



Autour des graphes et du routage

Laurent Viennot

► To cite this version:

| Laurent Viennot. Autour des graphes et du routage. Algorithme et structure de données [cs.DS].
| Université Paris-Diderot - Paris VII, 2005. tel-00471731

HAL Id: tel-00471731

<https://tel.archives-ouvertes.fr/tel-00471731>

Submitted on 9 Apr 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Autour des graphes et du routage

Laurent Viennot

Autour des graphes et du routage

Novembre 2005.

Thèse présentée par Laurent Viennot à l'université Paris 7 pour obtenir l'habilitation à diriger des recherches, rapportée par Vincent Blondel, Michel Habib, Laurent Massoulié et Stéphane Ubéda, écrite sous le parrainage de François Baccelli, soutenue avec la complicité de Pierre Fraigniaud et Philippe Jacquet, et préparée à l'Inria.

Table des matières

1	Introduction	7
2	Modélisation	9
2.1	Routage	9
2.2	Internet	10
2.3	Réseaux ad hoc	13
2.4	Graphe du Web	18
2.5	Réseaux de pair à pair	22
2.6	Exploration de réseau	29
3	De l'un vers tous	33
3.1	Inondation	33
3.2	Inondation optimisée par multpoints relais	34
3.3	Inondation optimisée par ensemble dominant	40
3.4	Diffusion de pair à pair	42
4	De l'un vers tel autre	53
4.1	En connaissant le graphe (protocoles classiques)	53
4.2	En connaissant un sous-graphe	55
4.2.1	Routage ad hoc par états de liens	55
4.2.2	Routage petit monde	58
4.3	Sans rien connaître du graphe (routage réactif)	59
5	De l'un vers celui qui	67
5.1	Routage dans Internet	67
5.2	Routage pair à pair	68
5.3	Gnutella et inondation optimisée par filtre de Bloom	68
5.4	Tables de hachage distribuées	71

6 De temps à autre	79
6.1 Routage par vecteur de distances et Bellman-Ford asynchrone	80
6.2 Itération asynchrones	81
6.3 Synchronisation des nœuds d'un réseau	83
6.4 PageRank et routage aléatoire	84
6.5 PageRank distribué	85
6.6 PageRank incrémental	88
6.7 Itérations asynchrones par colonnes	88
6.8 HITS incrémental	91
7 Quand ça bouge	97
7.1 Surcoût de contrôle	97
7.2 Connexité temporelle	98
7.3 Stabiliser le dynamisme	99
7.4 Équilibrer le dynamisme	101
8 Conclusion	105

*au robinet
et à la GX,*

Remerciements

Je voudrais remercier mes lecteurs et en particulier mes rapporteurs car ce document ne serait rien qu'une partition muette sans un œil bienveillant pour le mettre en musique.

Je voudrais remercier mes collègues expérimentés qui m'ont parrainé à divers moments.

Je voudrais remercier mes jeunes collègues qui se sont engagés en thèse avec moi avant même qu'un diplôme n'atteste de ma capacité à les accompagner.

Je voudrais remercier ceux qui recherchent leur nom ici, cette marque d'affection me va droit au cœur¹.

Enfin, je voudrais adresser un clin d'œil tout particulier à certaines personnes par quelques petites énigmes² :

- 0 Un lapin, un éléphant et une girafe sont sur un bateau. Un grand animal tombe à l'eau et un seul animal à grandes oreilles reste sur le bateau. Qui est tombé ?
- 10 Qui est le meneur d'un rebouteux, d'un amiral et d'un demi-euro ?
- 1 Quel est le style architectural des bouches du métro parisien ?
- 20 Alice et Bob communiquent. Qui pointe l'autre du doigt ?
- 11 Qui a tué Jeremy O'Malley ?
- 2 À qui s'adresse telle ou telle petite énigme ci-dessus ?
- 30 À qui s'adresse l'énigme précédente ?
- 21 Qu'est-ce qu'il y a après quatre ?

¹Beaucoup de noms me sont venus à l'esprit en rédigeant ces lignes et la peur d'en oublier un, ainsi qu'une certaine pudeur, me poussent à cette nomination implicite.

²Les petites énigmes sont pour moi des sortes de confiseries, j'espère que les intéressés les apprécieront aussi.

Chapitre 1

Introduction

L’habilitation est un exercice de style imposé dans une carrière de chercheur. Cependant, comment ne pas aborder cette sorte de bilan sans un sentiment mêlé d’ambition, de recherche de légitimité, d’humilité et d’angoisse devant l’apparente incohérence d’une pile d’articles accumulés au fil des années.

Le premier défi consiste donc à trouver un fil conducteur dans le chemin sinueux de quelques années de recherche. Il faut bien reconnaître qu’après deux inflexions de thématique de recherche, mon cas ressemblait à une sorte de quadrature du cercle en la matière. Pour résoudre cette équation, je décidai de me rallier à un vieux précepte qui consiste à résoudre une version simplifiée du problème. Je ne présenterai donc qu’une partie de mes travaux, ce qui me permet de centrer avec ferveur mon propos sur les réseaux et son problème central, celui du routage. Bien sûr, le terme de routage doit être pris dans un sens assez large sans quoi je ne pourrais pas présenter une nécessaire diversité.

Évidemment, il est prétentieux de penser qu’une fraction seule de ses travaux suffit à atteindre la masse critique nécessaire à une habilitation, mais comment passer cette épreuve sans une certaine dose d’égocentrisme aveugle. Par ailleurs, on reconnaîtra sans doute la noblesse du but qui consiste à essayer de rendre un tel document plus intéressant que la pile mentionnée ci-dessus. Pour aller dans ce sens, j’ai tenté d’organiser la suite dans une sorte d’inventaire d’astuces algorithmiques des réseaux et du routage.

Je laisse donc de côté mes activités concernant uniquement l’algorithmique des graphes et qui font le gros de ma culture scientifique. J’espère qu’on sentira néanmoins leur présence en filigrane. Ayant déjà rédigé dans ma jeunesse un document sur quelques points de l’algorithmique des graphes,

j'espère que l'on pardonnera cette omission.

Le chapitre 2 retrace rapidement la problématique du routage, notamment dans l'Internet (qui servira d'exemple introductif dans la plupart des chapitres), les réseaux ad hoc, le graphe du web et les réseaux de pair à pair. Le chapitre 3 est consacré au routage « de l'un vers tous », c'est à dire au problème de diffusion d'un message à tous les membres d'un réseau. Le chapitre 4 traite du routage dans son sens le plus classique, c'est-à-dire quand il s'agit d'envoyer un message « d'un nœud vers tel autre ». Nous considérerons ensuite le problème plus pratique qui consiste à envoyer un message vers un nœud défini de manière indirecte, ce que j'ai appelé « de l'un vers celui qui ». Les deux derniers chapitres s'intéressent enfin à la dynamique des réseaux concernant l'algorithme distribuée entre les nœuds ou le réseau lui-même. Le chapitre 6 décrit ainsi une classe très générale d'algorithmes de réseau à base d'itérations asynchrones qui fonctionne dès que des messages envoyés régulièrement sont reçus « de temps à autre ». Le chapitre 7 développe ensuite quelques points liés à l'aspect dynamique de certains réseaux : « quand ça bouge » tant du point de vue des connexions entre nœuds que de la présence des nœuds.

Chapitre 2

Modélisation

L'utilisation des graphes pour les réseaux commence nécessairement par une phase de modélisation dont l'importance est à rappeler. Avant de voir différents types de réseaux et comment les graphes interviennent dans leur modélisation, nous allons rapidement rappeler en quoi consiste le routage.

La question du routage est intimement liée à celle de l'adressage, c'est-à-dire comment on identifie la destination. Cette ambivalence se retrouvera souvent au fil de ce mémoire.

2.1 Routage

Nous considérons ici le routage par *paquet* dans lequel chaque paquet de données acheminé contient un *en-tête* indiquant entre autres quelle est la destination du paquet. Un *routeur* est une sorte d'aiguilleur qui possède des *liens* avec d'autres routeurs. Chaque lien est branché au routeur via une *interface*. La principale activité d'un routeur consiste à *router* des paquets :

Routage

1. Un paquet arrive sur une interface,
2. son *en-tête* est lu (et éventuellement modifiée),
3. il est retransmis sur une interface.

Le choix de l'interface de sortie dépend de l'en-tête du paquet. Pour faire ce choix, un routeur maintient à jour une *table de routage* qui contient pour une destination donnée le numéro d'interface où faire suivre le paquet. Un

protocole de routage spécifie les informations que s'échangent les routeurs pour pouvoir construire leurs tables de routage.

L'en-tête du paquet contient un identifiant de la destination du paquet que l'on appelle l'*adresse*. Pour une adresse donnée, chaque table de routage doit indiquer à qui faire suivre le paquet (en indiquant l'interface attachée au lien vers celui-ci et éventuellement l'adresse lien de celui-ci s'il y a plus de deux routeurs connectés au lien). Pour une destination donnée, cette relation entre routeur et *nœud suivant* (ou « *next hop* » en anglais) peut se représenter par un arc d'un routeur vers le suivant. Idéalement, l'ensemble de ces arcs forme un arbre de plus court chemin enraciné à la destination.

La bête noire du routage est la *boucle*, c'est-à-dire une incohérence dans les tables de routage qui fait qu'un paquet peut se mettre à faire une boucle. Si jamais cela arrive, les liens de la boucle peuvent vite être engorgés par les paquets qui vont se mettre à y circuler indéfiniment. Pour éviter qu'un tel dysfonctionnement ne devienne dramatique, les protocoles de routage prévoient généralement un champ TTL¹ dans l'en-tête des paquets. Le TTL est une sorte de temps de vie du paquet au bout duquel il disparaît du réseau.

TTL

1. Quand un paquet est reçu,
2. le champ TTL est décrémenté de 1.
3. Si le TTL atteint 0, le paquet est interdit de retransmission.

Ainsi, un paquet ne peut pas circuler indéfiniment dans le réseau.

2.2 Internet

À titre d'exemple introductif, nous allons reprendre le cas bien connu d'Internet, le réseau des réseaux, et considérer le problème classique du routage qui consiste à envoyer de proche en proche un paquet d'une machine source vers une machine destination. Le protocole IP² spécifie le format des paquets qui circulent dans Internet. Distinguons les routeurs des *hôtes* qui ne

¹Il serait vain de vouloir éviter les acronymes lorsqu'on traite des réseaux, aussi vais-je tenter de donner en bas de page la signification de tous ceux que je n'aurais pas réussi à éviter. TTL : « Time To Live »

² IP : « Internet Protocol »

participent en général que faiblement au routage. Les hôtes sont les machines des utilisateurs d'Internet, généralement reliés à un réseau local pour lequel une *passerelle* fait le pont avec Internet. La seule décision de routage prise par un hôte est d'envoyer un paquet directement à la destination si elle se trouve dans le réseau local ou à la passerelle sinon.

Un routeur appartient à un *réseau*. Tous les routeurs d'un réseau sont gérés par la même organisation et sont reliés entre eux de manière connexe. Certains des routeurs du réseau peuvent avoir des liens vers des routeurs d'autres réseaux, appelons-les des *routeurs de frontière* (pour « border gateway » en anglais). Le monde des destinations, vu d'un routeur Internet, se sépare donc en deux populations, celles qui sont accessibles sans sortir du réseau et celle qui sont en dehors du réseau.

Internet est constitué par un empilement hiérarchique de réseaux. Les réseaux du bas de la hiérarchie ne possèdent souvent qu'un seul routeur relié à un réseau de niveau supérieur. Tout paquet pour une destination hors de portée du réseau sera envoyé vers ce lien. On parle de route par défaut puisque les paquets sont envoyés par là si aucune information concernant la destination n'est trouvée dans la table. Elle ne contient en effet que des entrées concernant les destinations accessibles via le réseau. À l'inverse, le routeur auquel est connecté ce réseau doit connaître l'ensemble des destinations qui sont accessibles via ce réseau.

Les réseaux de plus haut niveau n'ont pas de route par défaut, on les appelle les systèmes autonomes (AS³) et ils constituent la nervure centrale (« backbone ») d'Internet⁴. Les réseaux de différentes organisations sont reliés entre eux au gré d'accords bilatéraux. Les deux organisations s'accordent sur les informations que s'échangent les routeurs et sur les conditions commerciales dans lesquelles ils s'échangent du trafic.

Les réseaux intermédiaires de la hiérarchie que nous appellerons *réseau d'accès* (ou « stub network » en anglais) peuvent aussi conclure de tels accords de « peering » à leur niveau. La figure 2.1 illustre ainsi schématiquement la structure d'Internet.

Pour s'intéresser au problème du routage dans Internet, il serait hâtif de modéliser cette structure par un simple graphe avec pour sommets les routeurs des systèmes autonomes et pour arêtes les liens entre ceux-ci. Cela reviendrait à oublier l'existence même des systèmes autonomes. De plus, le

³ AS : « Autonomous System »

⁴ Il y en a environ vingt mille à l'heure actuelle.

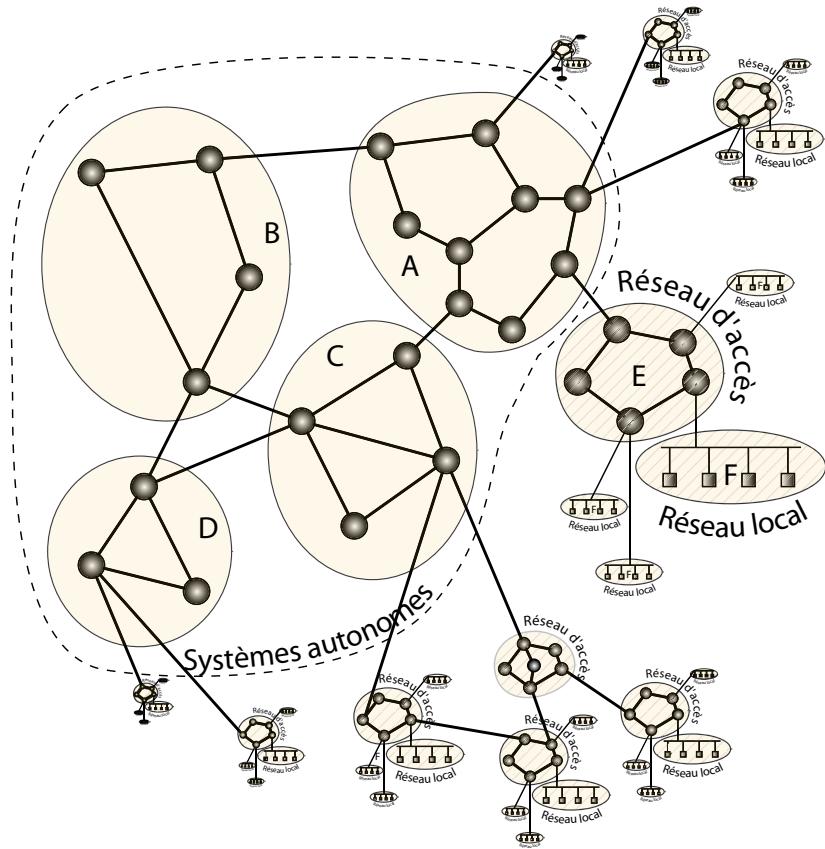


FIG. 2.1 – Internet

routage ne consiste pas à router vers un noeud donné mais vers la passerelle qui est en charge d'une adresse IP donnée. L'attribution des adresses IP constitue donc un deuxième point clé qui prend toute son importance dans le problème du routage.

Rassemblons ce dont nous disposons :

- chaque système autonome possède un numéro d'identification (AS ID⁵),
- chaque routeur est identifié par un numéro (Routeur ID) au sein de son système autonome,
- chaque interface d'un routeur (autrement dit l'extrémité d'un lien entre deux routeurs) possède une adresse IP,

⁵ ID : « IDentification number »

- chaque passerelle connaît l'ensemble des adresses IP dont elle a la charge.

À l'intérieur d'un système autonome, on se trouve dans le cadre idyllique de l'algorithme des graphes. Il suffit que chaque routeur connaisse l'ensemble des adresses IP gérées par tout autre routeur du système autonome pour que le problème du routage se modélise par un calcul de plus courts chemins. Il existe principalement deux solutions dans ce cadre :

- le routage par vecteur de distances (« distance vector » en anglais),
- le routage par état de liens (« link state » en anglais).

Ces algorithmes sont rappelés au chapitre 4. Précisons simplement ici comment l'ensemble des adresses gérées par une passerelle peut-être codé efficacement. Une adresse IP est constituée par n'importe quelle suite de 32 bits. L'attribution des adresses se fait en regroupant autant que faire se peut les adresses d'une même zone sous le même préfixe. Idéalement, toutes les adresses joignables via un routeur donné devraient pouvoir être identifiées par un même préfixe commun. En pratique, il s'agit de quelques-uns. Ce schéma est décrit plus en détail au chapitre 5.

2.3 Réseaux ad hoc

Dans un réseau radio, les connexions se font au gré de la transmission radio. Un signal est reçu s'il est K fois plus fort que le bruit ($K = 10$ typiquement). Les ondes radio s'atténuent avec la distance en $1/r^\alpha$ avec $\alpha = 2$ en espace libre. À bruit ambiant constant B un émetteur radio peut donc être reçu jusqu'à une distance $R = \left(\frac{P_e}{KB}\right)^{1/\alpha}$ où P_e est la puissance d'émission du signal. En espace homogène, l'ensemble des positions d'où l'on peut recevoir un signal constitue donc un disque centré sur l'émetteur. Avec des obstacles, cela devient une tâche de forme complexe.

Le modèle le plus simple de la connectivité d'un réseau ad hoc est ainsi celui du graphe de disques unitaires (« disk unit graph » en anglais) où deux nœuds peuvent communiquer s'ils sont à distance inférieure à R . Autrement dit, un disque unitaire est supposé centré sur chaque nœud et deux d'entre eux sont reliés si leur disque s'intersectent (les distances doivent être normalisées de sorte que $R = 2$). Les figures 2.2 et 2.3 illustrent un tel graphe avec des positions de nœuds tirées aléatoirement. Des raffinements de ce modèle sont possibles : le graphe de boules unitaires où l'on plonge les nœuds dans l'espace plutôt que dans le plan, ou encore supposer des puissances d'émission

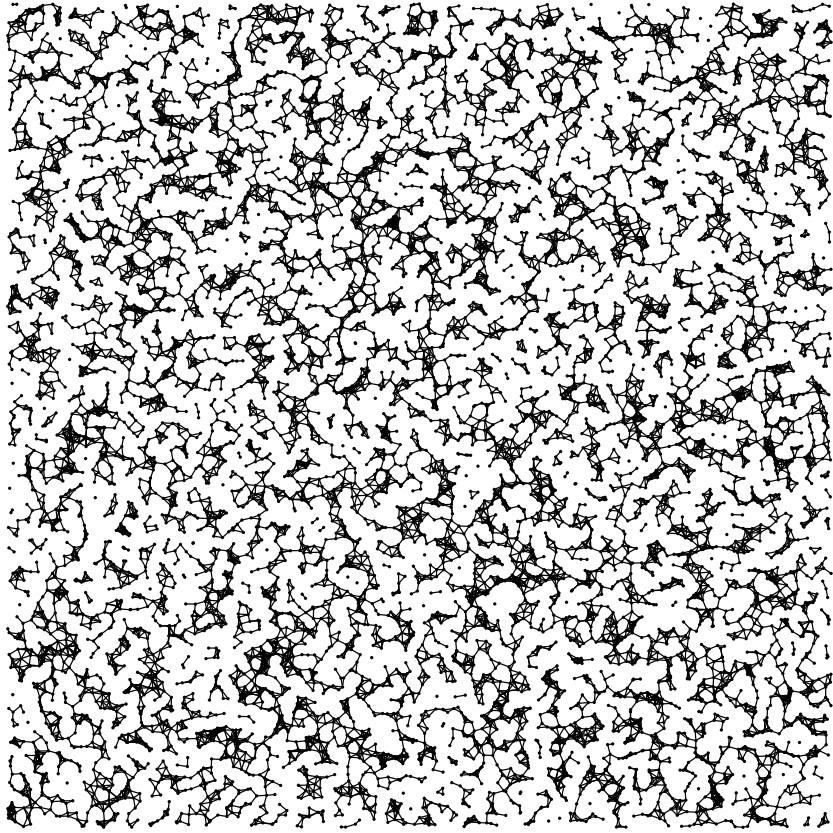


FIG. 2.2 – Un graphe de disques unitaires avec 10 000 nœuds, 24 608 liens, de degré moyen 2.5 et de degré maximal 14.

variable soit des disques ou des boules de rayon variable.

Ce modèle reste valide en espace homogène tant qu'un seul nœud émet à la fois. En réalité, les transmissions interfèrent les unes avec les autres (pour une transmission donnée, les autres transmissions doivent être considérées comme du bruit). Aussi le graphe des connexions possibles entre nœuds dépend des connexions en cours d'une manière complexe puisque les bruits provenant de plusieurs transmissions s'additionnent. Un fort trafic le long d'une route peut donc briser des connexions qui seraient possibles autrement.

Dans [3], nous proposons un modèle de réservation de bande passante permettant d'éviter de briser des connexions. Pour cela, nous définissons la zone d'interférence de chaque nœud. L'idée de cette zone est d'identifier les

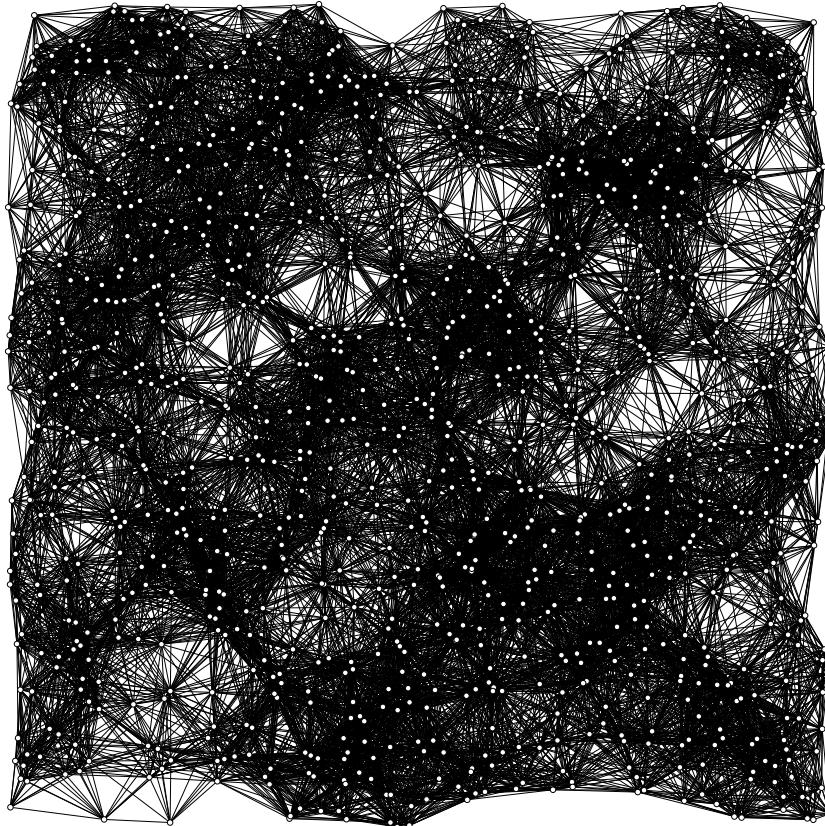


FIG. 2.3 – Un graphe de disques unitaires avec 1 000 nœuds, 22 677 liens, de degré moyen 22.7 et de degré maximal 70.

émissions qui pourraient gêner une transmission vers le nœud. En première analyse, on peut évaluer en espace homogène jusqu'à quelle distance R_b un brouilleur gêne la transmission par un nœud à distance R :

$$\frac{P_e}{R^\alpha} < K \frac{P_e}{R_b^\alpha} \text{ soit } R_b < K^{1/\alpha} R$$

En espace libre ($\alpha = 2$), on obtient qu'une émission simultanée doit provenir d'une distance au moins $3R$ environ pour ne pas brouiller une émission provenant d'une distance R . Cette analyse très simple a l'avantage de montrer qu'une émission perturbe une zone bien plus étendue que la zone où elle peut être reçue. Un modèle permettant de s'abstraire des aléas de la trans-

mission radio consiste à coder par une hyperarête la zone de couverture, soit l'ensemble des noeuds qui peuvent recevoir un émetteur donné. De même, la zone d'interférence devient l'hyperarête des noeuds qui brouilleraient une émission depuis la zone de couverture. Une réservation de bande passante pour la transmission vers un noeud s'effectue alors sur tous les noeuds de la zone d'interférence. Pour réserver la bande passante nécessaire à un trafic le long d'une route, il faut effectuer une réservation pour chaque zone d'interférence des noeuds de la route. Nous montrons dans [3] que ce problème s'apparente à celui du sac à dos multidimensionnel. Ainsi, satisfaire un ensemble de requêtes de réservation de bande passante en ad hoc est NP⁶-difficile, même pour diverses variantes simplifiées. Citons ici le résultat encore plus surprenant obtenu par Bernard Mans [2] qui montre que même en réduisant la zone d'interférence au voisinage, la réservation d'une seule route reste un problème NP-difficile.

Dans [1], nous tentons de définir le rayon de cette zone d'interférence dans le cas de plusieurs émissions simultanées. Une autre manière de voir le problème est de trouver la densité maximale d'émetteurs qui permette que toutes les émissions soient reçues jusqu'à une distance donnée.

Les approches ci-dessus reposent sur l'hypothèse que le protocole d'accès au médium (le mécanisme qui décide du moment où émettre un paquet) sera capable de sérialiser localement les émissions si la capacité en bande passante du médium n'est pas excédée. Cette hypothèse semble réaliste modulo le problème des noeuds cachés (voir ci-dessous). Citons ici une approche plus poussée dans ce domaine [6] ouverte par Baccelli, Blaszczyszyn et Mulhethaler et qui consiste à combiner dans le même protocole l'accès au médium et le routage.

