Par exemple, la condition est vérifiée si

Théorème de surprovisionnement

- ▶ Donne une condition suffisante pour qu'un système optimal atteigne d_{\max} dans la limite fluide du régime stationnaire :

$$\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} > k \left(1 - \frac{\bar{u}}{d_{\max}}\right)$$

- ▶ Par exemple, la condition est vérifiée si
 - ▶ $\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} \geq k$: tout débit cible peut être atteint.

Théorème de surprovisionnement

- ▶ Donne une condition suffisante pour qu'un système optimal atteigne d_{\max} dans la limite fluide du régime stationnaire :

$$\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} > k(1 - \frac{\bar{u}}{d_{\max}})$$

- ▶ Par exemple, la condition est vérifiée si
 - ▶ $\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} \geq k$: tout débit cible peut être atteint.
 - ▶ $\bar{T}_S \bar{u}_S \geq k$: tout débit cible peut être atteint, indépendamment de la popularité λ et de U_C . Par exemple

Théorème de surprovisionnement

- ▶ Donne une condition suffisante pour qu'un système optimal atteigne d_{\max} dans la limite fluide du régime stationnaire :

$$\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} > k \left(1 - \frac{\bar{u}}{d_{\max}}\right)$$

- ▶ Par exemple, la condition est vérifiée si
 - ▶ $\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} \geq k$: tout débit cible peut être atteint.
 - ▶ $\bar{T}_S \bar{u}_S \geq k$: tout débit cible peut être atteint, indépendamment de la popularité λ et de U_C . Par exemple
 - ▶ $T_S(u) = \frac{k}{u}$

Théorème de surprovisionnement

- ▶ Donne une condition suffisante pour qu'un système optimal atteigne d_{\max} dans la limite fluide du régime stationnaire :

$$\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} > k \left(1 - \frac{\bar{u}}{d_{\max}}\right)$$

- ▶ Par exemple, la condition est vérifiée si
 - ▶ $\bar{T}_S \bar{u}_S + \frac{U_C}{\lambda} \geq k$: tout débit cible peut être atteint.
 - ▶ $\bar{T}_S \bar{u}_S \geq k$: tout débit cible peut être atteint, indépendamment de la popularité λ et de U_C . Par exemple
 - ▶ $T_S(u) = \frac{k}{u}$
 - ▶ $T_S = \frac{k}{\bar{u}}$

Théorème de surprovisionnement

Preuve :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :
$$U_L + U_S + U_C > \lambda k$$
- ▶ Expression de U_S (Little) :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :
$$U_L + U_S + U_C > \lambda k$$
- ▶ Expression de U_S (Little) :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :
 $U_L + U_S + U_C > \lambda k$
- ▶ Expression de U_S (Little) :

$$U_S = \sum_{s \in S} u_s = N_S \frac{\sum_{s \in S} u_s}{N_S} = N_S \bar{u}_S = \lambda \bar{T}_S \bar{u}_S$$

- ▶ Expression de U_L :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :
 $U_L + U_S + U_C > \lambda k$
- ▶ Expression de U_S (Little) :

$$U_S = \sum_{s \in S} u_s = N_S \frac{\sum_{s \in S} u_s}{N_S} = N_S \bar{u}_S = \lambda \bar{T}_S \bar{u}_S$$

- ▶ Expression de U_L :

Théorème de surprovisionnement

Preuve :

- ▶ Bande passante consommée par le système : λk
- ▶ Condition pour qu'un système parfait sature en download :
 $U_L + U_S + U_C > \lambda k$
- ▶ Expression de U_S (Little) :

$$U_S = \sum_{s \in S} u_s = N_S \frac{\sum_{s \in S} u_s}{N_S} = N_S \bar{u}_S = \lambda \bar{T}_S \bar{u}_S$$

- ▶ Expression de U_L :

$$U_L = \lambda \bar{T}_L \bar{u}_L$$

Théorème de surprovisionnement

Expression de $U_L = \lambda \bar{T}_L \bar{u}_L$ (suite) :

- Rappel : $\bar{u}_L = \sum u_i p_L(u_i)$, avec

$$p_L(u) = \frac{T_L(u)p(u)}{\sum T_L(u_i)p(u_i)} = \frac{T_L(u)}{\bar{T}_L} p(u)$$

Théorème de surprovisionnement

Expression de $U_L = \lambda \bar{T}_L \bar{u}_L$ (suite) :

- Rappel : $\bar{u}_L = \sum u_i p_L(u_i)$, avec

$$p_L(u) = \frac{T_L(u)p(u)}{\sum T_L(u_i)p(u_i)} = \frac{T_L(u)}{\bar{T}_L} p(u)$$

- On déduit $\bar{T}_L \bar{u}_L = \bar{T}_L \sum u_i \frac{T_L(u_i)}{\bar{T}_L} p(u_i) = \sum u_i T_L(u_i) p(u_i)$

Théorème de surprovisionnement

Expression de $U_L = \lambda \bar{T}_L \bar{u}_L$ (suite) :

- Rappel : $\bar{u}_L = \sum u_i p_L(u_i)$, avec

$$p_L(u) = \frac{T_L(u)p(u)}{\sum T_L(u_i)p(u_i)} = \frac{T_L(u)}{\bar{T}_L} p(u)$$

- On déduit $\bar{T}_L \bar{u}_L = \bar{T}_L \sum u_i \frac{T_L(u_i)}{\bar{T}_L} p(u_i) = \sum u_i T_L(u_i) p(u_i)$
- Le download étant fini, on a

Théorème de surprovisionnement

Expression de $U_L = \lambda \bar{T}_L \bar{u}_L$ (suite) :

- Rappel : $\bar{u}_L = \sum u_i p_L(u_i)$, avec

$$p_L(u) = \frac{T_L(u)p(u)}{\sum T_L(u_i)p(u_i)} = \frac{T_L(u)}{\bar{T}_L} p(u)$$

- On déduit $\bar{T}_L \bar{u}_L = \bar{T}_L \sum u_i \frac{T_L(u_i)}{\bar{T}_L} p(u_i) = \sum u_i T_L(u_i) p(u_i)$
- Le download étant fini, on a

Théorème de surprovisionnement

Expression de $U_L = \lambda \bar{T}_L \bar{u}_L$ (suite) :

- Rappel : $\bar{u}_L = \sum u_i p_L(u_i)$, avec

$$p_L(u) = \frac{T_L(u)p(u)}{\sum T_L(u_i)p(u_i)} = \frac{T_L(u)}{\bar{T}_L} p(u)$$

- On déduit $\bar{T}_L \bar{u}_L = \bar{T}_L \sum u_i \frac{T_L(u_i)}{\bar{T}_L} p(u_i) = \sum u_i T_L(u_i) p(u_i)$
- Le download étant fini, on a $T_L(u_i) \geq \frac{k}{d_{\max}}$. En injectant, il vient

Théorème de surprovisionnement

Expression de $U_L = \lambda \bar{T}_L \bar{u}_L$ (suite) :

- Rappel : $\bar{u}_L = \sum u_i p_L(u_i)$, avec

$$p_L(u) = \frac{T_L(u)p(u)}{\sum T_L(u_i)p(u_i)} = \frac{T_L(u)}{\bar{T}_L} p(u)$$

- On déduit $\bar{T}_L \bar{u}_L = \bar{T}_L \sum u_i \frac{T_L(u_i)}{\bar{T}_L} p(u_i) = \sum u_i T_L(u_i) p(u_i)$
- Le download étant fini, on a $T_L(u_i) \geq \frac{k}{d_{\max}}$. En injectant, il vient

$$U_L = \lambda \bar{T}_L \bar{u}_L \geq \lambda \sum u_i \frac{k}{d_{\max}} p(u_i) \geq \lambda k \frac{\bar{u}}{d_{\max}}$$

