Plan

Problème à résoudre

- Stocker de l'information
- Trouver de l'information

$\Rightarrow$

- Utiliser l'ensemble des participants comme serveurs de stockage distribué
- Utiliser une partie des participants comme répertoire de nommage

Chaque nœud est à la fois client et serveur

Domaine d'application

Partage

- informations/fichiers
- ressources de calcul (grid computing)
- ressources de stockage (réplication)
- bande passante (CDN *Content delivery network*)
- interactions (jeux massivement multijoueur)

Réseaux de recouvrement

Réseau de recouvrement ou *overlay*

Réseau logique, virtuel, au-dessus d'un réseau physique existant

- Couche applicative
  - Réimplantation du routage
  - Ajout de fonctionnalité : nommage, stockage
- Pairs : réseau formé par les participants, tous (à peu près) égaux

Difficultés (1)

Maintenance du réseau de recouvrement

- Démarrage
  - Création du réseau ? (premier site : cas particulier)
  - Insertion/retrait d'un pair dans le réseau
- Maintenance continue
  - Faute, retrait involontaire, expulsion
- Terminaison : arrêt du réseau ?

Passage à l'échelle

- Éviter un (ou quelques) serveurs centralisés
- Distribuer la charge sur les pairs
- Borner la charge sur chaque pair (CPU, bande passante, stockage)

Difficultés (2)

Équité

- Équilibrer la charge : égalitairement ? proportionnellement ?
- Utilisateurs égoïstes : contrôles et incitations à l'équité

Défaillances

- Maintenance de l'overlay à tout prix
- Défaillances de pairs, de liens de communication
- Partition temporaire du réseau, réinsertion ?

Difficultés (3)

Adaptabilité

- Ajout/retrait de sites par vagues (heures ouvrables)
- Ajout/retrait d'informations parfois massif (plusieurs milliers d'un coup)

Performance

- Efficacité : localisation, accès
- Latence du réseau (localité d'accès)
- Parallélisation

Classification : contrôle

- Contrôle centralisé : un serveur central met en correspondance les pairs
  - \+ simple
  - – fragile
  - – faible capacité de croissance
- Contrôle totalement décentralisé : tous les nœuds jouent un rôle symétrique (client et serveur)
  - \+ pas de point central, confidentialité
  - \+ disponibilité élevée
  - – complexe, gestion hasardeuse
  - – performance indéterminée
- Contrôle partiellement décentralisé : un ensemble dynamique de nœuds jouent un rôle privilégié

