chapitre 3

Applications des GRI

Section 2

Peer-to-Peer

Fabien de Montgolfier
fm@irif.fr

11 mars 2022

Problématique : trouver une donnée

Limites du routage de requêtes par inondation

- Évidentes ! La *moitié* de la bande passante de gnutella 0.4 !

Limites des serveurs de requêtes

- contraire au paradigme pair-à-pair !
- Si pas assez de serveurs : attaques des serveurs
- Si trop de serveurs :
  - comment les trouver ? Maintenir la cohérence des données ?
  - on se ramène à l'inondation

La troisème solution

La DHT !

Hachage

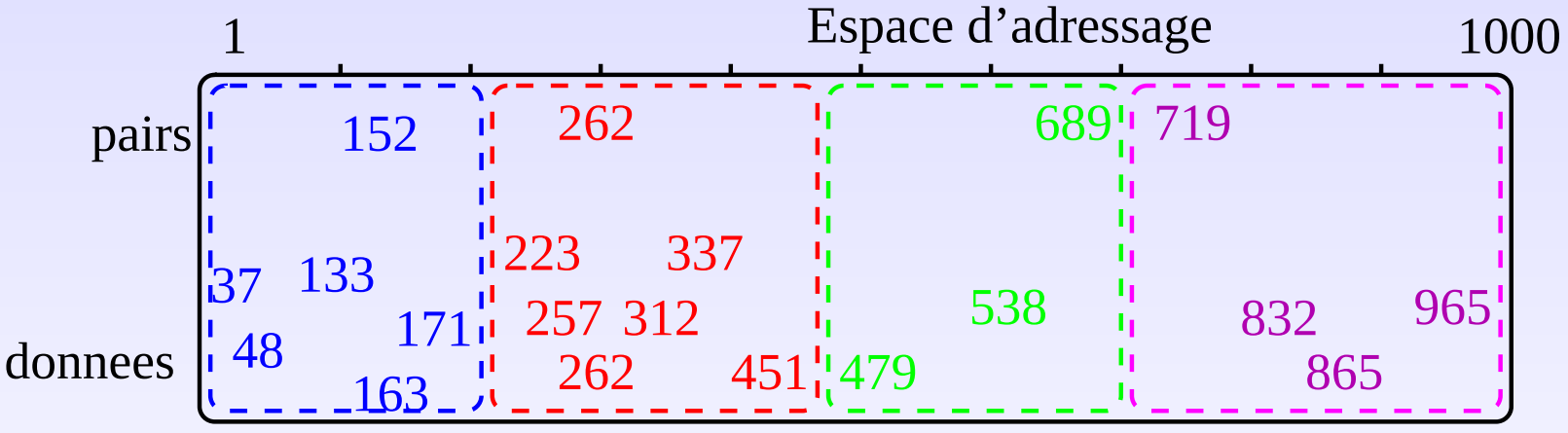
Fonction de hachage

fonction h : $E \mapsto [0, ..2^n[$
E = tout ! (ensemble de fichiers, noms de fichiers, adresses IP...)

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 héberge les données dont le numéro est le plus proche du sien

Indexation par les plus proches



Le pair de hashcode 152 indexe les données de hashcode 37, 48, 133, 163 et 171

Que mettre dans une DHT ?

Hash de quoi ?

- Nom de fichier pour trouver le contenu ?

Que mettre dans une DHT ?

Hash de quoi ?

- Nom de fichier pour trouver le contenu ?Non !
  - car collisions : quel contenu pour `linux.iso` ?
  - pas de recherche approchée `linux*.iso`
- Propriétaire ?

Que mettre dans une DHT ?

Hash de quoi ?

- Nom de fichier pour trouver le contenu ?Non !
  - car collisions : quel contenu pour `linux.iso` ?
  - pas de recherche approchée `linux*.iso`
- Propriétaire ? Non ! Plutôt dans une **blockchain**
  - NFT (propriété du contenu correspondant au hash)
  - cryptomonnaie (portefeuille $\neq$ hashcode)
  - une DHT n'a aucun mécanisme d'authentification

Que mettre dans une DHT ?

Hash de quoi ?

- Nom de fichier pour trouver le contenu ?Non !
  - car collisions : quel contenu pour `linux.iso` ?
  - pas de recherche approchée `linux*.iso`
- Propriétaire ? Non ! Plutôt dans une **blockchain**
  - NFT (propriété du contenu correspondant au hash)
  - cryptomonnaie (portefeuille $\neq$ hashcode)
  - une DHT n'a aucun mécanisme d'authentification
- Localisation unique d'une ressource ? Peu utilisé
  - nom de domaine$\rightarrow$ hash $\rightarrow$ IP
  - pseudonyme unique $\rightarrow$ hash $\rightarrow$ email ou URL
- Pair qui **hébergent un contenu** et peuvent l'offrir
  - *swarming* bittorrent
  - *swarming* eMule

Pour et contre

Avantages

- On peut supposer que la fonction de hachage est **uniforme** : les hash sont densément répartis dans $[0, ..2^n[$
- donc s'il y a N pairs et D données, chaque pair indexe D/N données
- isotropie totale, pas de serveur
- charge faible pour chaque pair

Pour et contre

Problèmes

- Le pair peut très bien s'en aller car les pairs restent peu connectés (en moyenne) donc :
  - redondance : copier les données chez k pairs
  - maintenir les copies (départ/arrivées)
- Requêtes peu souples : par mot-clé seulement
- Et indexation seule, d'une façon générale !
- collisions (très gros swarms)
- Comment trouver le pair qui possède une donnée ?

Routage par la clé

Algorithme de recherche

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

Analyse

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

Routage par la clé

Comment faire ?

- Le réseau des connaissances *essaie* d'avoir une certaine **topologie** :
  - Chord [Stoica & alii, 2001] : cercle
  - CAN [Ratnasamy et al., 2001] : tore multidimensionnel
  - Pastry [Rowstron et Druschel 2001] : hypercube
  - Viceroy [Malkhi et al., 2002] : papillon
  - Kademlia [Maymounkov Mazières 2002] : hypercube
  - D2B [Fraigniaud et Gauron, 2003] : de Bruijn
- On fait un **routage glouton** vers la clé
  - on marche de voisin en voisin
  - à chaque étape, on progresse vers le but
  - Or le diamètre du graphe est $O(\log n)$...
- On arrive donc sur le voisin recherché en $O(\log n)$ étapes

Chord [Stoica & alii, 2001]

Identifiant pris modulo 2^{160} (placés sur un cercle)

Chaque pair a un hash SHA-1

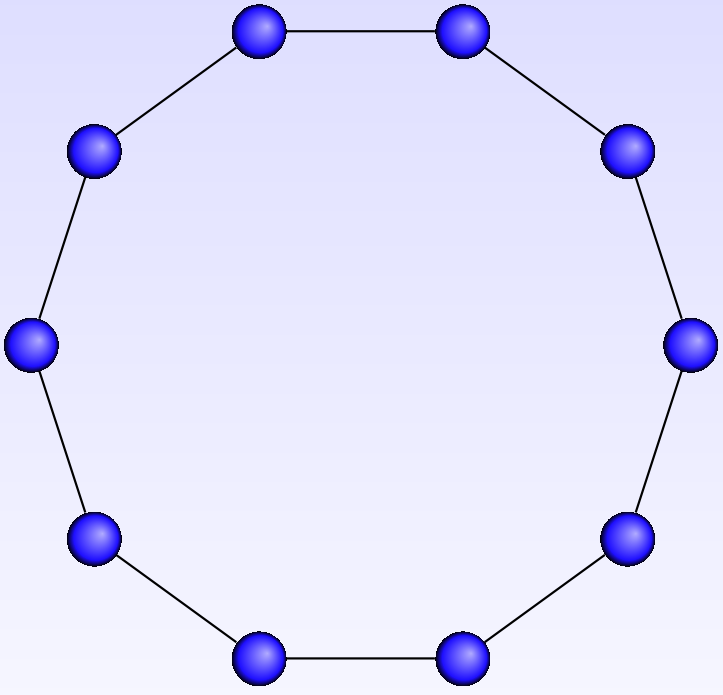
Chaque pair connaît son successeur et son prédecesseur

Chord doit maintenir cette information

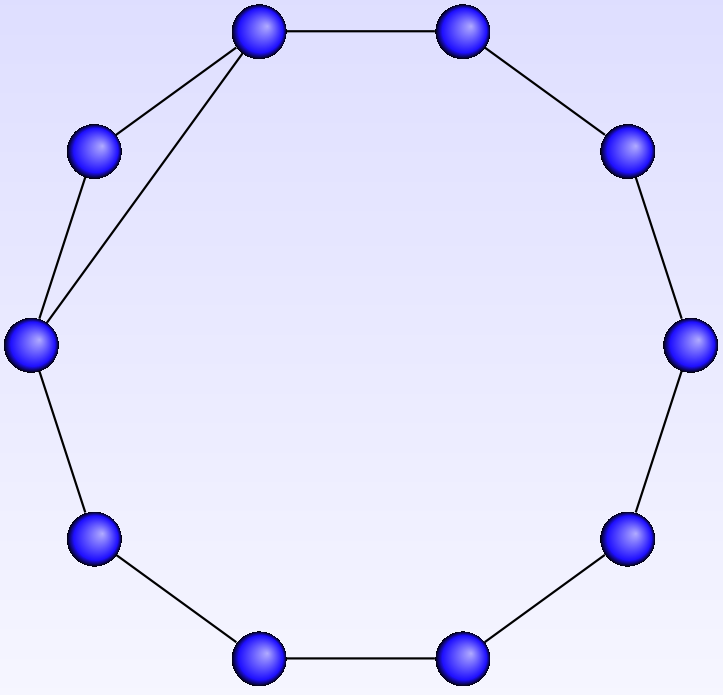
En plus il y a des **raccourcis** vers les pairs lointains

Ce sont eux qui vont permettre un routage plus rapide. k raccourcis en tout

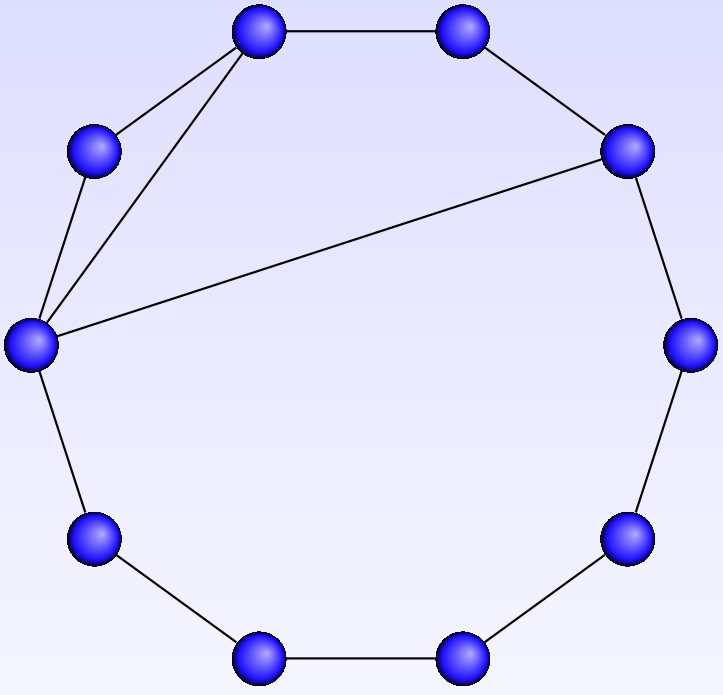
Chord : topologie schématique



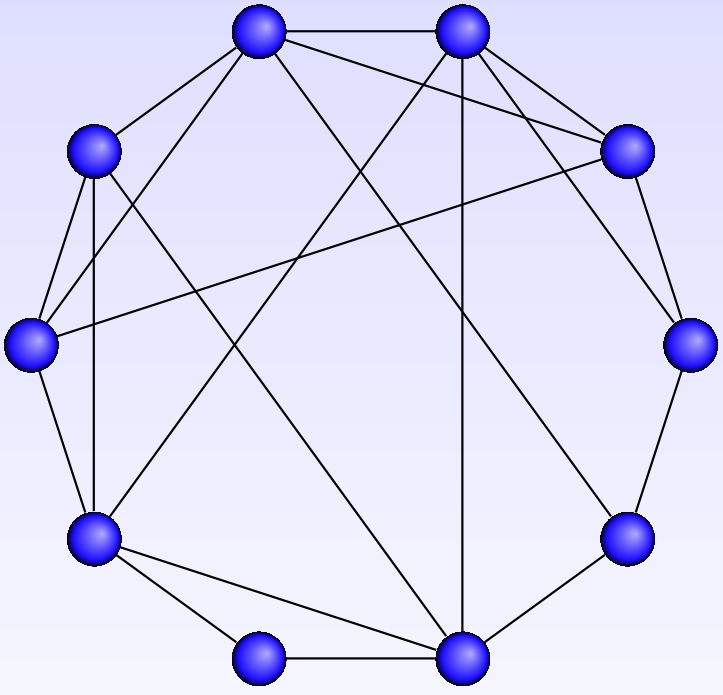
Chord : topologie schématique



Chord : topologie schématique



Chord : topologie schématique



Algorithme de routage

Algorithme

- Prendre le plus proche voisin du but parmis :
  - Sucesseur et prédécesseur
  - les k raccourcis
- Lui propager la requête

Analyse

Pour N pairs :

- Temps au pire : $N/2$ (via les successeurs ou predecesseurs)
- Temps moyen : $O(\log N)$ **si les raccourcis sont bien distribués**.