Liens unidirectionnels

Il se peut qu'un noeud soit entendu d'un autre mais pas l'inverse. On parle alors de noeud caché ou plus explicitement de lien unidirectionnel. Cette situation n'est pas rare en pratique et il existe quelques techniques pour essayer d'en résoudre les méfaits. S'il n'est pas évident que cela puisse arriver à puissance d'émission égale (il faut prendre en compte la capacité des récepteurs radio à décorrérer les chemins multiples et supposer que la configuration spatiale rassemble ces chemins dans un sens mais pas dans l'autre), on comprend

⁶ NP : « Non determinist Polynomial »

facilement l'apparition de ce genre de liens lorsque les puissances d'émission peuvent varier. Ce type de liens est en général à bannir dans les réseaux, il est sans doute bon de rappeler ici pourquoi. D'une part, l'envoi d'un paquet de données est en général suivi d'un acquittement en retour (ceci est nécessaire pour s'assurer que le paquet est bien passé). Un lien unidirectionnel interdit l'envoi direct d'un tel acquittement. D'autre part, supposons l'existence d'un lien unidirectionnel d'un noeud A vers un noeud B . Pour pouvoir utiliser ce lien dans une route, il faut que A connaisse l'existence du lien. Pour faire parvenir cette information à A , B est confronté au même problème que pour lui faire parvenir des acquittements. Retourner l'information de B vers A demande donc a priori un coût important (surtout si l'on veut faire parvenir des acquittements) qui rend l'utilisation des liens unidirectionnels plutôt prohibitive.

On peut donc définir le voisinage d'un noeud comme l'ensemble des nœuds de sa zone de couverture dont il peut recevoir les messages. Nous pouvons maintenant distinguer les deux modes d'émission de paquet en ad hoc : les paquets de contrôle (servant à l'établissement des routes) sont généralement envoyés en « broadcast », ce qui permet de diffuser des informations plus efficacement que s'il fallait envoyer le message individuellement à chaque voisin. Les paquets de données sont généralement envoyés en « unicast », c'est-à-dire en spécifiant le noeud suivant de la route (le « next hop ») à qui il est destiné, celui-ci doit alors envoyer un acquittement après réception. Comme précisé plus haut, ceci permet de ne pas perdre les paquets de données, ce qui pourrait dégrader considérablement le fonctionnement de TCP⁷ qui interpréterait ces pertes comme un signe de congestion. À l'inverse, la perte de quelques paquets de contrôle ne compromet pas en général le fonctionnement du réseau.

Enfin, un niveau supplémentaire de difficulté intervient lorsqu'on considère que les noeuds peuvent posséder plusieurs interfaces radio. Chaque interface aura alors ses interfaces voisines et sa zone de couverture. Au cours de la rédaction de la spécification du protocol OLSR⁸ [5], le document a pratiquement doublé en taille lors du passage d'une spécification mono-interface à multi-interfaces. Nous aborderons quelques-unes de ces subtilités au chapitre 3.

Terminons cette introduction sur les réseaux ad hoc par quelques points

⁷ TCP : « Transfert Control Protocol »

⁸ OLSR : « Optimized Link State Routing protocol »

illustrant la nécessité de savoir router dans ce modèle extrême qui combine toutes les difficultés d'un réseau radio :

- À puissance de traitement du signal donnée, la portée diminue avec le débit (voir la loi de Kepler-Jacquet [8]). Le relayage radio est donc indispensable si l'on ne veut pas tirer des nouveaux fils partout.
- Même si les relais ne bougent pas, un réseau radio est dynamique : une pluie, un camion qui passe, une armoire métallique qu'on déplace sont autant d'éléments qui peuvent perturber les connexions radio.

2.4 Graphe du Web

Le graphe du web est constitué par les liens hypertextes entre les pages web. Il s'agit donc d'un graphe orienté dont les sommets sont les pages web et les arcs sont les liens hypertexte qui permettent de naviguer d'une page à l'autre. Cette définition passe sous silence plusieurs aspects importants dans l'étude de la structure du web.

Premièrement, l'ensemble des pages web est une notion floue qui occulte toute la difficulté de découvrir le web. Remarquons tout d'abord qu'il est impossible de découvrir toutes les pages web accessibles sur Internet. Outre les pages protégées par mot de passe ou par une URL⁹ secrète, ou les pages dynamiques (générées à partir de formulaires), le temps nécessaire à la récupération d'un grand nombre de pages web rend l'opération impossible avant que de nouvelles pages soient apparues. Le « crawl » du web (qui consiste à télécharger toutes les pages web par exploration exhaustive du graphe du web construit à partir des pages déjà « crawlées ») est comme un scan de radar très lent.

De plus, l'identification des pages web n'est pas évidente. D'un point de vue pratique, toute page est accessible via son URL. Cependant, la même page peut-être accessible via plusieurs URLs (si un serveur possède plusieurs noms, ou si des liens symboliques font apparaître la même page avec différents chemins d'accès). Une solution consiste à hacher le contenu de la page pour détecter les répliques, mais cet identifiant ne permet pas d'identifier une page au cours du temps et de ses modifications.

Dans [4], nous tentons de définir le « web observable » en croisant les informations obtenues en « crawlant » et celles obtenues dans les fichiers de

⁹ URL : « Uniform Resource Locator »

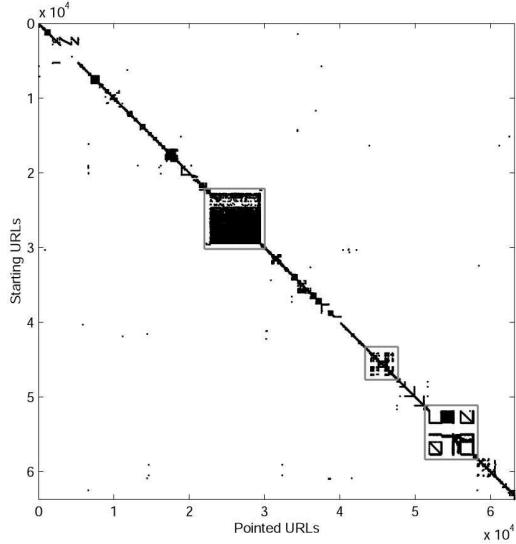


FIG. 2.4 – Un bloc $60\,000 \times 60\,000$ de la diagonale de la matrice d’adjacence d’un crawl du web en .fr. Des zooms sur les trois carrés entourés sont donnés dans les figures suivantes.

log d’un serveur. L’utilisation croisée des deux techniques permet de découvrir de nombreuses pages invisibles avec une seule des deux techniques, ce qui montre bien la difficulté de définir l’ensemble des pages web observables.

D’autre part, la structure du web ne se résume pas au graphe du web. Les pages web sont réparties sur des serveurs et en arborescence de fichiers à l’intérieur de chaque serveur. Cette structure arborescente se retrouve d’ailleurs sur les URLs associées aux pages web. Les noms de serveurs, par l’organisation hiérarchique des noms de domaine (par les DNS¹⁰), sont aussi organisés en arbre. Le web possède donc aussi une structure arborescente : *l’arbre du web*.

Dans [10], nous montrons comment les deux structures peuvent être utilisées conjointement pour observer une structure de blocs dans le web, pour arriver à une notion intrinsèque de site web. Cet aspect de la structure du web est assez évidente intuitivement : la plupart des liens sont locaux, c’est-à-dire qu’une page pointe souvent vers des pages proches d’elle dans l’arbre. C’est d’ailleurs cette simple observation qui nous avait permis de stocker de

¹⁰ DNS : « Domain Name Server »

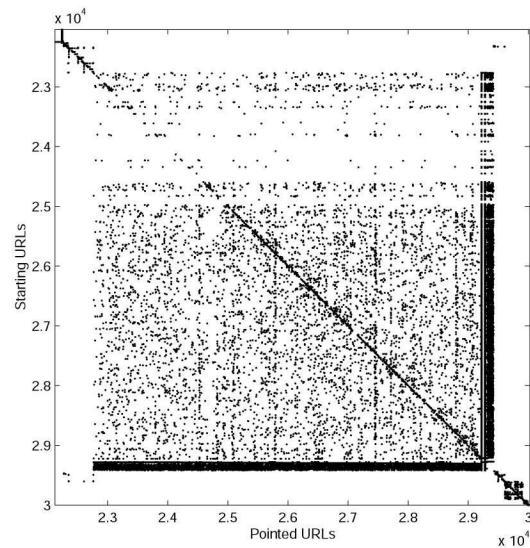


FIG. 2.5 – La matrice d'adjacence du site d'un dictionnaire interactif.

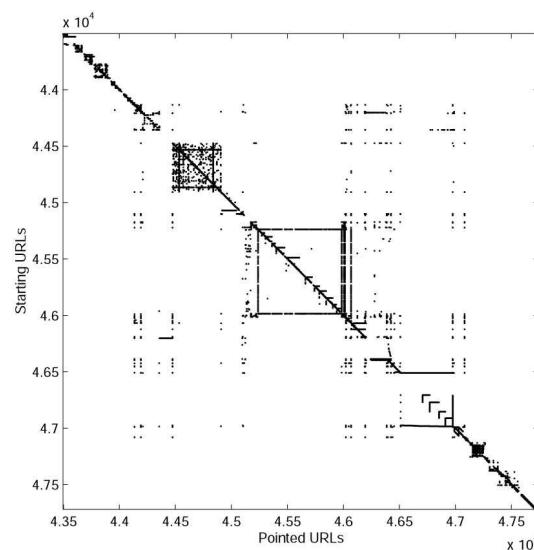
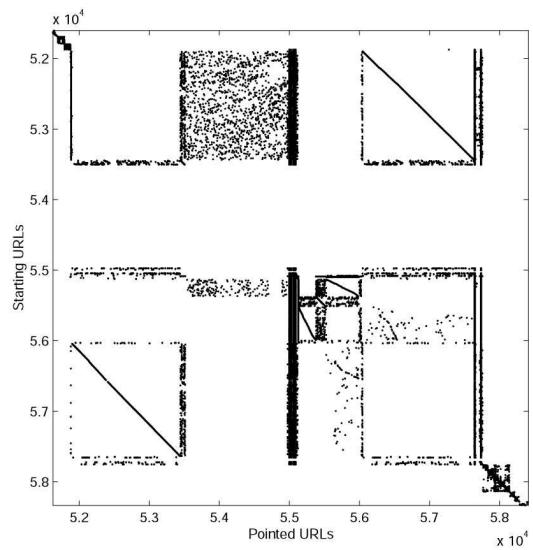
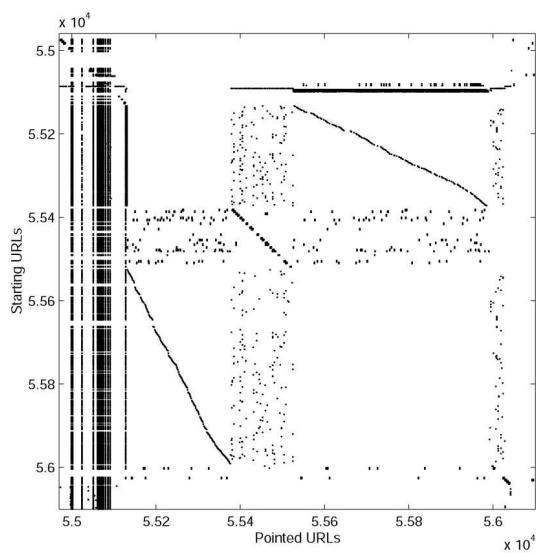


FIG. 2.6 – La matrice d'adjacence du site de l'équipe Algo de l'Inria

FIG. 2.7 – La matrice d'adjacence du site `allmacintosh`.FIG. 2.8 – Un zoom sur la partie OS X de `allmacintosh`.

manière efficace des parties du graphe du web [7].

Les figures 2.4, 2.5, 2.6, 2.7 et 2.8 illustrent cette structure de blocs [10]. Elle représentent des parties de la diagonale de la matrice d'adjacence du graphe obtenu d'un crawl de `.fr` de 8 millions de pages datant de 2001. Les URLs sont numérotées dans l'ordre alphabétique (qui constitue un parcours de l'arbre des URLs). Un point apparaît dans la matrice d'adjacence dans la ligne i et la colonne j quand i pointe vers j .

En algorithmique des graphes, partitionner un graphe de sorte que la plupart des liens soient intérieurs aux parties (l'opération peut être répétée récursivement à l'intérieur de chaque partie) s'appelle une décomposition en « clusters ». Une fois construite, cette décomposition est représentée par un arbre dont les feuilles sont les sommets du graphe. Chaque nœud interne représente un cluster par l'ensemble des feuilles du sous-arbre branché sur le nœud. L'arbre du web fournit naturellement une bonne décomposition en clusters.

Introduisons enfin ce que peut être la question du routage dans le graphe du web, même si cela peut ressembler à un exercice de style. Tout d'abord, naviguer de page en page en suivant des liens s'apparente à suivre une route. Une question légitime pour un auteur de page web est de savoir en combien de clics sa page est atteignable depuis telle ou telle page d'accueil. On voit donc bien apparaître la question de l'existence de routes et de leur longueur dans le contexte du web. D'autre part, le problème de l'indexation des pages web (auquel se confronte les moteurs de recherche) consiste à trouver des pages intéressantes étant donné des mots qu'elles contiennent. Comme il y a généralement beaucoup de pages qui contiennent les mots donnés d'une requête le principal problème consiste à identifier les plus intéressantes. Pour cela, l'équipe de Google a eu l'idée novatrice de juger l'importance d'une page par les routes qui y mènent. Ainsi, l'importance d'une page est évaluée comme la probabilité qu'un surfeur aléatoire tombe sur cette page (voir le paragraphe 6.4 sur le PageRank).

2.5 Réseaux de pair à pair

La principale application des réseaux de pair à pair consiste à partager des fichiers : un utilisateur met en accès à la communauté ses propres fichiers, recherche par des mots clés d'autres fichiers, contacte ceux qui les possèdent pour les récupérer directement de pair à pair. On peut inclure cela dans la

problématique du routage puisqu'il s'agit pour un nœud de contacter le nœud qui possède tel ou tel fichier.

Les réseaux de pair à pair (« peer to peer » en anglais) reposent généralement sur l'élaboration d'un réseau virtuel (« overlay network ») au-dessus d'Internet. Tout nœud pouvant se connecter à tout autre, il n'y a pas a priori de contrainte sur le graphe de connexion du réseau virtuel. Les restrictions viennent de la taille et du dynamisme du réseau. En effet, on imagine aisément plusieurs millions de noeuds former un réseau de pair à pair avec sans cesse l'arrivée de nouveaux nœuds et le départ d'autres. Il devient donc impensable que chaque nœud connaisse tous les autres. Les pairs doivent s'organiser pour former une structure dynamique efficace les connectant entre eux. Il existe en gros deux approches pour cela : construire un graphe aléatoire et reposer sur des explorations aléatoires du réseau, ou bien construire un graphe structuré dans lequel on sait router efficacement.

Graphes aléatoires

L'utilisation de graphes aléatoires permet de gérer efficacement le problème de la volatilité des nœuds et sont souvent adaptés à la diffusion de messages ou de données.

Le réseau Gnutella construit le réseau selon un processus d'exploration aléatoire du réseau : un nouvel arrivant parcours le réseau jusqu'à trouver des pairs qui acceptent de se connecter avec lui. Le protocole distingue deux niveaux de pairs : les « super-pairs » se connectent entre eux et assurent la connectivité du réseau, les « simples-pairs » doivent trouver un super-pair qui accepte de les prendre en charge. Les super-pairs sont en charge de la diffusion des requêtes dans le réseau. (Certaines techniques de Gnutella, décrites au paragraphe 5.3, donnent plus de détails sur ce réseau.) Les figures 2.9, 2.10, 2.11, 2.12, 2.13 et 2.14 issues du réseau Gnutella illustrent cette topologie à deux niveaux.

Ces figures sont réalisées à partir d'un crawl du réseau Gnutella v0.6 datant de fin 2003 [9]. Le crawl a duré une semaine pendant laquelle un robot a pu se connecter à 20 000 clients du réseau Gnutella v0.6. Chaque client fournissant la liste des pairs auxquels il est connecté, le crawl se présente sous la forme d'un graphe d'environ 100 000 sommets. Les images ci-dessous sont générées en effectuant une marche aléatoire sur un nombre variable de sommets du crawl et en intégrant le voisinage des sommets visités. Ce processus reproduit ainsi une exploration restreinte du réseau tel que photographié

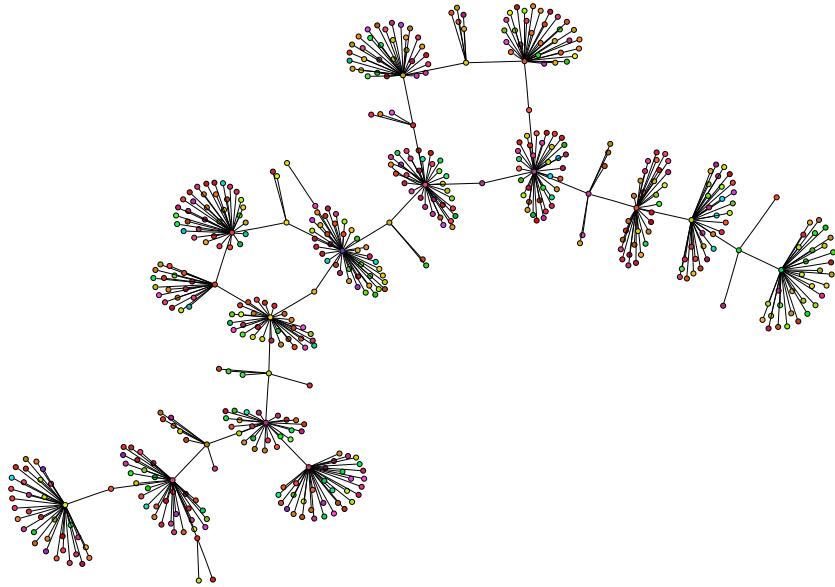


FIG. 2.9 – Une exploration de 30 noeuds du réseau Gnutella

dans notre crawl. Les dessins sont réalisés avec Graphviz. En augmentant peu à peu le nombre de sommets explorés (30, 40, 50, 100, 250, 400) on peut imaginer à quoi pourrait ressembler un dessin de la totalité du crawl (Graphviz devient inutilisable pour une exploration de plus de 1000 sommets).

Le protocole BitTorrent utilise un « tracker » pour mettre en relation les pairs intéressés par le téléchargement d'un fichier. Celui-ci fournit régulièrement à chaque pair une liste plus ou moins aléatoire de quelques dizaines de pairs. Le graphe construit sert alors à diffuser le fichier. Le protocole BitTorrent est décrit plus en détails au paragraphe 3.4.

Graphes structurés

Un domaine très actif ces dernières années concerne les tables de hachages distribuées. L'idée est de relier les pairs selon un réseau logique structuré, comme un hypercube ou un graphe de Bruijn par exemple, de manière à pouvoir utiliser des techniques de routage efficaces plutôt que des diffusions. Toute la difficulté consiste alors à maintenir un graphe proche d'une structure très irrégulière idéale malgré la volatilité des noeuds.

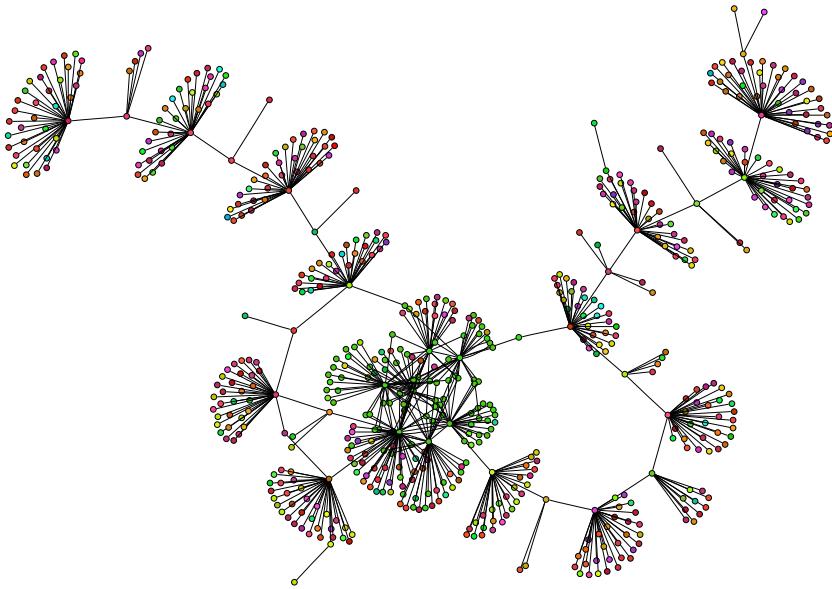


FIG. 2.10 – Une exploration de 40 noeuds du réseau Gnutella

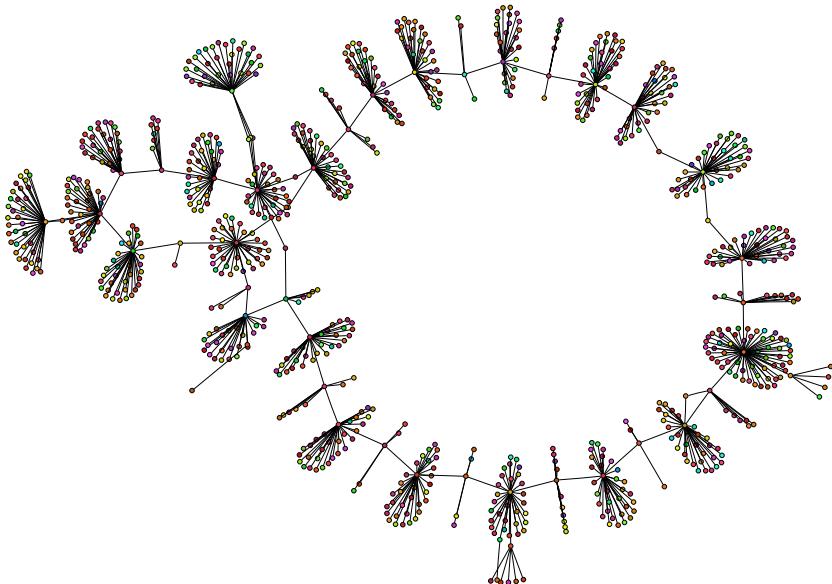


FIG. 2.11 – Une exploration de 50 noeuds du réseau Gnutella.

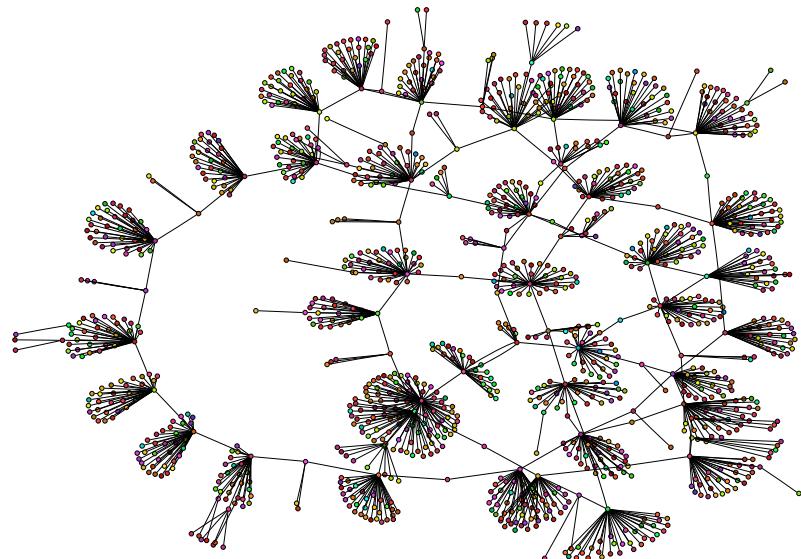


FIG. 2.12 – Une exploration de 100 nœuds du réseau Gnutella

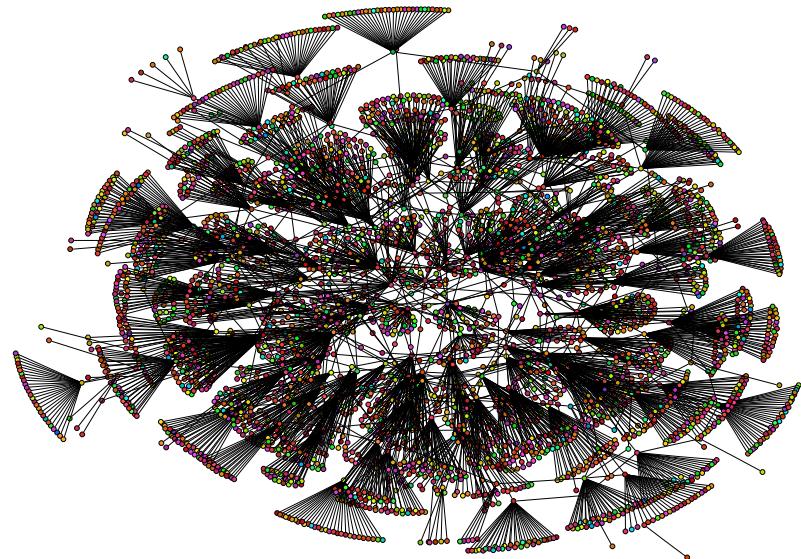


FIG. 2.13 – Une exploration de 250 nœuds du réseau Gnutella

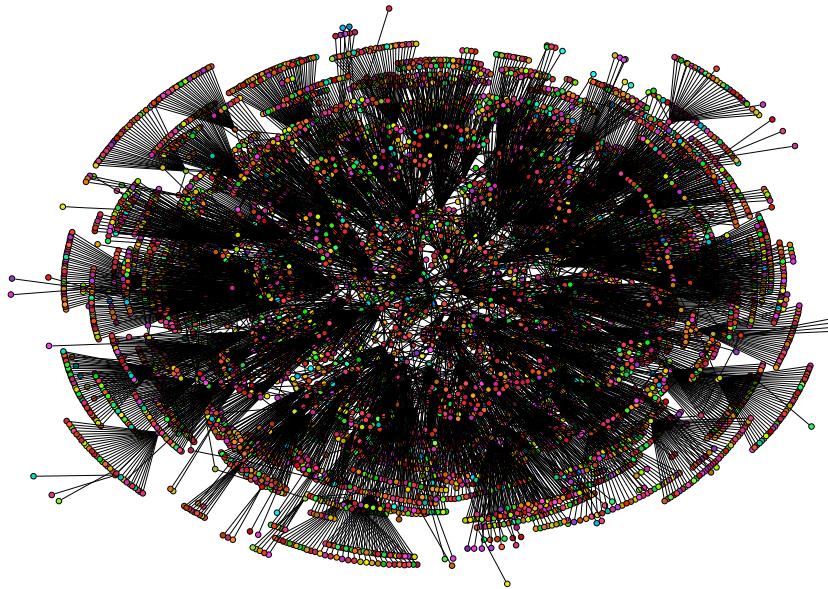


FIG. 2.14 – Une exploration de 400 nœuds du réseau Gnutella

Les tables de hachage distribuées se concentrent sur le problème de trouver un fichier lorsqu'on connaît son identifiant. L'idée est de construire une table distribuée dont les clés sont à la fois des identifiants de fichiers ou de nœuds. (En plus de son adresse IP, chaque nœud possède un identifiant.) Pour éviter les collisions, les clés sont généralement assez longues (128 ou 160 bits). Ainsi pour permettre de retrouver un fichier, le nœud qui le possède stocke dans la table une association de clé l'identifiant du fichier et de valeur sa propre adresse IP. Ainsi un nœud qui recherche le fichier retrouve cette association (connaissant l'identifiant du fichier) pour obtenir l'adresse IP d'un nœud qui a le fichier. Toute la difficulté consiste à répartir équitablement les associations sur tous les nœuds de sorte qu'il soit possible de retrouver le ou les nœuds qui sont en charge d'une clé donnée.