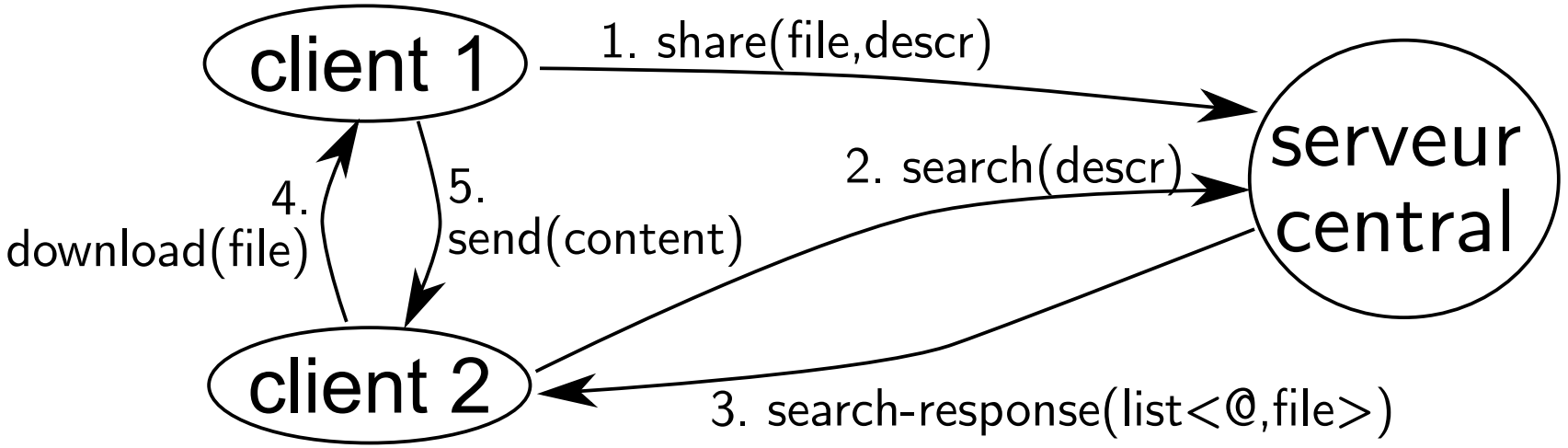
Classification : réseau virtuel

- Non structuré : le placement des données n'est pas lié à la topologie du réseau
  - \+ bonne adaptabilité avec un ensemble de nœuds très dynamique
  - − recherche inefficace en absence de contrôle centralisé
- Structuré : les données sont placées en des points prédéterminées $\Rightarrow$ recherche déterministe
  - \+ recherche rapide, en temps borné
  - − ensemble de nœuds dynamique ?
- Faiblement structuré : partiellement déterministe

Classification

	Réseau virtuel		
	non structuré	faiblement structuré	structuré
contrôle centralisé	Napster		
partiellement décentralisé	eMule, FastTrack, Gnutella		
totalement décentralisé	BitTorrent	Freenet	Chord, Pastry

Contrôle centralisé – Napster simplifié ♪♪♪



- Un serveur central : l'annuaire
- Des clients pairs : stockage

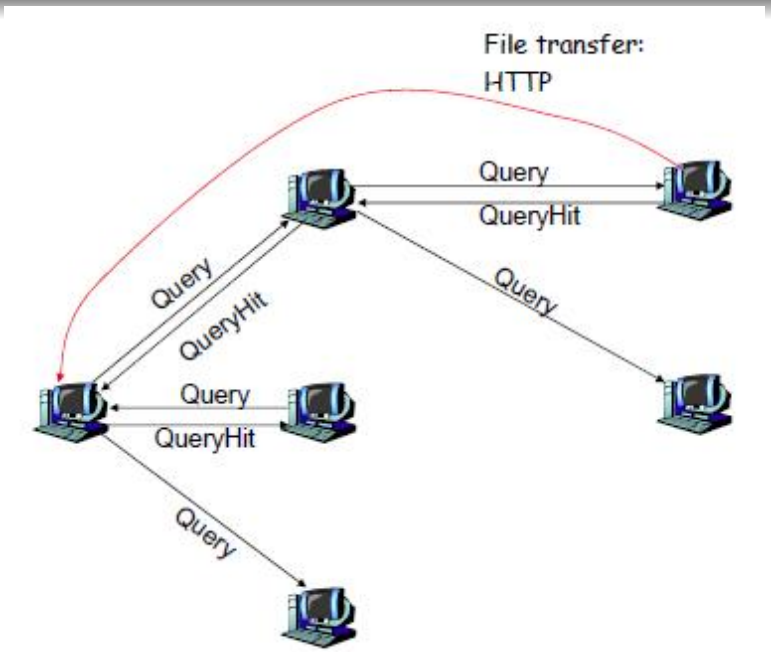
Contrôle décentralisé – Gnutella

Chaque nœud est client, serveur et routeur
Messages

- `ping` : pour découvrir des correspondants
  bootstrap : *gnutella caches* notoires puis par rebond
- `pong` : en réponse (+ informations : nb/taille des fichiers possédés)
- `query` : recherche (mots clefs)
- `query-hit` : en réponse (@ IP, id fichiers)

1. *On the Long-term Evolution of the Two-Tier Gnutella Overlay*, Amir Rasti, Daniel Stutzbach and Reza Rejaie. 25th IEEE Int'l Conf. on Computer Communications. April 2006.