Algorithme de routage

Raccourcis

Soit $H(P)$ le hashcode du pair P.
Pour tout $i = 0, 1, 2, 3...$: `finger[i]` est le pair responsable de la clé $H(P) + 2^i$
Les raccourcis suivent donc une distribution **harmonique** : leur distance à P est 1,2,4,8,16...2^i...

Algorithme

- Prendre le plus proche voisin du but parmis :
  - Sucesseur et prédécesseur
  - les k raccourcis
- Lui propager la requête

Illustration d'une requête

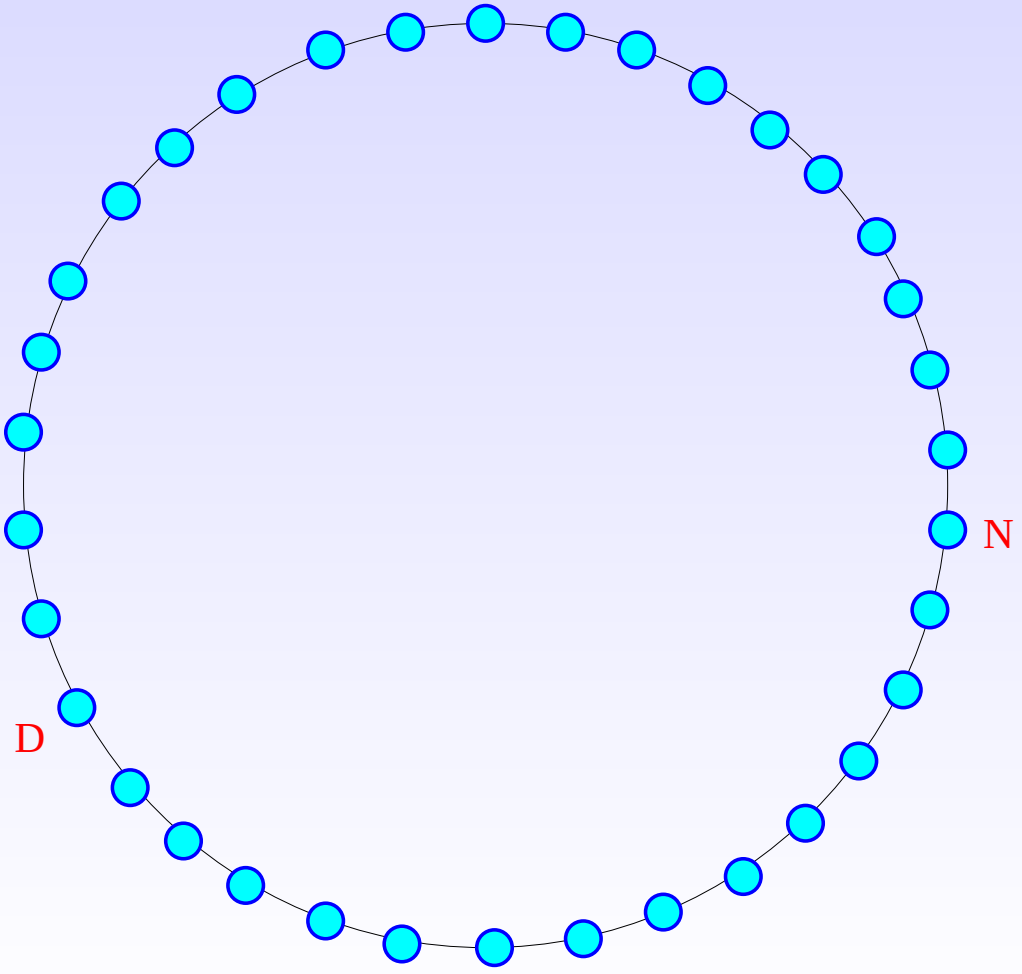


Illustration d'une requête

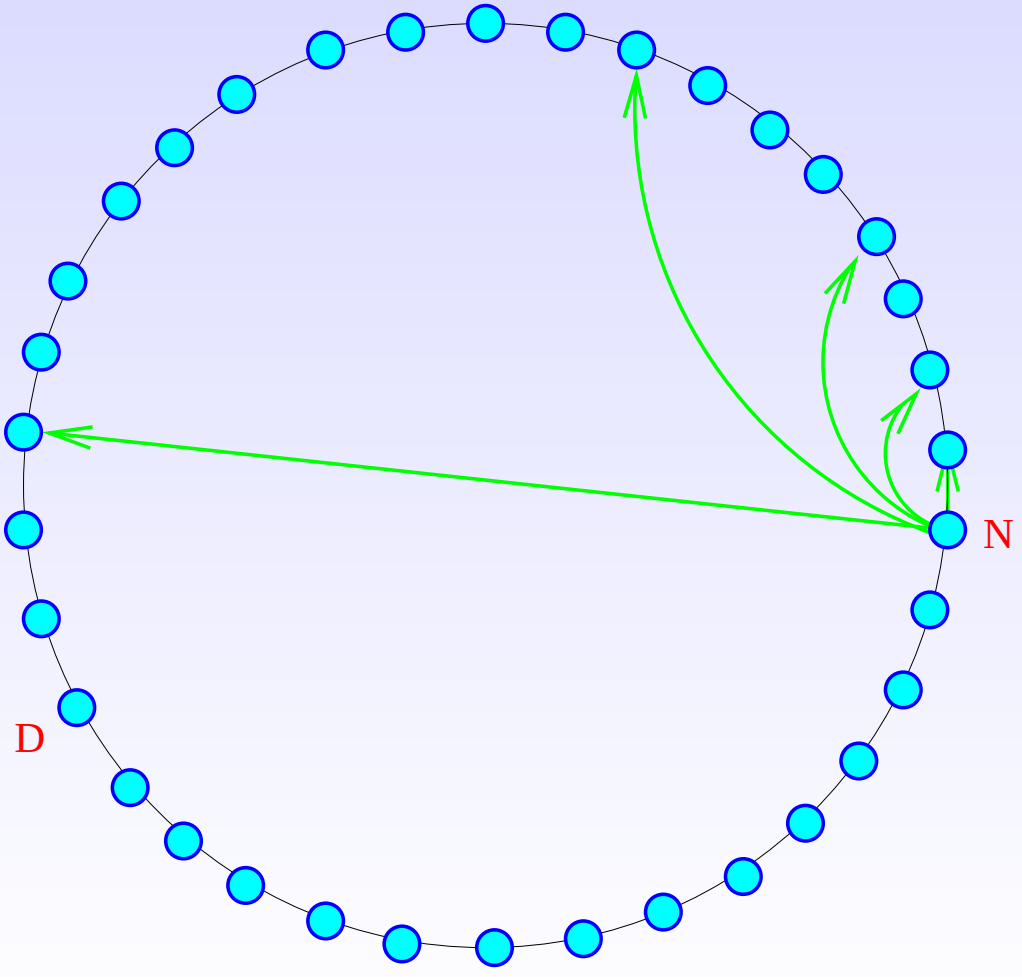


Illustration d'une requête

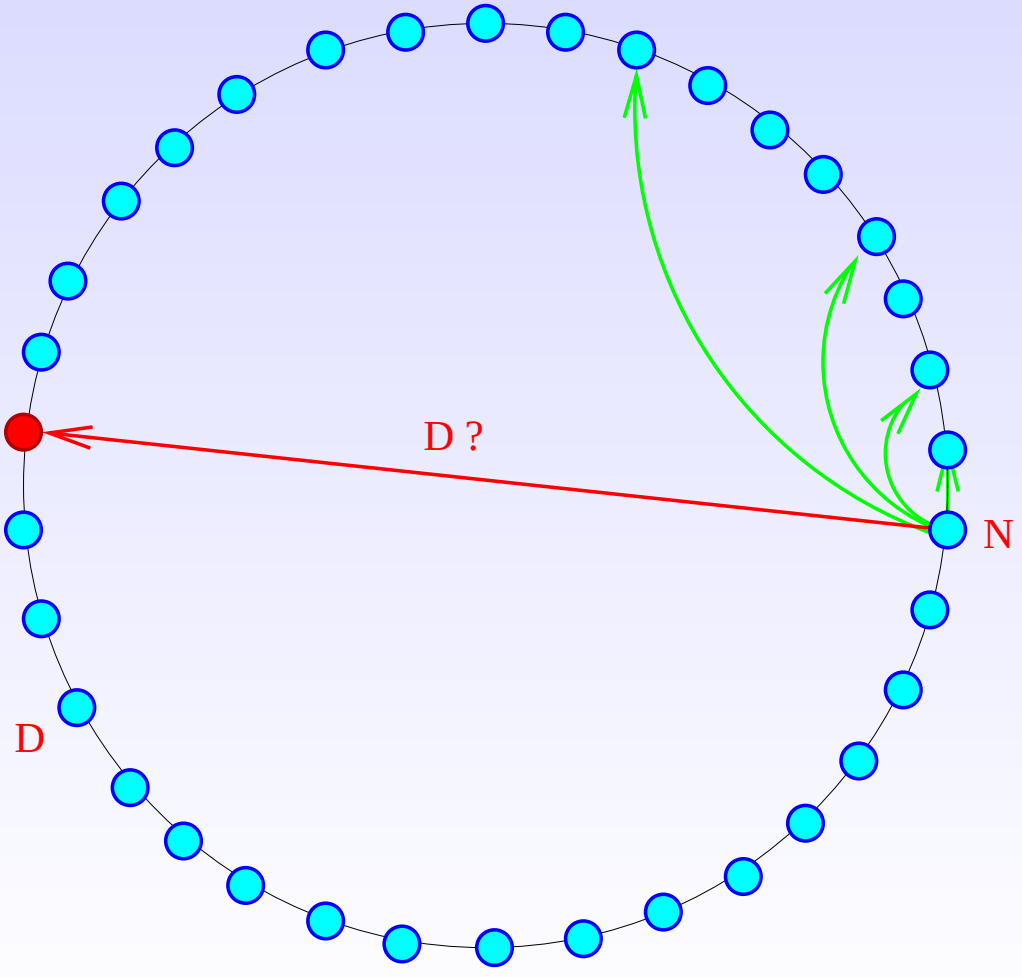


Illustration d'une requête

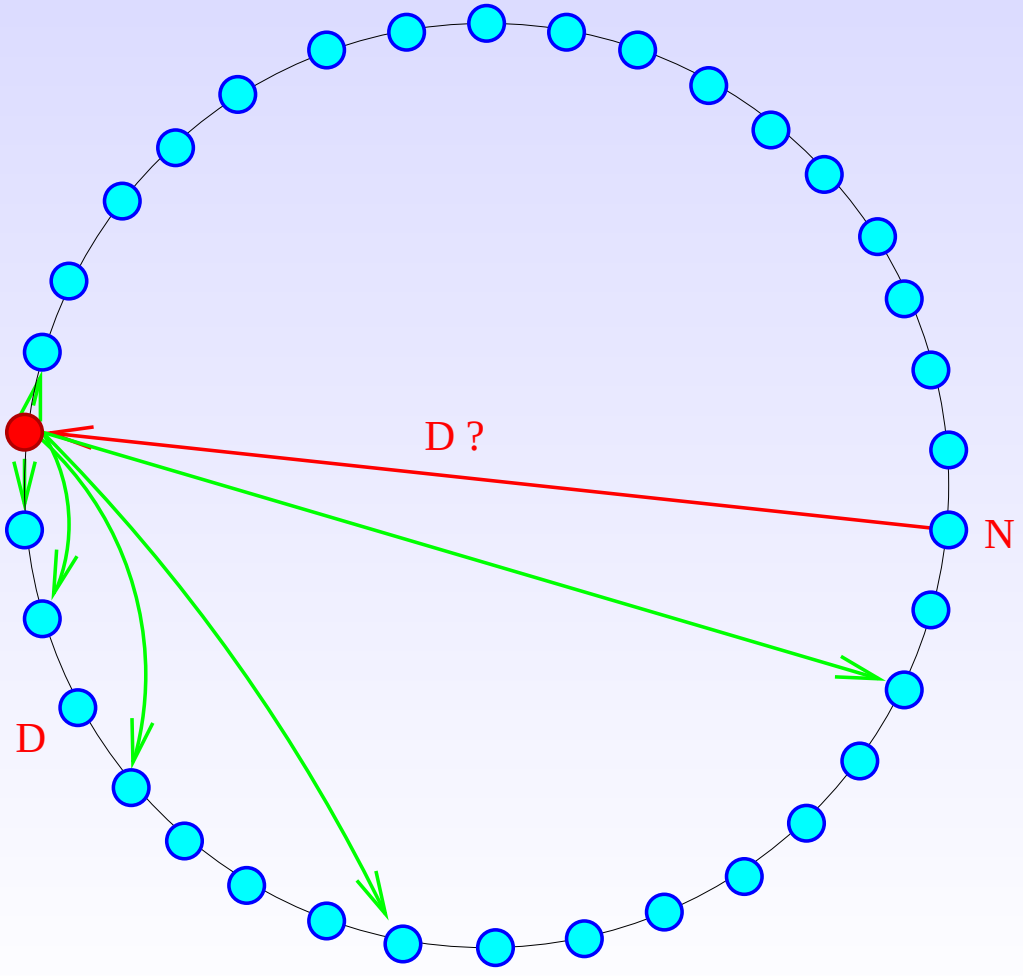


Illustration d'une requête

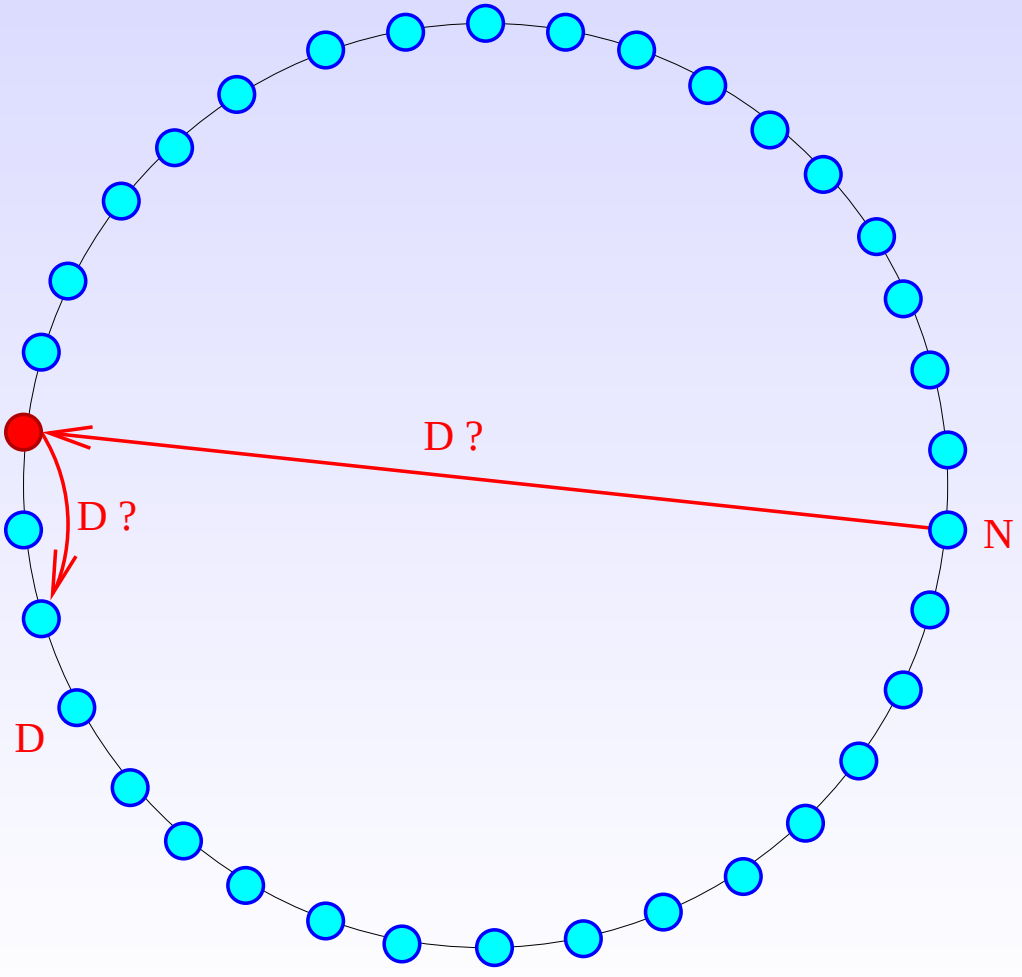


Illustration d'une requête

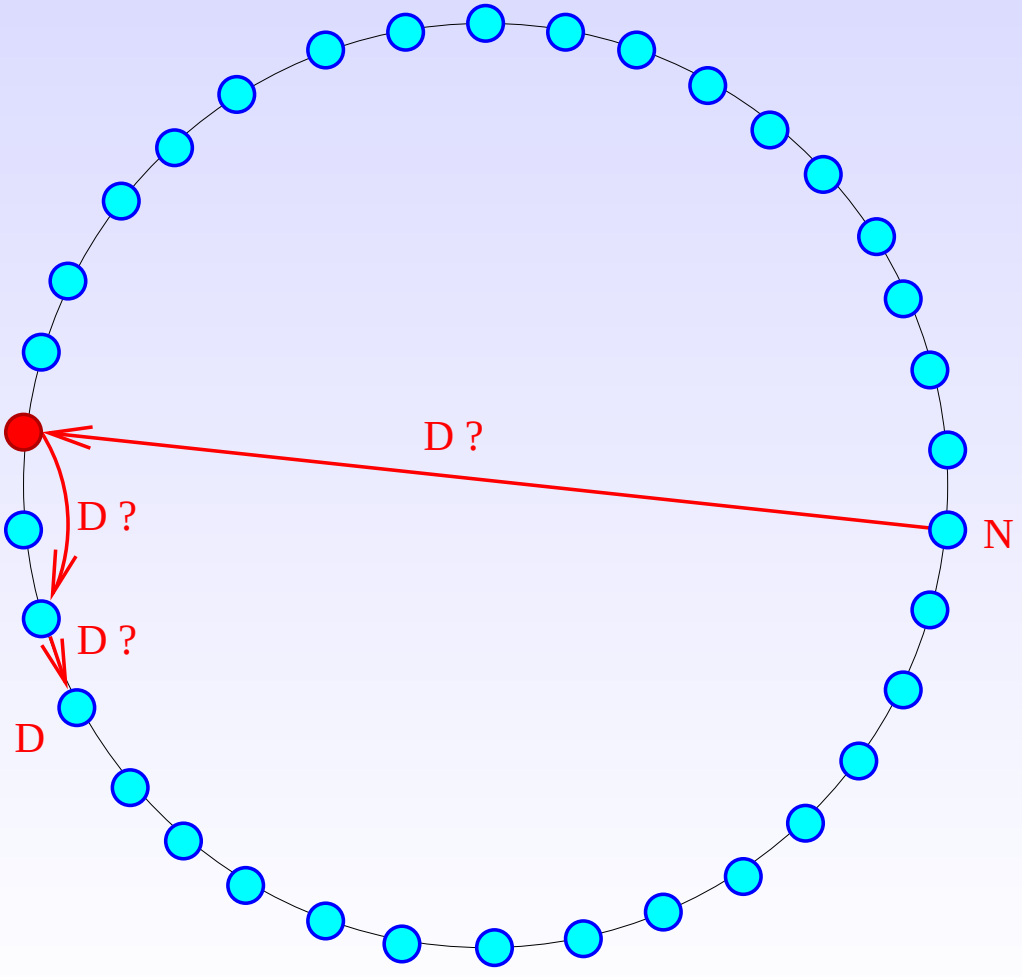
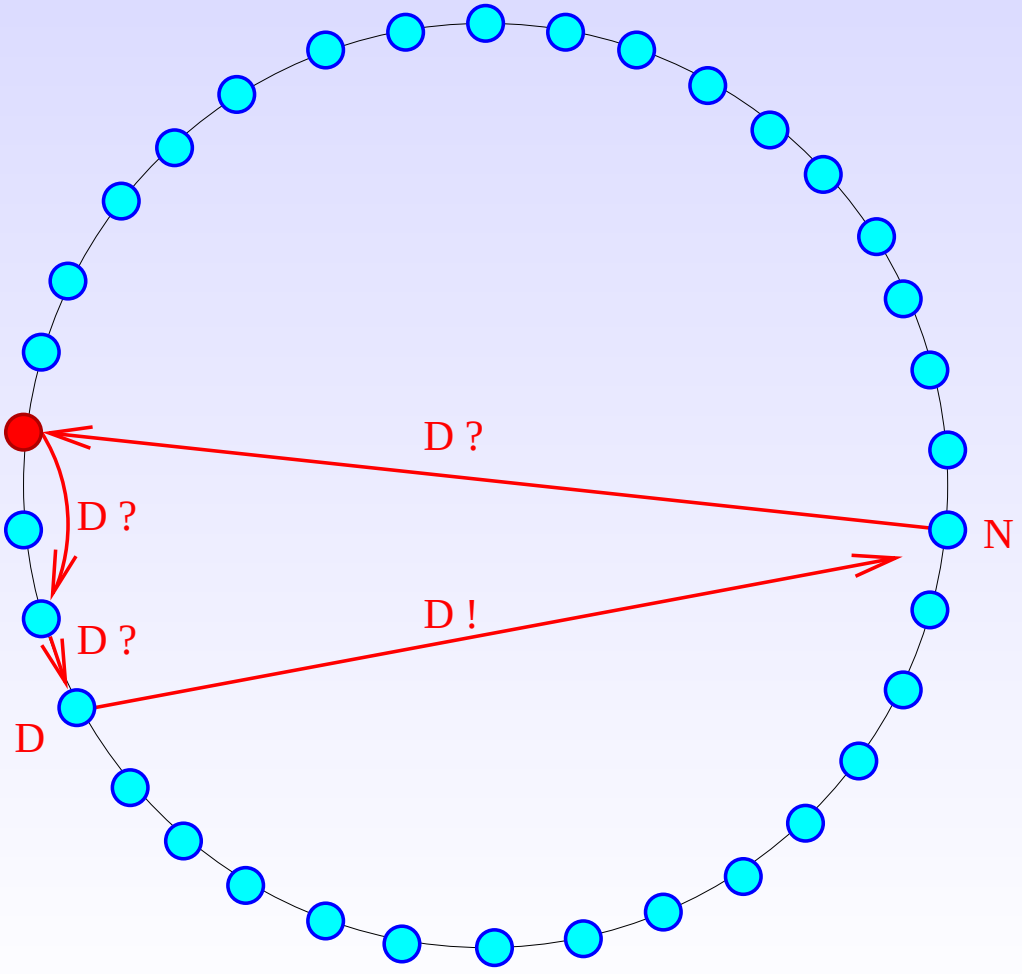