Les solutions existantes consistent à stocker une association sur le ou les nœuds dont l'identifiant est le plus proche de la clé pour une certaine métrique. D'autre part, les nœuds s'organisent en un réseau logique de sorte qu'il soit facile et rapide de trouver de proche en proche le nœud dont l'identifiant est le plus proche d'une clé donnée. Le problème ressemble alors à celui des machines parallèles dans lesquelles il faut relier entre eux des pro-

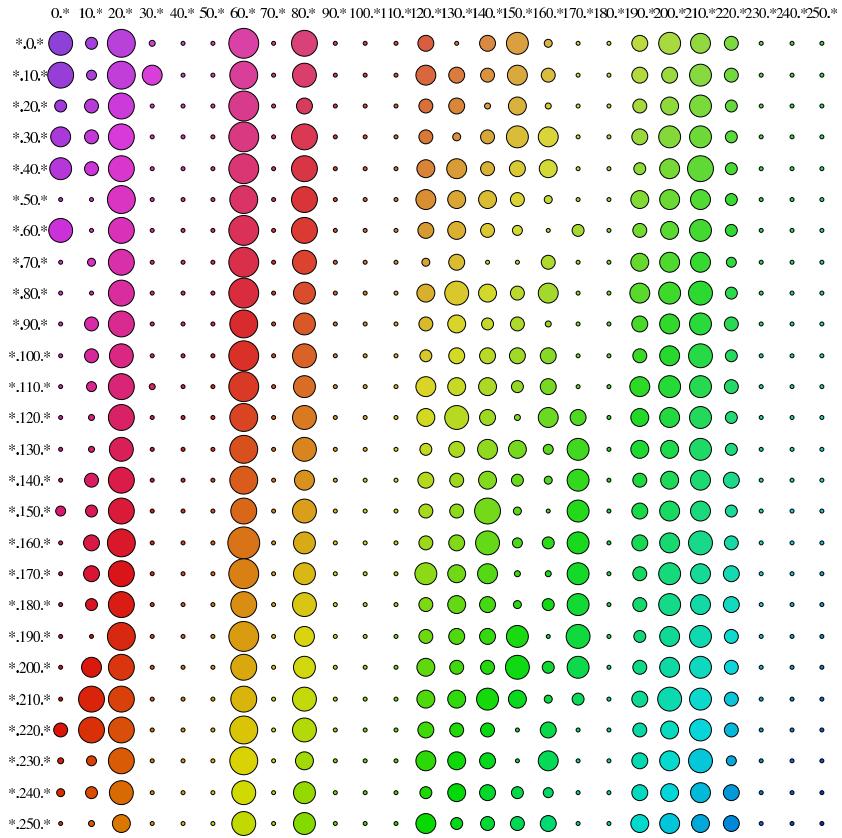


FIG. 2.15 – La couleur des noeuds dans les figures précédentes représente leur adresse IP selon la table ci-dessus. Le rond en colonne a et en ligne b indique la couleur des adresses dont le premier octet est entre $10a$ et $10a + 9$ et le deuxième entre $10b$ et $10b + 9$. La taille des ronds indique le logarithme du nombre de noeuds rencontrés dans le crawl dont l'adresse IP tombe dans la plage correspondante.

cesseurs avec un nombre de câbles limité et un diamètre en nombre de sauts¹¹ faible. De plus, le pair à pair nécessite de s'adapter à une nature fortement dynamique où des noeuds partent et d'autres arrivent sans cesse. Ainsi tous les travaux qui ont été faits autour des machines parallèles ont été revisités

¹¹Le nombre de sauts en réseau désigne la longueur d'une route, en nombre de liens généralement. (Dans le paquet acheminé jusqu'à la destination, un champs incrémenté chaque fois que le paquet est retransmis permet de mesurer ce nombre de sauts.)

dans ce cadre en adaptant les topologies classiques du tore, de l'hypercube, du papillon ou encore du graphe de Bruijn à ce contexte dynamique. Le paragraphe 5.4 fournit quelques algorithmes de routage dans ce contexte.

2.6 Exploration de réseau

La technique de base pour explorer un réseau consiste à parcourir le graphe.

Parcours de réseau

1. Connaître un nœud u_0 et l'ajouter dans la liste des nœuds à visiter.
2. Tant qu'il y a des nœuds à visiter :
 - (a) Enlever un nœud u de la liste des nœuds à visiter.
 - (b) Récupérer la liste de ses voisins et ajouter ceux qui n'ont pas encore été visités à la liste des nœuds à visiter.

En pratique, cela pose plusieurs problèmes. Le premier vient du mécanisme de découverte qui ne répond pas toujours (un serveur web peut être surchargé ou en panne, un pair peut refuser la connection) ou qui n'existe pas (un routeur d'Internet ne possède pas de tel mécanisme).

Dans le cas du graphe du web, la nature orientée du graphe fait qu'on ne trouvera que les pages accessibles par une suite de liens depuis u_0 . De plus, le processus d'exploration ne termine jamais en général (au bout de plusieurs mois d'exploration, il reste souvent une liste très importante de pages non visitées).

Dans le cas d'Internet, on ne dispose pas de mécanisme de découverte de voisins. Pour l'explorer, l'idée générale consiste à router vers n'importe qui (vers des destinations aléatoires). La technique consiste à effectuer des « traceroutes » vers des adresses aléatoires. Un « traceroute » revient à envoyer des messages de « ping » avec des TTLs de 1, 2, 3,... Le routeur intermédiaire qui voit le TTL expirer répond en général par un message de contrôle. On peut ainsi découvrir la suite des adresses IP des interfaces entrantes des routeurs de la route vers une destination donnée. Une deuxième difficulté consiste alors à retrouver la liste des interfaces d'un routeur.

Une approche de plus haut niveau consiste à observer les tables de routage BGP de systèmes autonomes, ce qui donne des arbres couvrants du graphe

de connectivité inter-systèmes autonomes. De plus, ces tables indiquent quels sont les préfixes gérés par chaque système autonome.

Dans les deux modes d'exploration de la topologie d'Internet, on obtient des listes de chemins uniquement depuis les points à partir desquels on a pu lancer le processus d'exploration.

Discussion

L'exploration des réseaux est un domaine très difficile d'un point de vue pratique. Nous n'en parlerons pas plus dans cet ouvrage plutôt dédié à l'algorithme. Cependant, il est nécessaire de bien identifier les informations dont un algorithme de réseau peut disposer pour que cet algorithme soit utilisable en pratique et le plus efficace possible.

Bibliographie

- [1] K. Al Agha and L. Viennot. Spatial reuse in wireless lan networks. In *Personal Wireless Communications (IFIP PWC'2001)*. Kluwer, 2001.
- [2] G. Allard, L. Georgiadis, P. Jacquet, and B. Mans. Bandwidth reservation in multihop wireless networks : Complexity, heuristics and mechanisms. *International Journal of Wireless and Mobile Computing*, may 2004.
- [3] K. Bertet, C. Chaudet, I. Guérin Lassous, and L. Viennot. Impact of interferences on bandwidth reservation for ad hoc networks : a first theoretical study. In *The IEEE Symposium on Ad-Hoc Wireless Networks (GLOBECOM SAWN'2001)*, 2001.
- [4] Y. Boufkhad and L. Viennot. The observable web. Technical Report RR-4790, INRIA, 2003.
- [5] T. Clausen, P. Jacquet (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhelethaler, A. Qayyum, and L. Viennot. Optimized link state routing protocol (olsr). RFC 3626, October 2003. Network Working Group.
- [6] .P Muhelethaler F. Baccelli, B. Blaszczyzyn. A spatial reuse aloha mac protocol for multihop wireless mobile networks. Technical Report RR-4955, INRIA, 2003.
- [7] J.-L. Guillaume, M. Latapy, and L. Viennot. Efficient and simple encodings for the web graph. In *The Third International Conference on Web-Age Information Management (WAIM)*, august 2002. Beijing.
- [8] P. Jacquet. Éléments de théorie analytique de l'information, modélisation et évaluation de performances. Technical Report RR-3505, INRIA, 1998.
- [9] S. Le-Blond and L. Viennot. Exploration du réseau gnutella 0.6. Stage Epitech, mai 2004.

- [10] F. Mathieu and L. Viennot. Local structure in the web. In *12-th international conference on the World Wide Web*, 2003. poster.

Chapitre 3

De l'un vers tous

Le routage d'un nœud vers tous les autres s'appelle la diffusion (ou « broadcast » en anglais). Il s'agit d'un cas particulier du routage multipoint (« multicast » en anglais), mais les techniques utilisées sont généralement différentes. L'opération de diffusion est une brique de base des protocoles de routage, elle est souvent nécessaire pour diffuser les informations nécessaires à la constitution des tables de routage. À l'inverse, les techniques de routage multipoint utilisent les tables de routage pour construire des arbres de diffusion restreints.

La difficulté de la diffusion réside dans son optimisation : l'opération doit consommer un minimum de ressources tout en assurant que tout nœud soit atteint.

3.1 Inondation

L'inondation est la technique la plus rudimentaire de diffusion. Elle consiste à répéter un message dans tout le réseau : chaque nœud qui reçoit le message pour la première fois répète le message. Ainsi, de proche en proche, le message inonde le réseau.

Inondation

1. Un message de diffusion est reçu sur une interface,
2. si le message n'a pas déjà été reçu,
3. il est retransmis sur toutes les interfaces.

Cette technique soulève tout de même quelques subtilités :

- Un nœud doit pouvoir détecter s'il reçoit un message pour la première fois. La solution la plus courante consiste à un inclure dans chaque message un numéro de séquence propre au nœud origine du message. À chaque nouveau message, le nœud origine incrémente ce numéro. Il y a évidemment un petit problème lorsque le compteur repasse à zéro mais nous ne développerons pas ici cette épingleuse question. Cette solution nécessite de retenir le dernier numéro de séquence rencontré pour chaque nœud origine possible. Une solution alternative consisterait à hacher le contenu du message et à cacher les empreintes des messages les plus récemment reçus. Une collision de clé de hachage peut alors entraîner la perte unilatérale d'un paquet. (Un champ de bits aléatoires pourrait permettre d'éviter les collisions avec les messages récents reçus par le noeud origine.)
- La technique n'est pas exactement la même dans un réseau filaire et dans un réseau ad hoc : en filaire, un nœud retransmet le message sur toutes ses interfaces sauf sur celle par laquelle il a reçu le message. En ad hoc, le nœud retransmet sur son unique interface s'il n'en a qu'une et sur toute ses interfaces dans le cas général.

3.2 Inondation optimisée par mulitpoints relais

Les réseaux radio ont une bande passante limité, mais la capacité d'atteindre plusieurs nœuds par une seule transmission laisse entrevoir la possibilité d'optimiser le processus de diffusion. Dans un réseau dense (où chaque nœud peut atteindre beaucoup de voisins), il doit être possible d'atteindre n nœuds avec moins de n émissions.

Dans ce but, Philippe Jacquet, Paul Muhlethaler et Pascale Minet imaginèrent pour la norme Hiperlan 2 [6] le concept de *multipoint relai* [11]. L'idée est d'optimiser localement la diffusion en espérant un gain global. Le principe est d'utiliser une connaissance locale de la topologie :

Messages « hello »

1. Chaque nœud émet régulièrement un message « hello » (avec TTL 1) contenant la liste des nœuds qu'il entend.

Au bout de quelques périodes d'émission, un nœud a non seulement une liste à jour des nœuds qu'il entend, mais de plus la liste des nœuds entendus par ceux-ci. Le premier avantage de cette technique est de repérer les liens unidirectionnels. Le second est de donner à chaque nœud une vision à deux sauts de la topologie du réseau. Nous appellerons voisins de u les nœuds qui sont entendus par u et dont le message « hello » inclut u , et voisins à deux sauts de u les voisins listés dans les messages « hello » entendus par u autres que u et ses voisins. Nous noterons ces deux ensembles $N(u)$ et $N^2(u)$ respectivement. Remarquons que cette connaissance du voisinage à deux sauts nécessite de distinguer dans les messages « hello » les nœuds qui sont simplement entendu des voisins.

L'idée des multipoints relais consiste à utiliser cette connaissance à deux sauts pour optimiser localement la diffusion. Ainsi, chaque nœud u désigne un ensemble de multipoints relais (parmi ses voisins) qui lui permettent d'atteindre tous les nœuds à deux sauts de lui (les voisins de ses voisins).

Calcul de multipoints relais

1. Chaque nœud u élit un ensemble $M(u) \subset N(u)$ de multipoints relais de sorte que $N^2(u) \subset N(M)$ (où $N(m) = \cup_{m \in M} N(m)$).

Remarquons la notion de multipoint relai est une relation binaire (un nœud est multipoint relai d'un autre).

L'idée ensuite est que seuls les multipoints relais de u doivent retransmettre un message de diffusion en provenance de u . Atteindre le voisinage à deux sauts de u requiert ainsi $|M(u)| + 1$ émissions. Pour obtenir un gain maximal, il faut bien sûr élire un nombre minimal de multipoints relais. Dans la figure 3.1 ci-contre par exemple, $\{a, c, e, b\}$ constitue un ensemble de multipoints relais acceptable pour u . (u n'a pas connaissance des arcs grisés.) Cependant, $\{a, b, d\}$ fournirait une meilleure solution. Remarquons que a est forcément multipoint relai de u puisqu'il est le seul qui lui permette d'atteindre f .

Nous avons montré dans [13] que calculer un tel ensemble optimal est un problème NP-difficile. On peut néanmoins prouver que l'heuristique proposée par Amir Quayyum calcule une solution dont la taille est au plus à un facteur logarithmique de l'optimal. L'idée est de sélectionner de manière gloutonne les multipoints relais en préférant ceux qui atteignent un maximum de voisins à deux sauts non encore atteints par les multipoints relais déjà choisis.

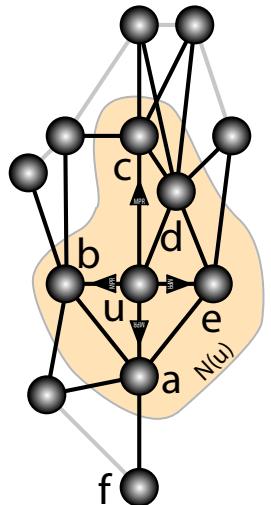


FIG. 3.1

Dans le modèle du graphe de disques unitaires, cette heuristique s'avère extrêmement proche de l'optimal. La preuve de NP-complétude montre que le calcul de multipoints relais est équivalent à celui d'un *ensemble dominant* dans un graphe. (Un ensemble dominant est un ensemble de noeuds tel que tout sommet du graphe est soit dans l'ensemble soit voisin d'un noeud de l'ensemble.) Un résultat d'inapproximabilité de ce problème [7] indique qu'il n'est pas possible d'obtenir une heuristique avec un facteur d'approximation meilleur que $O(\log n)$ dans le cas général.

Les multipoints relais permettent de construire un processus d'inondation optimisé :

Inondation par multipoints relais

1. Si v reçoit un message de diffusion par u ,
2. et u est voisin de v (uv n'est pas un lien unidirectionnel),
3. le message est marqué comme reçu.
4. Si le message est reçu pour la première fois,
5. et v est multipoint relai de u ,
6. et le TTL du message reste strictement positif après décrémentation,
7. alors v retransmet le message.

Il est une petite subtilité à faire noter : si v reçoit le message pour la première fois d'un noeud u_1 dont il n'est pas multipoint relai, il ne retransmettra jamais le message, même s'il reçoit ensuite d'un noeud u_2 dont il est multipoint relai. L'idée est que tout voisin w de v sera malgré tout atteint. En effet, w étant voisin à deux sauts de u_1 , u_1 a dû élire un multipoint relai lui permettant d'atteindre w ...

La première partie de la règle indique qu'il ne faut pas considérer un message comme reçu si on le reçoit par un lien unidirectionnel. Dans la figure 3.2 ci-contre par exemple, c ne doit pas considérer qu'il a reçu le message quand la transmission par le lien unidirectionnel $a \rightarrow c$ passe. Ainsi, lors de l'émission de b qui est multipoint relai de a , c retransmettra puisqu'il est multipoint relai de b .

On peut formellement prouver que tout noeud du réseau sera atteint par une inondation par multipoints relais en supposant que tous les voisins d'un émetteur reçoivent systématiquement le message émis et que les ensembles de multipoints relais sont valides (tout voisin à deux sauts d'un noeud doit

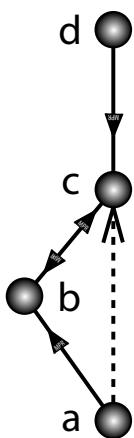


FIG. 3.2

être couvert par au moins l'un de ses multipoints relais) et que le graphe défini par les ensembles de voisins est connexe.

Nous avons pu vérifier l'efficacité de cette technique dans diverses simulations [13, 4]. Il est néanmoins possible de formuler une analyse très simple. Si un nœud a d voisins dont m l'ont choisi comme multipoint relai, on peut estimer qu'il émettra avec probabilité de l'ordre de m/d . On peut donc s'attendre à observer en moyenne nm/d émissions. Dans le cas où les nœuds sont distribués uniformément, m et d restent proches du nombre moyen de multipoints relais par nœuds et du degré moyen respectivement. Cette analyse empirique est en accord avec des simulations dans le modèle du graphe de disques unitaires [1]. Cette analyse rudimentaire laisse entrevoir un facteur d'optimisation global proportionnel au facteur d'optimisation local, ce qui n'est pas évident a priori [10].

Multipoints relais et interfaces multiples

Dans un souci de simplicité de présentation, les paragraphes ci-dessus supposent implicitement que chaque nœud ne possède qu'une seule interface. Une des plus grandes difficultés que nous avons rencontrée dans la rédaction du protocole OLSR [3] qui reprend ces idées a été de transposer les idées ci-dessus au cas où les nœuds peuvent posséder plusieurs interfaces. Le design d'OLSR repose sur l'idée que les nœuds mono-interface seront vraisemblablement les plus courants. Ainsi, le protocole doit rester simple quand les nœuds ne possèdent qu'une seule interface.

Pour cela, les nœuds sont identifiés par l'adresse d'une de leur interface (« main address ») pour éviter d'introduire un numéro de routeur (« routeur ID ») qui serait inutile dans le cas mono-interface. D'autre part, les messages « hello » spécifient la liste des interfaces entendues. Pour détecter les liens unidirectionnels, il est important de distinguer les interfaces entendues par telle ou telle interface du nœud. (Deux liens unidirectionnels pourraient donner l'illusion d'un lien symétrique.)

Dans ce cadre, les nœuds ont une vision légèrement incomplète de la topologie à deux sauts puisqu'un nœud connaît la liste des interfaces entendues par ses voisins mais pas la liste des nœuds à deux sauts (il se peut que deux interfaces appartiennent au même nœud, mais rien ne permet de le savoir sans augmenter encore la complexité des messages « hello »). Dans le doute, le calcul des multipoints relais est modifié de sorte que toutes les interfaces à deux sauts soient atteintes depuis chaque interface. La procédure de diffusion

subit les modifications les plus subtiles :

Inondation par multipoints relais avec interfaces multiples

1. Si v reçoit sur son interface j un message de diffusion émis par l'interface i de u ,
2. et j est voisine de i (ij n'est pas un lien unidirectionnel),
3. le message est marqué comme reçu sur j .
4. Si le message est reçu pour la première fois sur j ,
5. et v est multipoint relai de u ,
6. et le TTL du message reste strictement positif après décrémentation,
7. alors v retransmet le message sur toutes ses interfaces.

Il est nécessaire d'inclure un mécanisme particulier si le message est reçu à nouveau sur une autre interface. Dans OLSR, un multipoint relai risque alors de devoir retransmettre plusieurs fois le message. Ceci permet de garantir que tout noeud recevra bien le message si toutes les transmissions sont fiables. Comme seule la preuve du cas mono-interface apparaît dans la littérature, nous allons inclure ici une preuve du cas multi-interfaces, ce qui nous permettra de voir un exemple de preuve de type théorie des graphes appliquée au domaine des réseaux.

Supposons qu'un noeud u ne soit pas atteint par une inondation initié par une source s d'un réseau ad hoc connexe. Considérons la distance d en nombre de sauts des noeuds les plus proches de u ayant émis ($d > 1$ puisque u n'a pas reçu le message). Considérons la première interface i d'un noeud v à distance d de u à avoir émis le message vers au moins un noeud à distance $d - 1$ de u . i a forcément un voisin à deux sauts w à distance $d - 2$ de u et un multipoint relais m qui couvre w . Soit j l'interface sur laquelle m reçoit le message de i . En supposant que les messages reçus par j sont lus dans par ordre d'émission, m reçoit le message sur j pour la première fois (il se peut par contre que m ait déjà reçu le message par une autre interface). D'après la règle de retransmission, m doit retransmettre, ce qui est contradictoire puisque m est à distance $d - 1$ de u .

L'alternative concurrente : une autre option aurait consisté à distinguer la découverte des topologies à un saut et à deux sauts. Ainsi, les messages « hello » auraient eu une section 1 saut décrivant les interfaces entendues

par l'interface émettrice du message et une section 2 sauts listant les « main address » des noeuds avec lesquels la symétrie du lien a été établie. Cette alternative qui aurait permis de généraliser plus élégamment le concept de multipoint relai n'a pas été retenue pour deux raisons. La première est qu'elle s'éloigne d'un choix de conception de OLSR : le cas mono-interface doit rester intuitif et ne pas trop compliquer le protocole (même message « hello » envoyé sur toutes les interfaces). Une autre raison était de rester éloigné de solutions proposées par un protocole concurrent à l'IETF¹, l'organisme de standardisation d'Internet.

Optimisation versus fiabilité

Les transmissions radio étant peu fiables (surtout en mode « broadcast »), il peut être intéressant de s'assurer que chaque noeud reçoive plusieurs fois un message de diffusion. Les multipoints relais permettent facilement d'intégrer une telle facilité. Cela apparaît dans OLSR sous le nom de « MPR² coverage », un paramètre qui indique la redondance k voulue dans la diffusion. Il suffit de calculer les multipoints relais de sorte que tout voisin à deux sauts soit atteint par au moins k multipoints relais si cela est possible. L'heuristique originelle de Amir Qayyum se prête naturellement à ce calcul. Dans l'exemple de la figure 3.1, $\{a, b, c, e\}$ offre un « MPR coverage » de 2 au lieu de 1 pour $\{a, b, d\}$.

Une autre fonctionnalité similaire de OLSR consiste à prendre en compte la propension des noeuds à vouloir relayer (« MPR willingness »). Les noeuds indiquent dans leur messages « hello » s'ils sont prêts à relayer. Les autres noeuds peuvent alors sélectionner préférentiellement comme multipoints relais ceux qui ont le plus de propension à ce rôle. Bien sûr, en creusant ce sujet, on bute sur l'un des dilemmes des réseaux ad hoc : comment inciter les noeuds à coopérer et à bien vouloir servir de relai. Ce débat dépasse le cadre de ce chapitre.

¹ IETF : « Internet Engineering Task Force »

² MPR : « Multi-Point Relay »

3.3 Inondation optimisée par ensemble dominant

Maintenant que nous avons vu comment optimiser localement la diffusion, étudions comment optimiser globalement la diffusion. Considérons tous les émetteurs d'une diffusion optimisée. Tout nœud du réseau est soit l'un d'eux, soit a reçu le message de l'un d'eux. L'ensemble des émetteurs doit donc nécessairement former un ensemble dominant du graphe de connexions. Le nœud source de la diffusion fait partie de cet ensemble. D'autre part, tout émetteur a dû recevoir le message depuis la source, il doit donc exister un chemin d'émetteurs de la source au nœud. L'ensemble des émetteurs de la diffusion doit donc de plus être connexe. Quitte à opérer une émission supplémentaire, un ensemble dominant connexe donné peut servir à optimiser toutes les diffusions quelles que soient leurs sources.