Gnutella : recherche



- Recherche par inondation : requête `query` transmise de voisin en voisin
- Id unique : éviter les retransmissions en boucle
- nombre de retransmissions (hops) limité

Améliorations :

- Envoi aléatoire, effectué en parallèle
- Distinction entre nœuds feuilles (connectés à 2 ou 3 ultranœuds) et ultranœuds (puissants, fortement interconnectés) $\Rightarrow$ nombre réduit de hops

(source : Original uploader was ACNS at en.wikipedia – Commons CC BY-SA 3.0)

Table de hachage répartie (DHT)

Table de hachage répartie = *distributed hash table*

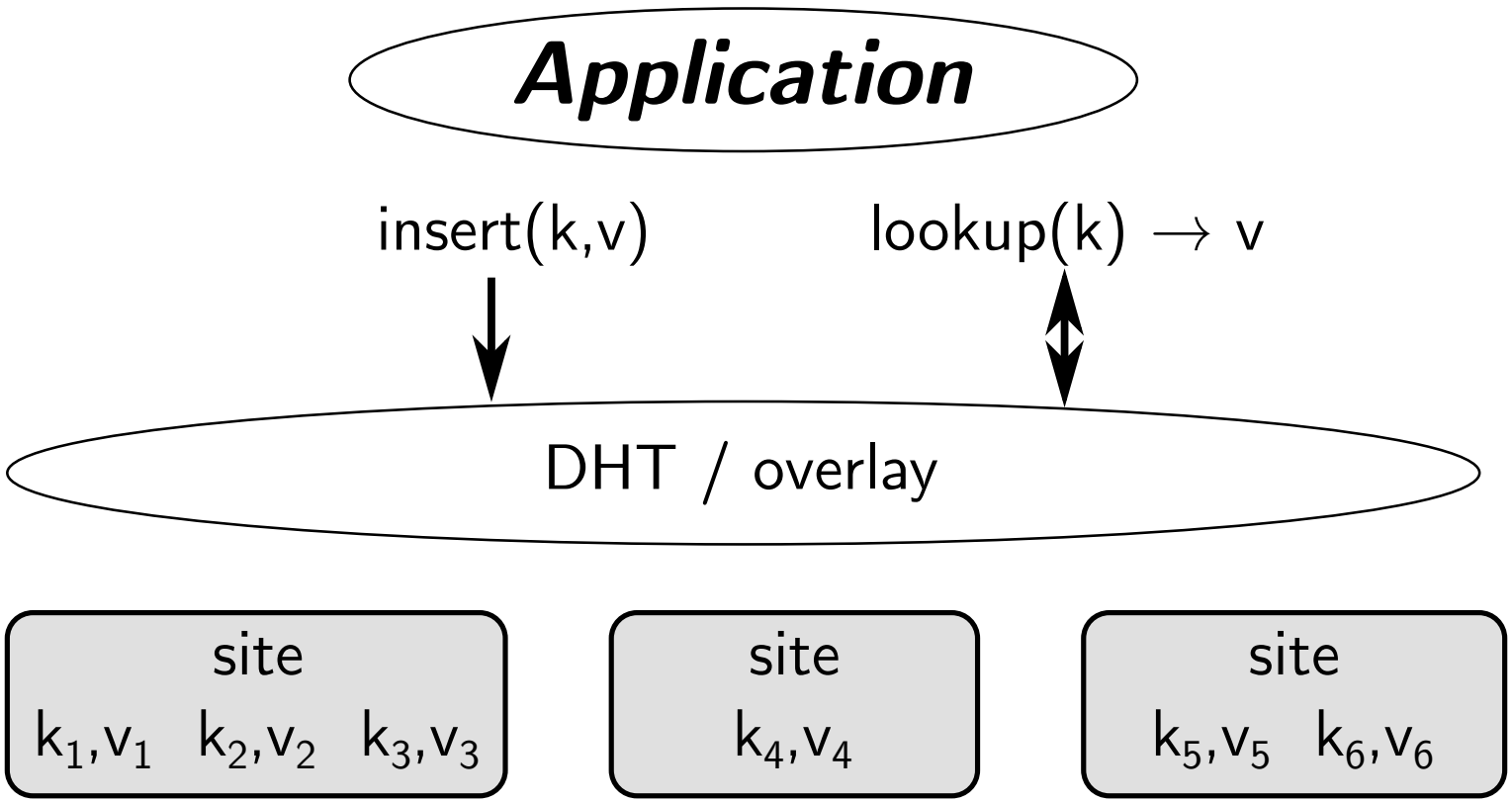


Table de hachage répartie (DHT)