Illustration d'une requête



Analyse : Avantages

Avantages

- Hachage $\rightarrow$ Bonne répartition des données sur les pairs
- **passage à l'échelle** le nombre de pairs peut croître indéfiniment
  - Sans augmenter la charge
  - Avec une influence marginale sur le temps de requêtes (2 fois plus de pairs $\rightarrow$ une requête de plus...)
- Requêtes en $O(\log N)$ messages et $O(\log N)$ temps, contre $O(N)$ pour l'inondation

Analyse : Inconvénients

Il faut maintenir une **sur-topologie** (ici, l'anneau)

- Les départs (sans prévenir) brisent l'anneau
- Il faut sans cesse reconfigurer
- Les arrivées modifient aussi la topologie ! (les intervalles deviennent faux)
- Et en plus, il faut pouvoir s'insérer dans l'anneau. Coût : $O((\log N)^2)$ messages

Maintenir la topologie

Chord a du mal a maintenir que `finger[i]` est à distance 2^i.
Heureusement, il n'en a pas vraiment besoin.

Observation

Si chaque pair connait, pour tout i un autre pair situé à distance $d \in [2^i, 2^{i+1}[$ **alors** le routage peut se faire en $O(\log N)$

Liste étendue de sucesseurs

Au lieu de garder 1 successeur Chord en stocke r pour parer aux pannes... mais pas aux attaques !

Attaques

Attaque par rupture de la topologie

1. Déclarer des tas de faux pairs d'indentifiants proches
2. Les éteindre tous en même temps : l'anneau est rompu

Attaques par pollution

Déclarer des tas de fausses données
Les DHTs ont du mal à s'en prémunir

Topologie : l'hypercube

Idée déjà présente dans Pastry [2001] puis dans Kademlia[2002]

Idée de base

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

Topologie : l'hypercube

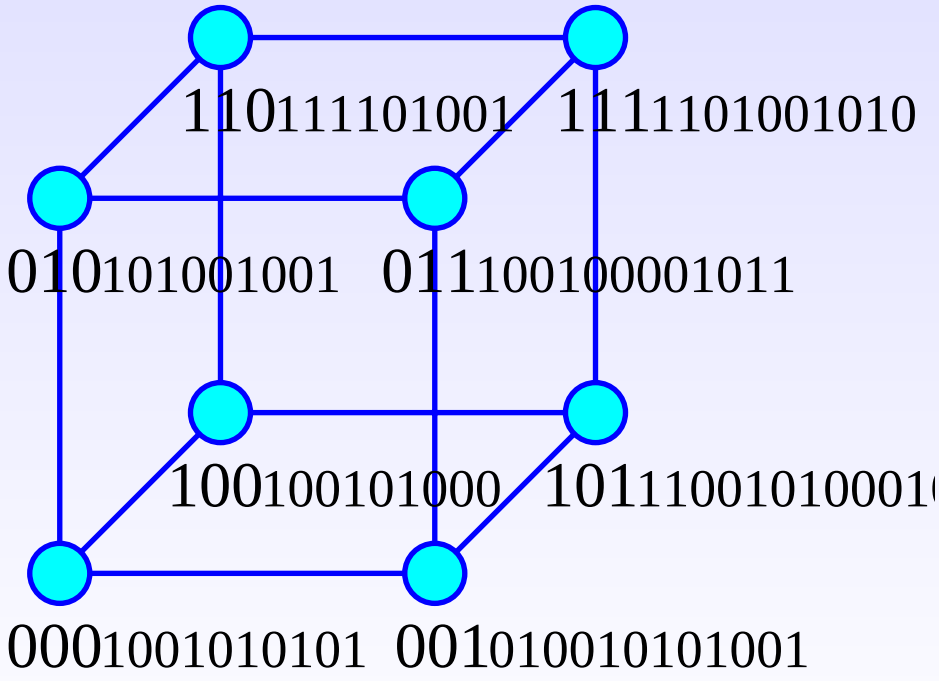
Idée déjà présente dans Pastry [2001] puis dans Kademlia[2002]

Exemple

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

Topologie : l'hypercube

Si v a la même valeur pour tout le monde :
le graphe des connaissances est l'hypercude de dimension v



Dimension du cube ?

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

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

Théoreme des anniversaires

Soit E un ensemble. La probabilité que après N tirages avec remise on ait au moins 2 éléments identiques est

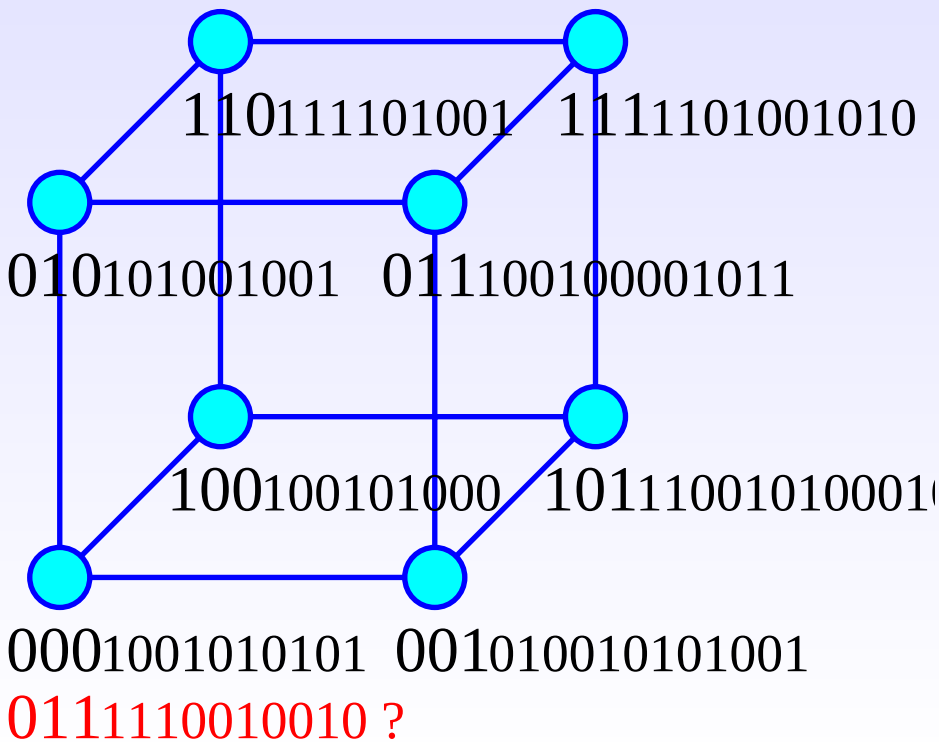
$$p(N) \approx 1 - e^{-\frac{N^2}{2 \cdot |E|}}$$

exemple : E=$\{1...365\}$ tirage = date d'anniversaire

Prenons $N = 2^{v_{moy}}$ pairs et $|E| = 2^{v_{max}}$ préfixes $\neq$. Pour avoir $p(N) = 1/2$ on a $\frac{N^2}{2.2^{|E|}} = \ln 2$ soit $2^{2v_{moy}} = \ln 2.2^{v_{max}+1}$ donc $2v_{moy} = v_{max} + 1 + \log \ln 2$