Théorème de surprovisionnement

Fin de la preuve :

► On a $U_L + U_S + U_C \geq \lambda k \frac{\bar{u}}{d_{\max}} + \lambda \bar{T}_S \bar{u}_S + U_C$

Théorème de surprovisionnement

Fin de la preuve :

- ▶ On a $U_L + U_S + U_C \geq \lambda k \frac{\bar{u}}{d_{\max}} + \lambda \bar{T}_S \bar{u}_S + U_C$
- ▶ Donc, si $\lambda k \frac{\bar{u}}{d_{\max}} + \lambda \bar{T}_S \bar{u}_S + U_C > \lambda k$, le download est saturé

Théorème de surprovisionnement

Fin de la preuve :

- ▶ On a $U_L + U_S + U_C \geq \lambda k \frac{\bar{u}}{d_{\max}} + \lambda \bar{T}_S \bar{u}_S + U_C$
- ▶ Donc, si $\lambda k \frac{\bar{u}}{d_{\max}} + \lambda \bar{T}_S \bar{u}_S + U_C > \lambda k$, le download est saturé
- ▶ Il n'y a plus qu'à ré-arranger les termes.

Conclusions sur la bande passante

- ▶ Des règles de dimensionnement valables pour tous types de distribution de contenu
- ▶ C'est du premier ordre (faisabilité / ordre de grandeur)
- ▶ Deuxième ordre : dépend du contexte
 - ▶ Latences réseau
 - ▶ Goureaux cœur, pertes paquets
 - ▶ Comportement des protocoles underlay (TCP/ μ TP, AQM)
 - ▶ Propagation / disponibilité de l'information (chunk)
 - ▶ Overhead
 - ▶ ...

Sommaire

Introduction

Motivation

Modèle

Conservation de la bande passante

Formule générale

Exemple du streaming

Théorème de surprovisionnement

Zoom sur BitTorrent

Flashcrowd, régime stationnaire, mort

Coopérer ou pas : théorie des jeux

Les trois moments clés d'un torrent

Naissance (flashcrowd)

- ▶ Un seul possesseur (seeder initial)
- ▶ Burst d'arrivées

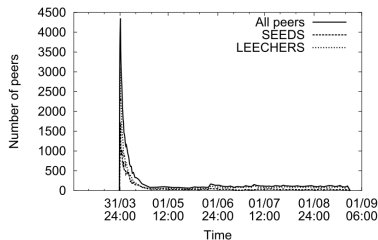
Régime stationnaire

- ▶ Départs et arrivées à peu près équilibrées
- ▶ Correspond aux formules précédentes

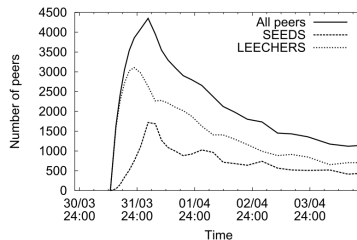
Mort

- ▶ De moins en moins d'arrivées
- ▶ Décrochage de disponibilité

Les trois moments clés d'un torrent



(a) Complete trace



(b) Zoom on the first five days

Source : Dissecting BitTorrent : Five Months in a Torrent's Lifetime

Flashcrowd

Facteur limitant ?

En flashcrowd, une question importante est de savoir quel est le goulot d'étranglement.

- ▶ Si le seeder est suffisamment rapide, le goulot est la bande passante des leechers → bande passante classique.
- ▶ Si seeder trop lent, le goulot est le contenu, donc la bande passante du seeder.
- ▶ La frontière dépend du taux d'arrivées
- ▶ Seeder bottleneck = risques d'instabilité

Flashcrowd : bascule

Beaucoup de choses se passent au moment de la distribution du dernier chunk si le contenu est limitant

- ▶ Juste avant le dernier chunk
 - ▶ Tout le monde a presque tous les chunks sauf un
 - ▶ Écart-type énorme sur la disponibilité des chunks
- ▶ Si tout se passe bien
 - ▶ L'écart de disponibilité se réduit petit à petit
 - ▶ Transition vers un régime stationnaire
- ▶ Si tout se passe mal
 - ▶ Dès que quelqu'un récupère le dernier chunk, il s'en va
 - ▶ Dès que quelqu'un arrive, il récupère très vite tous les chunks sauf un
 - ▶ Syndrome du chunk manquant

Syndrome du chunk manquant

- ▶ Attaque utilisée IRL par les majors pour ralentir des torrents
- ▶ Traîtement assez simple
 - ▶ Pour le seeder : mode furtif (superseed)
 - ▶ Pour les leechers : rarest first + rester seeder

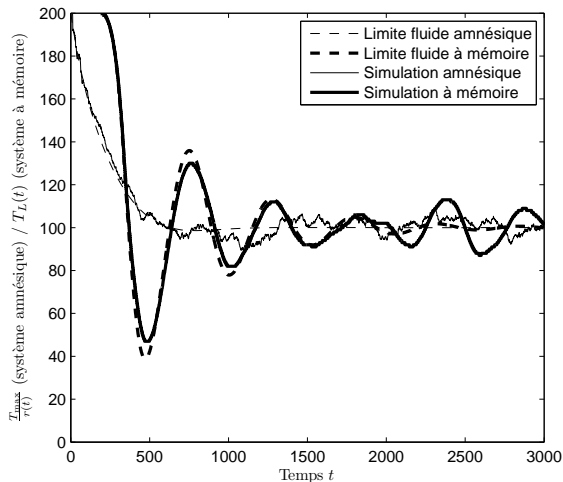
Transition vers le régime stationnaire

Se modélise par des équations différentielles plus ou moins compliquées.

$$\begin{pmatrix} \dot{n}_L(t) \\ \dot{n}_S(t) \end{pmatrix} = \begin{pmatrix} -\sigma_L & -\sigma_L \\ \sigma_L & \sigma_L - \sigma_S \end{pmatrix} \begin{pmatrix} n_L(t) \\ n_S(t) \end{pmatrix} + \begin{pmatrix} \lambda - C\sigma_L \\ C\sigma_L \end{pmatrix}$$

Transition vers le régime stationnaire

Se modélise par des équations différentielles plus ou moins compliquées.



Mort

- ▶ Un ou plusieurs chunks deviennent indisponibles
- ▶ Peut recréer une fausse flashcrowd
- ▶ Deux options : reseed ou game over

Le problème des free riders

À qui envoyer les données ? Pourquoi ?

- ▶ Deux problèmes à résoudre :
 - ▶ Comment le côté “serveur” d’un pair traite les demandes de données ? La solution file d’attente est équitable sur le temps d’attente mais pas très performante. . .
 - ▶ Comment motiver les pairs à uploader ?
 - ▶ Bande passante limitée : l’envoi coûte
 - ▶ Problèmes juridiques
 - ▶ → Pourquoi envoyer ?? → free riders : clients mais pas serveurs
- ▶ Bram Cohen → résoudre les deux problèmes en même temps

Le dilemme du prisonnier

- ▶ Deux bandits, Al et Vito, sont arrêtés
- ▶ Détenus séparément
- ▶ Preuves indirectes
- ▶ il faut donc les témoignages ! Incitation du procureur :

| Al / Vito | Al se tait | Al dénonce |
|--------------|----------------|-----------------|
| Vito se tait | 1 an / 1 an | libre / 20 ans |
| Vito dénonce | 20 ans / libre | 15 ans / 15 ans |

Le dilemme du prisonnier

Quelles sont leurs options ?