Diffusion par ensemble dominant connexe

1. Un ensemble dominant connexe a été calculé.
2. La source émet son message.
3. Tout nœud de l'ensemble dominant répète le message la première fois qu'il le reçoit.
4. Les autres nœuds ne répètent pas le message.

Dans [1], nous comparons par des simulations diverses approches pour optimiser la diffusion. En très forte densité, les méthodes à base d'ensemble dominant l'emportent. En densité moyenne, l'inondation par multipoints relais semble mieux bénéficier des effets du bord du terrain. L'inondation multipoints relais possède l'avantage de faire tourner les charges de relayage entre les nœuds au gré de la source initiant la diffusion et de l'ordonnancement des émissions.

Dans un souci d'inventaire algorithmique des réseaux, donnons ici l'une des heuristiques les plus efficaces de calcul distribué d'ensemble dominant connexe. (Rappelons que calculer un ensemble dominant connexe de taille minimale est un problème NP-difficile pour lequel il existe des algorithmes séquentiels approchés.)

Élection d'ensemble dominant connexe à la « Wu et Li généralisé » [14]

1. Chaque noeud u s'élit comme membre de l'ensemble dominant connexe sauf si :
 - l'ensemble des ses voisins d'identifiant plus petit que le sien forme un ensemble connexe et qui couvre $N(u)$ (tout noeud de $N(u)$ est voisin d'au moins l'un d'eux).

Une manière assez simple de montrer que cet algorithme construit bien un ensemble dominant connexe est de considérer les noeuds par identifiants décroissants et de les retirer au fur et à mesure si la règle l'exige. L'ensemble des noeuds restants forme toujours un ensemble dominant connexe. Bien sûr en réalité, tous les noeuds qui s'éliminent le font en parallèle, mais la séquentialité de la preuve ne la rend pas moins exacte.

Optimisation versus fiabilité

La littérature sur le sujet des ensembles dominants connexes se concentre principalement sur l'obtention d'une réduction maximale du nombre d'émissions. Comme nous l'avons vu pour OLSR, il peut cependant être utile de proposer un paramétrage de l'algorithme de sélection pour obtenir un compromis entre l'optimisation du nombre d'émissions et la fiabilité de la diffusion. Notons tout d'abord que l'heuristique peut utiliser n'importe quel ordre total sur les noeuds (pas forcément celui donné par les identifiants). La règle de redondance (« coverage ») peut facilement être intégrée à l'heuristique précédente de calcul d'ensemble dominant connexe : un noeud ne s'élimine que si ses voisins plus petits que lui forment un ensemble connexe tel que tout noeud de $N(u) \cup \{u\}$ est voisin de k de ces noeuds (dans la mesure du possible). Il serait intéressant d'étudier quelle heuristique est la meilleure à facteur de redondance k donné.

Pour intégrer la propension à vouloir relayer (« willingness »), il suffit de considérer un ordre total sur les noeuds qui respecte cette propension : un noeud est plus petit qu'un autre si sa propension est plus grande ou s'il a même propension mais un identifiant plus petit.

Une autre voie d'étude concerne la fiabilité du mécanisme de diffusion lors que le graphe des connexions est fortement dynamique. Ainsi les voisinages peuvent avoir légèrement évolué depuis le moment où la structure de diffusion a été calculée. Étudier la résistance d'un mécanisme de diffusion selon l'importance des changements de voisinages semble une piste importante.

Diffusion de données

Nous avons présenté la diffusion comme une brique de base pour le protocole de routage. Elle peut aussi être utilisée pour le broadcast de données voire pour une fonctionnalité rudimentaire de multicast. Cependant, l'utilisation des multipoints relais pose une difficulté technique : il faut savoir de quel nœud on reçoit un message. Cela ne pose pas de problème avec un message de protocole puisqu'il suffit d'inclure un champ spécifiant l'émetteur du message. Dans le cas de paquet de données (dont on ne peut rien utiliser de plus que ce que fournit l'en-tête IP), il devient techniquement difficile de savoir de quel nœud provient le message. La technique par ensemble dominant connexe ne souffre pas de ce défaut, ce qui justifie dans le cadre d'OLSR de trouver un moyen de définir un ensemble dominant connexe. Dans [1], nous montrons comment élire un ensemble dominant connexe à partir des ensembles de multipoints relais. Dans le cas d'OLSR, les identifiants sont les « main address » :

Élection d'un ensemble dominant par multipoints relais

1. Un noeud u s'élit comme membre de l'ensemble dominant connexe si :
 - u a un identifiant plus petit que ses voisins,
 - ou u est multipoint relai de son voisin de plus petit identifiant.

Une analyse rudimentaire permet à nouveau d'estimer le nombre de noeuds qui vont s'élire dans l'ensemble. La probabilité d'être le nœud de plus identifiant dans son voisinage est $1/(d + 1)$ et la probabilité d'être multipoint relai s'estime à m/d où d est le degré moyen d'un nœud et m le nombre moyen de multipoints relais d'un nœud. On peut donc estimer la taille de l'ensemble dominant connexe construit par les multipoints relais à $nm/d + n/(d + 1)$. Les simulations dans le graphe de disques unitaires [1] confirme cette analyse qui montre que l'ensemble dominant ainsi calculé est légèrement moins optimisé que la diffusion par multipoints relais. (Cela dit, dans le cas d'interfaces multiples, il permet d'éviter les émissions multiples et pourrait alors s'avérer plus efficace.)

3.4 Diffusion de pair à pair

Le contexte du pair à pair fournit de nombreuses occasions de revisiter certains problèmes classiques du routage, la diffusion en est un. Une des ap-

plications légales du pair à pair réside dans la diffusion de fichiers volumineux à large échelle. Citons le protocol BitTorrent [5] qui est utilisé pour diffuser la distribution Linux RedHat³.

Dans sa version la plus simple, le problème consiste à diffuser un fichier possédé par une source à $n - 1$ autres nœuds. L'idée est d'utiliser la capacité d'« upload⁴ » de tous les nœuds et non de celle de la source uniquement. En effet, la capacité d'« upload » de tous les nœuds est proportionnelle à n alors que celle de la source est limitée. L'idée rudimentaire consiste alors à faire redistribuer le fichier par les premiers nœuds qui ont reçu le fichier :

Diffusion de fichier de pair à pair

1. Dès qu'un nœud a reçu une copie du fichier, il le redonne à un autre nœud.

Si le téléchargement du fichier prend un temps 1, tout le monde aura le fichier au bout d'un temps de l'ordre de $\lceil \log_2 n \rceil$ (la population des nœuds possédant le fichier double à chaque unité de temps). On peut cependant encore améliorer le temps de diffusion en découplant le fichier en b blocs. Dès qu'un bloc est reçu, il peut être redistribué sans attendre la fin du fichier.

Les premiers travaux de modélisation sur le sujet sont dûs à Pascal Felber [8]. Étudions à titre récréatif la faisabilité d'un ordonnancement optimal de diffusion⁵. Donnons tout d'abord une borne inférieure sur le temps de diffusion du fichier par blocs. La source ne peut avoir donné une copie de chaque bloc qu'au bout d'un temps 1. Le dernier bloc qu'elle a donné n'est donc possédé que par deux nœuds au temps 1. Le téléchargement de ce bloc prenant un temps $1/b$, il ne peut être possédé par tous les pairs qu'au bout d'un temps $(\lceil \log_2 n \rceil - 1)/b$ (au mieux, tous les nœuds qui le possèdent le redonnent tous les $1/b$ et leur population double tous les $1/b$). On arrive donc à la borne inférieure suivante pour le temps de diffusion :

$$1 + \frac{\lceil \log_2 n \rceil - 1}{b}$$

³Le lecteur averti soulèvera le fait que BitTorrent est aussi utilisé pour diffuser des copies pirates de films... Le protocole reste néanmoins l'un des plus beaux aboutissements des développements du pair à pair.

⁴On peut traduire « upload » par « téléchargement par la voie montante » en français, ce qui est un peu long pour faire la distinction entre « upload » et « download ».

⁵Cette petite récréation provient de discussions avec Anh-Tuan Gai.

Remarquons que cette analyse tient dès que la capacité d'« upload » des nœuds est limitée. Nous allons voir un peu plus loin un algorithme optimal où la capacité de « download » de chaque nœud est égale à la capacité d'« upload ». Ceci montre bien la nature intrinsèquement symétrique du problème.

De plus, cette analyse montre l'intérêt de découper le fichier en blocs. Cependant, une taille trop petite de bloc n'est pas envisageable pour deux raisons. La première est due à la mise en route de TCP⁶ qui ne permet pas d'atteindre un débit maximal sur un volume de données à transférer trop faible. La deuxième, plus légère, vient de la vérification des blocs. L'intégrité de chaque bloc est vérifiée par le téléchargement préalable des empreintes des blocs (calculées par MD5⁷ ou SHA1⁸). Cette opération a un coût non négligeable si le nombre de blocs est trop grand⁹.

Donnons à titre de cas d'école un algorithme d'ordonnancement optimal. Le protocole suppose simplement que chaque nœud peut recevoir et envoyer un bloc tous les $1/b$. Les nœuds sont numérotés de 0 à $n - 1$, la source ayant le numéro $s = 0$. Les numéros des nœuds sont lus comme des nombres entiers de $l - 1$ bits ($l = \lceil \log_2 n \rceil$) avec les bits de poids fort à gauche.

Diffusion de fichier de pair à pair ordonnancée avec 2^l nœuds

1. La source donne le bloc i au temps i/b au nœud $1 \lll i$ (où $a \lll k$ désigne l'opération de décalage circulaire des bits de a de k positions vers la gauche modulo l).
2. Au temps t/b , chaque nœud u redonne le dernier bloc reçu à $u \oplus (1 \lll t)$ (\oplus désigne le ou exclusif bit à bit).

L'idée de l'algorithme est que le bloc i est possédé (et redistribué mis à part à la dernière étape) par :

- le nœud $1 \lll i$ au temps $(i + 1)/b$,

⁶ TCP : « Transfert Control Protocol »

⁷ MD5 : « Message Digest 5 »

⁸ SHA1 : « Secure Hash Algorithm 1 »

⁹ À l'extrême, on peut envisager d'avoir une empreinte de chaque paquet envoyé sur le réseau, soit 20 octets d'empreinte pour 1000 octets de données environ, ce qui induit un surcoût de 2 %. Ce surcoût est néanmoins raisonnable et l'idée de vérifier les données au niveau de chaque paquet peut être envisagée dans certains contextes (par exemple, pour détecter quels nœuds fournissent des données erronées dans le contexte de BitTorrent où un bloc serait téléchargé depuis plusieurs pairs).

- les noeuds $1 \lll i$ et $11 \lll i$ au temps $(i + 2)/b$,
- ...
- les noeuds $b_1 \dots b_k 1 \lll i$ au temps $(i + k + 1)/b$,
- ...
- tous les noeuds au temps $(i + l + 1)/b$.

À la dernière phase, tous les noeuds dont le bit en position i vaut 1 donnent le bloc à tous ceux qui ont ce bit à 0. (La transmission vers $s = 0$ qui est la source est inutile.) Tous les noeuds impliqués dans les redistributions des blocs suivants diffèrent par le bit à cette position et il n'y a donc pas de conflit (à un instant donné, les noeuds impliqués dans la phase en court d'un bloc donné forment bien des ensembles disjoints). Du point de vue du graphe de connexion, les noeuds sont organisés en hypercube puisque chaque noeud échange des blocs avec les noeuds dont le numéro diffère d'un bit. Au temps t/b , les noeuds s'échangent deux à deux des blocs (chacun envoie un bloc à celui dont le numéro diffère par le bit en position t modulo l). Le trafic est donc parfaitement symétrique : à chaque instant, chaque noeud envoie un bloc et reçoit un bloc.

Une autre manière de voir le graphe est une union de l arbres arêtes disjoints enracinés aux noeuds du type $1 \lll i$. Chaque bloc est diffusé le long d'un de ces arbres. Ce type de construction a déjà été produit dans l'étude de la diffusion dans une machine parallèle dont les processeurs sont reliés via un hypercube [12].

Cet algorithme se termine au bout d'un temps $1 + l/b$ (le dernier bloc a numéro $b - 1$). Pour atteindre l'optimal, il faut que la source participe à la diffusion du dernier bloc. Pour cela, il suffit que la source donne le dernier bloc au noeud $1 \lll t$ au temps t/b . La dernière phase sera alors inutile pour ce dernier bloc, ce qui permet finalement d'obtenir un algorithme optimal.

Dans le cas où le nombre de noeuds n'est pas une puissance de 2, il est possible d'adapter l'algorithme en regroupant les noeuds en groupes de puissances de 2. S'il montre que l'optimal est atteignable, cet algorithme n'est malheureusement pas très utilisable en pratique. En effet, il presuppose que tous les noeuds cibles sont connus à l'avance. C'est le cas dans une mise à jour d'un parc de machines par exemple, ou lorsqu'il s'agit de diffuser le fichier à une liste d'abonnés... Mais le défaut principal de cet algorithme réside dans l'hypothèse que tous les noeuds sont fiables. Il suffit qu'un noeud tombe en panne pour perturber tout le processus. Dans le cas où les noeuds arrivent et partent de manière continue, et ont des capacités d'upload variables, BitTorrent [5] est sans doute la solution la plus évoluée à l'heure actuelle. Pour ne

pas laisser le lecteur dans l'expectative, donnons-en rapidement l'algorithme ici.

BitTorrent [5]

1. Chaque client se connecte à une centaine d'autres et échange avec eux la « bitmap » des blocs qu'il possède (un nœud spécial appelé « tracker » est chargé de mettre les clients intéressés par le téléchargement du fichier en relation).
2. Un client répète en continu le processus suivant jusqu'à ce que le programme soit arrêté par l'utilisateur (une fois le fichier récupéré, le client continue de le redistribuer) :
 - (a) Chaque client demande un bloc à chacun des autres clients (les blocs qui paraissent les plus rares sont demandés de préférence).
 - (b) Chaque client ne sert qu'une partie des demandes qui lui sont adressées selon le principe d'« un prêté pour un rendu » :
 - les trois clients qui lui ont le plus donné dans les 10 dernières secondes sont servis de préférence,
 - une quatrième demande sélectionnée au hasard est de plus honorée.

Le point clé de BitTorrent réside dans l'utilisation du principe du « prêté pour un rendu » dans le choix de redistribution du fichier, ce qui est une heuristique connue en théorie des jeux pour résoudre le dilemme du « téléchargeur égoïste » (ou « free-rider » en anglais) qui veut télécharger le fichier en utilisant le moins possible sa capacité d'upload.

Un problème similaire consiste à diffuser un flot de données continu, audio ou vidéo par exemple, de pair à pair. Si l'utilisation d'une topologie en hypercube paraissait naturelle dans une opération synchrone des échanges de blocs de fichiers, elle ne tient plus dans le cadre asynchrone général d'un réseau de pair à pair. Nous allons voir qu'une approche par arbres de degrés fixes nous conduit naturellement à la structure de graphe de Bruijn.

Diffusion d'un flot continu

La diffusion d'un flot continu s'apparente au problème du multicast. Cependant, dans le contexte du pair à pair, seuls les nœuds intéressés par le flot

participent à sa redistribution et ressemble donc plus à un problème de diffusion continu dans un réseau logique. Ce problème admet des solutions assez évidentes si l'on ne se donne pas les contraintes suivantes assez naturelles :

- chaque client n'est prêt à consommer qu'une partie minimale de son upload,
- le délai doit être de l'ordre du délai de diffusion optimale.

Une solution pour cela consiste à construire une famille de d arbres de degré d qui soient noeuds internes disjoints (chaque noeud est interne dans un arbre et feuille dans les $d - 1$ autres). L'algorithme introduit dans le papier séminal de SplitStream [2] consiste alors à utiliser chacun des arbres à tour de rôle pour diffuser les paquets du flot. L'utilisation d'arbres de degrés $d > 1$ garantie un délai proche de l'optimal. Chaque noeud doit retransmettre l'équivalent d'un flot en débit, ce qui est le minimum viable pour que chacun reçoive le flot. Maintenir une telle famille cohérente d'arbres est cependant loin d'être évidente dans un contexte où les noeuds arrivent et partent en continu. L'algorithme proposé dans [2] utilise une table de hachage distribuée et un processus de reconstruction des arbres qui nécessite que certains noeuds acceptent d'utiliser une partie plus importante de leur upload pour donner le mou nécessaire permettant de limiter l'ampleur des reconstructions. La figure 3.3 illustre deux arbres de degré 2 noeuds internes disjoints construite à partir d'un graphe de Bruijn (et non d'un hypercube comme dans [2]).

Dans [9], nous proposons de construire une famille d'« arbres clusters » pour éviter un processus de reconstruction et garantir une charge d'upload toujours égale au débit du flot pour chaque noeud. Pour cela, nous utilisons des *clusters* de quelques noeuds pour la retransmission. La taille de ces clusters peut varier sans changer la charge de ses membres. Il suffit d'un noeud vivant dans chaque cluster pour assurer la pérennité du flot.

Diffusion par d arbres de clusters

1. Un chef de cluster est élu dans chaque cluster.
2. La source transmet les paquets aux chefs des d clusters racine à tour de rôle.
3. Quand un chef de cluster reçoit un paquet,
 - (a) il le retransmet à un délégué dans son cluster chargé de retransmettre aux autres membres du cluster (le rôle de délégué tourne à chaque fois),

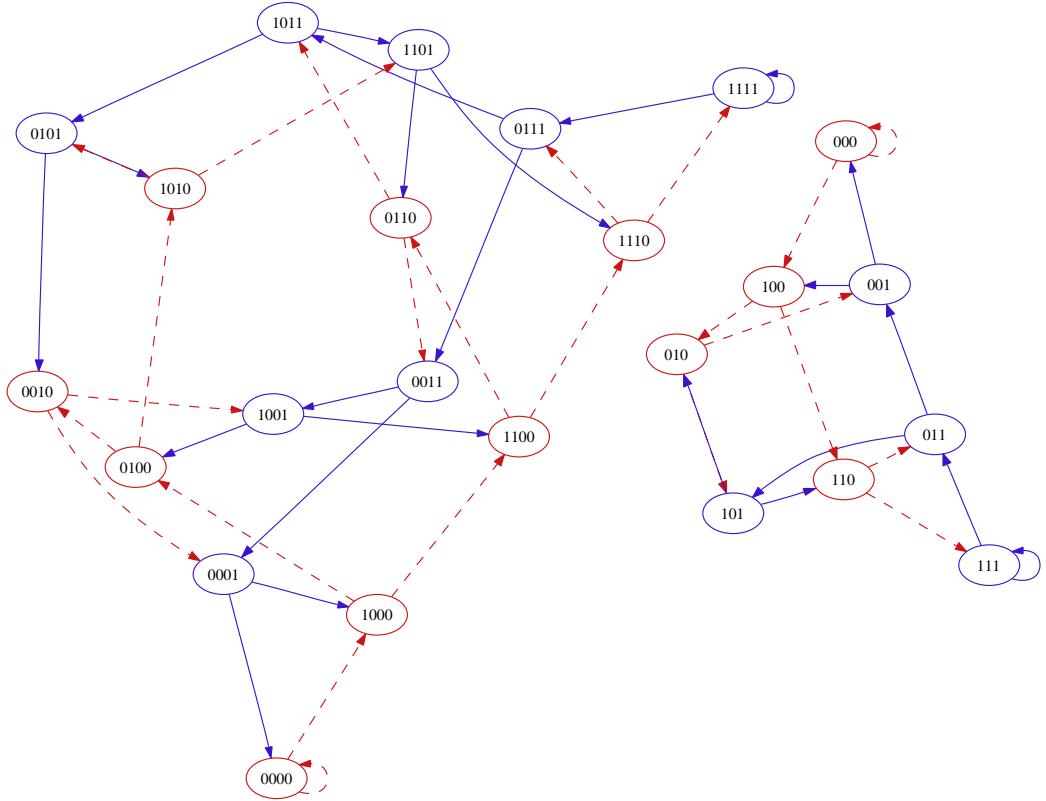


FIG. 3.3 – Deux arbres de degré 2 noeuds internes disjoints dans le graphe de Bruijn à 16 sommets et dans le graphe de Bruijn à 8 sommets.

- (b) si le cluster est noeud interne pour l'arbre impliqué dans la diffusion de ce paquet, il le retransmet aux d chefs des clusters fils dans l'arbre.

Nous proposons en fait d'utiliser 2 familles de d arbres de clusters avec un chef différent selon la famille utilisée. De plus, à l'intérieur d'une famille un cluster qui est noeud interne dans un arbre est feuille dans les $d - 1$ autres arbres. La source utilise ainsi un cycle de longueur $2d$ dans l'envoi des paquets aux racines d'une famille puis de l'autre. La figure 3.4 illustre les arbres utilisés dans cet algorithme de diffusion avec $n = 26$ et $d = 2$: les chefs de clusters sont colorés en plein ; les arcs internes au clusters changent en fonction des délégués.

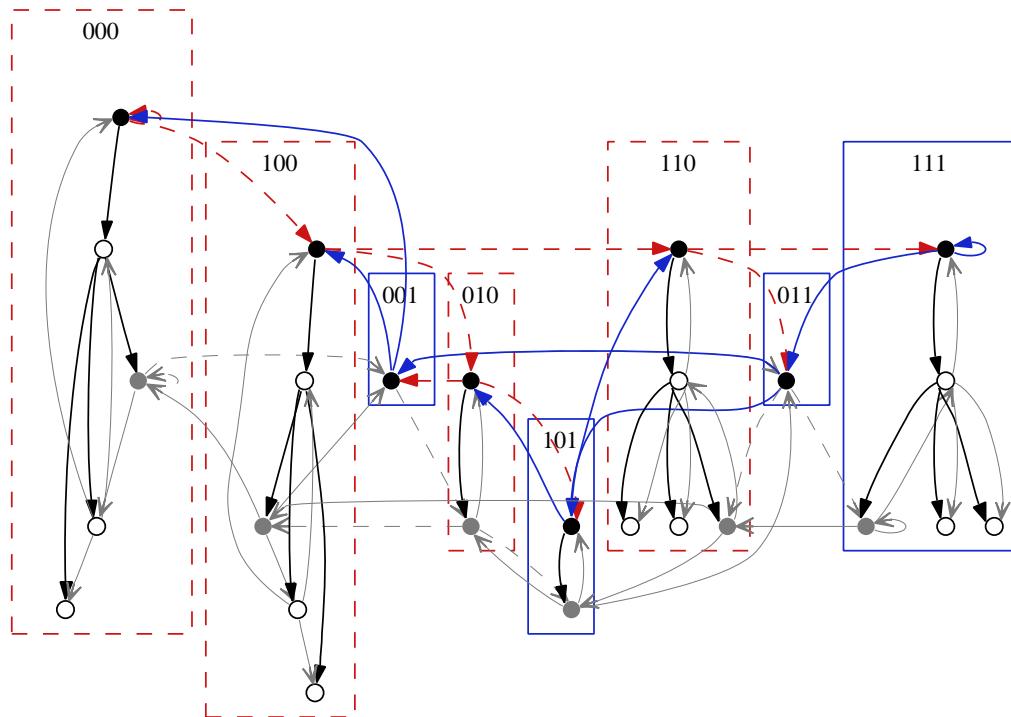


FIG. 3.4 – Une famille de deux arbres de clusters construite à partir du graphe de Bruijn à 8 sommets. (Les arcs fins et grisés fournissent une deuxième famille de deux arbres.)

Analysons la charge d'upload d'un nœud selon la taille m de son cluster :

- Si $m = 1$, on se retrouve dans le cas de SplitStream où le seul nœud qui est donc deux fois chef doit retransmettre d fois 2 paquets sur $2d$.
- Si $m = 2$, chacun des deux chefs retransmet $d + 1$ fois un paquet sur $2d$ et une fois $d - 1$ paquets sur $2d$.
- Si $m > 2$, les deux chefs fournissent toujours le même travail, tandis que les délégués retransmettent $m - 2$ fois un paquet sur $m - 2$.

Dans tous les cas, la charge d'upload d'un nœud est égale au débit du flot.

Ce regroupement en cluster permet alors de regrouper un nombre variable de nœuds en un nombre de clusters de la forme l^d . (On sait construire des arbres arêtes disjointes de temps de diffusion optimale mais qui n'ont pas degré constant et qui ne sont pas nœuds internes disjoints.) La difficulté consiste

alors à former les clusters de manière distribuée et à construire d arbres de clusters qui soient clusters internes disjoints.

Un paradigme simple pour former les clusters consiste à laisser chaque noeud choisir un identifiant au hasard écrit en base d et à regrouper ensemble les noeuds qui ont même préfixe d'identifiant de longueur l . (La longueur l peut être décidée par la source et changer au cours du temps en étant incluse dans l'en-tête de chaque paquet par exemple.)

La structure en hypercube exposé plus tôt n'est pas utilisable ici pour construire les arbres car la contrainte d'égalité en upload impose la construction d'arbres de degré constant d . Une structure qui s'impose alors assez naturellement est celle du graphe de Bruijn où les clusters fils d'un cluster de préfixe d'identifiant $x_1 \dots x_l$ sont ceux de préfixes d'identifiants $1x_1 \dots x_{l-1}$, $2x_1 \dots x_{l-1}$, ... $dx_1 \dots x_{l-1}$ obtenus par décalage et rajout d'un nouveau chiffre. (Une construction similaire en décalant les bits vers la gauche permet de construire l'arbre binaire de la structure de tas.) En utilisant les clusters de préfixes d'identifiant $1 \dots 1, \dots, d \dots d$, on obtient d arbres clusters internes disjoints. En décalant de manière symétrique vers la gauche on obtient une deuxième famille de d arbres clusters internes disjoints de mêmes racines. La figure 3.4 donne un exemple obtenu pour un préfixe de longueur 3 avec $d = 2$.