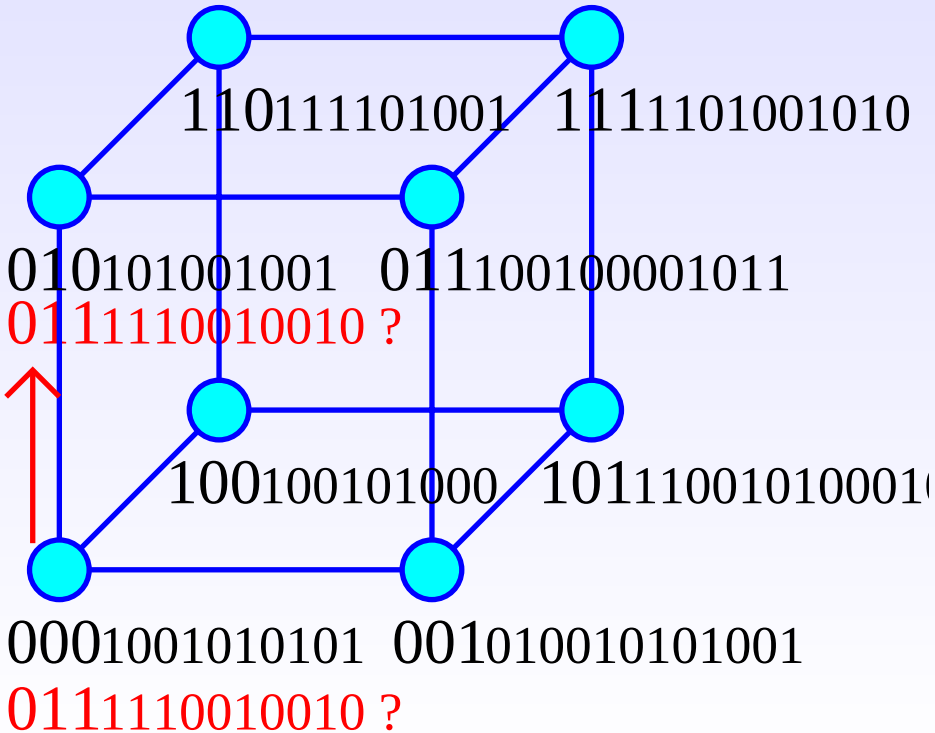
Algorithme de recherche

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



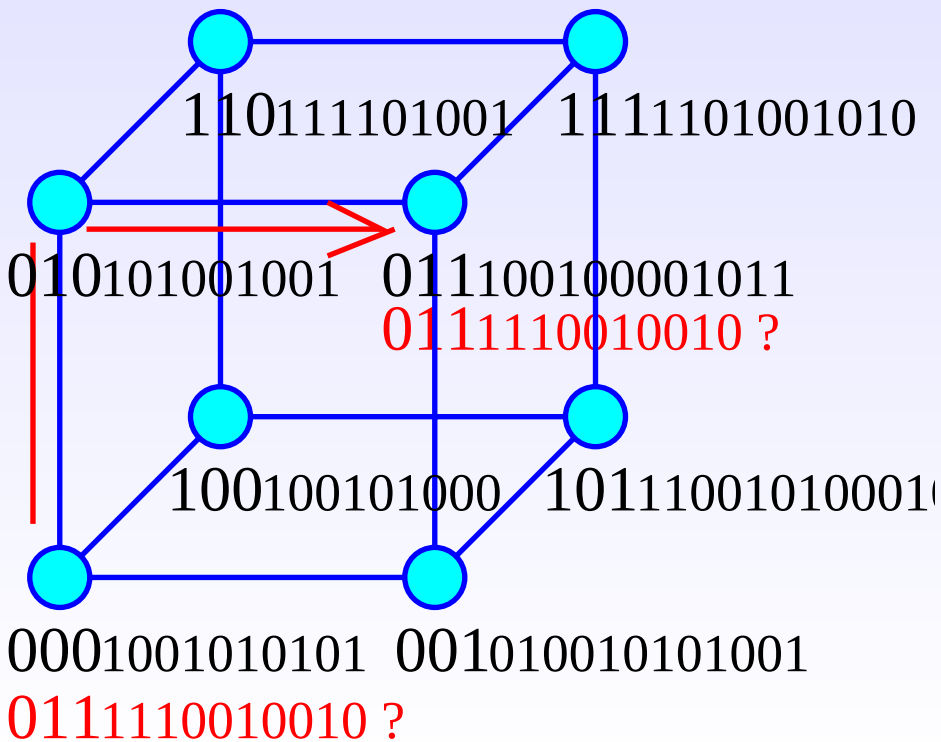
Algorithme de recherche

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



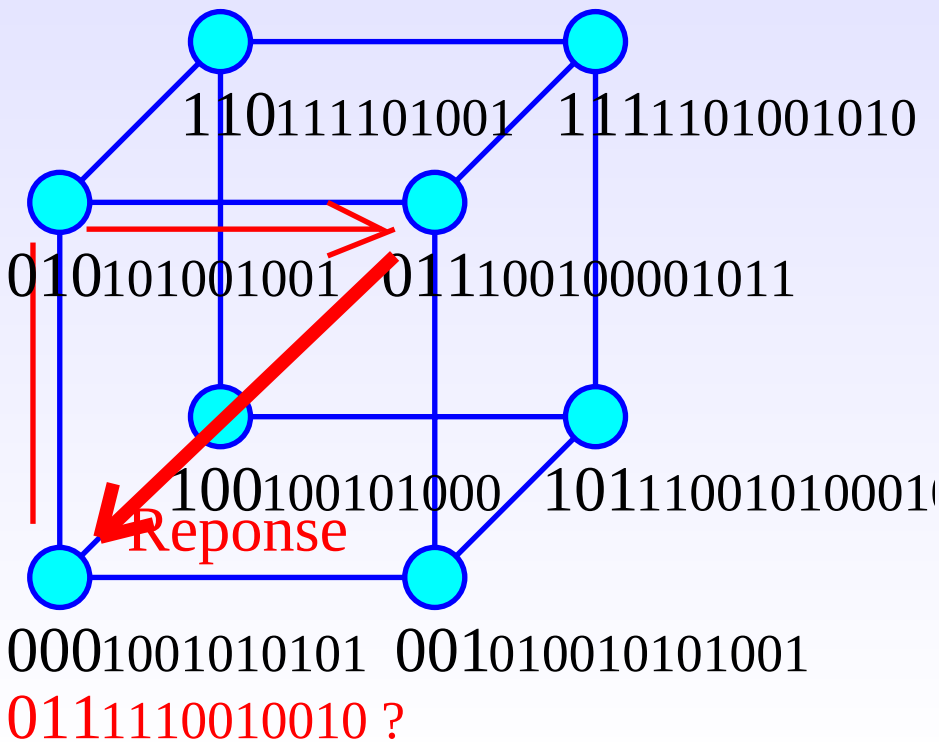
Algorithme de recherche

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



Algorithme de recherche

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



Algorithme de recherche

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

donc :

pour N pairs

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

Améliorations de la vitesse de recherche

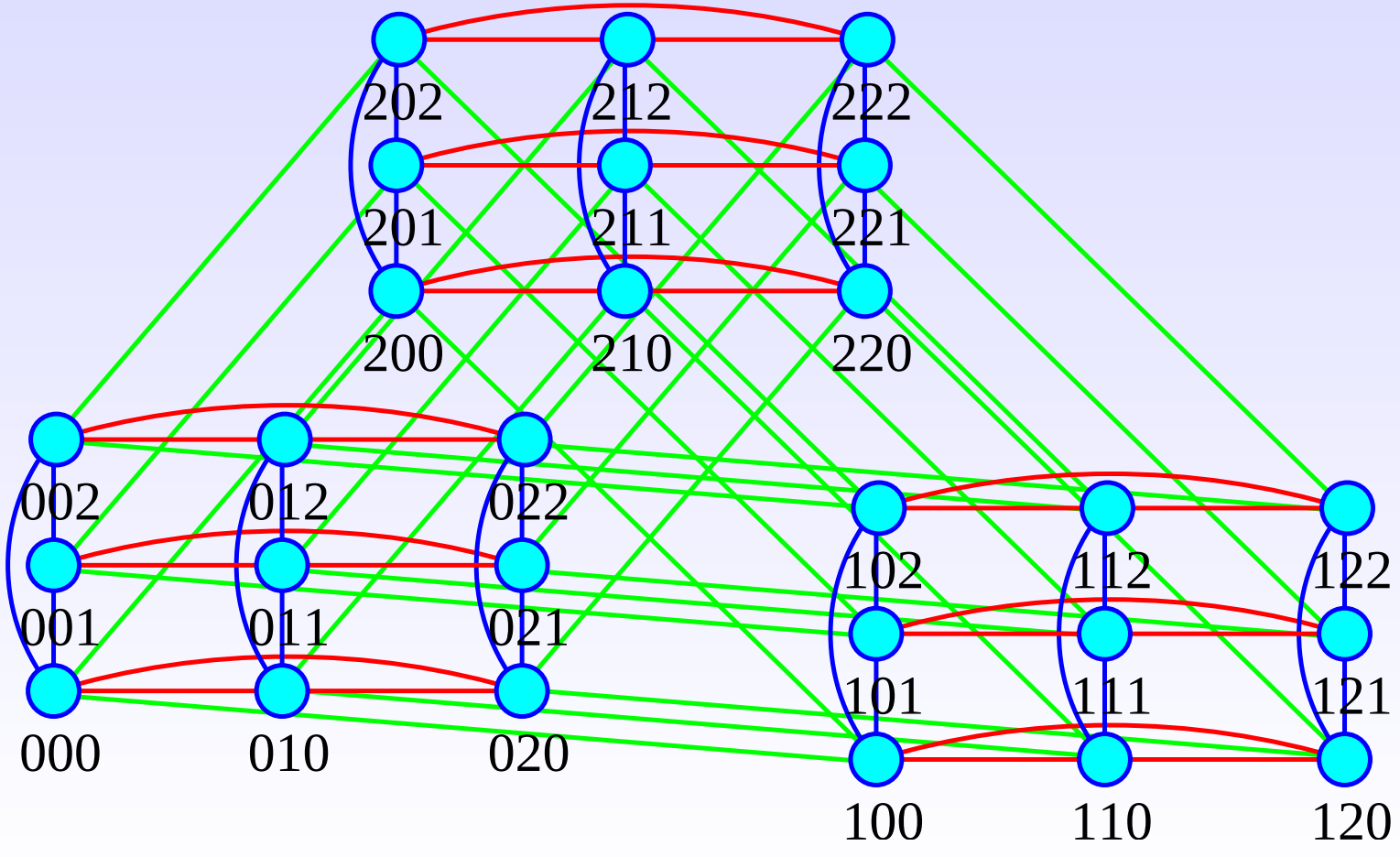
Base plus grande

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

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

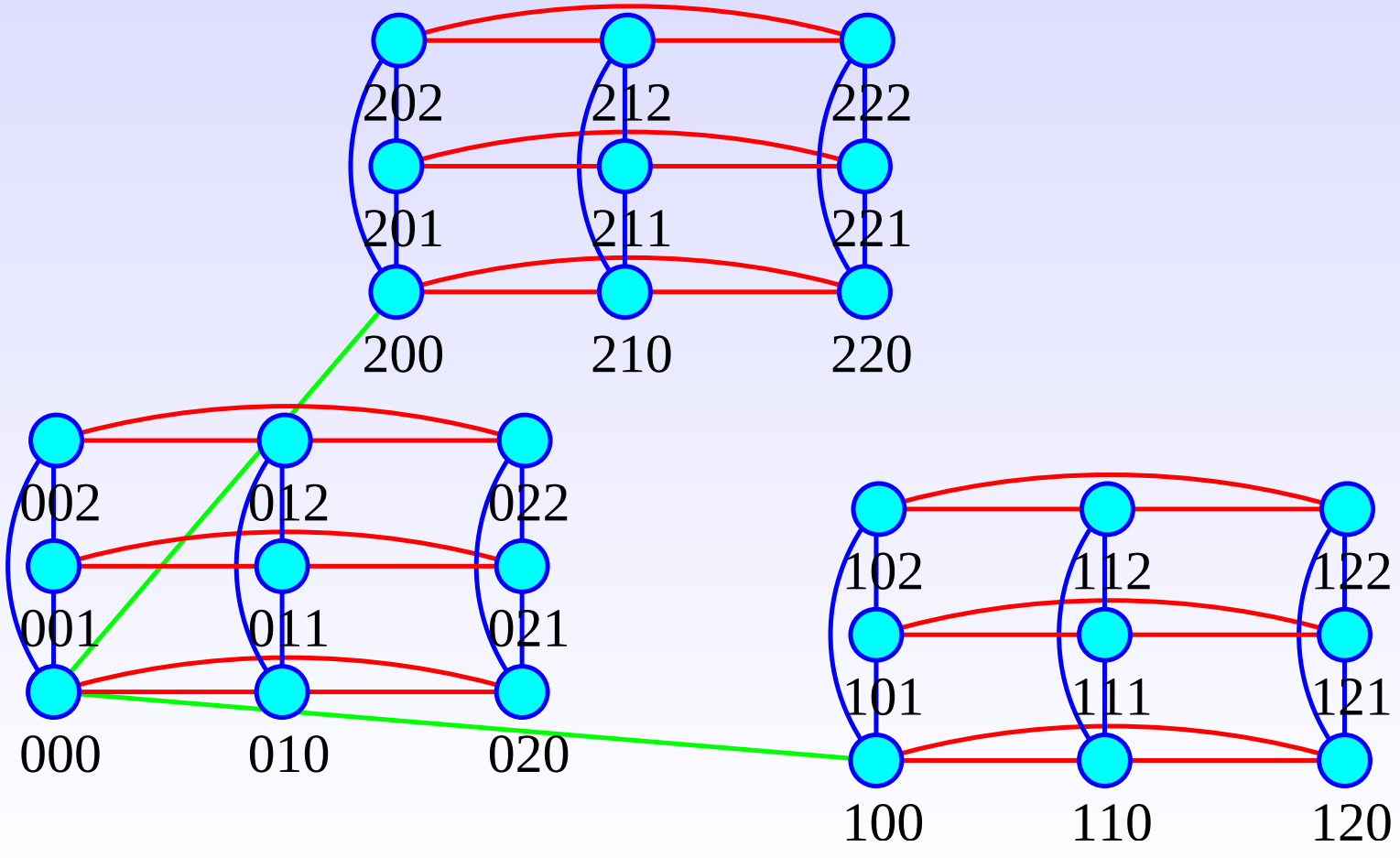
Améliorations de la vitesse de recherche

En base 3, dimension 3 :



Améliorations de la vitesse de recherche

En base 3, dimension 3 :



Mieux que $\log N$?

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

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

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

Améliorations de la robustesse

Nombre de voisins

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

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

Publication multiple

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

Taille des tables

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

Insertion dans le réseau ?