- ▶ Pour chaque choix de Vito, Al a intérêt à dénoncer
- ▶ item pour Al : à chaque fois, il a moins de prison s'il dénonce
- ▶ Or l'intérêt commun est évidemment le silence des deux

| Al / Vito | Al se tait | Al dénonce |
|--------------|----------------|-----------------|
| Vito se tait | 1 an / 1 an | libre / 20 ans |
| Vito dénonce | 20 ans / libre | 15 ans / 15 ans |

Le dilemme du prisonnier

Équilibre de Nash

Atteint quand aucun joueur ne peut améliorer son sort en changeant son choix (toutes choses égales par ailleurs)

Dilemme

L'équilibre de Nash n'est pas Pareto-optimal : $(15, 15) > (1, 1)$

| Al / Vito | Al se tait | Al dénonce |
|--------------|----------------|-----------------|
| Vito se tait | 1 an / 1 an | libre / 20 ans |
| Vito dénonce | 20 ans / libre | 15 ans / 15 ans |

Le dilemme du prisonnier, version free-riders !

Satisfaction

- ▶ Télécharger un fichier : +10
- ▶ risque juridique et bande passante : -1

| Alice / Bob | Alice partage | Alice free-rider |
|----------------|---------------|------------------|
| Bob partage | 9 / 9 | 10 / -1 |
| Bob free-rider | -1 / 10 | 0 / 0 |

En fait....

On joue au dilemme du prisonnier **itéré**
(plusieurs parties consécutives)

Observation empirique

Les stratégies “gentilles” gagnent

[JP Delahaye “L'altruisme récompensé ?” Pour la Science 1992]

Très bonne stratégie simple : Tit For tat : jouer comme l'autre au coup d'avant

Commencer par coopérer : la confiance mérite des sacrifices

Remarques

Un problème fréquent

Le dilemme du prisonnier apparaît dans de nombreuses situations réelles

- ▶ Équilibre de la terreur (MAD)
- ▶ Taux d'imposition des états
- ▶ Politique

Une recherche active

Récemment, Freeman Dyson a découvert une nouvelle classe de stratégies permettant de gruger l'adversaire.

L'idée de Bram Cohen



LE P2P est une partie de prisonnier itéré.

Il ne faut pas envoyer à ceux qui attendent depuis le plus longtemps... mais à ceux qui coopèrent le plus !
Mieux encore : à ceux qui me sont le plus utiles !

The Tit-for-Tat paradigm

Tit for tat

Un prêté pour un rendu
give and ye shall receive

BitTorrent

- ▶ c connexions d'upload ($c = 4$, puis $c = 0.8\sqrt{u}$)
- ▶ Toutes les 10s, calcul des meilleurs pairs (ont le plus envoyé)
- ▶ Envoi aux $c - 1$ meilleurs d'entre eux
- ▶ En plus, une connexion d'envoi vers un pair au hasard (optimistic unchoke)
 - ▶ sert à proposer de nouvelles connections
 - ▶ gagner la confiance en coopérant

The Tit-for-Tat paradigm

Lien avec conservation de la bande passante

$$d(u) \approx \frac{1}{c} \bar{u}_L + \frac{c-1}{c} u + \dots \text{ (seeders/serveurs)}$$

- ▶ Idée : loi de l'offre et de la demande (bourse)
- ▶ Plus formellement : réseaux à préférences (pas dans ce cours)
- ▶ Formule en moyenne temporelle
- ▶ Ne tient pas compte du bootstrap initial
- ▶ Les plus riches finissent plus vite → distribution déformée

The Tit-for-Tat paradigm

Avantages

- ▶ On gagne quelque chose à envoyer
- ▶ Du coup, la moyenne des envois grimpe beaucoup
 - ▶ La QoS totale du réseau y gagne !
 - ▶ Celle d'un pair aussi : débit émission \simeq débit réception
- ▶ Pas de calcul global de la coopération d'un pair
 - ▶ donc système impossible à gruger (sur ce point-là)
 - ▶ On évalue la qualité d'un lien, sur une très courte période

The Tit-for-Tat paradigm

Inconvénients

- ▶ Meilleure QoS globale → mieux pour les free riders...
- ▶ Démarrage (quand on n'a rien à offrir en retour)
- ▶ Pas d'incitation aux serveurs purs (seeds) → inefficace en surseeding

The Tit-for-Tat paradigm

Tolérance aux free riders

On peut montrer qu'avec le TfT, il peut y avoir trop de freeriders : tant que

$$p_f \leq 1/c + \frac{c-1}{c} \left(\frac{\bar{T}_S \bar{u}_S}{k} + \frac{U_C}{\lambda k} \right)$$

c'est bon, sinon explosion du système et des free-riders.

Share Ratio : Tit-for-Tat pour seeders

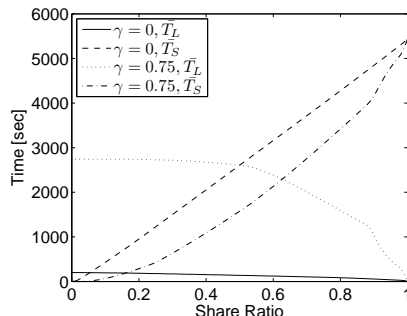
- ▶ But : conserver des seeders dans le système
- ▶ Share ratio : rapport quantité uploadée sur quantité téléchargée
- ▶ Politique de Share Ratio : punir (niveau tracker) les pairs ne respectant pas un certain share ratio minimal SR
- ▶ Générosité : quota initial pour bootstraper
- ▶ Quelle valeur choisir pour SR ?

Share Ratio : Tit-for-Tat pour seeders

- ▶ But : conserver des seeders dans le système
- ▶ Share ratio : rapport quantité uploadée sur quantité téléchargée
- ▶ Politique de Share Ratio : punir (niveau tracker) les pairs ne respectant pas un certain share ratio minimal SR
- ▶ Générosité : quota initial pour bootstraper
- ▶ Quelle valeur choisir pour SR ?
 - ▶ Il vaut mieux choisir $SR < 1$ (car la moyenne vaut 1)
 - ▶ Plus c'est proche de 1, plus la taille des swarms grossit

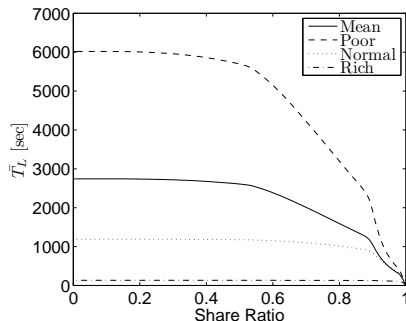
Share Ratio : influence générale

- ▶ $T_S(u) = \max(0, \frac{SRF}{u} - T_L(u))$
(F : taille fichier)
- ▶ $\gamma = \frac{k-1}{k}$: part du TfT
- ▶ T_S est impacté par la politique de TfT
- ▶ Remarque : sans SR, il faut plus longtemps avec le TfT ?



Share Ratio : influence par classe

- ▶ Supposons trois classes de puissance
- ▶ Le SR agit par seuils sur les différentes classes
- ▶ premier seuil observé à 0.6
- ▶ Deuxième seuil à 0.9 : risqué



Share ratio à 1 ?

Comme annoncé sur le [Forum](#) , le **ratio minimum passe de 0,75 à 1**.

Tous les ratios ont été augmentés en conséquence.

Si votre ratio était à 0,75, il est passé automatiquement à 1.

Ceux dont le ratio était légèrement inférieur à 0,75 auront un ratio légèrement inférieur à 1.