Discussion

La diffusion est une brique élémentaire de l'algorithme des réseaux. C'est pourquoi les algorithmes de diffusion ne doivent utiliser qu'une information obtenue localement tout en assurant des propriétés globales, une espèce de quête récurrente en algorithmique distribuée. D'un autre côté, la diffusion dans un réseau logique peut s'apparenter à la problématique du « multicast » où il s'agit de construire des arbres entre les noeuds du réseau. Notons cependant le cas du protocole BitTorrent qui garde une approche purement locale lui conférant une forte robustesse à la volatilité des noeuds et à leur hétérogénéité, mais sans garantie de délai sur le temps de diffusion d'un bloc du fichier.

Bibliographie

- [1] C. Adjih, P. Jacquet, and L. Viennot. Computing connected dominated sets with multipoint relays. *Ad Hoc and Sensor Wireless Networks*, 1(1-2), january 2005.
- [2] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream : High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP)*, 2003.
- [3] T. Clausen, P. Jacquet (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhllethaler, A. Qayyum, and L. Viennot. Optimized link state routing protocol (olsr). RFC 3626, October 2003. Network Working Group.
- [4] T. Clausen, N. Larsen, T. Olesen, and L. Viennot. Investigating data broadcast performance in mobile ad hoc networks. In *The 5th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, october 2002. Honolulu.
- [5] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.
- [6] ETSI STC-RES10 Committee. Radio equipment and systems : High performance radio local area network type 1, functional specifications. Technical Report ETS 300-652, ETSI, December 1995.
- [7] U. Feige. A threshold of $\ln n$ for approximating set cover. *JACM*, 1998.
- [8] P. Felber and E.W. Biersack. Self-scaling networks for content distribution. In *Proceedings of the International Workshop on Self-* Properties in Complex Information Systems (Self-*)*, 2004.
- [9] A.T. Gai and L. Viennot. Prefixstream : A balanced, resilient and incentive peer-to-peer multicast algorithm. Technical report, Inria, 2005.

- [10] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot. Performance analysis of olsr multipoint relay flooding in two ad hoc wireless network models. In *The second IFIP-TC6 NETWORKING Conference*, may 2002. Pise.
- [11] Philippe Jacquet, Pascale Minet, Paul Muhlethaler, and Nicolas Rivierre. Increasing reliability in cable-free radio LANs : Low level forwarding in HIPERLAN. *Wireless Personal Communications*, 4(1) :65–80, January 1997.
- [12] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9) :1249–1268, 1989.
- [13] A. Laouiti, A. Qayyum, and L. Viennot. Multipoint relaying : An efficient technique for flooding in mobile wireless networks. In *35th Annual Hawaii International Conference on System Sciences (HICSS'2001)*. IEEE Computer Society, 2001.
- [14] J. Wu and F. Dai. A generic distributed broadcast scheme in ad hoc wireless networks. In *ICDCS*, 2003.

Chapitre 4

De l'un vers tel autre

Nous considérons ici le cas le plus classique du routage : celui où il s'agit de trouver une route jusqu'à un nœud précisément identifié, les routeurs du réseau possèdent par exemple un identifiant de routeur (« routeur ID ») ou une adresse privilégiée (« main address »).

4.1 En connaissant le graphe (protocoles classiques)

Le protocole de routage historique d'Internet qui porte le nom de RIP¹ est sans doute le protocole de routage le plus simple (un seul format de paquets de contrôle échangés) et le plus élégant algorithmiquement. Il s'agit d'une version distribuée de l'algorithme de plus courts chemins de Bellman-Ford [1, 9].

Routage par vecteur de distances (RIP)

1. Chaque nœud échange régulièrement avec ses voisins un vecteur de distances spécifiant, pour chaque destination connue, sa distance estimée en nombre de sauts.
2. Pour atteindre une destination, un nœud choisit son voisin le plus proche de la destination et estime sa distance à 1 de plus que celui-ci.

¹ RIP : « Routing Information Protocol »

La validité de cet algorithme est analysée plus en détail au paragraphe 6.1. Le principal défaut de ce protocole est sa faible tolérance au dynamisme du réseau. Ainsi si un nœud tombe en panne, ses voisins vont successivement se choisir les uns les autres comme prochain saut pour l'atteindre. Ils ne détecteront la défaillance du nœud que lorsque la distance estimée atteint l'infini (qui vaut typiquement 16 pour ce protocole). C'est le problème du comptage à l'infini (« count to infinity »). Outre son temps de convergence un peu lent, le principal problème réside dans l'existence de boucles durant toute sa durée. Il existe différentes techniques pour réduire les cas de comptage à l'infini. La plus robuste consiste à échanger des vecteurs de chemins plutôt que de simples vecteurs de distance. Cela offre l'avantage de pouvoir détecter immédiatement les boucles qui sont ainsi évitées. Cette variante est notamment utilisée dans le protocole BGP² qui est le protocole de routage utilisé entre les systèmes autonomes d'Internet.

Routage par vecteur de chemins (BGP)

1. Chaque nœud échange avec ses voisins un vecteur de chemins spécifiant, pour chaque destination connue, le chemin qui lui permet de l'atteindre.
2. Pour atteindre une destination, un nœud u choisit son voisin de plus court chemin jusqu'à la destination (les chemins qui contiennent u sont ignorés). Le chemin jusqu'à la destination s'obtient en ajoutant l'identifiant du voisin au début de son chemin.

Pour le routage à l'intérieur d'un réseau, un nouveau protocole est venu supplanter RIP, son nom est OSPF³. L'idée consiste à donner la connaissance du graphe de connexion à tous les nœuds du réseau. Il en résulte un protocole assez complexe (toute la base de données représentant le réseau doit être spécifiée) mais à l'algorithme extrêmement simple : chaque nœud calcule les plus courts chemins localement par un algorithme séquentiel (comme celui de Dijkstra [8] par exemple).

Routage par états de liens (OSPF)

1. Chaque nœud découvre ses voisins par l'échange régulier de messages « hello ».

² BGP : « Border Gateway Protocol »

³ OSPF : « Open Shortest Path First »

2. Chaque nœud diffuse régulièrement dans le réseau la liste de ses voisins.
3. Chaque nœud calcule un arbre de plus courts chemins vers toutes les destinations connues pour construire sa table de routage.

En cas de panne, le nœud qui détecte la perte d'un lien peut immédiatement diffuser dans tout le réseau une nouvelle liste de voisins qui invalide la précédente. (Ce sont ces listes de voisins qui sont appelées états de liens et diffusée dans des messages du nom de « link state advertisement ».) Ainsi, tous les nœuds peuvent rapidement mettre à jour leur vision du réseau et leur table de routage. Toute la difficulté d'OSPF réside dans la cohérence des visions du réseau de chaque nœud. Pour cela, la gestion des numéros de séquence, qui permettent d'identifier la liste de voisins la plus à jour, est critique.

Enfin, il faut noter que les deux protocoles de routage ci-dessus reposent sur l'hypothèse que les liens sont symétriques. En effet, dans RIP, un routeur va utiliser comme prochain saut un routeur dont il a reçu un message annonçant la destination. Dans OSPF, seuls les liens symétriques sont diffusés.

4.2 En connaissant un sous-graphhe

Les algorithmes ci-dessus calculent des plus courts chemin du graphe de connexion entier. Dans certains cas, il peut être intéressant de ne calculer les routes que d'après un sous-graphe. C'est par exemple le cas des réseaux ad hoc où le graphe de connexion peut être extrêmement dense.

4.2.1 Routage ad hoc par états de liens

Dans OLSR [3], nous proposons une adaptation du principe de OSPF aux réseaux ad hoc. Considérons tout d'abord une analyse rapide du coût qu'aurait OSPF en ad hoc. Un réseau ad hoc peut posséder $O(n^2)$ liens. Si chaque nœud diffuse régulièrement la liste de ses voisins, on obtient un trafic en $O(n^3)$. Nous avons déjà vu comment l'utilisation de la diffusion par multipoints relais permet de multiplier le coût de la diffusion par un facteur m/d (dans le cas d'un réseau très dense, ce facteur peut atteindre $\log n/n$). De plus, le protocole OLSR propose de ne diffuser que la liste des multipoints relais sélecteurs, ce qui permet encore de gagner un facteur m/d . Les multipoints relais sélecteurs d'un nœud sont l'ensemble de ses voisins

qui l'ont choisi pour multipoint relai. Chaque noeud calcule donc ses routes sur le graphe orienté constitué par l'ensemble des liens multipoint relai vers multipoint relai sélecteur et l'ensemble des liens vers ses voisins :

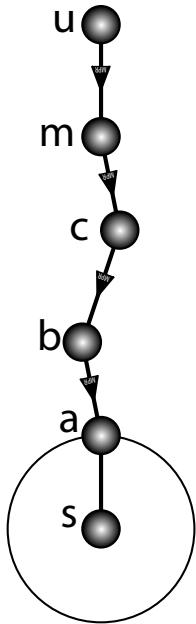


FIG. 4.1

Routage par multipoints relais (OLSR)

1. Chaque noeud émet régulièrement un message « hello » avec la liste de ses voisins et de ses multipoints relais.
2. Chaque noeud diffuse régulièrement par inondation par multipoints relais la liste des voisins qui l'ont pris pour multipoint relai.
3. Pour construire sa table de routage, chaque noeud calcule un arbre de plus courts chemins sur le sous-graphe constitué par les liens vers ses voisins et les liens multipoint relai – noeuds qui sont diffusés dans le réseau.

Notons que les liens noeud – multipoint relai sont utilisés en sens inverse par rapport aux diffusions multipoint relai ainsi qu'illustré par la figure 4.1 qui montre une route calculée par la source s pour atteindre u .

Même si les routes sont calculées sur un sous-graphe, on peut montrer que ce sont des routes de plus court chemin pour le graphe de connexions entier. Donnons-en la preuve ici : montrons par récurrence sur d que tout noeud à distance d est atteint par une route optimale. Pour $d = 1$, cela provient de l'inclusion des liens avec les voisins dans la topologie. Supposons la propriété vraie pour $d \geq 1$ et considérons un noeud u à distance $d + 1 \geq 2$. u possède forcément un voisin v à distance d et v un voisin w à distance $d - 1$. u doit donc posséder un multipoint relai m qui couvre w . Comme m est à distance au plus d , l'hypothèse de récurrence nous permet de conclure à l'existence d'une route optimale de longueur $d + 1$ jusqu'à u en passant par m . Cela termine la preuve.

Nous touchons là à un des aspects les plus élégants d'OLSR, les multipoints relais permettent à la fois d'optimiser la diffusion et de définir une sous-topologie suffisante pour la construction de routes optimales. Cette optimisation a néanmoins un défaut par rapport à OSPF : un noeud ne peut pas calculer la table de routage d'un autre noeud dont il ne connaît pas le voisinage. De tels calculs sont, par exemple, utilisés dans MOSPF⁴, le protocole de routage multicast basé sur OSPF.

⁴ MOSPF : « Multicast extensions to OSPF »

On peut aussi se poser la question à l'envers : quelles sont les sous-topologies qui permettent d'obtenir des routes optimales ? L'inclusion des liens avec les voisins est nécessaire. Considérons un noeud u à distance 2, il est nécessaire qu'un lien avec un des voisins soit connu. Si l'on suppose que tous les noeuds connaissent les même liens pour les noeuds à plus de deux sauts (comme ces liens doivent être diffusés, cette hypothèse est raisonnable), l'ensemble des liens qui connectent à u définit nécessairement un ensemble de multipoints relais pour u (tout noeud à distance 2 de u peut atteindre u par l'un de ces liens). La notion de multipoint relai apparaît donc de manière intrinsèque dans le problème de diffuser une sous topologie la plus petite possible qui soit suffisante pour construire des routes optimales.

On pourrait néanmoins pousser l'optimisation plus loin dans le contexte du routage de proche en proche (« hop by hop »). En effet, il n'est pas nécessaire qu'un noeud connaisse une route optimale pour que le paquet suive néanmoins une route optimale. Il suffit simplement que le noeud suivant sur la route soit un saut moins loin de la destination. S'il ne semble pas évident de définir un protocole sur ce principe, une approche similaire consiste à diffuser les états de liens plus fréquemment à courte distance qu'à longue distance. Pour cela, il suffit d'émettre régulièrement des messages d'états de liens avec des TTLs croissants. L'optimisation se fait alors au détriment de la fraîcheur des informations et donc de l'optimalité des routes. L'idée est que la vision de la topologie doit être de plus en plus fine à mesure que l'on se rapproche de la destination. Voir [14] pour une analyse de cette technique.

Expliquons enfin une dernière subtilité d'OLSR : pourquoi les noeuds diffusent-ils leur liste de multipoints relais sélecteurs plutôt que simplement leur liste de multipoints relais ? La raison est de pouvoir réagir efficacement lorsqu'un lien casse. La cassure d'un lien doit être répercutee sur la vision générale de la topologie assez rapidement lorsqu'il s'agit d'un lien entre un noeud et un multipoint relai. Dans ce cas, le multipoint relai sélecteur peut immédiatement diffuser une nouvelle liste où le lien cassé n'apparaît plus, ce qui l'invalidera dans les bases de données des autres noeuds. Toute source de trafic qui utilise ce lien est assurée de recevoir cette information car le message diffusé va suivre la route dans le sens inverse. Ainsi, dans l'exemple de la figure 4.1, si le lien bc casse, b pourra envoyer immédiatement une nouvelle liste de MPR sélecteurs ne contenant pas c que la source s de traffic vers u est assurée de recevoir. Si l'invalidation du lien était diffusée par le multipoint relai sélecteur plutôt que le multipoint relai, on ne serait pas assuré de la propagation du message jusqu'aux sources de trafic utilisant le lien. Il faudrait

d'abord que le noeud recalcule ses multipoints relais, informe ses nouveaux multipoints relais (par un message « hello ») avant de pouvoir effectuer une diffusion qui atteigne les sources de trafic concernées. De plus, si jamais le réseau est déconnecté par la cassure du lien, le noeud n'a aucune chance de pouvoir les contacter.

Dans [5, 4], nous comparons par des simulations les performances de OLSR avec celles de protocoles réactifs tels que AODV qui est décrit un peu plus loin au paragraphe 4.3. Dans [6], nous étudions le compromis entre l'optimisation de OLSR par la réduction de la sous-topologie diffusée (en faisant varier le paramètre de « MPR coverage » décrit au paragraphe 3.2) et la fiabilité de l'acheminement des paquets obtenus.

4.2.2 Routage petit monde

Jon Kleinberg [10] a donné une vision algorithmique des petits mondes qu'il serait dommage de ne pas citer dans ce chapitre. La notion de « petit monde » (dans le domaine des graphes) est due à Milgram [12], un sociologue qui mena une expérience sur le graphe des connaissances entre les hommes. Il donna quelques lettres mentionnant un destinataire, sa ville de résidence et son métier à des personnes choisies au hasard. La consigne pour elles était de faire passer la lettre à une connaissance susceptible de connaître le destinataire. Environ 40 % des lettres arrivèrent avec une moyenne de 5 ou 6 sauts. La conclusion de Milgram fut que le monde était effectivement petit. Watts et Strogatz [15] tentèrent de modéliser les propriétés des graphes petits mondes, mais Kleinberg donna une vision algorithmique [10] de l'expérience de Milgram qui est aujourd'hui à l'origine de toute une série de résultats tournant autour du routage.

Ce qui surprend le plus Kleinberg dans l'expérience de Milgram n'est pas tant l'existence de chemins courts mais le fait qu'on arrive à les trouver. En effet, si le graphe était aléatoire, il existerait des chemins courts mais difficiles à trouver. Pour expliquer cela, il modélise le graphe des connaissances par un tore de dimension d (la profession, le lieu de résidence, ... peuvent être vus comme une position dans un espace de dimension d qui modélise les interactions sociales). De plus, il suppose l'existence de liens aléatoires (dûs aux rencontres fortuites de la vie). Le graphe est ainsi constitué d'un graphe régulier connu de tous plus des liens aléatoires propres à chacun. Kleinberg définit ainsi l'algorithme de routage glouton suivant :

Routage petit monde

1. Parmi tous ses voisins dans le tore plus ses voisins aléatoires, un nœud fait passer le message à celui qui est le plus près de la destination pour la distance dans le tore.

Kleinberg montre que si chaque nœud a un lien aléatoire et que ce lien aléatoire mène à un nœud à distance x avec probabilité proportionnelle à $1/x^d$, alors l'algorithme de routage ci-dessus mène à destination en $O(\log^2 n)$ étapes avec forte probabilité. Le lien aléatoire doit être choisi avec cette probabilité précise car le nombre de nœuds à distance au plus x de la destination est de l'ordre de x^d .

Notons que la distance dans le tore à la destination diminue toujours. De plus, un nœud à distance $2x$ de la destination a une probabilité $\Theta(1)$ d'avoir un lien vers un nœud à distance inférieure à x de la destination. Au bout de $\log n$ étapes, la probabilité de trouver un tel lien devient forte, ce qui donne la complexité globale en $O(\log^2 n)$. Une fois qu'on a atteint un nœud à distance au plus $O(\log n)$ de la destination, le routage petit monde se comportant au moins aussi bien que le routage dans le tore, on est assuré d'atteindre la destination en $O(\log n)$.

Une application naturelle de ce paradigme réside dans la construction de tables de hachages distribuées (voir le paragraphe 5.4). D'autre part, il reste un graphe célèbre où il reste difficile de trouver des chemins. Il s'agit du graphe du web. On sait qu'il existe des chemins courts entre les pages web de la composante fortement connexe géante [2], mais on ne sait pas comment les trouver sans explorer la composante entière. Utiliser le routage à la Kleinberg dans ce contexte semble une piste intéressante.

4.3 Sans rien connaître du graphe (routage réactif)

L'inondation qui permet d'atteindre tous les nœuds d'un réseau dont on ignore la topologie peut être utilisée pour atteindre un nœud particulier. Ce mécanisme peut ensuite être complété et optimisé pour cette utilisation.

Construction de routes par inondation

L'inondation est ainsi utilisée de manière séduisante dans les protocoles de routage ad hoc dits réactifs : toute route est construite à la demande lorsque le premier paquet est envoyé. La source déclenche alors une inondation. Lorsque la destination reçoit le message, une route est établie : la suite des nœuds qui ont retransmis le message de la source à la destination. Cette approche est très séduisante dans un réseau où il y a peu de routes actives. Dans le cas où une seule route serait créée dans le réseau par exemple, une seule diffusion est nécessaire. On imagine mal un protocole de routage générant moins de trafic que cela.

Si chaque nœud retient par qui il a reçu un message d'inondation initié par un nœud origine, on obtient un arbre enraciné au nœud origine. Une inondation permet donc de construire des routes de n'importe quel nœud vers le noeud origine. Dans un réseau de n nœuds, n inondations sont donc suffisantes pour construire des routes de tout nœud vers tout nœud avec un coût de n^2 messages.

En présence de liens unidirectionnels, cette technique soulève un petit problème technique⁵ : si l'inondation passe par un lien unidirectionnel, la route construite peut le contenir et sera inutilisable pour un trafic de données. Dans certaines configurations, il se peut que l'inondation passe toujours en priorité par un lien unidirectionnel. Dans la figure 4.2 ci-contre par exemple, le noeud c recevra toujours un message inondé par a par le lien unidirectionnel $a \rightarrow c$ avant de le recevoir par b . Aussi la route $abcd$ ne pourra jamais être construite. Deux solutions sont alors possibles : détecter les liens unidirectionnels ou mettre sur une liste noire un nœud qu'on suspecte d'être à l'origine d'un lien unidirectionnel (les messages en sa provenance sont alors ignorés).

Construire les routes par inondation soulève un autre handicap : on peut obtenir des routes non optimales en nombre de sauts, ce qui peut coûter très cher en bande passante puisque les paquets de données transitant sur la route devront être émis une ou plusieurs fois de plus qu'en utilisant une route optimale. Dans [7], nous proposons une analyse permettant d'estimer la longueur moyenne des routes obtenues par inondation dans le cas d'un réseau unidimensionnel de forte densité (une route embouteillée de véhicules futuristes

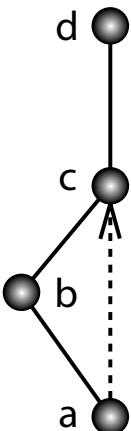


FIG. 4.2

⁵Ce point a été relevé par Philippe Jacquet, que l'honneur lui en revienne au moins dans ce document à défaut des spécifications des protocoles qui ont intégré ses remarques.

par exemple). Nous avons ainsi pu montrer que les routes par inondation sont en moyenne $4/3$ plus longues que l'optimal. Dans le cas bidimensionnel, le problème est encore ouvert, les simulations laissent conjecturer un facteur $5/3$ dans le modèle du graphe de disques unitaires avec une distribution continue de noeuds. Citons le premier pas [11] de Dmitri Lebedev et Jean-Marc Steyaert qui analysent le cas bidimensionnel quand l'inondation n'est propagée que dans le quartant où se trouve la destination recherchée.

Pour obtenir des routes de longueur optimale, nous avons de plus proposé une technique d'inondation plus lourde [7] : si un message est reçu une nouvelle fois mais en ayant fait un nombre de sauts moindre, alors il est à nouveau retransmis. Cette technique garantie l'optimalité des routes obtenues si la destination choisit la route empruntée par un paquet d'inondation ayant fait un minimum de sauts. Les simulations montrent que le nombre d'émissions s'en trouve multiplié par deux environ sur des scénarios classiques.

Dans la philosophie originelle des concepteurs de protocoles réactifs, il fallait éviter tout trafic de contrôle en cas d'inactivité du réseau. Mis à part de basses raisons de furtivité militaire, on comprend mal l'intérêt de ce précepte. En effet, cela n'empêche pas les batteries des récepteurs en écoute de se vider, cela économise de la bande passante lorsqu'il n'y a pas de trafic, et surtout cela interdit d'utiliser une technique de diffusion plus optimisée que la brutale inondation (voir le chapitre 3).

Inondation par TTLs croissants

Notons tout de même une optimisation proposée par certains protocoles réactifs comme AODV⁶ [13] sous la dénomination « expanding ring search » : limiter le rayon de l'inondation par le champ TTL. La source tente ainsi une suite d'inondations de plus en plus larges par des TTL de plus en plus grands jusqu'à recevoir une réponse de la destination.

L'intérêt de cette technique n'est pas évident a priori. Développons ici cette question puisqu'elle est livrée sans plus d'explication dans le protocole AODV. Analysons tout d'abord le cas où l'on effectue une inondation de TTL initial d suivi d'une inondation de TTL initial maximal si aucune réponse n'est reçue après la première inondation. Notons n_d le nombre de noeuds atteints par la première inondation, et e_d le nombre d'émissions produites par la première inondation. Pour une destination atteinte au premier essai,

⁶ AODV : « Ad hoc On demand Distance Vector routing »

on observe donc e_d émissions et $n + e_d$ sinon. Le nombre moyen d'émissions pour une destination aléatoire du réseau sera donc

$$\frac{n_d}{n}e_d + \frac{n - n_d}{n}(e_d + n) = n + e_d - n_d$$

La technique est donc utile dès que $e_d < n_d$, ce qui est à priori vérifié pour tout d inférieur à l'excentricité du nœud source. Le gain provient du fait qu'une inondation ainsi bornée atteint plus de noeuds qu'elle ne produit d'émissions.

Par contre, plusieurs phases d'inondation successives ne permettent pas forcément d'améliorer encore les performances. En effet, supposons que l'on tente des inondation de TTLs d , puis f , puis TTL maximal. Dans ce cas, le nombre moyen d'émissions devient :

$$\frac{n_d}{n}e_d + \frac{n_f - n_d}{n}(e_d + e_f) + \frac{n - n_f}{n}(e_d + e_f + n) = n + \left(e_d - \frac{n_d}{n}e_f\right) + e_f - n_f$$

Ce qui offre un gain par rapport à la stratégie en deux phases avec TTL f puis TTL maximal seulement quand $\frac{e_d}{n_d} < \frac{e_f}{n}$. Dans un modèle de répartition aléatoire uniforme de disques unitaires où $e_d = \mu\pi d^2$ et $n_d = \mu\pi(d+1)^2$ (μ est la densité de nœuds)⁷, on obtient la condition $\frac{f}{g+1} > \frac{d}{d+1}$ avec $n = \mu\pi(g+1)^2$ (g est le TTL minimal permettant d'atteindre tous les nœuds). Comme $e_f - n_f$ n'est alors négatif que pour $f \leq g$, le gain n'est observé que pour d petit et f proche de g . On imagine donc mal que cette technique puisse bénéficier de plus de trois ou quatre phases. La spécification d'AODV propose quatre phases avec TTLs 1, 3, 5 et TTL maximal. Pour $d = 3$ et $f = 5$, la condition garantissant un gain de la technique donne $g \leq 6$, ce qui correspond certainement aux scénarios utilisés dans les simulations qui ont permis de mettre au point ces paramètres. Une grande difficulté de la technique réside dans l'ignorance de g a priori. Commencer par une inondation de TTL 1 optimise néanmoins le processus dans presque tous les cas.

Discussion

Trouver son chemin est sans doute un très vieux problème que l'on pourrait faire remonter à des histoires de Minotaure. J'espère montrer dans ce

⁷Pour être plus précis, il faudrait tenir compte du facteur r de non-optimalité des routes obtenues par inondation que l'on peut estimer compris entre 3/5 et 1 d'après le paragraphe précédent. On obtient alors $e_d = \mu\pi(rd)^2$ et $n_d = \mu\pi(rd+1)^2$. Cela ne change pas sensiblement l'analyse qui est faite par la suite.