Algorithme d'insertion

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

Bootstrap

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

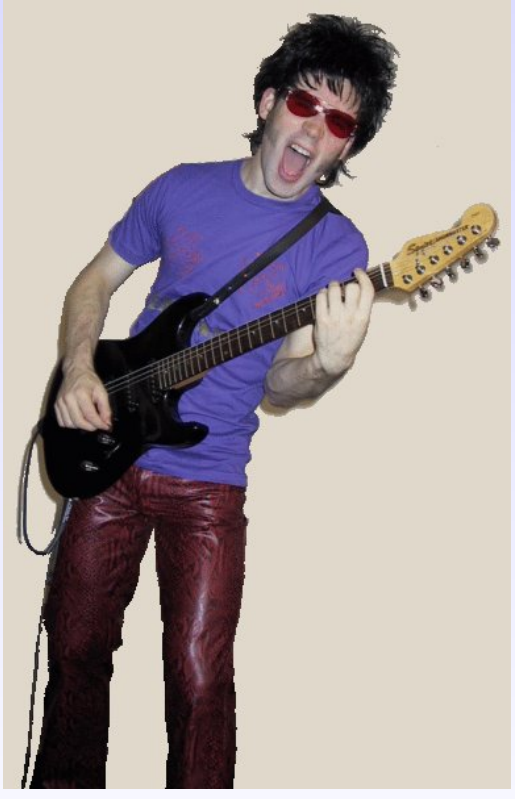
Analyse

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

Kademlia [Maymounkov Mazières 2002]

Implémentation : 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 $A + B \geq A \oplus B$)
- **unidirectionelle** : La sphere de centre A et de rayon r contient **un seul** point $B = A \oplus r$

Un tiroir (*bucket*)

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

Table de routage

Table de routage

Chaque pair possède 160 tiroirs. Ceux de numéro $\geq \log N$ sont vide en général
Contenu d'une entrée : triplet (IP, port UDP, hashcode du pair)

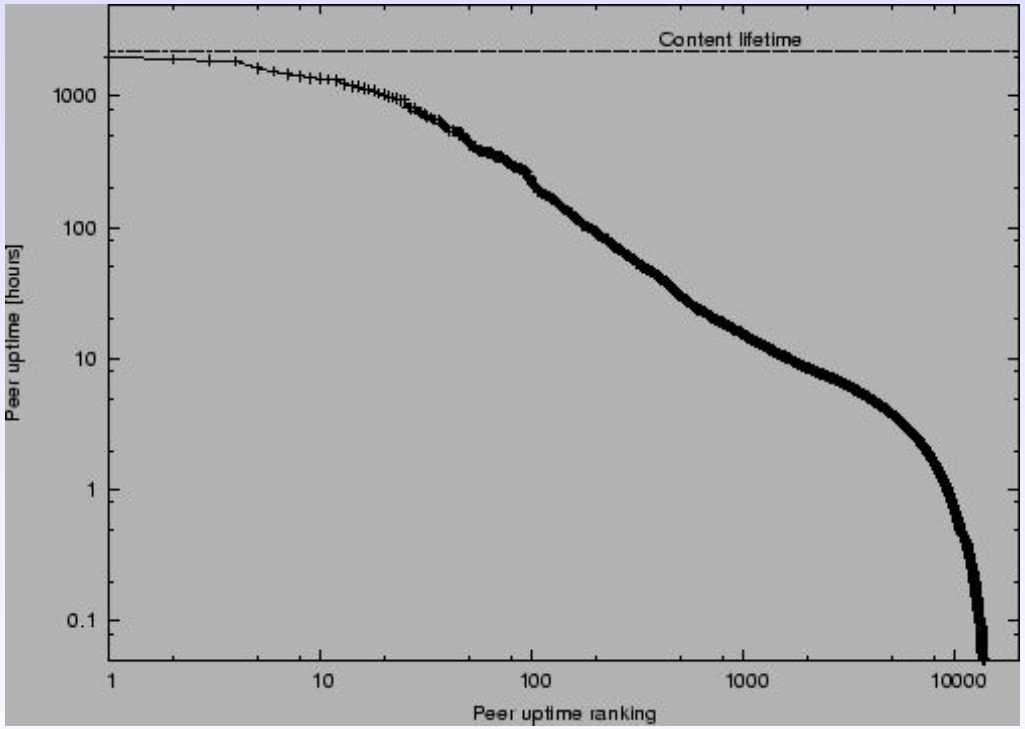
Maintenance du tiroir : LRU amélioré

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

Pourquoi garder les anciens pairs ?

Loi de décroissance

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



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

Protocole

Paquets : 4 types (UDP)

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

Requêtes en parallèle

- Première tentative : 3 parmi les 20 du tiroir, 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

Preuve

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 seul algo de routage et non 2 comme pastry et cache plus efficace
- requêtes concurrentes (3 puis 17)
- utilise efficacement la distribution du temps de présence

Conclusion sur DHT

Topologies existantes

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

Conclusion sur DHT

Avantages des DHT

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

Inconvénients

- table de hachage seulement $\rightarrow$ pas de requêtes complexes !
- plus lent qu'un serveur
- protocoles plus complexes mais bugs plus durs à corriger

Problématiques de diffusion de contenu

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

Une borne inférieure

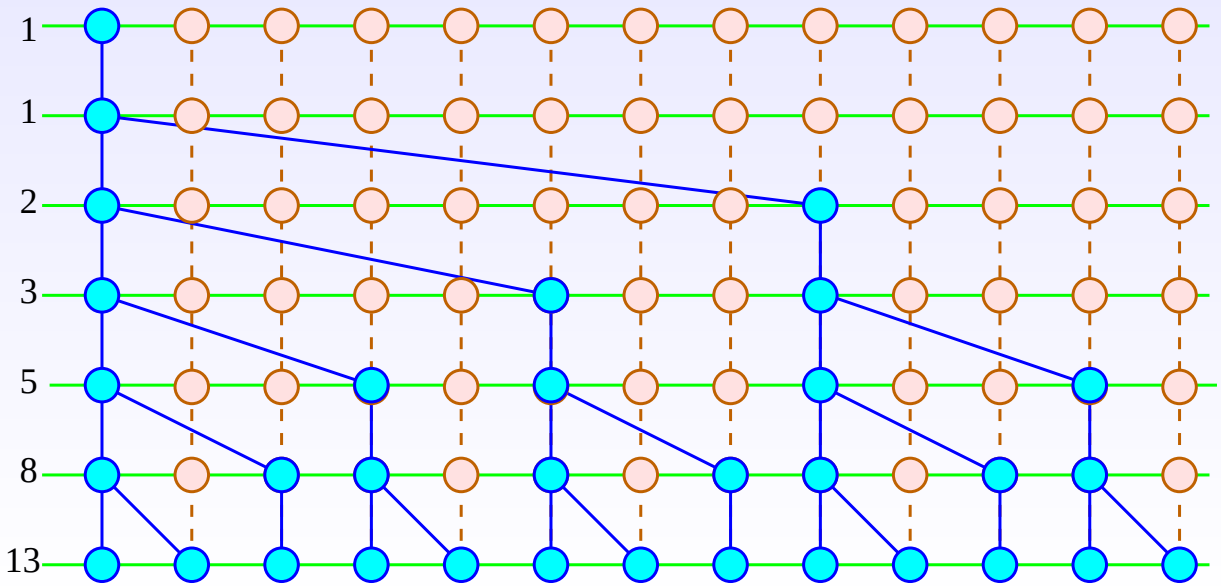
Prenons un modèle simple

- Tous les pairs se connaissent
- tous doivent avoir un bloc de taille 1
- Un seul l'a au début
- Bande passante : 1 / unité de temps

Quel temps faut-il ?

Arbre de Fibonacci

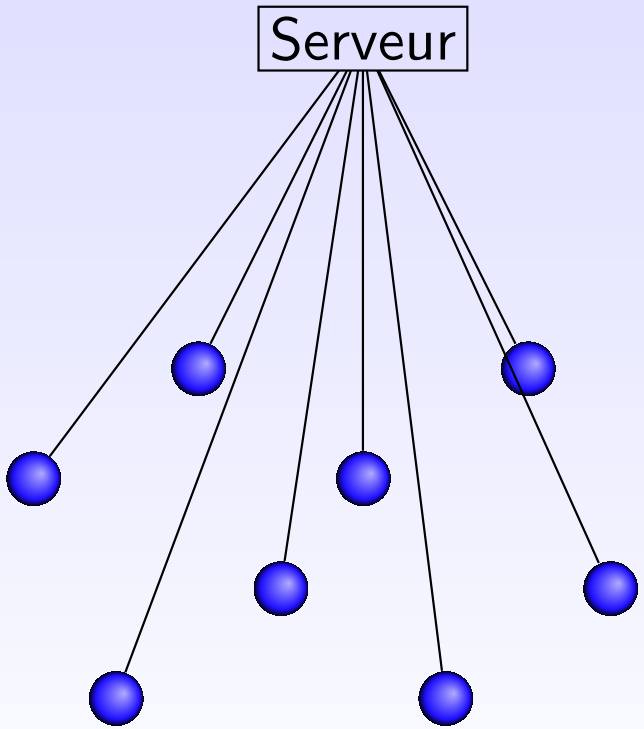
- Il faut une unité de temps pour être prêt à émettre
- Il faut une unité de temps pour émettre un bloc
- Totalement coopératif : tous participent
- Tous ont la même borne passante
- Borne inférieure au temps de diffusion : $\Phi_n \simeq \phi^n$ avec $\phi = \frac{1+\sqrt{5}}{2}$ et n rounds d'émission $\rightarrow n = O(\log N)$



Streaming : problématique

Pourquoi du streaming P2P ?

Un diffuseur de contenu (chaîne de télé.) diffuse à plusieurs pairs

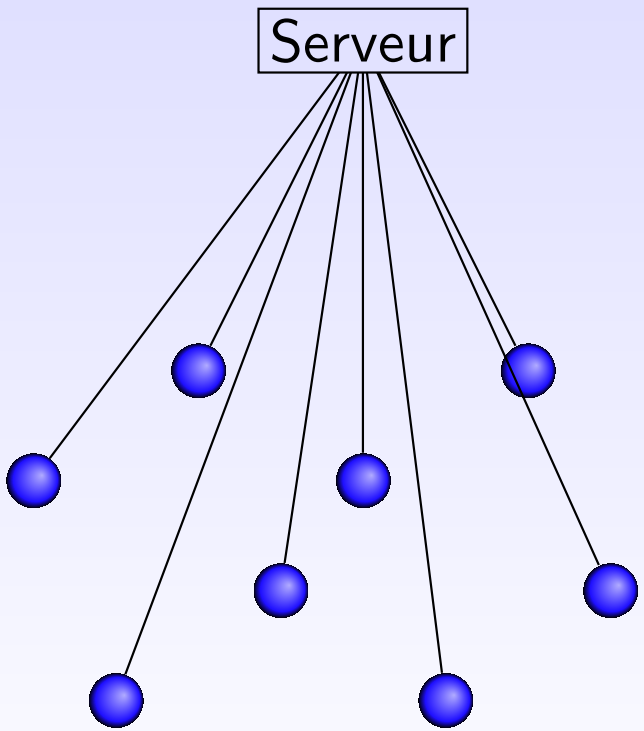


$\frac{k}{n}$ par client

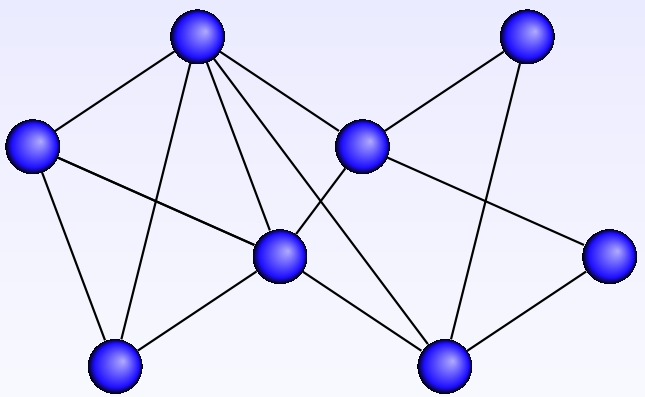
Streaming : problématique

Pourquoi du streaming P2P ?