Afin de permettre aux membres de mieux appréhender cette nouvelle division dans le partage, Admin offre

une semaine complète de **free-leech**. (qui prendra fin le 1er septembre à 23h59 GMT)

Pour compenser de façon plus durable, sera instauré un free leech estival régulier.

Donc, en plus des 4 jours de free leech des fêtes du réveillon, seront offerts 4 jours de free leech 6 mois plus tard (dates non fixées pour l'instant).

Il y aura ainsi 2 bouées de sauvetage dans l'année.

Nous tenons également à signaler à tous que c'est bien de seeder mais que c'est bien aussi de télécharger.

Ceux qui ont un total uploadé astronomique ne doivent pas oublier qu'en "dépensant" cette avance, ils feront non seulement vivre des torrents mais aideront également des seeders dans le rouge. Cela fait aussi partie du partage.

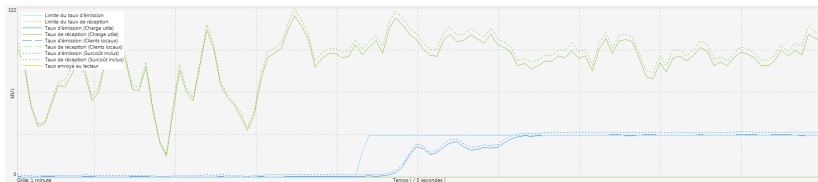
merci à tous

Le staff t411

En pratique : optimiser son BitTorrent

Règles de base pour télécharger vite

- ▶ Tit-for-tat $\rightarrow d = f(u)$: il faut uploader



En pratique : optimiser son BitTorrent

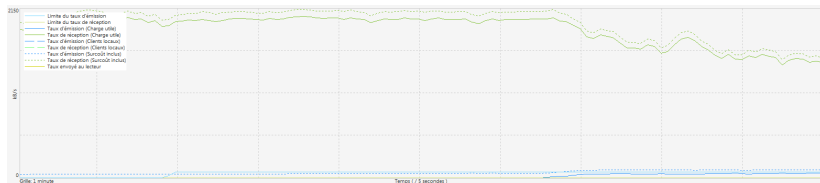
Règles de base pour télécharger vite

- ▶ Tit-for-tat $\rightarrow d = f(u)$: il faut uploader
- ▶ Les seeders comptent \rightarrow privilégier les trackers privés

En pratique : optimiser son BitTorrent

Règles de base pour télécharger vite

- ▶ Tit-for-tat $\rightarrow d = f(u)$: il faut uploader
- ▶ Les seeders comptent \rightarrow privilégier les trackers privés
- ▶ Si surprovisionnement, coupez (temporairement) l'upload !



En pratique : optimiser son BitTorrent

Règles de base pour télécharger vite

- ▶ Tit-for-tat $\rightarrow d = f(u)$: il faut uploader
- ▶ Les seeders comptent \rightarrow privilégier les trackers privés
- ▶ Si surprovisionnement, coupez (temporairement) l'upload !
- ▶ Remarque : une vitesse élevée ne garantit pas un faible délai de streaming

Partie IV

(Live) Streaming

Sommaire

Il y a délai et délai

Propager un chunk

Streaming structuré

Non-structuré

Rappel

On veut diffuser un contenu le plus rapidement possible.
Un diffuseur, N pairs.

Temps différé (diffusion)

Exemples

- ▶ 1ere diffusion d'un fichier (release). Une source, beaucoup de téléchargeurs (flashcrowd)
- ▶ Vidéo à la demande (VoD)

Temps réel (streaming)

- ▶ Exemple : télévision
- ▶ Des contraintes temporelles s'ajoutent

Delai : transmission point-à-point

- ▶ Combien de temps dans l'underlay pour communiquer de l'information entre deux voisins dans l'overlay ?
- ▶ Approche stripes (flux continu de données) :
latence réseau ($\approx RTT/2$)
- ▶ Approche chunks (unité atomique de données) :
Négociation+latence+TailleChunk/Bandwidth
- ▶ Pour les “gros” chunks :
TailleChunk/Bandwidth

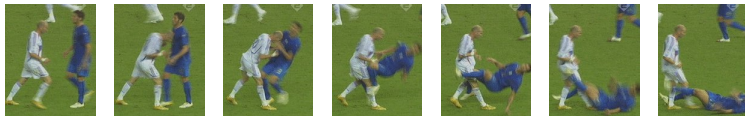
Délai de propagation P2P

- ▶ Rappel : décalage (play-out delay)
 - ▶ Temps entre un évènement et son affichage
 - ▶ Spécifique au direct
- ▶ Rappel : délai d'initialisation (start-up delay)
 - ▶ Temps pour commencer à regarder
 - ▶ Contrainte générique pour le streaming
- ▶ Délai de propagation
 - ▶ Temps pour qu'une donnée atteigne tous les pairs une fois injectée
 - ▶ Borne inf pour le décalage
 - ▶ Bonne estimation pour l'initialisation

Un exemple simple

- ▶ Une source unique
 - ▶ Direct (durée inconnue)
 - ▶ Ne peut transmettre qu'une copie du flux
- ▶ n spectateurs (pairs)
 - ▶ Veulent voir le flux
 - ▶ Chaque pair a la bande passante pour retransmettre un flux
- ▶ Comment faire ?

Approche chunks



- ▶ Le flux est découpé en morceaux à la volée
- ▶ La source injecte le flux chunk après chunk
- ▶ Les pairs peuvent envoyer les chunks (complètement) reçus

Sommaire

Il y a délai et délai

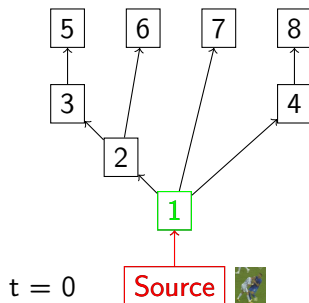
Propager un chunk

Streaming structuré

Non-structuré

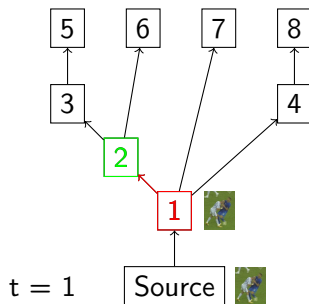
Simple chunk : transmission optimale

- ▶ Un client reçoit le chunk ($t = 0$)
- ▶ Ceux qui l'ont donnent aux autres
- ▶ Individuellement séquentiel, globalement parallèle
- ▶ C'est optimal :
 - ▶ Temps de propagation en $\log_2(n)$,
 - ▶ Une panne touche $\log_2(n)$ pairs en moyenne



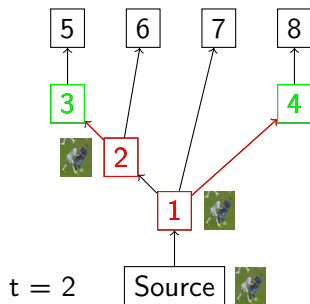
Simple chunk : transmission optimale

- ▶ Un client reçoit le chunk ($t = 0$)
- ▶ Ceux qui l'ont donnent aux autres
- ▶ Individuellement séquentiel, globalement parallèle
- ▶ C'est optimal :
 - ▶ Temps de propagation en $\log_2(n)$,
 - ▶ Une panne touche $\log_2(n)$ pairs en moyenne