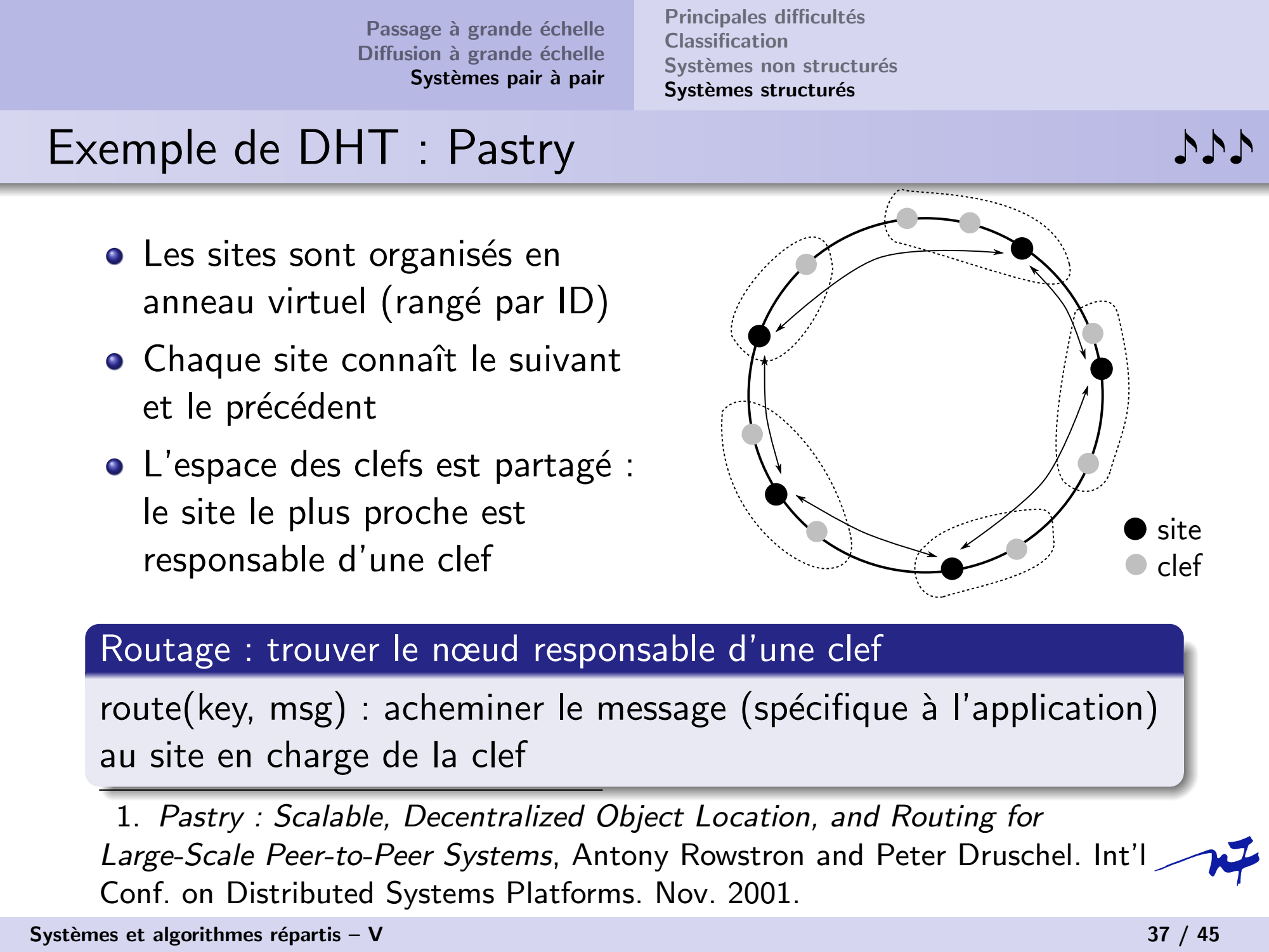
L'infrastructure P2P établit le lien entre clef et site :

- Chaque site possède une ID : p.e. hachage de son adresse IP
- Chaque objet possède une clef et une valeur
- La clef d'un objet est p.e. le hachage de sa valeur, ou de son nom, ou de sa description...
- Chaque site est responsable d'une partie de l'espace de hachage, p.e. des clefs qui sont proches de son ID

$\Rightarrow$

- Pas de connaissance globale centralisée
- Pas de point unique de défaillance
- Passage à l'échelle
- Répartition de la responsabilité

Exemple de DHT : Pastry



- Les sites sont organisés en anneau virtuel (rangé par ID)
- Chaque site connaît le suivant et le précédent
- L'espace des clefs est partagé : le site le plus proche est responsable d'une clef

Routage : trouver le nœud responsable d'une clef

route(key, msg) : acheminer le message (spécifique à l'application) au site en charge de la clef

1. *Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, Antony Rowstron and Peter Druschel. Int'l Conf. on Distributed Systems Platforms. Nov. 2001.