Un diffuseur de contenu (chaîne de télé.) diffuse à plusieurs pairs



$\frac{k}{n}$ par client

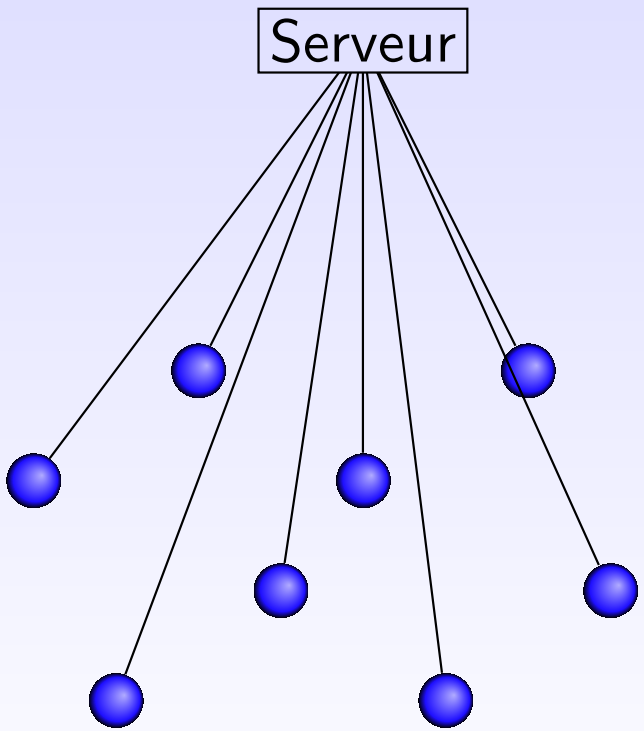


$\frac{n}{n} = 1$ par client

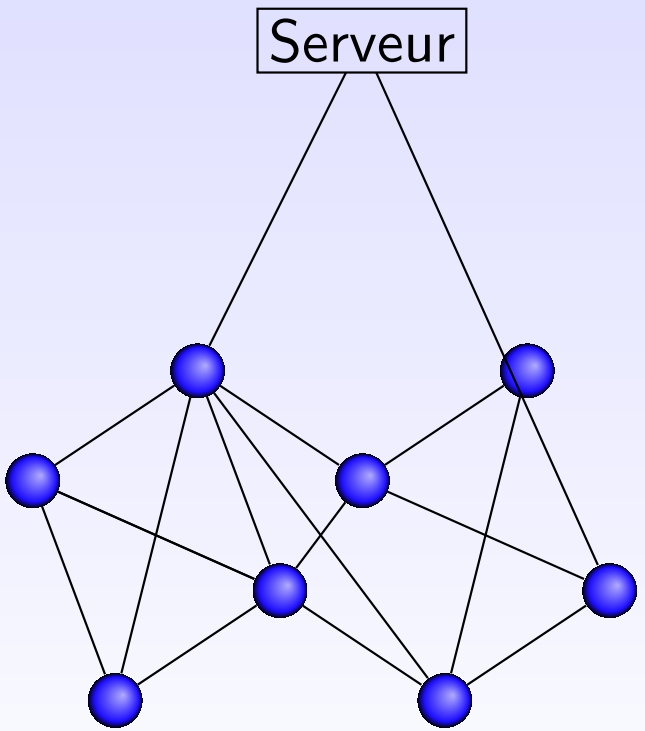
Streaming : problématique

Pourquoi du streaming P2P ?

Un diffuseur de contenu (chaîne de télé.) diffuse à plusieurs pairs



$\frac{k}{n}$ par client



$\frac{n+k}{n} = 1 + \frac{k}{n}$ par client

Streaming : problématique

Minimiser le délai entre l'émission d'un bloc et sa réception

- Temps légèrement différé : quelques secondes au plus (football...)
- Donc pas beaucoup de relais
- Mais alors, problème de bande passante !

Maximiser la qualité

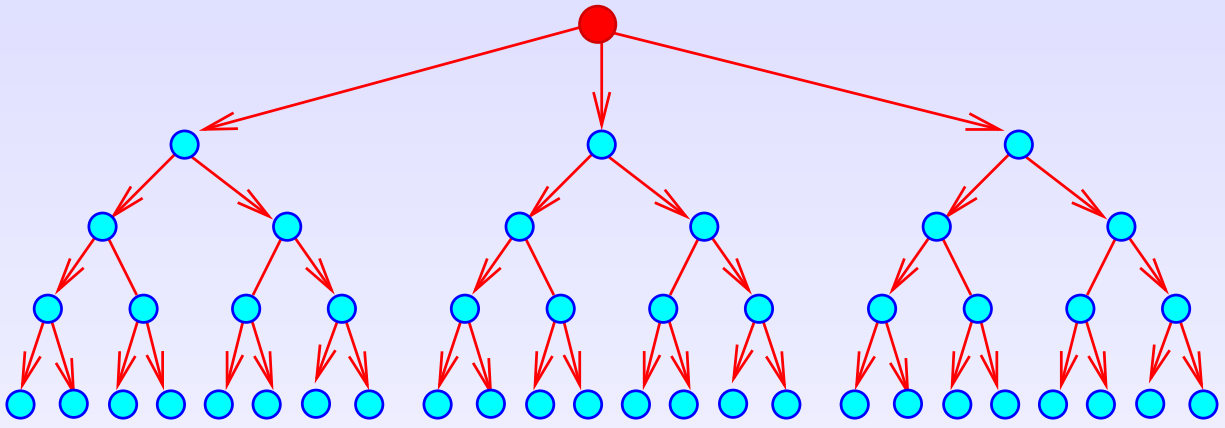
En cas de données manquantes

- Redemander ? Prend du temps
- Ignorer ? Dégrade l'image (exemple : TNT)

Éviter l'enregistrement pirate ?

Protocoles propriétaires cryptés...

Une bonne idée : l'arbre binaire



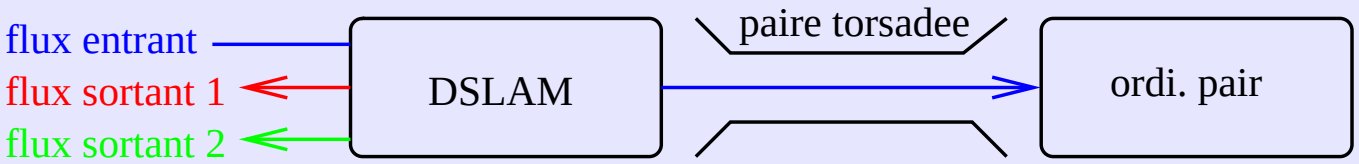
Avantage

Délai court : log N

Remarque sur l'usage des câbles

vrai Multicast

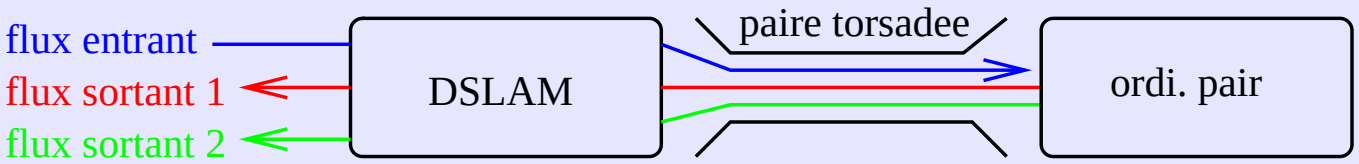
Le routeur reçoit un flux et en renvoie k (implémenté dans IP).
Exemple : télé par ADSL



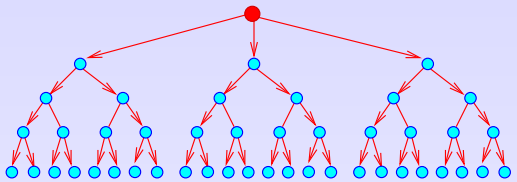
Remarque importante : faisable seulement par les FAI !

Multicast P2P

C'est le pair qui doit s'en charger



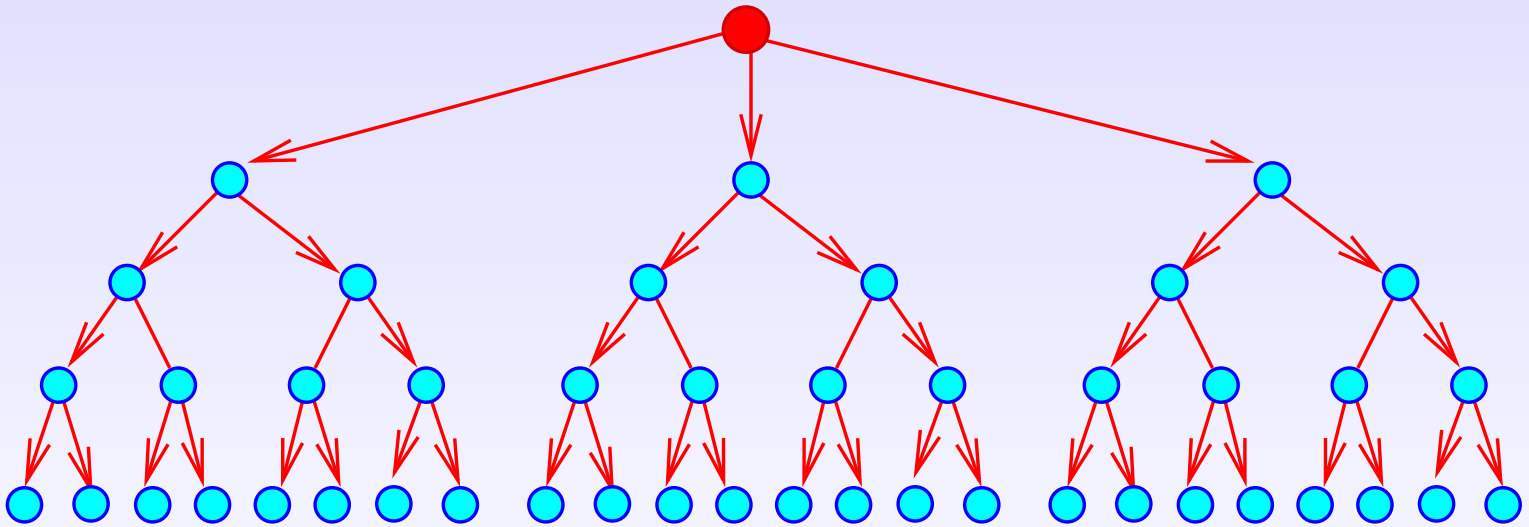
Une bonne idée : l'arbre binaire



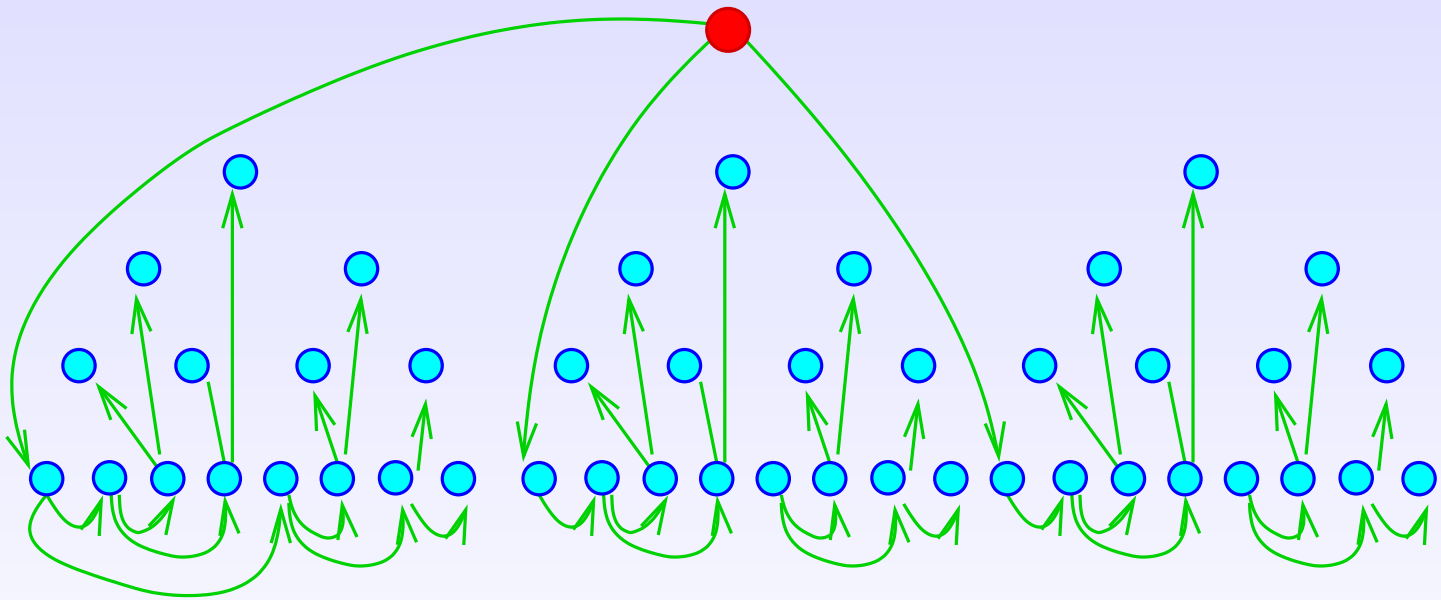
Inconvénients

- déconnection racine = déconnection de dizaines de clients !
- gérer la hiérachie
- bande passante des feuilles inutilisée !
- au contraire chaque nœud interne émet 2 fois plus
- Si flux TCP : protocole inadapté au temps réel (peut tout bloquer pour 1 paquet perdu)
- free riders $\rightarrow$ feuilles