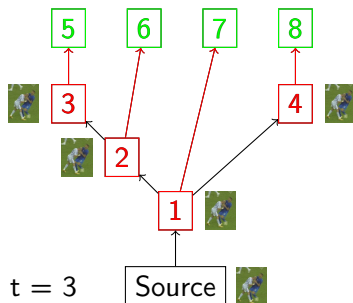
Simple chunk : transmission optimale

- ▶ Un client reçoit le chunk ($t = 0$)
- ▶ Ceux qui l'ont donnent aux autres
- ▶ Individuellement séquentiel, globalement parallèle
- ▶ C'est optimal :
 - ▶ Temps de propagation en $\log_2(n)$,
 - ▶ Une panne touche $\log_2(n)$ pairs en moyenne



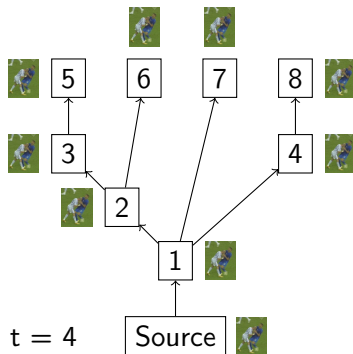
Simple chunk : transmission optimale

- ▶ Un client reçoit le chunk ($t = 0$)
- ▶ Ceux qui l'ont donnent aux autres
- ▶ Individuellement séquentiel, globalement parallèle
- ▶ C'est optimal :
 - ▶ Temps de propagation en $\log_2(n)$,
 - ▶ Une panne touche $\log_2(n)$ pairs en moyenne



Simple chunk : transmission optimale

- ▶ Un client reçoit le chunk ($t = 0$)
- ▶ Ceux qui l'ont donnent aux autres
- ▶ Individuellement séquentiel, globalement parallèle
- ▶ C'est optimal :
 - ▶ Temps de propagation en $\log_2(n)$,
 - ▶ Une panne touche $\log_2(n)$ pairs en moyenne



Propagation simple

Différents facteurs peuvent affecter la propagation

- ▶ Parallélisme individuel
- ▶ Nombre de copies par pair limité
- ▶ Effet retard

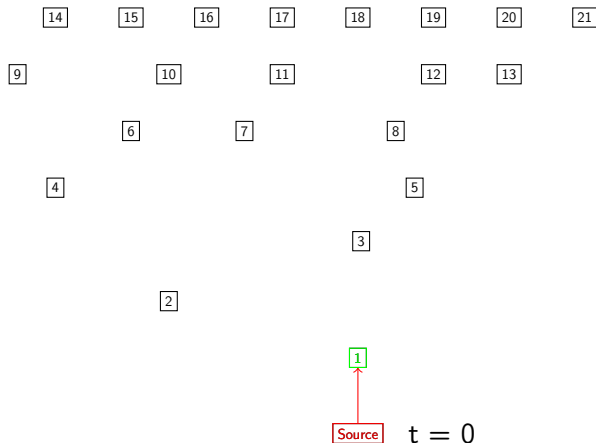
Le délai est toujours augmenté par rapport à l'optimal. L'origine de ces facteurs est :

- ▶ à cause de contraintes
- ▶ pour gagner ailleurs

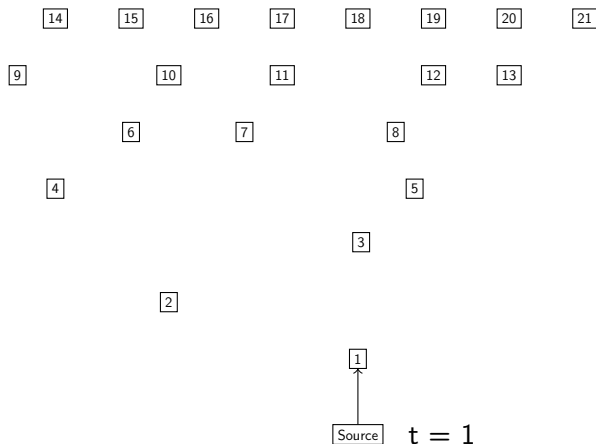
Exemple de contrainte

- ▶ Il faut une unité de temps pour être prêt à émettre
- ▶ Il faut une unité de temps pour émettre un bloc
- ▶ Totalement coopératif : tous participent
- ▶ Tous ont la même borne passante
- ▶ Délai de propagation ?