4.3. SANS RIEN CONNAÎTRE DU GRAPHE (ROUTAGE RÉACTIF) 63

document qu'il est toujours besoin de le revisiter. En ce qui concerne les réseaux, il faut cependant noter qu'on ne sait pas en général identifier la destination explicitement mais plutôt par une propriété qu'elle doit vérifier, ce qui est le sujet du chapitre suivant.

Bibliographie

- [1] Richard Bellman. On a routing problem. *Quaterly of Applied Mathematics*, 16(1), 1958.
- [2] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 2000.
- [3] T. Clausen, P. Jacquet (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlthaler, A. Qayyum, and L. Viennot. Optimized link state routing protocol (olsr). RFC 3626, October 2003. Network Working Group.
- [4] T. Clausen, P. Jacquet, and L. Viennot. Comparative study of cbr and tcp performance of manet routing protocols. In *Workshop MESA*, 2002. ETSI.
- [5] T. Clausen, P. Jacquet, and L. Viennot. Comparative study of routing protocols for mobile ad hoc networks. In *Med-hoc-Net*, september 2002. Sardaigne.
- [6] T. Clausen, P. Jacquet, and L. Viennot. Investigating the impact of partial topology in proactive manet routing protocols. In *The 5th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, october 2002. Honolulu.
- [7] T. Clausen, P. Jacquet, and L. Viennot. Optimizing route length in reactive protocols for ad hoc networks. In *Med-hoc-Net*, september 2002. Sardaigne.
- [8] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1959.
- [9] Lester Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

- [10] J. Kleinberg. The small-world phenomenon : An algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
- [11] D. Lebedev and J.M. Steyaert. Path lengths in ad hoc networks. In *IWWAN*, 2004.
- [12] S. Milgram. The small world problem. *Psychology Today*, 1967.
- [13] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. RFC 3561, July 2003. Network Working Group.
- [14] C. Santivanez, R. Ramanathan, and I. Stavrakakis. Making link-state routing scale for ad hoc networks. In *Proc. 2001 ACM Int'l Symp. Mobile Ad Hoc Net. Comp.*, 2001.
- [15] D.J. Watts and S.H. Strogatz. Small world. *Nature*, 1998.

Chapitre 5

De l'un vers celui qui

Dans de nombreux cas, les routeurs ne connaissent pas toutes les destinations possibles du réseau. Dans ce cas, le problème consiste généralement à router vers le (ou un) routeur qui connaît la destination. La destination peut être spécifiée de diverses manières : son adresse, celle qui possède le fichier avec tel identifiant ou encore celle qui possède un fichier avec tel ou tel mot. La découverte de service s'apparente aussi à cette problématique de routage.

5.1 Routage dans Internet

Internet possède des milliards de destinations possibles (il y a 2^{32} adresses IP possibles) et des centaines de millions de machines sont connectées au réseau. Cependant, les routeurs des systèmes autonomes d'Internet¹ ont moins de deux cents mille entrées dans leur table de routage (ce qui est déjà difficile à gérer quand des milliers de paquets arrivent à la seconde). Pour arriver à cette réduction, la façon dont les identifiants de destinations (les adresses IP²) sont alloués est fondamentale.

Pour cela, les adresses sont regroupées par préfixes : l'ensemble des adresses commençant par un préfixe p donné sont représentées par ce préfixe. Par exemple, $p = 128.93.0.0/16$ représente toutes les adresses dont les 16 premiers bits sont communs avec ceux de l'adresse 128.93.0.0 (une adresse de 32 bits est souvent écrite en base 256). Ainsi quand une organisation connecte un ensemble de machines à Internet, elle réserve un ensemble d'adresses de

¹Voir le paragraphe 2.2 pour une description d'Internet.

² IP : « Internet Protocol »

même préfixe et n'annonce que ce préfixe aux routeurs de plus haut niveau. Idéalement, on pourrait faire des regroupements de plus haut niveau comme regrouper toutes les adresses accessibles par un système autonome donné sous le même préfixe. Cela n'est envisageable d'un point de vue pratique qu'avec IPv6³.

Il y a environ 160 000 préfixes à l'heure actuelle [8]. Ce nombre pourrait être réduit à 100 000 si certains routeurs BGP agrégeaient mieux leurs préfixes. Le nombre de systèmes autonomes étant d'environ 20 000, ce nombre de préfixes pourrait idéalement descendre à 20 000 en n'attribuant qu'un seul préfixe par système autonome.

Le routage consiste alors à envoyer un paquet destiné à une adresse a donnée au noeud qui annonce le plus long préfixe de a . Un algorithme critique dans un routeur consiste donc à trouver dans un ensemble de préfixes celui qui a le plus de bits communs avec une suite de bits donnée.

5.2 Routage pair à pair

Le partage de fichiers de pair à pair consiste dans un premier temps à résoudre le problème de l'indexation distribuée des fichiers mis à disposition par les pairs de manière distribuée grâce à des techniques de routage. Notons tout d'abord qu'il existe deux manières de rechercher un fichier dans ce contexte. Soit l'on connaît l'identifiant du fichier et le problème est alors très similaire au routage dans Internet. Soit l'on cherche un fichier dont le nom contient les mots d'une requête (comme lorsqu'on effectue une recherche sur le « web »). La principale différence avec le routage classique provient du fait qu'un fichier peut être présent chez de nombreux pairs.

5.3 Gnutella et inondation optimisée par filtre de Bloom

Gnutella est le premier réseau à avoir proposé un système de recherche de fichier totalement décentralisé. Dans sa première version (v0.4), les noeuds se connectent entre eux de manière plus ou moins ad hoc et les requêtes sont inondées dans le réseau logique ainsi formé. Lorsqu'un noeud possède un

³ IPv6 : « Internet Protocol version 6 »

fichier qui répond à une requête qui lui parvient, il y répond par le chemin inverse suivi par la requête. (Il ne répond pas directement pour des raisons de confidentialité.) La technique est donc tout à fait similaire à la construction de routes de manière réactive en ad hoc (voir le paragraphe 4.3) et repose donc sur le mécanisme d’inondation.

Cependant, cette architecture souffre de l’hétérogénéité des connexions des noeuds : ceux qui ont une connexion lente ralentissent les autres. Pour palier ce problème, la génération suivante (FastTrack aussi nommé Kazaa et Gnutella v0.6) utilise une organisation du réseau logique en deux niveaux : les super-pairs ou « ultra-peer » en anglais (qui ont une connexion plus rapide) et les autres. Les noeuds de faible capacité doivent trouver un super-pair à qui se relier. (Le réseau Edonkey utilise une architecture plus différenciée encore avec des serveurs dédiés en place de super-pairs.) Les super-pairs sont reliés entre eux à la manière de Gnutella v0.4. Dans ce cas, l’inondation ne propage que dans le réseau logique reliant les super-pairs.

Gnutella v0.6 propose une utilisation des filtres de Bloom [2] assez astucieuse pour limiter la propagation de l’inondation. Un filtre de Bloom permet de coder un ensemble de manière compacte (de l’ordre de $O(n)$ bits pour un ensemble de n éléments). La structure de données permet uniquement de tester si un élément donné est dans l’ensemble avec une probabilité de faux positif bornée⁴. Gnutella utilise ainsi des filtres pour coder l’ensemble des mots des noms des fichiers possédés par un noeud. En faisant l’union des filtres de ses voisins, un noeud peut ainsi fournir un filtre des mots accessibles à un saut. En s’échangeant leurs filtres, les super-pairs peuvent ainsi construire un filtre de l’ensemble des mots accessibles à deux sauts et ainsi de suite. En testant les mots d’une requête de TTL donné, avec le filtre associé au nombre de sauts correspondant, un noeud peut ainsi éviter de propager la requête sur certains liens. Les filtres permettent, à l’inverse des tables de routage, d’indiquer les liens qui ne mènent pas à la destination. Bien sûr, pour un nombre de sauts trop important, les filtres peuvent devenir saturés et répondre presque toujours affirmativement aux tests d’appartenance. Ils ne permettent donc de réduire l’inondation des requêtes que sur les derniers sauts.

Pourrait-on utiliser des mécanismes similaires en routage ad hoc, notam-

⁴Par exemple, pour obtenir une probabilité de faux positif de 0.02 il suffit de $8n$ bits pour coder un ensemble de n éléments. Une solution alternative consisterait à stocker un « hash » de l’identifiant de chaque élément, ce qui nécessite $\Omega(n \log n)$ bits au total pour garantir une probabilité de faux positif constante.

ment en IPv6⁵ où les adresses sont 4 fois plus longues qu'en IPv4⁶ (la version classique de IP) ? Il est possible de représenter dans les limites de la taille d'un paquet radio (de l'ordre de 1 Ko⁷) un ensemble de plusieurs centaines d'éléments. Il est donc parfaitement envisageable de diffuser dans le réseau un ensemble codant le voisinage (voire le voisinage à deux sauts) de chaque nœud. La technique pourrait donc facilement être adaptée dans les protocoles de routage réactif qui utilisent de la détection de voisinage pour limiter le coût des inondations. Pour l'exercice de style, proposons ici un nouveau protocole de routage ad hoc à base de filtres de Bloom :

Routage ad hoc par état de liens à la Bloom-Gnutella

1. Chaque nœud découvre ses voisins par l'échange régulier de messages « hello ».
2. Un ensemble dominant connexe est élu (à la Wu et Li généralisé par exemple, voir le paragraphe 3.3).
3. Chaque nœud de l'ensemble dominant diffuse ses voisins appartenant à l'ensemble dominant ainsi qu'un filtre de Bloom codant l'ensemble de ses voisins.
4. Chaque nœud construit une table de routage pour atteindre les nœuds de l'ensemble dominant en calculant un arbre de plus court chemin sur la topologie diffusée.
5. Pour envoyer un paquet à une destination, une source l'envoie à un des nœuds de l'ensemble dominant dont le filtre indique l'appartenance de la destination. (En cas de faux positif, il faudra peut-être essayer plusieurs nœuds de l'ensemble dominant avant d'en trouver un relié à la destination.)

Un tel protocole permettrait de réduire encore davantage que OLSR le nombre de nœuds diffusant régulièrement des paquets de contrôle dans le réseau (le nombre de nœuds élus multipoints relais est nettement plus grand que le nombre de nœuds d'un ensemble dominant connexe [1]).

Si ce protocole est loin d'assurer des routes optimales, il montre une propriété intéressante vis-à-vis de l'anonymat. Pour atteindre un autre nœud, il

⁵ IPv6 : « Internet Protocol version 6 »

⁶ IPv4 : « Internet Protocol version 4 »

⁷ Ko : « Kilo Octet »

faut préalablement connaître son adresse et tester si l'un des filtres de Bloom la contient. Pour limiter les calculs d'appartenance aux filtres de Bloom, le paquet peut être envoyé à un membre de l'ensemble dominant qui est voisin de la destination et qui se charge de lui faire suivre. Pour protéger leur anonymat, les membres de l'ensemble dominant connexe peuvent utiliser un identifiant factice et inclure leur propre adresse dans le filtre de Bloom codant leur voisinage. On pourrait même rendre anonymes les messages « hello ». Il suffit que chaque noeud diffuse le filtre de Bloom restreint à sa seule adresse. Le filtre de Bloom codant le voisinage se construit alors comme le « ou » bit à bit des filtres de Bloom restreints des voisins.

Optimisation de l'inondation à la Gnutella

À l'inverse, une idée naturelle serait d'utiliser les techniques d'optimisation de la diffusion issues des réseaux ad hoc pour optimiser l'inondation des requête en pair à pair (notamment pour les premiers sauts où les filtres de Bloom s'avèrent peu utiles). Pour cela, nous avons effectué des « crawls » du réseau Gnutella [4]. Malheureusement, nous avons observé peu de redondance dans les voisinages de voisins, ce qui laisse peu d'espoir sur l'efficacité des techniques utilisant la connaissance locale de la topologie. Une évolution proposée pour le protocole Gnutella consiste à n'autoriser que les inondations de TTL au plus 2 pour obliger les pairs qui veulent faire une recherche plus large à interroger directement d'autres super-pairs (qui peuvent être découverts de proche en proche). Ce fonctionnement rapprocherait le protocole de Edonkey et suit la philosophie selon laquelle l'effort doit être fourni par les noeuds qui recherchent plutôt que par ceux qui offrent leur ressources.

5.4 Tables de hachage distribuées

Dans les tables de hachage, chaque noeud possède un identifiant de m bits dans l'espace des clés de la table. La technique consiste généralement à stocker une association clé-valeur sur le noeud d'identifiant le plus proche de la clé pour une certaine métrique. Un algorithme de routage est alors nécessaire pour retrouver le noeud en charge d'une clé donnée.

La solution la plus classique qui repose sur une topologie similaire à l'hypercube consiste pour chaque noeud d'identifiant u à connaître des noeuds d'identifiants de préfixes similaires sauf pour le i^e bit (pour i variant de 1

à $\log n + O(\log \log n)$). Le routage revient ensuite à trouver des noeuds dont l'identifiant partage un préfixe commun avec la clé destination de plus en plus long. On notera par $u[1 : i]$ le préfixe de i bits de u .

Routage par préfixe dans l'hypercube

1. Chaque noeud d'identifiant u connaît des noeuds dont l'identifiant commence par $u[1 : i - 1]\bar{u[i]}$ pour $i = 1, \dots, l$.
2. Quand un noeud recherche les noeuds d'identifiants proches d'une clé w , il répète pour $i = 1, \dots, l$:
 - trouver un noeud vivant d'identifiant commençant par $w[1 : i]$,
 - pour cela, les noeuds contactés à l'étape précédente pour $i - 1$ connaissent de tels noeuds.

Si les noeuds choisissent leurs identifiants aléatoirement, la probabilité de trouver des noeuds avec un identifiant de préfixe donné de longueur $l \geq \log n + O(\log \log n)$ devient très faible. Un noeud peut donc trouver qui gère un identifiant donné en $O(\log n)$ étapes. De même, chaque noeud gère une liste de contacts de l'ordre de $O(\log n)$.

Les plus anciennes solutions de tables de hachages distribuées [7, 9] reposent sur le routage par préfixe. Chord [10], une solution populaire utilise un principe similaire en utilisant une métrique euclidienne (plutôt qu'une métrique liée au préfixe) : un noeud d'identifiant u connaît pour chaque i le noeud qui gère la clé $u + 2^{m-i}$ (pour retomber exactement dans le cadre ci-dessus, il suffirait de prendre $u \oplus 2^{m-i}$). Citons aussi une solution à base de routage petit monde (ainsi que décrit dans le paragraphe 4.2.2) : Symphony [5] où chaque noeud choisit un nombre constant de contacts aléatoirement avec une probabilité inversement proportionnelle à la distance. La probabilité d'avoir un contact avec préfixe de longueur i donnée est donc constante. Dans tous les cas, l'idée est qu'un noeud à distance x de la destination permet de trouver des contacts à distance $x/2$. Kademlia, un des protocoles les plus évolués repose directement sur le routage par préfixe.

Kademlia [6] est le premier protocole ayant donné naissance à un réseau utilisé en pratique (dans le réseau eDonkey). Pour cela, la topologie est rendue très robuste à la volatilité des noeuds en y introduisant beaucoup de redondance. Ainsi les opérations de maintenance avant le départ d'un noeud qui rendaient les protocoles précédents peu utilisables en pratique devient inutile. Le routage à la Kademlia possède une petite subtilité : il faut détecter

si les noeuds intermédiaires de la route sont encore là (puisque aucune mise à jour n'est faite quand les noeuds partent). Ceci implique de recevoir d'une manière ou d'une autre un acquittement à chaque saut. Plutôt que de faire passer le message de noeud intermédiaire en noeud intermédiaire, Kademlia préconise donc que le noeud qui effectue une recherche contacte un à un tous les noeuds intermédiaires de la route jusqu'à la source. Ceux-ci lui fournissent en retour une liste de contacts possibles pour le saut suivant.

Dans [3], nous proposons une solution similaire à Kademlia basée sur la topologie de Bruijn, ce qui permet de réduire la taille des listes de contacts générés par chaque noeud (à quelques centaines au lieu de quelques milliers pour Kademlia typiquement). Le routage dans le graphe de Bruijn consiste à décaler les bits des identifiants pour introduire ceux de la clé recherchée :

Routage par décalage dans le De Bruijn

1. Chaque noeud d'identifiant u connaît des noeuds d'identifiants proches de $0u[1 : m - 1]$ et $1u[1 : m - 1]$ (m est la longueur en bits des identifiants).
2. Quand un noeud recherche les noeuds d'identifiants proches d'une clé w , il répète pour $i = l, \dots, 1$:
 - trouver un noeud vivant d'identifiant proche de $w[i : l]u[1 : m - (l - i + 1)]$,
 - pour cela, les noeuds contactés à la phase précédente pour $i - 1$ connaissent de tels noeuds.

L'idée est de contacter des noeuds de préfixes respectifs $w[l], w[l - 1 : l], w[l - 2 : l], \dots$. Une petite difficulté du protocole réside dans le fait qu'il faut estimer borne inférieure de l , ce qui revient à estimer le nombre de noeuds dans le réseau. Cette approche permet à nouveau de trouver en $O(\log n)$ le noeud en charge d'une clé avec des listes de contacts de taille constante.

La figure 5.1 illustre la topologie obtenue pour $n = 20$ avec $k = 2$. Les arcs pleins illustrent les liens d'un noeud u vers les k noeuds les plus proches de $0u[1 : m - 1]$ et les arcs pointillés illustrent les liens vers les noeuds proches de $1u[1 : m - 1]$. La figure 5.2 illustre de même un exemple de topologie obtenue pour $n = 500$ et $k = 3$.

Une subtilité dans l'introduction de redondance consiste à décaler vers la droite durant l'algorithme de routage alors que les solutions précédentes de tables de hachage sur le graphe de Bruijn proposaient de décaler vers la

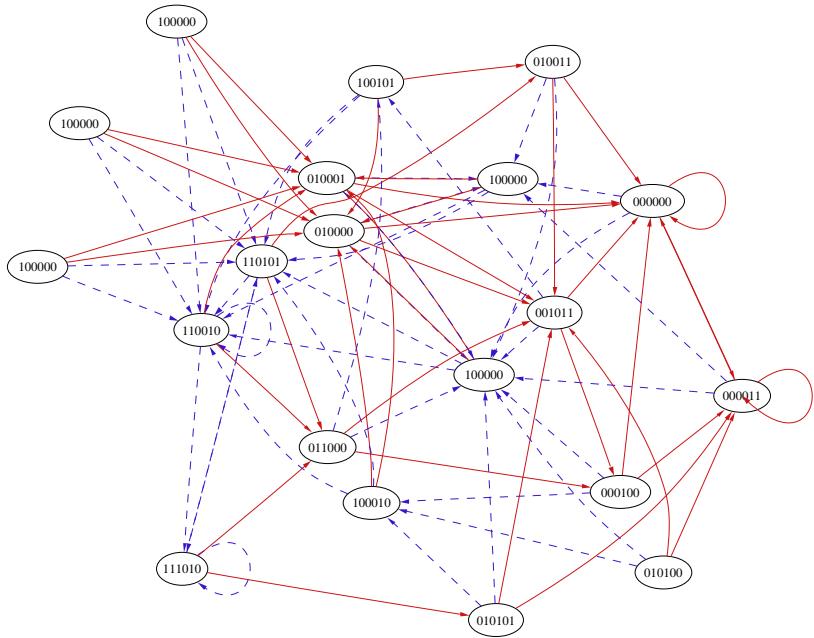


FIG. 5.1 – Une topologie de Bruijn souple avec $n = 20$ nœuds et $k = 2$.

gauche (ce qui évite d'avoir à estimer l). La raison intuitive est qu'un nœud intermédiaire u de préfixe $w[i : l]$ choisit des contacts proches de $0w[i : l]$ et $1w[i : l]$ et connaît donc certainement des nœuds de préfixe $w[i - 1 : l]$ partageant ainsi 1 bit de plus avec w . À l'inverse, si on décalait vers la gauche, un nœud dont l'identifiant partagerait i bits avec w risquerait de n'avoir que des contacts ne partageant que $i - 1$ bits avec w .

Discussion

De nombreuses applications se rapprochent plus ou moins de problématiques liées au routage. Cependant, chaque application définira à sa manière ce que l'on cherche, ce qui influe nécessairement sur la façon de le joindre.

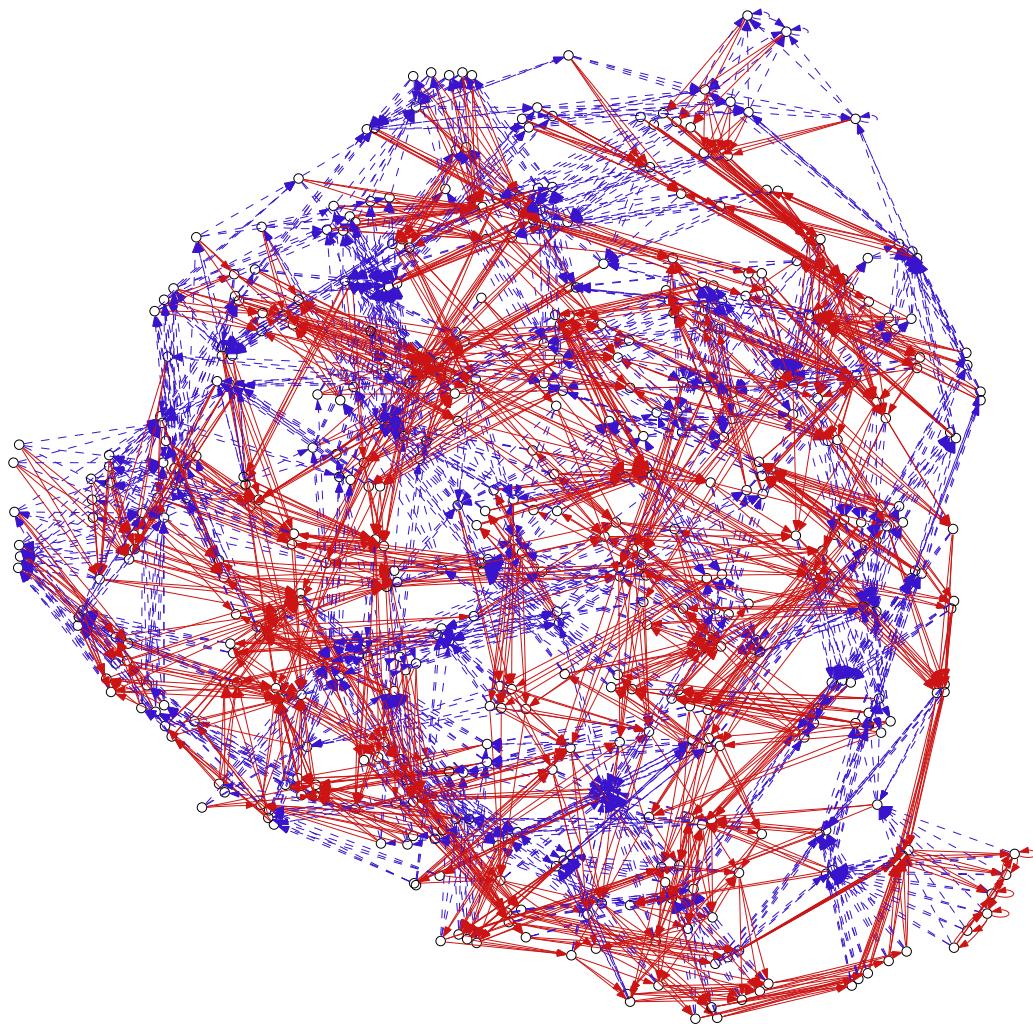


FIG. 5.2 – Une topologie de Bruijn souple avec $n = 500$ nœuds et $k = 3$.

Bibliographie

- [1] C. Adjih, P. Jacquet, and L. Viennot. Computing connected dominated sets with multipoint relays. *Ad Hoc and Sensor Wireless Networks*, 1(1-2), january 2005.
- [2] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Commun. ACM*, 13(7), 1970.
- [3] A.T. Gai and L. Viennot. Broose : A practical distributed hashtable based on the de-bruijn topology. In *The Fourth IEEE International Conference on Peer-to-Peer Computing*, 2004.
- [4] S. Le-Blond and L. Viennot. Exploration du réseau gnutella 0.6. Stage Epitech, mai 2004.
- [5] G. Manku, M. Bawa, and P. Raghavan. Symphony : Distributed hashing in a small world, 2003.
- [6] P. Maymounkov and D. Mazieres. Kademia : A peerto -peer information system based on the xor metric. In *1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
- [7] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 1999.
- [8] CIDR Reports. Analysis of bgp tables. <http://www.cidr-report.org/>.
- [9] A. Rowstron and P. Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, LNCS 2218, pages 329–350, 2001.
- [10] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications.

In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

Chapitre 6

De temps à autre

Une grande difficulté des algorithmes de réseau distribués provient de l'asynchronisme des envois de messages : le temps au bout duquel un message arrive est incertain, le fait qu'un message arrive est incertain, l'ordre dans lequel plusieurs messages envoyés arrivent est incertain. Nous allons considérer dans ce chapitre une classe d'algorithmes très générale opérant à condition que les messages soient reçus de temps à autre. Le principe consiste à opérer des itérations asynchrones (au gré de l'arrivée des messages) qui convergent vers un point fixe. Nous verrons ainsi le lien entre le routage par vecteur de distances, la synchronisation des nœuds d'un réseau ou encore le calcul de PageRank incrémental.