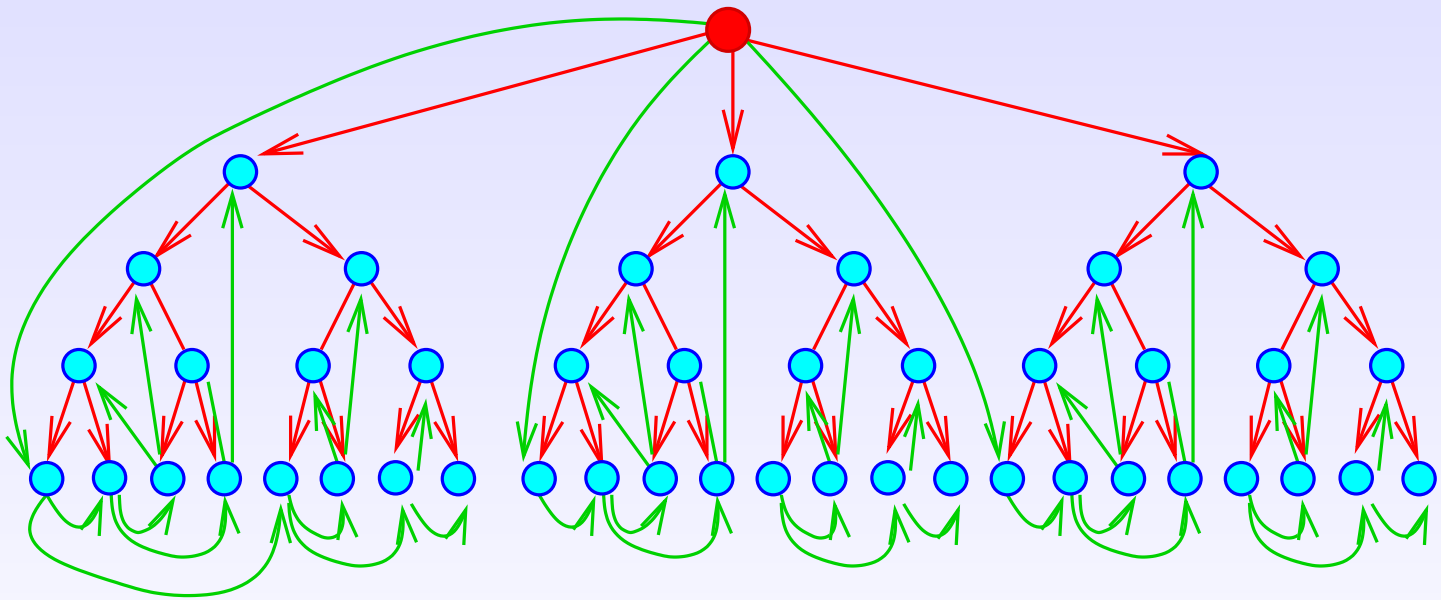
Meilleure idée : **deux** arbres binaires !



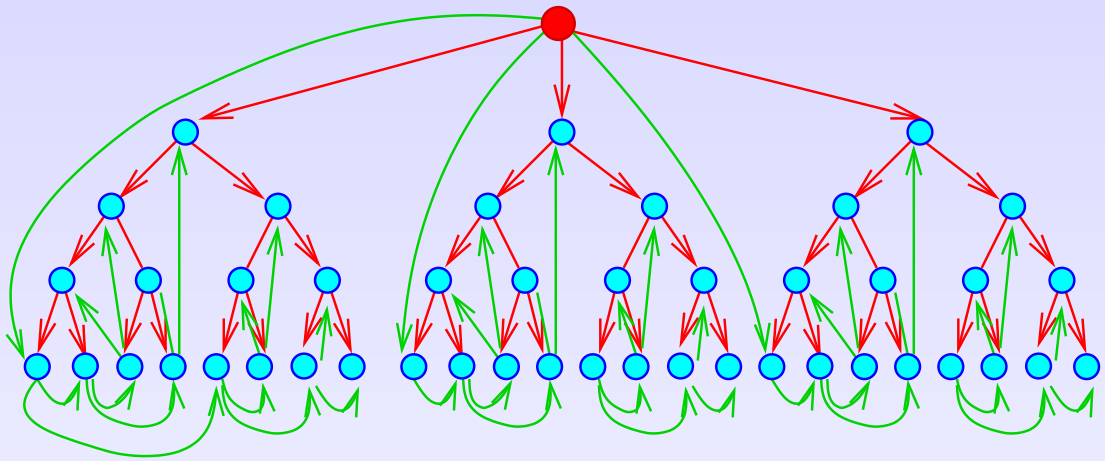
Meilleure idée : **deux** arbres binaires !



Meilleure idée : **deux** arbres binaires !



Meilleure idée : **deux** arbres binaires !



Comparaison

- Délai aussi court : log N
- Plus de bande passante gaspillée : émission = réception chez tout le monde (ou presque)
- mais toujours : TCP, free riders, déconnections et gestion de la topologie

Mieux que TCP ?

force et faiblesse de TCP

Produire un flux **sans erreur** $\rightarrow$ réemissions $\rightarrow$ délais

Alternative UDP

Redondance : Envoyer $1 + \epsilon$ fois plus.
Exemple : envoyer A, B, C, D, et $A \oplus B \oplus C \oplus D$

Mieux que TCP ?

force et faiblesse de TCP

Produire un flux **sans erreur** $\rightarrow$ réémissions $\rightarrow$ délais

Alternative UDP

Redondance : Envoyer $1 + \epsilon$ fois plus.
Exemple : envoyer A, B, C, D, et $A \oplus B \oplus C \oplus D$
Perte de B ?

Mieux que TCP ?

force et faiblesse de TCP

Produire un flux **sans erreur** $\rightarrow$ réemissions $\rightarrow$ délais

Alternative UDP

Redondance : Envoyer $1 + \epsilon$ fois plus.
Exemple : envoyer A, B, C, D, et $A \oplus B \oplus C \oplus D$
Perte de B ? $A \oplus C \oplus D \oplus (A \oplus B \oplus C \oplus D) = B$! !
Ici redondance : 20%. Mieux que XOR : Reed-Solomon,...

Mieux que TCP ?

force et faiblesse de TCP

Produire un flux **sans erreur** $\rightarrow$ réemissions $\rightarrow$ délais

Alternative UDP

Redondance : Envoyer $1 + \epsilon$ fois plus.
Exemple : envoyer A, B, C, D, et $A \oplus B \oplus C \oplus D$
Perte de B ? $A \oplus C \oplus D \oplus (A \oplus B \oplus C \oplus D) = B$! !
Ici redondance : 20%. Mieux que XOR : Reed-Solomon,...

Multiplexage temporel : Stripes

- Couper le flux en k sous-flux : stripes
- La réception de r stripes suffit pour recevoir
- Si moins que r : dégradé mais possible

Splitstream [Castro & al 2004]

Hypothèse

Les pairs n'ont pas tous le même upload

- Beaucoup $\rightarrow$ fort degré
- Peu $\rightarrow$ faible degré

Architecture

- k Stripes
- donc k arbres de diffusion : un par stripe
- On aimerait bien :
  - qu'un sommet soit nœud interne dans un arbre seulement (et donc feuille dans les $k-1$ autres)
  - Donc ratio feuilles / nœuds internes $= k-1$
    $\rightarrow$ arbres k-aires complets, *idéalement*

Stratigraphie de Splitstream

Pastry

Comme on a vu : c'est une DHT (topologie : anneau plus raccourcis, comme Chord)

Scribe

Maintient un arbre de diffusion grâce à Pastry

Splitstream

A première vue, semblable à k fois Scribe pour maintenir k arbres.

Architecture de Splitstream

- Chaque stripe reçoit un hash → numéros "uniformément" répartis dans $[0, ..2^{128}[$
- Chaque pair aussi a un hash
- Un pair va transmettre *prioritairement* la stripe de hash la plus proche du sien
- Le responsable de la stripe est le pair le plus proche (indexation DHT normale)
- Chaque pair doit se connecter à k autres
  (pour avoir toutes les stripes) k : paramètre global
- Chaque père accepte f fils et leur redistribue ses stripes
  f dépend de la capacité d'upload

Protocoles de connection

Du côté du père

- Accepte jusqu'à f fils
- Si trop de fils : rejette ceux à qui il ne transmet pas sa stripe prioritaire
- Si encore trop : rejette les plus loin de lui (en distance de hash)

Du côté d'un fils

- Doit se connecter à un père par stripe
- Essai par parcours en profondeur
- On part du père de la stripe, qui est connu grâce a la DHT
- Si rejet : un de ses fils
- etc etc (parcours en profondeur)

Analyse

Avantages

- Preuve (probabiliste) que la transmission se fait (si $f \geq k$)
- Délai court
- Bande passante bien répartie : émission = réception (optimal)

Inconvénients

- Problème si $f < k$ en moyenne : plus assez de nœuds internes, trop de feuilles $\rightarrow$ effondrement
- Maintenir cette toplogie compliquée : dur
- Toujours le problème des free riders ($f = 0$)

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 !

Propagation épidémique [Sanghavi, Hajek, Massoulié, 2006]

Propagation épidémique = inondation

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

Efficacité

En termes de délai, la stratégie 3 est à un facteur constant de l'optimal !

Inconvénient

Il est trop facile d'être free-rider !

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 !

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'emettre 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 déconnections font perdre du temps : moins intérêt qu'en BitTorrent

Arbre de clusters : Prefixstream [Gai Viennot 2003]

Topologie hybride

- Localement : non-structuré (cluster de plusieurs nœuds)
- Globalement : les clusters forment des arbres nœuds-internes-disjoints

Protocole

- Dans un cluster : diffusion
- Entre les clusters : suit la topologie
- Incitations à l'émission (échanges bidirectionnels)
  $\rightarrow$ plus de free riders
- Basé sur le graphe de De Bruijn

Conclusion

Beaucoup d'applications

- partage de fichiers personnels
- radio, télévision
- Sauvergarde coopérative
- Petites annonces, jeux, VoD etc etc...

Et aussi de domaine de recherche

- DHT (recherches plus puissantes, robustesse...)
- Anonymat
- Incitations à la coopération (à la BitTorrent)
- Protocoles etc etc