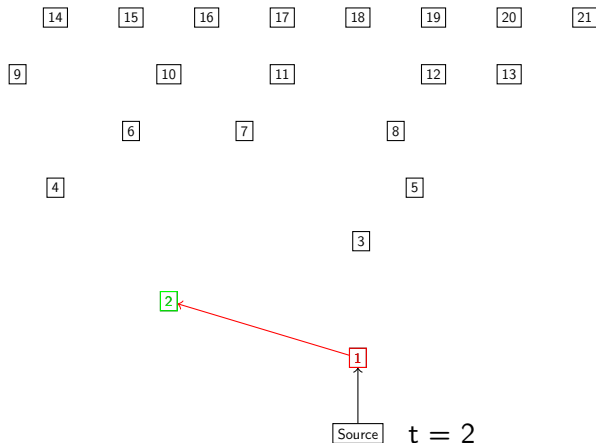
Exemple de contrainte



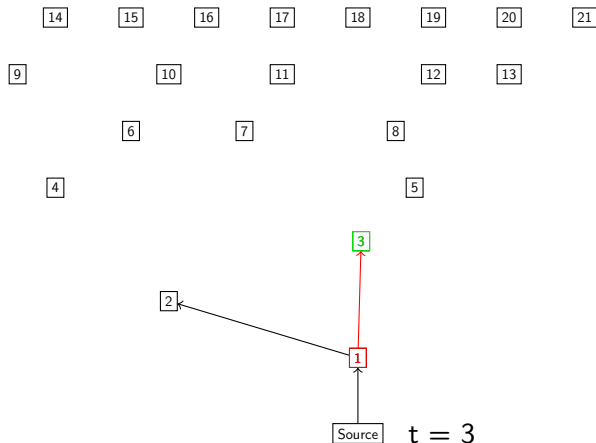
Exemple de contrainte



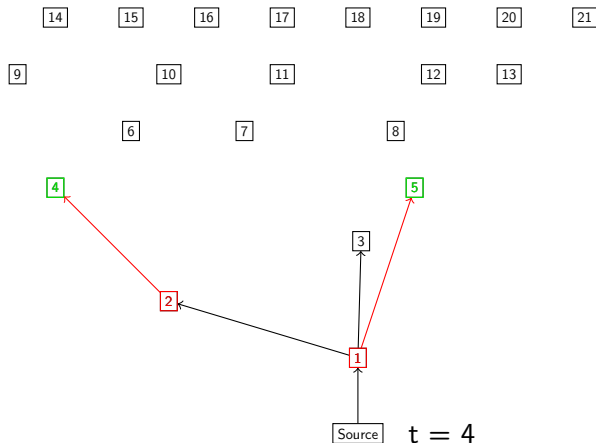
Exemple de contrainte



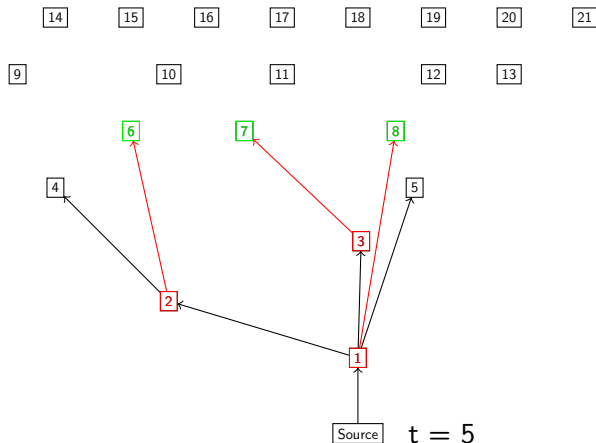
Exemple de contrainte



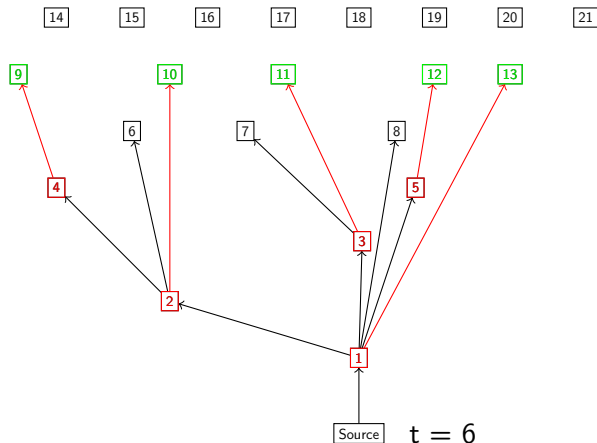
Exemple de contrainte



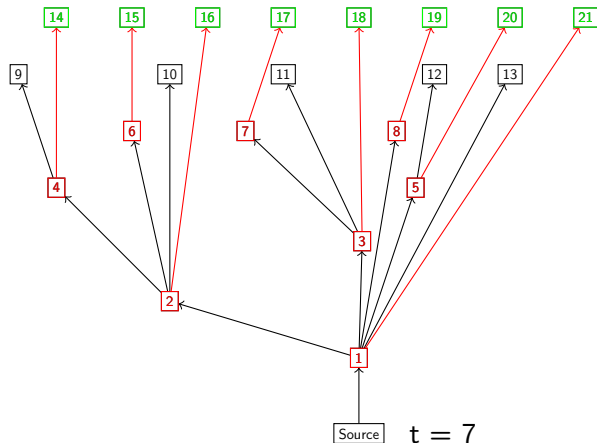
Exemple de contrainte



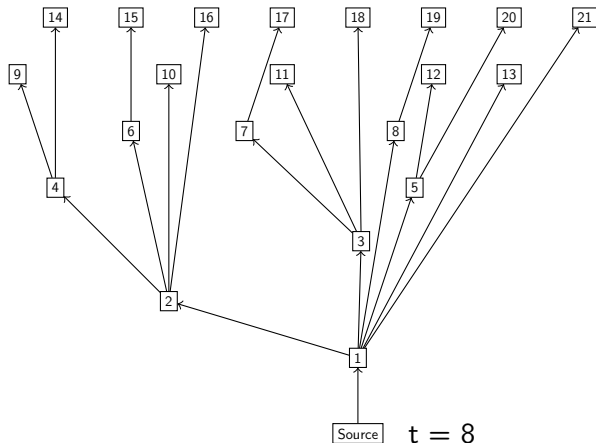
Exemple de contrainte



Exemple de contrainte



Exemple de contrainte



Exemple de contrainte

- ▶ On a une suite de Fibonacci !
- ▶ Rappel / exo : borne inférieure au temps de diffusion : $\Phi_n \simeq \phi^n$ avec $\phi = \frac{1+\sqrt{5}}{2}$ et n rounds d'émission $\rightarrow n = O(\log N)$

Sommaire

Il y a délai et délai

Propager un chunk

Streaming structuré

Non-structuré

Streaming structuré

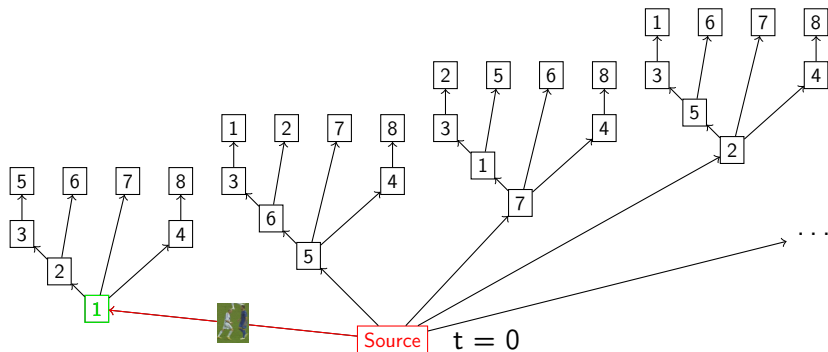
- ▶ But : passer d'un chunk à un flot continu
- ▶ Problème : concurrence entre chunks

Solution #1 : transmission linéaire

- ▶ Envoi à la queue leu leu
- ▶ Ça marche : tout le monde regarde
- ▶ Délai de propagation horrible ($O(n)$)
- ▶ Pas très robuste (un départ crée 50% de pertes en moyennes)

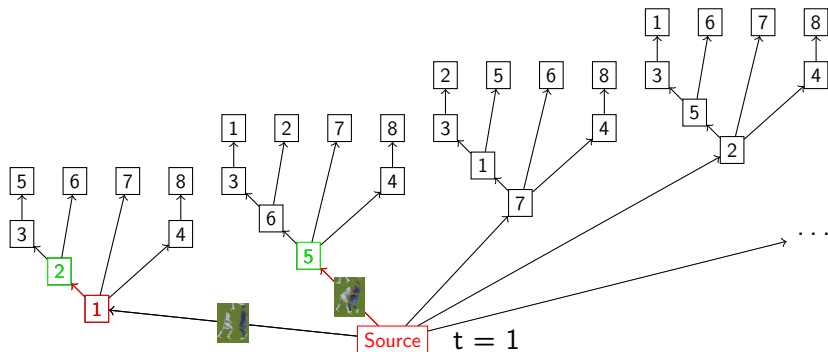
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



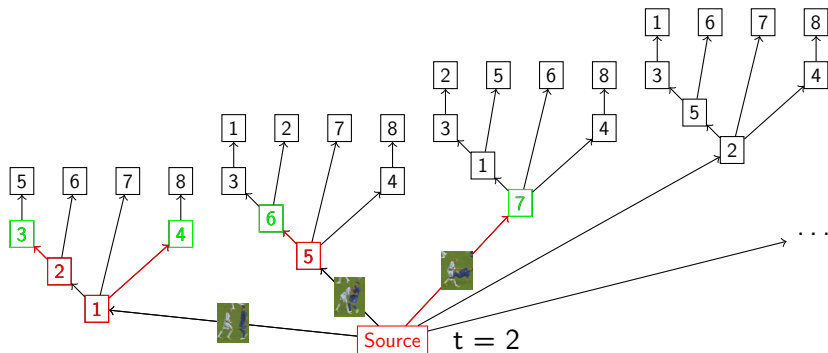
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



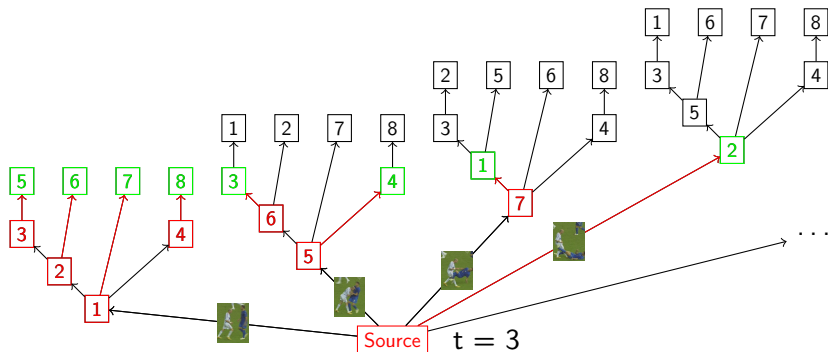
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