De nombreux problèmes de réseau consistent à maintenir un état $e_i \in E$ vérifiant certaines propriétés dans chaque nœud i du réseau. Par exemple, dans le cas du routage, il s'agit de maintenir pour chaque destination le routeur suivant à qui faire passer des paquets pour cette destination. Si cet état peut s'exprimer comme le point fixe d'une fonction F de E^n dans E^n alors il existe une méthode très générale pour maintenir cet état dans chaque nœud au moyen d'itérations asynchrones de F . On consultera [4] pour une introduction générale sur le domaine.

L'idée est de reposer sur un mécanisme de routage extrêmement simple : chaque nœud envoie régulièrement son état e_i à ses voisins. Il faut bien sûr que F s'exprime conformément aux voisinages des nœuds. Nous avons déjà vu un tel mécanisme dans la découverte de voisinage dans un réseau ad hoc avec les messages « hello ». Nous avons aussi vu un cas beaucoup moins trivial avec le routage par vecteur de distances que nous allons détailler maintenant.

6.1 Routage par vecteur de distances et Bellman-Ford asynchrone

Dans le routage par vecteur de distances, chaque routeur envoie régulièrement à ses voisins sa distance à toutes les destinations qu'il connaît. Concentrons-nous sur une destination k donnée. On ne considère qu'une seule entrée dans les vecteurs échangés car le traitement de chaque destination est indépendant. Chaque routeur i maintient à l'instant t une estimation d_i^t de sa distance à k (k est fixé ici) et connaît le temps l_{ij} qu'il met à envoyer un paquet à un voisin j . Initialement, $d_i^0 = \infty$ pour $i \neq k$ et $d_k^0 = 0$ pour tout t .

Routage par distance vers une destination k

1. Répéter régulièrement :

- (a) Si le routeur i reçoit de j l'estimation $d_j^{t_j}$, alors il la stocke dans v_j .
- (b) Le routeur i estime au temps t sa distance à

$$d_i^t = \min_{j \text{ voisin de } i} v_j + l_{ij} = \min_{j \text{ voisin de } i} d_j^{t_j} + l_{ij}$$

Un voisin qui réalise la plus petite distance est inséré dans la table de routage comme routeur suivant pour atteindre la destination k .

- (c) Le routeur i envoie régulièrement son estimation d_i^t à chacun de ses voisins.

Il faut remarquer qu'aucune hypothèse n'est faite sur la fréquence des opérations ci-dessus. Ceci permet d'affirmer que la perte de messages ne change pas l'algorithme. Du moment que le routeur i arrive régulièrement à contacter chacun de ses voisins, on peut montrer que l'algorithme va converger vers une estimation correcte des distances.

Remarquons tout d'abord la ressemblance avec l'algorithme de Bellman-Ford [3, 10] de calcul d'arbre de plus courts chemins vers une destination donnée :

Bellman-Ford [3, 10]

1. Initialiser la distance de chaque sommet $i \neq k$ à $d_i = \infty$.

2. $d_k = 0$.
3. Répéter n fois :
 - (a) Pour chaque sommet i , faire :

$$d_i = \min_{j \text{ voisin de } i} d_j + l_{ij}$$

Cet algorithme correspond donc au cas synchrone où chaque sommet effectue une itération au même temps t avec $t_j = t - 1$ pour chaque voisin. Dans le routage par vecteur de distances, on peut très bien utiliser de vieilles estimations pour certains voisins et de plus récentes pour d'autres. Certains nœuds peuvent effectuer leur itérations à une fréquence plus élevée que d'autres. Bertsekas [5] donne un cadre général pour montrer la convergence de telles itérations asynchrones.

6.2 Itération asynchrones

L'algorithme général d'itérations asynchrones pour une suite de vecteurs $X^t = (x_1^t, \dots, x_n^t) \in \mathbb{R}^n$ et une fonction $F = (f_1, \dots, f_n)$ de \mathbb{R}^n dans \mathbb{R}^n s'écrit ainsi :

Itérations asynchrones [5]

1. Pour chaque i , répéter régulièrement à un instant t :

$$x_i^t = f_i(x_1^{last_{i1}(t)}, \dots, x_n^{last_{in}(t)})$$

(on a toujours $t \geq last_{ij}(t) \geq 0$).

Un cas particulier est celui des itérations à la Gauss-Seidel où chaque x_i est calculé de manière cyclique et où la version la plus récente de x_j est toujours utilisée :

Itérations à la Gauss-Seidel

1. Répéter pour $k = 0$ à ∞ :

$$\begin{aligned} x_1^{k+1} &= f_i(x_1^k, x_2^k, \dots, x_{n-1}^k, x_n^k) \\ x_2^{k+1} &= f_i(x_1^{k+1}, x_2^k, \dots, x_{n-1}^k, x_n^k) \end{aligned}$$

$$\begin{array}{c} \vdots \\ x_n^{k+1} = f_i(x_1^{k+1}, x_2^{k+1}, \dots, x_{n-1}^{k+1}, x_n^k) \end{array}$$

Des hypothèses très minimales permettent de montrer la convergence de $X^t = (x_i^t)_{1 \leq i \leq n}$ vers le point fixe de $F = (f_i)_{1 \leq i \leq n}$ (le vecteur P tel que $P = F(P)$). Une hypothèse nécessaire est que chaque x_i soit réévalué infiniment souvent et que $\text{last}_{ij}(t) \rightarrow \infty$ quand $t \rightarrow \infty$ si f_i dépend de x_j . Pour montrer la convergence de X^t , Bertsekas suppose l'existence d'une suite décroissante d'ensembles $(E_k)_{k \geq 0}$ telle que :

- $\cap_{k \geq 0} E_k = \{P\}$,
- $X^0 \in E_0$,
- si $X = (x_1, \dots, x_n) \in E_k$, alors le vecteur $(x_1, \dots, x_{i-1}, f_i(X), x_{i+1}, \dots, x_n)$ est aussi dans E_k ,
- un vecteur X obtenu en mélangeant les coordonnées de deux vecteurs $X_1 \in E_k$ et $X_2 \in E_k$ est aussi dans E_k ,
- enfin, si $X \in E_k$ alors le vecteur obtenu par une itération synchrone $F(X)$ est dans E_{k+1} .

Un cas plus particulier de convergence est obtenu quand F est contractante : c'est-à-dire qu'elle vérifie pour tous X et Y , $|F(X) - F(Y)| \leq M|X - Y|$ où M est une matrice positive telle que $\lim_{k \rightarrow \infty} M^k = 0$ (l'inégalité est vérifiée ligne par ligne).

Les premières théories sur les itérations asynchrones datent de la paralléllisation de résolution de systèmes linéaires [8]. La technique consiste à effectuer des puissances itérées sur une matrice M . Pour résoudre $AX = B$, on utilisera ainsi $F(X) = MX + C$ où :

- $A = D - N$ avec D facilement inversible (par exemple diagonale),
- $M = D^{-1}N$ a un rayon spectral strictement inférieur à 1,
- $C = D^{-1}B$;
- $P = F(P)$ implique donc $(I - D^{-1}N)P = D^{-1}B$, ou encore $(D - N)P = B$, soit finalement $AP = B$.

Chaque processeur gère une partie des x_i et communique leurs valeurs de temps à autres aux autres processeurs. Chaque processeur effectue les itérations correspondant aux lignes qu'il gère avec les valeurs de x_j qu'il possède. Ce relâchement entre itérations est aussi connu sous le nom d'itérations chaotiques [8, 15, 2].

Un cas particulier d’itérations asynchrones provient de l’algorithme du gradient. En effet, la minimisation d’une fonction H de n variables en se déplaçant le long du gradient peut s’écrire sous forme d’itérations $X^{t+1} = (I - \epsilon G^t)X^t$ où G^t est la matrice des dérivées partielles de H (la méthode des itérations asynchrones peut être généralisée quand F dépend du temps, mais nous n’en n’aurons pas besoin ici). Ce contexte peut être vu comme point de départ de la synchronisation d’horloges des nœuds d’un réseau.

6.3 Synchronisation des nœuds d’un réseau

Il existe au moins deux cas dans lesquels un réseau radio a besoin de synchronisation. Dans le cas d’un réseau ad hoc par exemple, pour économiser de l’énergie, les nœuds peuvent avoir besoin de n’allumer leur interface radio que par intermittence ; il est alors nécessaire de synchroniser les périodes d’activité des nœuds. D’autre part, quand les communications sont planifiées dans des créneaux temporels (ou « slots » en anglais), la synchronisation permet de minimiser les recouvrements entre les slots de différents nœuds. C’est par exemple le cas des stations de base d’un réseau cellulaire de type GSM¹.

Dans [9], nous étudions ce dernier cas. Pour cela, notons θ_i l’heure de la station de base i . Cette heure est définie modulo une période T . Plus précisément, chaque fois qu’un mobile passe d’une cellule d’une station de base j voisine à la station de base i , le décalage temporel $\Delta_{ij} = \theta_i - \theta_j$ entre les deux stations est mesuré modulo T . L’algorithme de base consiste à recalcer l’horloge de la station i vers la moyenne de ses voisines en décalant régulièrement θ_i de $\epsilon \sum_{j \text{ voisine de } i} \Delta_{ij}$. On obtient ainsi l’algorithme du gradient (en version asynchrone) pour minimiser la fonction $H(\theta_1, \dots, \theta_n) = \sum_{ij \text{ voisines}} \Delta_{ij}^2$.

La difficulté du problème vient ici de l’aspect modulaire des heures. Il peut ainsi exister des situations de blocage où les horloges peuvent faire un tour de période le long d’un cycle : chaque horloge doit se recalcer dans un sens pour se synchroniser avec une voisine et dans l’autre pour se synchroniser avec l’autre. Pour sortir de telles situations, nous proposons dans [9] de briser la symétrie des cycles en utilisant un arbre couvrant, ce qui permet de synchroniser préférentiellement une horloge vers son père (en affectant par exemple un poids plus fort au décalage avec son père qu’avec les autres stations voisines). En pratique, il n’est pas possible de calculer un arbre couvrant. Cependant, en

¹ GSM : « Global System for Mobility »

désignant comme père la station voisine de plus petit identifiant, on obtient une forêt couvrante. L'existence de cycles de blocage entre arbres de la forêt paraît assez improbable.

On peut d'ailleurs pousser le principe plus loin en synchronisant préférentiellement une station sur toutes ses voisines de plus petit identifiant. Les horloges se synchronisent alors le long d'un graphe orienté acyclique qui a l'avantage d'être connexe si le graphe de détection de décalages est connexe.

Synchronisation asynchrone de réseau

1. Chaque noeud i du réseau stocke la dernière valeur mesurée de son décalage Δ_{ij} avec un noeud voisin j ,
2. et décale régulièrement son horloge de $\epsilon \sum_{j < i} \Delta_{ij}$.
3. Un noeud d'identifiant i minimal dans son voisinage se recale régulièrement de $\epsilon \sum_{j \text{ voisin de } i} \Delta_{ij}$.

Il est intéressant de noter la similitude des règles locales inventées ici et celles développées pour la construction d'ensembles dominants connexes vue au paragraphe 3.3.

Dans le cas où les variables ne sont pas modulaires, il est possible de prouver la convergence générale de tels systèmes [7] qui ont notamment été proposés pour résoudre le problème du consensus de manière asymptotique.

6.4 PageRank et routage aléatoire

Une des grandes difficultés de l'indexation du web réside dans le classement des pages. En effet, il existe souvent plusieurs milliers de pages qui contiennent les mots d'une requête donnée. La difficulté consiste donc à présenter en premier les pages les plus importantes. Pour répondre à ce besoin, Page et Brin [16] utilisèrent le graphe du web avec l'idée intuitive que les pages importantes étaient référencées par de nombreuses autres pages, et que les pages pointées par des pages importantes sont importantes aussi. Formellement, ils en arrivèrent à une définition de « PageRank » pour chaque page comme la probabilité pour un surfeur aléatoire de se retrouver sur la page. Plus cette probabilité est forte, plus la page est considérée comme importante. Pour pouvoir faire ce calcul, la matrice d'adjacence du graphe du web (celui formé par les pages qu'on indexe) est transformée en une matrice de transition markovienne irréductible et apériodique.

Dans sa version la plus simple, la probabilité d'aller de la page p vers la page q est $a_{pq} = 1/d$ où d est le degré sortant de p si p pointe vers q et 0 sinon. On définit ainsi la matrice d'adjacence normalisée $A = (a_{pq})$. (Des modifications de cette matrice sont en général nécessaires pour obtenir une matrice irréductible et apériodique, voir par exemple [16] ou [13] pour une analyse fine des détails de convergence de l'algorithme.) Le vecteur de PageRank P s'obtient en itérant

$$P^{t+1}(q) = \sum_{p \text{ pointe vers } q} a_{pq} P^t(p)$$

Cette suite de vecteurs converge vers le vecteur propre P de Perron-Frobenius associé à la valeur propre 1 de A :

$${}^T P = {}^T P A, \text{ soit } P = {}^T A P$$

Le résultat en est un algorithme d'évaluation de l'importance des pages assez efficace si on en juge le succès du moteur de recherche dont le nom commence par un G basé sur ce principe. La conclusion empirique est assez rassurante du point de vue de l'exploration du web puisqu'il dit que les pages les plus importantes sont les plus faciles à trouver puisqu'on a plus de chances de les rencontrer en suivant aléatoirement les liens, c'est-à-dire en routant aléatoirement.

6.5 PageRank distribué

Dans [14], nous montrons comment décomposer le calcul du PageRank vis-à-vis d'une partition des pages en sites. On considère qu'une partition en sites $\mathcal{S} = S_1, \dots, S_k$ des pages est donnée. On note $S(p)$ le site qui contient une page p . On peut ainsi différencier les liens entrants externes pq tels que $S(p) \neq S(q)$ et les liens entrants internes pq tels que $S(p) = S(q)$. Cette définition induit naturellement une décomposition du PageRank :

$$\begin{aligned} P(q) &= \sum_{p \text{ tel que } S(p) \neq S(q)} a_{pq} P(p) + \sum_{p \text{ tel que } S(p) = S(q)} a_{pq} P(p) \\ &= P_e(p) + P_i(p) \end{aligned}$$

Le PageRank se décompose donc en PageRank entrant externe $P_e = ({}^T A - {}^T A_{\mathcal{S}})P$ et $P_i = {}^T A_{\mathcal{S}} P$ où ${}^T A_{\mathcal{S}}$ est la matrice diagonale par bloc obtenu

de A en mettant à zéro toutes les entrées pq telles que $S(p) \neq S(q)$. On peut donc écrire :

$$(Id - {}^T A_S)P = P_e$$

Cette équation permet de définir le PageRank en fonction du PageRank entrant externe :

$$P = (Id - {}^T A_S)^{-1} P_e = \sum_i ({}^T A_S)^i P_e$$

Connaissant P_e , on peut retrouver P en itérant :

$$P^{t+1} = P_e + {}^T A_S P^t$$

Cet algorithme est intéressant car il peut être calculé localement à l'intérieur d'un site : connaissant P_e pour les pages d'un site S et la matrice d'adjacence normalisée A_S du graphe restreint au site S , il suffit d'itérer les opérations ci-dessus avec ${}^T A_S$. Cela peut être intéressant pour construire un moteur de recherche local au site qui classe ses pages selon le PageRank global (et non une version locale du PageRank qui serait moins pertinente).

La difficulté réside dans l'estimation de P_e . D'un point de vue pratique la seule information locale au site qui pourrait permettre d'estimer P_e sont les « hits » de surfeurs réels (et non aléatoires) enregistrés par le serveur du site. On peut aussi définir P_e par l'équation :

$$P_e = ({}^T A - {}^T A_S)(Id - {}^T A_S)^{-1} P_e$$

On peut montrer que la matrice $({}^T A - {}^T A_S)(Id - {}^T A_S)^{-1}$ est stochastique par colonnes², ce qui montre l'importance intrinsèque de P_e et en fait un bon candidat pour définir une notion d'importance de site. La définition d'un « SiteRank » reste cependant un domaine ouvert. La raison provient à la fois de la difficulté de définir les sites et de l'impossibilité d'explorer totalement un site.

Toujours est-il que la décomposition ci-dessus débouche sur un algorithme distribué de calcul du PageRank. On peut considérer chaque serveur web comme le site des pages qu'il sert. En coopérant entre eux, les serveurs web peuvent alors calculer le PageRank par les itérations doubles :

$$\begin{aligned} P^{t+1} &= P_e^t + {}^T A_S P^t \\ P_e^{t+1} &= ({}^T A - {}^T A_S)P^t \end{aligned}$$

En version asynchrone, on obtient l'algorithme suivant :

²L'entrée p, q de la transposée de cette matrice correspond à la probabilité pour le surfeur aléatoire d'arriver en $q \notin S(p)$ après une ballade dans $S(p)$ puis un saut vers q .

PageRank distribué

1. Chaque serveur S considère régulièrement la page $p \in S$ à un temps t .
2. Pour toute page $q \notin S$ pointant vers p , S a reçu une estimation $P^{last_q(t)}(q)$ de son PageRank ainsi que l'entrée a_{qp} de la matrice d'adjacence normalisée A .
3. S met à jour l'estimation du PageRank entrant externe de p par :

$$P_e^t(p) = \sum_{\substack{q \text{ pointe vers } p, \\ q \notin S}} a_{qp} P^{last_q(t)}(q)$$

4. S met à jour l'estimation du PageRank de p par :

$$P^t(p) = P_e^t(p) + \sum_{\substack{q \text{ pointe vers } p, \\ q \in S}} a_{qp} P^{t-1}(q)$$

5. Pour toute page $q \notin S$ pointée par p , envoyer au serveur S' qui la gère l'estimation $P^t(p)$ du PageRank de p ainsi que l'inverse a_{pq} du degré de p .

Il s'agit là d'une simple version asynchrone du PageRank. Cet algorithme nécessite la confiance entre les serveurs et ne semble applicable qu'entre les serveurs d'un site distribué. Notre modélisation permet de détecter un serveur tricheur. Le PageRank sortant doit en effet être égal au PageRank entrant :

$$\sum_{\substack{p \text{ pointe vers } q, \\ p \in S, q \notin S}} a_{pq} P(p) = \sum_{\substack{p \text{ pointe vers } q, \\ p \notin S, q \in S}} a_{pq} P(p)$$

Ceci peut se déduire simplement en remarquant que l'étiquetage des arcs pq du graphe du web par $a_{pq}P(p)$ définit un flot (le flot de PageRank tel qu'inventé par Fabien Mathieu [13]). Le flot entrant dans le site est donc nécessairement égal au flot sortant. Une possibilité d'arbitrage consisterait à redistribuer le défaut éventuel de flot sortant d'un serveur entre tous les autres serveurs. Se passer de serveur central paraît difficile dans un contexte où l'incitation à la tricherie est forte.

D'un autre côté, dissocier le calcul du PageRank de la manière dont on découvre le graphe du web n'est pas forcément désirable. De nombreux auteurs ont proposé d'utiliser le PageRank lors du « crawl » pour se concentrer plus rapidement sur les pages importantes. Mais ce sont Abiteboul, Cobena et Preda [1] qui formalisent le mieux cette idée en proposant un calcul incrémental du PageRank au fur et à mesure du « crawl ».

6.6 PageRank incrémental

Pour cela, ils considèrent que le web est « crawlé » en continu. Chaque fois qu'une page est visitée, une itération de calcul du PageRank est effectuée pour cette page.

PageRank incrémental [1]

1. Initialement, chaque page p a un historique $h_p = 0$ et un « cash » $c_p = 1/n$.
2. Répéter régulièrement pour chaque page p de degré sortant d :
 - (a) pour chaque page q pointée par p , faire $c_q = c_q + \frac{c_p}{d}$ (soit $c_q = c_q + a_{pq}c_p$),
 - (b) $h_p = h_p + c_p$,
 - (c) $c_p = 0$.
3. Le PageRank de p est estimé à $(h_p + c_p)/\sum_{q=1}^n(h_q + c_q)$.

Si toutes les pages sont visitées infiniment souvent, cet algorithme converge vers le PageRank. Il faut noter que la matrice d'adjacence n'a pas besoin d'être stockée et qu'en gardant $C = (c_p)_{1 \leq p \leq n}$ en mémoire, l'algorithme effectue un seul accès disque par page visitée.

En notant $H^t = (h_p^t)_{1 \leq p \leq n}$, $|H^t| = \sum_{1 \leq p \leq n} h_p^t$ et $C^t = (c_p^t)_{1 \leq p \leq n}$, les auteurs prouvent la convergence de $H^t/|H^t|$ vers le PageRank en établissant par récurrence :

$$H^t + C^t = C^0 + MH^{t-1} \quad (6.1)$$

Cependant cette équation masque le lien avec les algorithmes d'itérations asynchrones.

6.7 Itérations asynchrones par colonnes

Montrons ici que l'algorithme de PageRank incrémental est en fait une version asynchrone de la méthode des puissances itérées $P^{t+1} = MP^t$ avec $M = {}^T A$ la transposée de A . Cette méthode est généralisable à n'importe quelle matrice M accédée par colonnes. Écrivons ainsi une version généralisée pour un calcul d'itérations du type :

$$X^{t+1} = MX^t + S \quad (6.2)$$

On considère au temps t une colonne d'indice q_t de M (les variables sont indicées par le temps pour des besoins d'analyse ultérieure) :

Itérations asynchrones par colonnes (inspiré de [1])

1. Initialement, $c_p^0 = 0$ et $h_p^0 = 0$ pour tout $1 \leq p \leq n$.
2. Répéter pour $t = 1$ à l'infini :
 - (a) Considérer la colonne d'indice $q = q_t$ de M .
 - (b) Rajout de « cash » : faire $c_q^{t-1} = c_q^{t-1} + d_q^{t-1}$.
 - (c) Pour chaque p , faire $c_p^t = c_p^{t-1} + m_{pq} c_q^{t-1}$.
 - (d) $h_q^t = h_q^{t-1} + c_q^{t-1}$.
 - (e) $c_q^t = m_{qq} c_q^{t-1}$.
 - (f) (Pour $p \neq q$, $h_p^t = h_p^{t-1}$.)

Notons que pour être général, nous ne remettons c_q^t à zéro à l'étape 2.e que lorsque $m_{qq} = 0$ (ce qui semble être vérifié pour tout q dans [1]). Le rajout de « cash » peut être nécessaire pour une matrice de rayon spectral inférieur à 1. On peut en fait modifier le « cash » d'une colonne p juste avant une itération sur celle-ci ($d_p^{t-1} = 0$ si la colonne p n'est pas visitée au temps t). Dans le cas de [1], on a $d_p^0 = 1/n$ et $d_p^t = 0$ pour $t > 0$.

Pour un instant donné t , notons par $\text{last}_p(t)$ le dernier instant de calcul sur la colonne p , par $\text{last}_p^2(t) = \text{last}_p(\text{last}_p(t))$ l'instant précédent de calcul sur la colonne p , et ainsi de suite pour $\text{last}_p^3(t), \text{last}_p^4(t), \dots$ (Pour un temps t antérieur à la première itération sur p , on convient de $\text{last}_p(t) = 0$.) Quand on itère sur la colonne q à l'instant t , on a :

$$c_q^{t-1} = h_q^t - h_q^{\text{last}_p(t)}$$

D'autre part, le calcul des c_p nous indique qu'à tout instant t où la page p n'est pas visitée, on a :

$$c_p^t = \sum_{t \geq t' \geq \text{last}_p(t)} m_{pq_{t'}} c_{q_{t'}}^{t'-1} = \sum_{t \geq t' \geq \text{last}_p(t)} m_{pq_{t'}} \left(h_{q_{t'}}^{t'} - h_{q_{t'}}^{\text{last}_{q_{t'}}(t')} \right)$$

En décomposant h_p^t comme une somme de $c_p^{t'}$, on obtient au moment t d'une mise à jour de h_p^t :

$$h_p^t = c_p^{t-1} + c_p^{\text{last}_p(t)-1} + c_p^{\text{last}_p^2(t)-1} + \dots$$

$$\begin{aligned}
&= \sum_{t>t' \geq 0} d_p^{t'} + \sum_{t'>0} m_{pq_{t'}} \left(h_{q_{t'}}^{t'} - h_{q_{t'}}^{\text{last}_{q_{t'}}(t')} \right) \\
&= \sum_{t>t' \geq 0} d_p^{t'} + \sum_{1 \leq q \leq n} m_{pq} \left(h_q^{\text{last}_q(t)} - h_q^{\text{last}_q^2(t)} + h_q^{\text{last}_q^2(t)} - h_q^{\text{last}_q^3(t)} + \dots \right) \\
&= \sum_{t>t' \geq 0} d_p^{t'} + \sum_{1 \leq q \leq n} m_{pq} \left(h_q^{\text{last}_q(t)} - h_q^0 \right) \\
&= \sum_{t>t' \geq 0} d_p^{t'} + \sum_{1 \leq q \leq n} m_{pq} h_q^{\text{last}_q(t)}
\end{aligned}$$

On peut aussi écrire une équation similaire pour $x_p^t = h_p^t + c_p^t$ (en remarquant que $h_q^{\text{last}_q(t)} = x_q^{\text{last}_q(t)-1}$) qui a l'avantage de donner une estimation valable à tout instant t , en notant $s_p^t = \sum_{t>t' \geq 0} d_p^{t'}$:

$$x_p^t = s_p^t + \sum_{1 \leq q \leq n} m_{pq} x_q^{\text{last}_q(t)-1}$$

On retrouve bien une version asynchrone d'une équation du type $X^{t+1} = MX^t + S^t$ ($S^t = (s_p^t)_{1 \leq p \leq n}$). Quand les colonnes sont considérées de manière cyclique, on retrouve les itérations à la Gauss-Seidel. Dans le cas où S^t converge vers S , la convergence de $X^t = (x_p^t)_{1 \leq p \leq n}$ vers le point fixe solution de $X = MX + S$ est assez bien connue quand M a rayon spectral strictement inférieur à 1 [8] ou quand M a rayon spectral 1 et $S = 0$ [12].