Pastry : routage

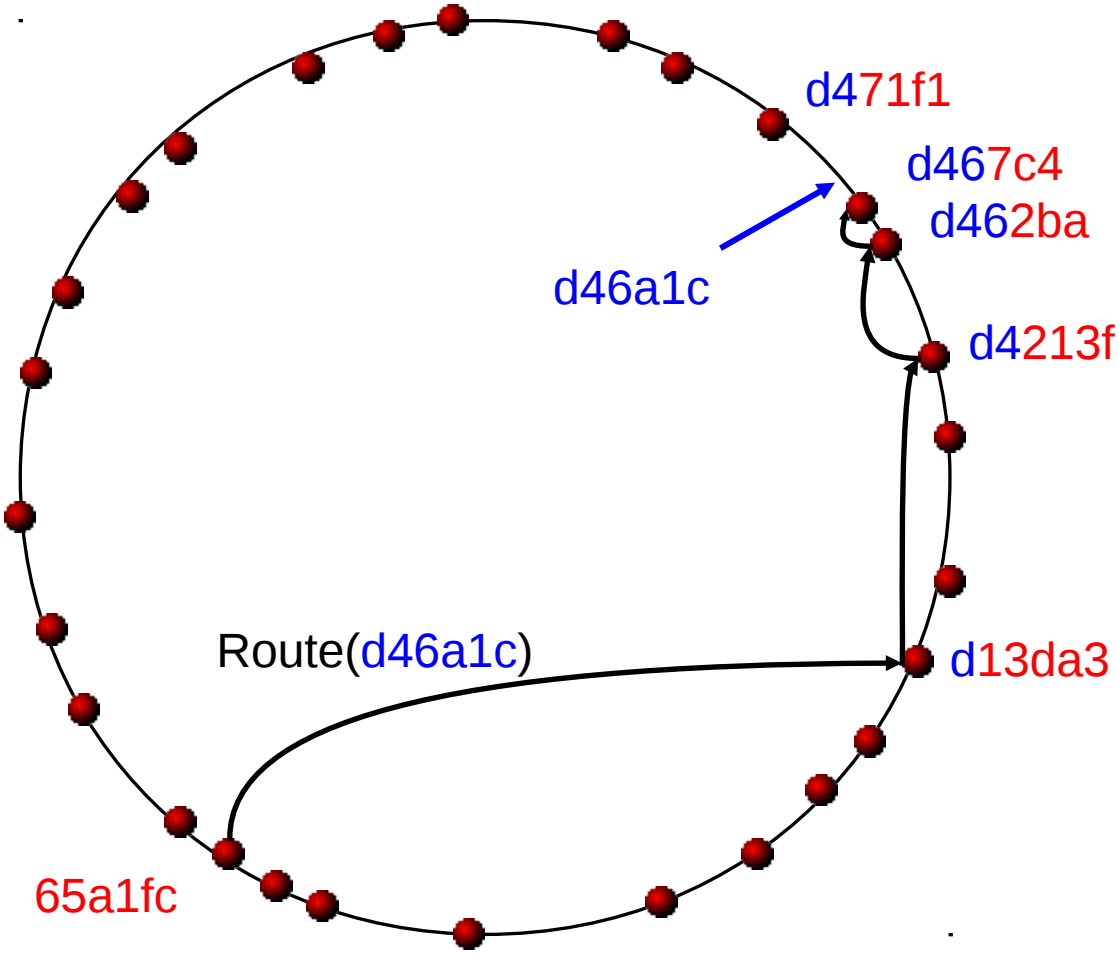
Routage en suivant les liens suivants/précédents inefficace ⇒ table de routage