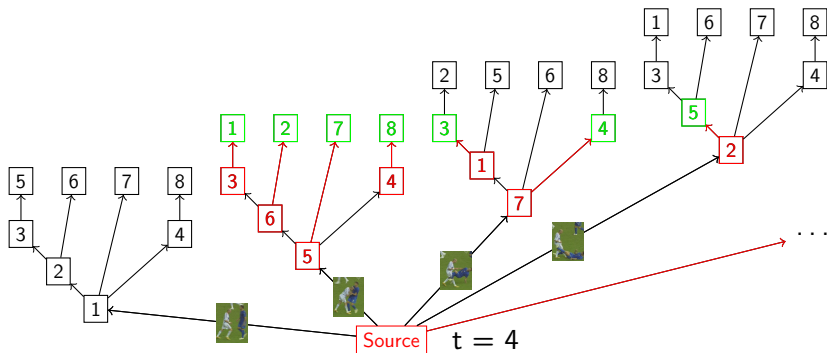
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



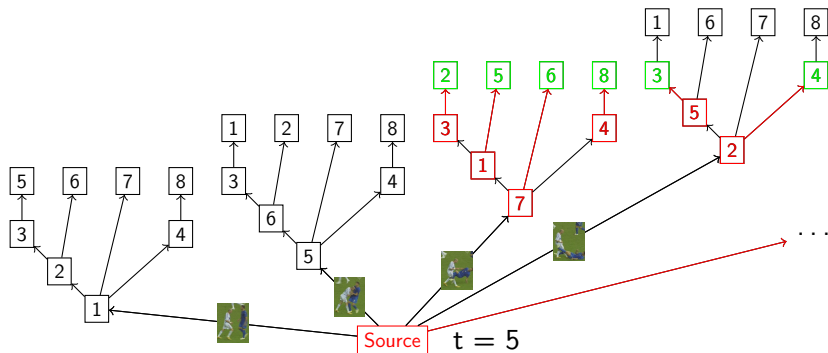
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



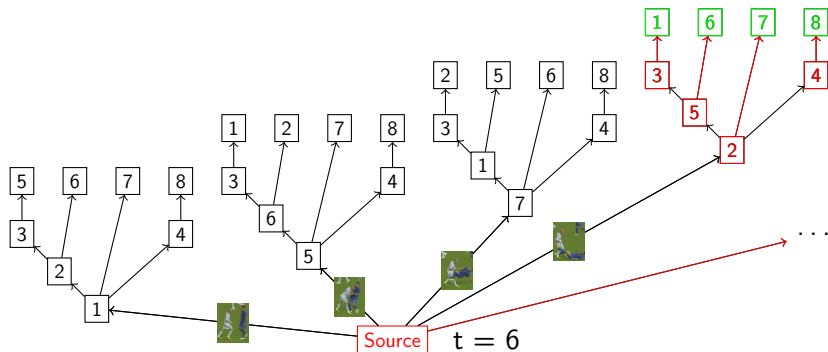
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



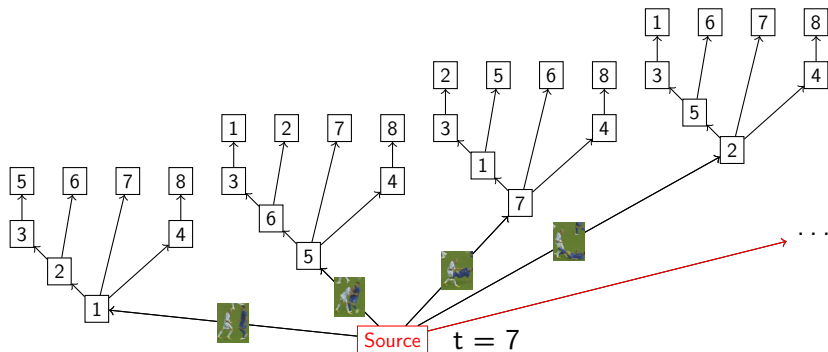
Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



Solution #2 : transmission optimale

- ▶ Il faut permuter sur l'optimal d'un simple chunk
- ▶ Problème : éviter les contentions
 - ▶ à l'envoi (rouge)
 - ▶ à la réception (vert) si download critique
- ▶ Faisable (mais un peu compliqué) sur l'exemple simple !



Une borne théorique

La propagation du bloc simple est une borne inf

- ▶ Cas homogène : valeurs identiques

- ▶ stripes

$$D = \tau \log_d \left(\left\lceil \frac{N_L}{N_C} \right\rceil \right), \text{ avec}$$

τ : latence ; d : degré sortant

- ▶ Gros chunks

$$D = \frac{c}{u} \log_2 \left(\left\lceil \frac{N_L}{N_C} \right\rceil \right), \text{ avec}$$

c : taille du chunk

- ▶ Extension au cas hétérogène : compliqué

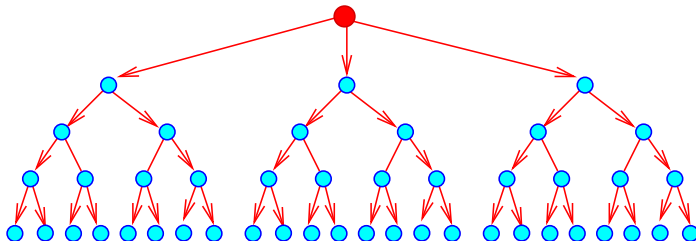
Délai constant ?

- ▶ Le délai est logarithmique en $\frac{N_L}{N_C}$
- ▶ Si $\frac{N_L}{N_C}$ est borné, le délai aussi
 - ▶ Obligé si pas de scalabilité

$$\frac{N_L}{N_C} \leq \frac{1}{1 - (\alpha_L + \beta\alpha_S)}$$

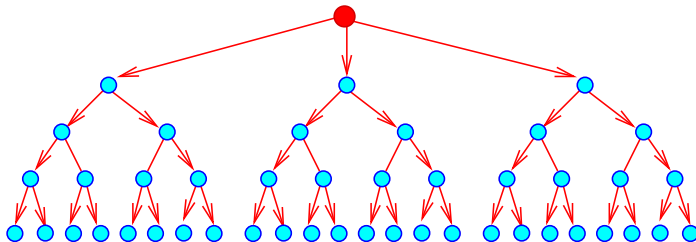
- ▶ Même si, un gros N_C est bénéfique pour le délai.

L'arbre binaire (Solution #3?)

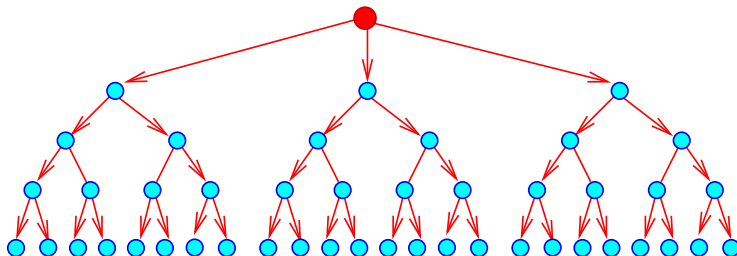


- Avantage : on a le log !

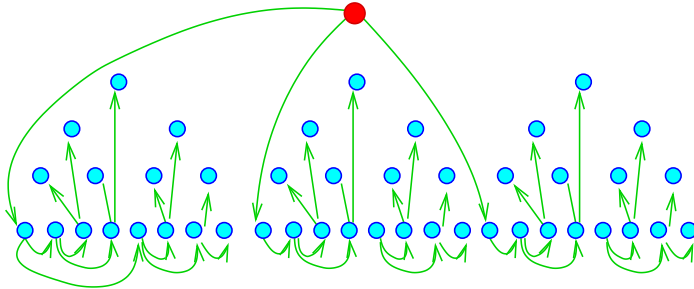
L'arbre binaire (Solution #3?)