Dans le cas du calcul du PageRank incrémental [1], $S = S^t = S^0$ et X^t croît indéfiniment, mais $X^t/|X^t|$ converge vers le PageRank P . Une solution pour avoir convergence vers un vecteur fini consisterait à retirer petit à petit le « cash » (par exemple, $d_p^{t-1} = -\epsilon C_p^{t-1}$). On aura alors $S^t \rightarrow 0$, $C^t \rightarrow 0$ et $X^t \rightarrow P$.

Pour avoir un algorithme qui tourne en continu, les auteurs du PageRank incrémental préfèrent conserver un « cash » constant ($|C^t| = 1$ pour tout t). La stochasticité par colonnes de M assure cette propriété. Pour éviter le problème de divergence de H^t , ils proposent d'utiliser une fenêtre de temps T et de considérer $Y^t = X^t - X^{t-T}$. Si les pages sont visitées assez régulièrement pour avoir $\text{last}_p(t-T) \approx \text{last}_p(t) - T$ alors on aura :

$$y_p^t = (s_p^t - s_p^{t-T}) + \sum_{1 \leq q \leq n} m_{pq} y_q^{\text{last}_q(t)-1}$$

Dans le cas d'une matrice quelconque, il suffirait de rajouter du « cash » en continu pour maintenir $s_p^t - s_p^{t-T} = s_p$ afin de maintenir Y^t sur le point

fixe solution de $Y = MY + S$. La méthode de Abiteboul, Cobena et Preda peut donc se généraliser à tous les calculs linéaires par itérations asynchrones sur les colonnes.

6.8 HITS incrémental

Kleinberg [11] a proposé une alternative au PageRank pour trouver les pages importantes du web sous le nom de HITS³. Sa méthode attribue un poids de «hub» x_p et un poids d'autorité y_p à chaque page et définit ces poids par les itérations suivantes sur la matrice A et sa transposée ${}^T A$ d'une partie du graphe du web :

$$\begin{aligned} X^{t+1} &= AY^t \\ Y^{t+1} &= {}^T A X^t \end{aligned}$$

L'idée intuitive de la modélisation est qu'il y a des pages de contenu et des pages d'index (ou « hub »). Les pages importantes de contenu sont celles qui sont pointées par des pages d'index importantes et les pages importantes d'index sont celles qui pointent sur des pages importantes de contenu... On trouvera une analyse détaillée de la convergence d'un tel système dans [6].

Réaliser une version incrémentale de l'algorithme de Kleinberg restait encore un problème ouvert jusqu'à la rédaction de ce document. Considérons plus généralement des itérations doubles :

$$\begin{aligned} X^{t+1} &= AY^t + E \\ Y^{t+1} &= BX^t + F \end{aligned}$$

La convergence d'un tel système est par exemple garantie quand $\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}$ a un rayon spectral strictement inférieur à 1.

Si les deux matrices sont obtenues ligne par ligne, on se retrouve dans le cadre des itérations asynchrones classiques. Si elles sont obtenues par colonnes alors on peut réécrire ce système sous la forme d'une seule itération asynchrone par colonne. La difficulté dans le cas de la méthode de Kleinberg provient du fait que A est obtenue par lignes alors que B est obtenue par colonnes. Proposons donc un algorithme d'itérations asynchrones doubles pour ce cas :

³ HITS : « Hypertext-Induced Topic Search »

Itérations asynchrones doubles par lignes et par colonnes

1. Initialement, $x_p^0 = 0$, $y_p^0 = 0$ et $xlast_p^0 = 0$ pour tout $1 \leq p \leq n$.
2. Répéter pour $t = 1$ à l'infini :
 - (a) Considérer la colonne d'indice $r = r_t$ de B .
 - (b) Pour chaque p , faire $y_p^t = y_p^{t-1} + b_{pr}(x_r^{t-1} - xlast_r^{t-1})$.
 - (c) $xlast_r^t = x_r^{t-1}$.

ou

- (a) Considérer l'estimation y_r^t d'indice $r = r_t$.
- (b) Rajout de « cash » : $y_r^t = y_r^{t-1} + d_r^{t-1}$.

ou

- (a) Considérer la ligne d'indice $r = r_t$ de A .
- (b) $x_r^t = e_r + \sum_{1 \leq q \leq n} a_{rq} y_q^{t-1}$.

Dans le cas où $B = {}^T A$, les trois itérations peuvent être effectuées en même temps, ce qui évite d'avoir à stocker $xlast_r^{t-1}$. Le vecteur de poids d'autorité Y^t doit être stocké en mémoire vive (si on s'autorise peu d'accès disques par page visitée), mais le vecteur de poids de « hub » X^t peut être stocké sur disque.

Dans le cas où $A = Id$ est l'identité et $B = {}^T A$ est la matrice d'adjacence normalisée utilisée pour le calcul du PageRank (avec $E = F + 0$), on reconnaît l'algorithme de PageRank incrémental en identifiant c_p^t avec $x_p^t - xlast_p^t$, h_p^t avec x_p^t et $h_p^t + c_p^t$ avec y_p^t .

Faisons maintenant le lien entre l'algorithme ci-dessus et les itérations asynchrones. Le vecteur X^t est clairement mis à jour suivant des itérations asynchrones de $X^{t+1} = AY^t + E$. Considérons le cas plus difficile du vecteur Y^t . Pour cela, on dira qu'un temps t est *colonne* s'il correspond à une itération sur la colonne r_t de B , *cash* s'il correspond à une itération de rajout de « cash » pour l'indice r_t de Y^t , ou *ligne* s'il correspond à une itération sur la ligne r_t de A . Notons à nouveau $last_r(t)$ le dernier instant de calcul sur la colonne r avant t , et $last_r^2(t) = last_r(last_r(t))$ l'instant précédent, et ainsi de suite pour $last_r^3(t), \dots$. En utilisant le fait que $xlast_r^t = x_r^{last_r(t)}$, on peut alors écrire :

$$y_p^t = \sum_{t' \text{ cash}} d_p^{t'-1} + \sum_{t' \text{ colonne}} b_{pr_{t'}} (x_{r_{t'}}^{t'-1} - xlast_{r_{t'}}^{t'-1})$$

$$\begin{aligned}
&= \sum_{t' \text{ cash}} d_p^{t'-1} + \sum_r b_{pr} (x_r^{last_r(t)} - x_r^{last_r^2(t)} + x_r^{last_r^2(t)} - x_r^{last_r^3(t)} + \dots) \\
&= \sum_{t \geq t'} d_p^{t'-1} + \sum_r b_{pr} x_r^{last_r(t)}
\end{aligned}$$

On retrouve donc bien une version asynchrone de $Y^{t+1} = BX^t + F$ si les d_p^t vérifient $\sum_{t \geq t'} d_p^{t'-1} \rightarrow f_p$. On peut, par exemple, prendre $d_p^0 = f_p$ et $d_p^t = 0$ pour $t > 0$. On peut aussi faire un algorithme sur une fenêtre temporelle de temps T en maintenant $\sum_{t \geq t' > t-T} d_p^{t'-1} \approx f_p$ et en estimant les poids d'autorité par $Y^t - Y^{t-T}$ (l'ordre des itérations doit alors vérifier $last_r(t-T) \approx last_r(t) - T$).

Discussion

Ce chapitre existerait-il si je n'avais pas dû faire le lien entre l'algorithmique du routage et les algorithmes utilisés sur le graphe du web ? La question a valeur d'anecdote. Disons simplement qu'il faudrait faire ici le lien avec le domaine de l'auto-stabilisation (que je connais mal). Les algorithmes auto-stabilisants sont ceux qui convergent vers un état valide au bout d'un temps fini après une panne. Dans ce document, on s'intéresse plutôt à maintenir un état proche de l'état valide malgré des modifications incessantes...

Bibliographie

- [1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proc. of the 12th Int. world wide web conf.*, 2003.
- [2] G.M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM*, 1978.
- [3] Richard Bellman. On a routing problem. *Quaterly of Applied Mathematics*, 16(1), 1958.
- [4] D. Beretsekas and J. Tsitsiklis. *Parallel and Distributed Computation : Numerical Methods*. Prentice Hall, 1989.
- [5] D.P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 1983.
- [6] V. Blondel and P. Van Dooren. Similarity matrices for pairs of graphs. In *ICALP*, 2003.
- [7] V. Blondel, J. Hendrickx, A. Olshevsky, and J. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *IEEE Conference on Decision and Control*, 2005.
- [8] D. Chazan and W. Miranker. Chaotic relaxation. *Linear algebra and applications*, 2, 1969.
- [9] J.-L. Dornstetter, D. Krob, M. Morvan, and L. Viennot. Some algorithms for synchronizing clocks of base transceiver stations in a cellular network. *Journal of Parallel and Distributed Computing*, 61 :855–867, 2001.
- [10] Lestor Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
- [11] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. of the 9th annual ACM-SIAM symp. on discrete algorithms*, 1998.

- [12] B. Lubachevsky and D. Mitra. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *Journal of the ACM*, 33(1), 1986.
- [13] F. Mathieu. *Graphes du Web, Mesures d'importance à la PageRank*. PhD thesis, Univ. Montpellier II, 2004.
- [14] F. Mathieu and L. Viennot. Aspects locaux de l'importance globale des pages web. In *5es rencontres francophones sur les Aspects Algorithmiques des Télécommunications (ALGOTEL'2003)*, 2003.
- [15] J.C. Miellou. Itérations chaotiques à retards, étude de la convergence dans le cas d'espaces partiellement ordonnés. *Comptes rendus de l'Academie des sciences*, A(278), 1974.
- [16] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking : Bringing order to the web. Technical report, Stanford University, 1998.

Chapitre 7

Quand ça bouge

Toute la difficulté du routage vient du dynamisme des connexions entre les nœuds. Ceci implique de mettre à jour de manière continue les informations de routage. Le routage point à point classique voit l'évolution du graphe de connexions comme une suite de graphes statiques. Dès qu'une modification s'opère, il faut rapidement changer les tables de routages pour prendre en compte la nouvelle topologie. Dans le cas des réseaux ad hoc, les modifications sont constantes. Dans le cas des réseaux de pair à pair, des nœuds arrivent ou quittent le réseau sans cesse.

7.1 Surcoût de contrôle

Dans [3], nous proposons un modèle élémentaire du dynamisme du graphe de connexions. Nous supposons simplement que chaque lien du graphe casse avec une fréquence μ . Ce seul paramètre nous permet de modéliser la mobilité des noeuds et le dynamisme du réseau. Ce modèle s'avère adapté aux réseaux ad hoc où tous les nœuds se déplacent, scénario que l'on retrouve presqu'invariablement dans toute la littérature sur le sujet. Ce modèle permet d'analyser simplement le surcoût de contrôle (c'est-à-dire la quantité de trafic de paquets de contrôle du protocole de routage) qui vont résulter d'une mobilité donnée. En reprenant des simulations apparues dans la littérature, notre modèle permet d'expliquer une croissance linéaire du trafic de contrôle dans les protocoles réactifs en fonction de la mobilité et du nombre de routes actives (celles sur lesquelles transitent des paquets). Dans le cas pro-actif, le trafic de contrôle augmente linéairement avec la mobilité indépendamment

du nombre de routes actives [3].

Dans un protocole réactif, seule la cassure d'un lien sur une route active va provoquer un trafic de contrôle, typiquement une inondation. Dans le cas d'un protocole pro-actif, seule la cassure d'un lien de la sous-topologie diffusée va provoquer un trafic de contrôle, typiquement la diffusion d'un nouvel état de lien. En première analyse, un protocole pro-actif aura un trafic de contrôle inférieur à un protocole réactif dès que plusieurs routes partagent des liens. En effet, une seule diffusion suffit alors à réparer toutes les routes passant par le lien dans le cas pro-actif alors qu'il faut une inondation par route dans le cas réactif. Cette analyse souligne une fois de plus l'importance de n'utiliser qu'une sous-topologie en routage ad hoc pro-actif pour favoriser l'utilisation des mêmes liens par plusieurs routes.

Contrairement à l'idée selon laquelle les protocoles réactifs étaient mieux adaptés à la mobilité, cette analyse montre que la différence entre les deux approches se fait sur le nombre de routes actives, les protocoles réactifs étant mieux adaptés à un petit nombre de routes actives.

7.2 Connexité temporelle

Dans [4], nous montrons comment étendre la notion de routage dans un graphe dont on connaît l'évolution. Un graphe dynamique peut être représenté par une liste temporelle d'événements d'arcs (u, v, t, s) . avec $s = \text{up}$ ou $s = \text{down}$ pour représenter respectivement l'activation ou la perte d'un lien. Ainsi, l'événement (u, v, t, up) (respectivement (u, v, t, down)) représente l'apparition (respectivement la perte) du lien $u \rightarrow v$ au temps t .

La calcul de plus courts chemins se fait alors par une version temporisée de l'algorithme de Dijkstra :

Dijkstra temporisé

1. Une source s calcule de plus courts chemins vers les autres noeuds. Elle initialise $d(s) = 0$ et $d(u) = \infty$ pour les autres sommets. Tous les sommets sont insérés dans une file de priorité F .
2. Tant que F n'est pas vide, faire :
 - défiler de F le sommet u de distance $d(u)$ minimale,
 - le temps minimal d'atteinte de u est $t_{\text{pres}} = d(u)$,
 - pour tous les événements (u, v, t, up) tels qu'il n'y a pas d'événement (u, v, t', down) avec $t \leq t' \leq \max(t, t_{\text{present}}) + t_{\text{transm}}(u, v)$,

si $d(v) > \max(t, t_{present}) + t_{transm}(u, v)$ alors mettre à jour F avec
 $d(v) = \max(t, t_{present}) + t_{transm}(u, v)$ et poser $Pere(v) = u$.

3. Utiliser l'arborescence $Pere$ pour construire la table de routage de s .

L'algorithme ci-dessus consiste à calculer des distances temporelles $d(u)$ correspondant au temps minimal au bout duquel un message peut être acheminé jusqu'au noeud.

Ce modèle généralise la notion de routage car il devient possible d'envoyer un message vers un noeud même si à aucun moment n'existe une route jusqu'au noeud. Une connexité temporelle suffit. On peut ainsi définir une sorte de diamètre temporel T_d d'un graphe dynamique comme le délai minimal tel qu'à tout instant t , un noeud peut atteindre n'importe quel autre avant $t+T_d$.

Dans un réseau pratique, on peut imaginer disposer de prédictions à court terme (par exemple en suivant l'évolution de la puissance d'émission de ses voisins dans un réseau ad hoc). Le routage sera alors possible si on sait prédire l'évolution du réseau jusqu'à T_d . Notons que le routage ad hoc classique presuppose une connexité constante (soit $T_d = 0$ si l'on néglige les délais de transmission). Le modèle présenté ici s'applique donc à des réseaux peu denses avec beaucoup de mobilité (comme les véhicules d'un réseau de transport, par exemple).

L'algorithme de Dijkstra temporisé pourrait être directement utilisé dans un protocole par état de lien. Il reste néanmoins à résoudre le problème de la diffusion des événements d'arcs, ce qui requiert alors une prédiction jusqu'à $2T_d$. Donnons un algorithme simple de diffusion pour ce modèle :

Inondation temporelle

1. La source émet un message avec une date d'expiration t_f .
2. Tant qu'un noeud a un message d'inondation non expiré, il le transmet à tout nouveau voisin.

L'idéal serait d'utiliser un délai d'expiration initial de T_d , ce qui garantit de toujours pouvoir router si l'on dispose d'une prédiction à $2T_d$, tout en minimisant la quantité de message en cours d'inondation.

7.3 Stabiliser le dynamisme

Dans le contexte du pair à pair, le dynamisme provient de l'arrivée incessante de nouveaux noeuds et du départ continu d'autres noeuds. Le réseau

logique formé entre pairs doit s'adapter à ce dynamisme. L'approche proposée par Kademlia [7] consiste à inclure assez de redondance dans la topologie pour obtenir une stabilité dans la connexité. Ainsi k contacts sont stockés au lieu d'un seul de sorte que $k - 1$ d'entre eux peuvent quitter le réseau sans qu'il soit nécessaire de mettre à jour l'entrée correspondante dans la table de routage. Dans [5], nous transposons cette technique pour obtenir une topologie similaire au graphe de Bruijn. Si la probabilité que k noeuds quittent le réseau dans une période de temps T est suffisamment faible, chaque entrée de la table de routage peut être rafraîchie une fois seulement par période de durée T . Pour obtenir une fiabilité similaire avec un seul contact par entrée, il faudrait remettre à jour chaque entrée de la table de routage plus souvent.

En effet, si un noeud a une probabilité $\lambda e^{-\lambda t} dt$ de quitter le réseau au temps t , la probabilité qu'un noeud reste au moins T est donc $e^{-\lambda T}$, il s'agit là d'un modèle de départ poissonnien¹. Ainsi, avec k contacts rafraîchis tous les T_k , la probabilité qu'ils soient tous partis au bout de T_k est $p = (1 - e^{-\lambda T_k})^k$. Pour garantir une probabilité p d'échec donnée, il faut donc utiliser une période de rafraîchissement $T_k = -(1/\lambda) \log(1 - p^{1/k})$ ($T_m = 1/\lambda$ est la durée moyenne de présence dans le réseau.)

La figure 7.1 montre la durée entre deux « pings » nécessaire pour obtenir au moins un contact valide sur k avec une certaine probabilité d'échec p fixée. Le maximum de la courbe $p = 1e - 6$ est obtenu pour $k = 20$, ce qui semble étayer le choix $k = 20$ fait dans [7] sans justification. Il est à noter que la valeur de k donnant une période maximale de « pings » ne dépend pas du paramètre λ de la loi de Poisson. Cette analyse montre le compromis qui peut exister entre la quantité de contacts stockés et le nombre de messages qu'il faut échanger avec eux.

¹Une étude du réseau Gnutella [7] montre que la probabilité de départ baisse pour les noeuds présents depuis longtemps dans le réseau, ce qui ne cadre pas avec le modèle poissonnien où cette probabilité ne dépend pas du temps de présence. En effet, on peut imaginer que de nombreux utilisateurs tentent une connexion pour essayer et quittent rapidement le réseau alors que les utilisateurs réels restent connectés plus longuement. Cependant, en ne ne considérant que les noeuds de temps de présence supérieure à une heure, on observe une probabilité de rester au moins une heure de plus comprise entre 0.7 et 0.9, ce qui se rapproche du modèle poissonnien avec λ entre 0.36 et 0.1.

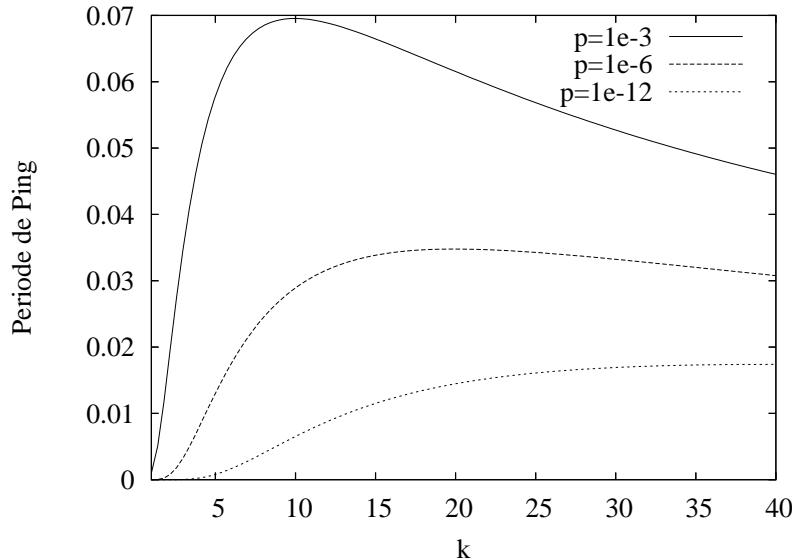


FIG. 7.1 – Période de ping $T_k/k/T_m$ nécessaire pour maintenir au moins un contact vivant parmi k avec probabilité $1 - p$ donnée.

7.4 Équilibrer le dynamisme

Une autre difficulté dans la volatilité des nœuds d'un réseau de pair à pair réside dans le choix d'identifiants équidistribués pour les nœuds. En effet, de nombreux protocoles utilisent des identifiants logiques pour construire un réseau logique entre les nœuds. Pour que le réseau logique fonctionne correctement, il faut que les identifiants soient distribués de manière proche de l'uniforme (sans quoi un nœud risque d'avoir un degré exagérément grand par exemple). Dans les protocoles qui utilisent la similitude de préfixe pour mettre en relation les nœuds, il est important, par exemple, de bien répartir les nœuds par préfixe de longueur donnée (voir le routage par préfixe au paragraphe 5.4).

L'algorithme le plus simple pour cela consiste à donner un identifiant aléatoire à chaque nouvel arrivant. Les bornes de Chernoff permettent alors d'affirmer que le nombre de nœuds partageant un préfixe de longueur $l = \log n / \log \log n$ (en bits) donné est $\Theta(1)$ en moyenne et $O(\log n)$ avec une forte probabilité. Cela peut aussi se voir comme un problème de balles et

de paniers : les identifiants choisis au hasard sont les balles et tombent dans le panier associé à leur préfixe de longueur l . Le nombre de paniers étant $n/\log n$, au moins une balle tombe dans chaque panier avec forte probabilité. Le nombre moyen de balle par panier est $\log n$ et le nombre de balles dans un panier est $O(\log n)$ avec forte probabilité.

Cependant, il serait plus intéressant encore d'atteindre une distribution en $\Theta(k)$ où k est une petite constante. Pour cela, on peut s'inspirer des algorithmes de placement de balles dans des paniers. Un résultat célèbre [1] montre qu'en tirant deux paniers au hasard et en plaçant la balle dans le panier le moins rempli, on obtient une borne en $O(\log \log n)$ au lieu de $O(\log n)$ sur le nombre de balles dans un panier (n balles sont jetées dans n paniers). En ce qui concerne l'attribution d'identifiants de préfixes équi-répartis, on peut donc tenter l'algorithme suivant :

Insertion équilibrée

1. Un nouvel arrivant tire au hasard deux identifiants u_1 et u_2 .
2. Pour chaque identifiant u_i , il trouve le nœud v_i de plus long préfixe commun avec u_i (u_i et v_i partagent un préfixe de longueur l_i).
3. Le nouvel arrivant choisit l'identifiant u_i de longueur l_i minimale.

Analyser la distribution des identifiants selon l'algorithme ci-dessus est un problème ouvert. Une idée similaire [2] est explorée pour améliorer l'équilibrage des charges dans Chord. Une autre solution pour équilibrer les préfixe [6] repose sur une marche de longueur $O(\log n)$ au moment de l'insertion d'un nouvel arrivant.

Discussion

Traiter du dynamisme d'un graphe ne me semble pas encore un sujet mûtre. J'espère que ce chapitre donne quelques points de départ intéressants.

Bibliographie

- [1] Y. Azar, A.Z. Broder, A.R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 2000.
- [2] J. Byers, Considine J, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *IPTPS*, 2003.
- [3] T. Clausen, P. Jacquet, and L. Viennot. Analyzing control traffic overhead versus mobility and data traffic activity in mobile ad-hoc network protocols. *ACM Wireless Networks journal (Winet)*, 10(4), july 2004.
- [4] A. Ferreira and L. Viennot. A note on models, algorithms, and data structures for dynamic communication networks. Technical Report RR-4403, INRIA, mars 2002.
- [5] A.T. Gai and L. Viennot. Broose : A practical distributed hashtable based on the de-bruijn topology. In *The Fourth IEEE International Conference on Peer-to-Peer Computing*, 2004.
- [6] G.S. Manku. A randomized id selection algorithm for peer-to-peer networks. In *PODC*, 2004.
- [7] P. Maymounkov and D. Mazieres. Kademlia : A peerto -peer information system based on the xor metric. In *1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

Chapitre 8

Conclusion

Il existe tout un dégradé de techniques de routage selon la structuration du réseau. Quand rien n'est connu de la topologie du réseau, celle-ci doit être découverte et il faut calculer quels sont les meilleurs chemins, c'est le cadre classique du routage dans un réseau. À l'inverse, quand la topologie est suffisamment maîtrisée, on sait trouver des chemins efficaces sans plus de calculs, c'est le cas d'une machine parallèle ou d'un réseau logique de pairs. En situation intermédiaire, le problème peut-être découpé en instances plus simples dans un réseau hiérarchique comme Internet ou une semi-structuration peut-être possible en sélectionnant une sous-topologie suffisante si la topologie est assez dense comme dans un réseau ad hoc de forte densité. La structuration peut aussi être probabiliste comme dans le routage à la Kleinberg ou des réseaux logiques de pairs construits aléatoirement comme le réseau Gnutella ou un téléchargement en torrent de BitTorrent.

Rappelons rapidement quelques aspects saillants des techniques présentées dans ce document.

Échange régulier de messages avec les voisins

Messages vides	Découverte du voisinage
Messages « hello »	Découverte du voisinage à deux sauts
Vecteur d'états (itérations asynchrones)	Routage distance vecteur Synchronisation PageRank distribué
Filtres de Bloom	Optimisation des inondations de requêtes

Techniques locales

Connaissance du voisinage	Inondation
Connaissance du voisinage à deux sauts	Multipoints relais Heuristique ensemble dominant connexe
Voisins d'identifiants inférieurs	Heuristique ensemble dominant connexe Synchronisation

Structuration de réseau

Agrégation par préfixe	Routage hiérarchique (Internet) Regroupement en clusters
Préfixes similaires	Hypercube
Préfixes décalés	De Bruijn
Multipoints relais	Sous-topologie optimale pour la distance

Graphes structurés

Hypercube	Routage préfixe Arbres arêtes disjoints
De Bruijn	Routage par décalage Arbres noeuds internes disjoints

Ténacité et joie.