- N sites ($N = 16$ millions)
- Chaque site possède une ID : P digits sur B valeurs (B $= 16$, $P = \lceil \log_B N \rceil$). ex : 65a1fc
- Chaque site a une table de routage à P lignes, B colonnes
- La case (i, j) est l'adresse d'un site ayant les i premiers digits identiques au site, et j en $i + 1$ digit.
  ex : sur le site 65a1fc, ligne 3, col 4 = site 65a4xx

*L*0	0	1	2	3	4	5	6	7	8	9	*a*	*b*	*c*	*d*	*e*	*f*
*L*1	60	61	62	63	64	65	66	67	68	69	6*a*	6*b*	6*c*	6*d*	6*e*	6*f*
*L*2	650	651	652	653	654	665	656	657	658	659	65*a*	65*b*	65*c*	65*d*	65*e*	65*f*
*L*3	65*a*0	65*a*1	65*a*2	65*a*3	65*a*4	65*a*5	65*a*6	65*a*7	65*a*8	65*a*9	65*aa*	65*ab*	65*ac*	65*ad*	65*ae*	65*af*
...																

- Trouver une clef = chercher le premier digit distinct et transmettre à ce site, qui poursuit le routage

Pastry : Routage



Dessin : Peter Druschel

- max $\log_{16} N$ hops
- table de routage (par site) : $O(\log N)$

Pastry : voisinage (*Leaf sets*)

Chaque site maintient les $L/2$ plus proches sites en deçà et en delà (où L est un paramètre, valant généralement 2 ou 4)

- efficacité du routage
- résistance du routage
- détection de fautes (ping périodique)

Pastry : algorithme de routage

```
if (destination ∈ voisinage) then
    transmettre le message au membre concerné
else
    let l ≜ longueur du préfix commun entre ce site
                                        et la destination
    let d ≜ valeur du l-ième digit
    if routage[l,d] est défini et répond then
        transmettre le message à routage[l,d]
    else
        transmettre à un site qui
            - a au moins un préfixe commun de taille l
            - est numériquement plus proche que ce site
```

Pastry : jonction

1. Le site d'id i veut s'insérer
2. Il envoie un message *join* à n'importe quel site
3. Ce message est routé comme précédemment au site j, actuellement responsable de i. Le site i sera inséré entre j et k (précédent ou suivant de j selon la valeur de i)
4. Le site j transmet à i son voisinage pour que i construise son voisinage initial
5. La table de routage de i est établie à partir des tables de j et k
6. Le site i interroge tous les sites de son voisinage initial :
   - i établit son voisinage définitif en gardant les $L/2$ plus proches dans chaque sens
   - Les sites du voisinage apprennent i et mettent à jour leurs propres table de routage et voisinage
7. Le site i devient actif

Seule difficulté : 2 insertions simultanées entre 2 même nœuds !

Pastry : départ (défaillance)

Les membres d'un voisinage s'échangent périodiquement des messages de vie.
Absence de réponse sur un message de vie ou un message de routage $\Rightarrow$ considéré défaillant, enlevé du voisinage et de la table de routage si présent.

- Réparation du voisinage : augmenter son voisinage en interrogeant le site le plus loin de son voisinage actuel
- Réparation du routage : obtenir la table des sites sur la même rangée que le site supprimé, puis en remontant

Pastry : bilan

Points positifs

- Passe à l'échelle, résistant à $L/2$ fautes simultanées (et en pratique bien plus), non centralisé (auto-organisation)
- Routage efficace : $O(\log N)$ hops en situation normale, $O(N)$ en cas pire (improbable : tables de routage détruites)
- Information par site modeste : $O(\log N)$ pour le routage
- Partage équitable de la responsabilité et du stockage

Améliorations

- Routage : prendre en compte la distance (en temps) pour établir la table de routage
- Réplication : un même objet clef/valeur est répliqué sur plusieurs nœuds voisins
- Sécurité : hachage non inversible avec peu de collision (SHA)

Conclusion

- Systèmes non structurés : performant si contrôle assez centralisé $\Rightarrow$ capacité de croissance ? anonymat ?
- Systèmes structurés : dynamicité ?
- Contrôle décentralisé $\Rightarrow$ qualité de service ?
  - sécurité, confiance ?
  - disponibilité non garantie (mais plutôt bonne)
  - site parasite ?
- Aucun standard, même architecturellement