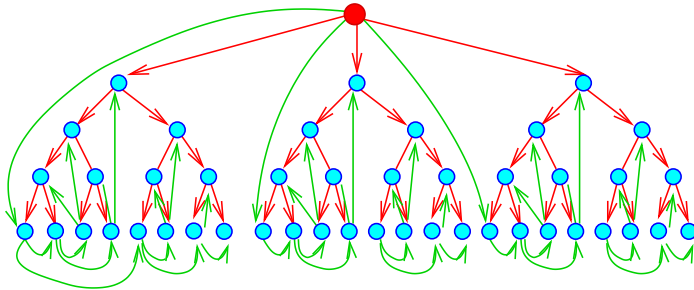
- Avantage : on a le log !
- Inconvénients
 - bande passante des feuilles inutilisée !
 - au contraire chaque nœud interne émet 2 fois plus
 - free riders → feuilles

Solution #3 : **deux** arbres binaires !

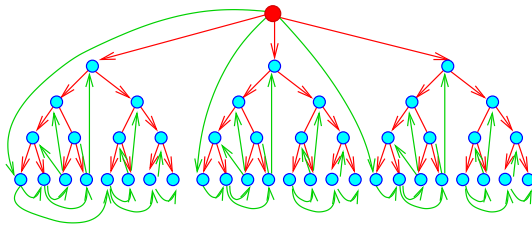
Solution #3 : **deux** arbres binaires !



Solution #3 : **deux** arbres binaires !



Solution #3 : **deux** arbres binaires !



Comparaison

- ▶ Délai aussi court : $\log N$
- ▶ Plus de bande passante gaspillée : émission = réception chez tout le monde (ou presque)

Splitstream [Castro & al 2004]

- ▶ Utilise k -arbre à nœuds internes disjoints
- ▶ Une DHT (Pastry) pour maintenir la structure
- ▶ Les pairs «riches» aident dans plusieurs arbres
- ▶ Plus 2/3 propositions (stripes, RS, MDC, ...)

Sommaire

Il y a délai et délai

Propager un chunk

Streaming structuré

Non-structuré

Abandonner la structure ?

Maintenir la topologie est coûteux et complexe

Essayer des réseaux non-structurés !

Topologie

Le **graphe des connections** entre pairs est “n’importe quel” sous-graphe du **graphe des connaissances**

Algorithme de diffusion

Essentiellement une inondation intelligente !

Propagation épidémique

Propagation épidémique = inondation intelligente

Étudient plusieurs stratégies : donner

1. Un bloc au hasard à un voisin au hasard
2. Le bloc le plus récent à un voisin au hasard
3. Le bloc le plus récent à un voisin qui ne l'a pas
4. À un voisin le bloc le plus récent qu'il n'a pas

Efficacité

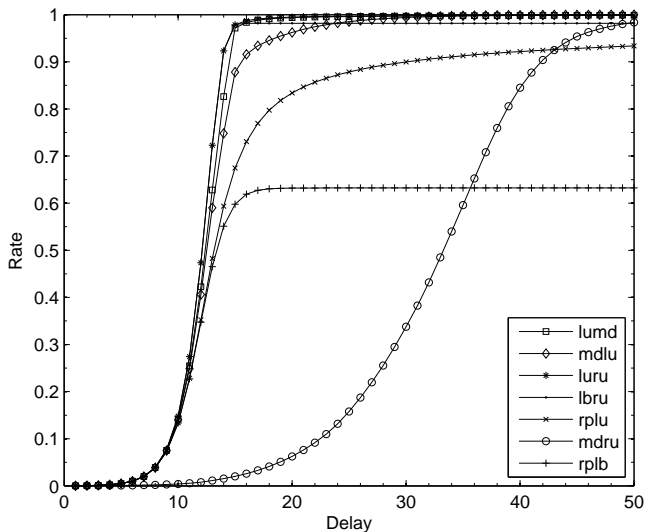
3 et 4 sont à un facteur constant de l'optimal (délai et débit) !

Inconvénient

- ▶ Il est trop facile d'être free-rider !
- ▶ Attention à l'overhead

Propagation épidémique : simulations

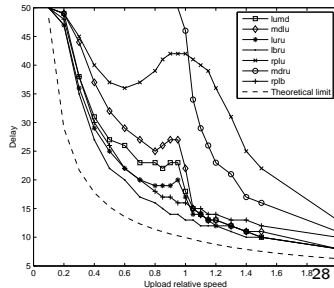
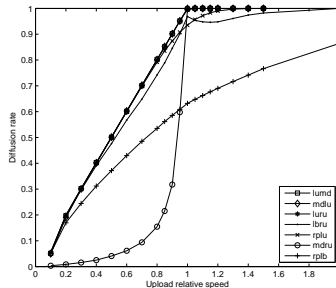
On peut évaluer numériquement différentes stratégies



Propagation épidémique : simulations

Plusieurs facteurs sont à prendre en compte :

- ▶ Bande passante
- ▶ Hétérogénéité
- ▶ Type d'overlay
- ▶ Fraicheur des maps
- ▶ ...



BitTorrent à fenêtre

Avantages de BitTorrent

- ▶ Protocole efficace.
- ▶ Le **graphe des échanges** est non-structuré.
- ▶ Grâce à l'**incitation à l'émission** (tit-for-tat) la bande passante est bonne

Faiblesse pour le temps réel

La politique de téléchargement du bloc le plus rare “garantit” même équirépartition des blocs. Ce qu'on ne veut pas : si chunks infinis, start-up = T_L !

Patch

Modifier la priorité des blocs

BitTorrent à fenêtre

Approche par tronçons [PPLive, RedCarpet, ...]

- ▶ La source découpe le flux en gros blocs
- ▶ On télécharge le bloc $B(t)$ (découpé en petits sous-blocs) en BitTorrent
- ▶ Pendant qu'on regarde $B(t)$ on télécharge $B(t + 1)$
- ▶ Puis on passe aux blocs $B(t + 2)$, $B(t + 3)$...
- ▶ La source peut gérer le démarrage de chaque pair en donnant un sous-bloc du bloc $B(t)$ à chacun, puis ils se débrouillent
- ▶ Inconvénient : temps de démarrage = temps d'un bloc ou deux
- ▶ Le délai est aussi le temps d'émettre deux blocs

BitTorrent à fenêtre

Approche par fenêtre glissante [Pulse, BASS, ...]

- ▶ Petits blocs (très vite échangeables)
- ▶ La priorité est une fonction de la **rareté** et de **l'urgence** (temps avant l'affichage du bloc)
- ▶ Problème : si on veut un petit délai, tous les blocs sont très urgents !
- ▶ Même si la connection est très rapide, le buffering ne sert à rien
- ▶ Les déconnexions font perdre du temps : moins intérêt qu'en BitTorrent

BitTorrent à fenêtre

Approche par scheduler modifié

μ -torrent possède une fonction flux

- ▶ Idée : changer les priorités des chunks (fenêtre/rarest)
- ▶ Avantage : permet de lancer la vidéo assez vite dès qu'il y a assez de monde
- ▶ Inconvénients :
 - ▶ Risque de perturbation de la distribution des chunks
 - ▶ Syndrome du chunk lent
 - ▶ VoD, pas Live ;)

Remerciements

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