Conception et analyse d'algorithmes distribués d'ordonnancement dans les réseaux sans-fil

Dorian Mazauric

Rapport de stage de fin d'études

Département Mathématiques Appliquées et Modélisation
Ecole Polytechnique Universitaire

Directeurs de stage : Jean-Claude Bermond et Philippe Nain
Encadrant EPU : Johny Bond

22 septembre 2008

Remerciements

Je remercie Jean-Claude Bermond et Philippe Nain de m'avoir encadré durant ce stage.
Je remercie Johny Bond, mon encadrant EPU.
Je remercie également tous les membres de l'équipe-projet Mascotte, particulièrement David Coudert pour ses précieux conseils et Patricia Lachaume pour son aide.
Je remercie Joseph Peters de m'avoir accueilli à l'Université Simon Fraser de Vancouver pour la seconde partie de mon stage.

Cadre du stage

Le centre de recherche INRIA Sophia Antipolis - Méditérranée regroupe **31 équipes de recherche** dont **MASCOTTE** et **MAESTRO**. Il a été crée en 1983 et est actuellement dirigé par Gérard Giraudon. Il est en partenariat avec entre autres :
— l'I3S : Informatique Signaux et Systèmes de Sophia Antipolis (CNRS et Université de Nice-Sophia Antipolis)
— l'ENS : Ecole Normale Supérieure de Paris
— le laboratoire J.A. Dieudonné (CNRS et Université de Nice - Sophia Antipolis)
— l'INRA : Institut Nationale de la Recherche Agronomique

Jean-Claude Bermond et Philippe Nain, responsables respectivement de Mascotte et de Maestro, m'ont encadré tout au long de mon stage. Décrivons succintement ces deux projets de recherche.

Le projet Mascotte ou **M**éthodes **A**lgorithmiques, **S**imulation, **C**ombinatoire et **O**p**T**imisation pour les **TE**lécommunications est un projet commun entre l'INRIA, le CNRS et l'Université de Nice Sophia-Antipolis (UNSA), dans le cadre du laboratoire I3S. L'objectif du projet Mascotte, dirigé par Jean-Claude Bermond (CNRS), est de développer des méthodes et outils algorithmiques qui s'appliquent en particulier à la conception de réseaux de télécommunication. La réalisation de cet objectif implique la poursuite de recherches dans les domaines de l'algorithmique, de la simulation et des mathématiques discrètes.

Les différents axes de recherche de l'équipe-projet sont les suivants :
— algorithmique, mathématiques discrètes et optimisation combinatoire,
— algorithmique des communications,
— dimensionnement de réseaux optiques (WDM, SDH, ATM, embarqués, radio et satellites),
— simulation orientée objet en environnement réparti,
— parallélisme et réseaux d'interconnexion.

Le projet Maestro, Modèles pour l'Analyse des performances et le contrôle des réseaux, est un projet INRIA commun avec le laboratoire CNRS LIRMM et l'Université de Montpellier II. Maestro a pour ambition de mettre à la disposition de la communauté des méthodes et des outils logiciel pour l'évaluation des performances, l'optimisation et le contrôle des réseaux. Plus précisément, les objectifs de Maestro sont :
— de développer des méthodes et des outils logiciel de portée générale pour évaluer, optimiser et contrôler les systèmes à événements discrets, et notamment les réseaux et leurs applications ;
— d'évaluer les performances et de contrôler les réseaux de communication (protocoles, architectures, applications) et, en premier lieu, les réseaux IP.

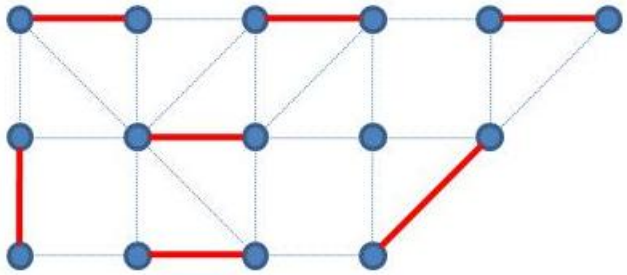
Table des matières

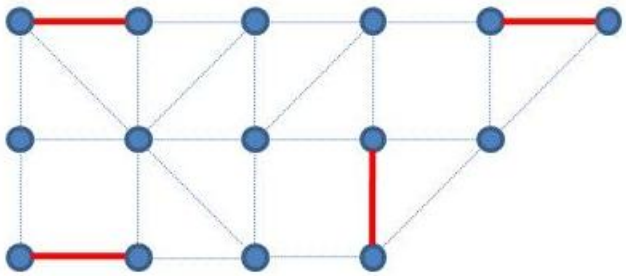
1 Introduction

L'ordonnancement des transmissions ("call scheduling") est un problème fondamental dans les réseaux de télécommunication. Il a été intensément étudié dans les réseaux filaires classiques. Pour les réseaux sans-fil (radio, ad hoc, senseurs), une contrainte supplémentaire vient des problèmes d'interférence. On considère ici des réseaux synchrones. Dans ce contexte, il faut que durant une étape (round) donnée de communication, les appels (calls) qui ont lieu n'interférent pas. Une étape est composée de 2 phases : une partie de contrôle où on décide quels appels vont avoir lieu en respectant les contraintes d'interférence et une phase de transmission où les appels retenus transmettent de l'information.

Nous considérons ici le cas d'antennes omnidirectionnelles. La modélisation des interférences peut être faite de différentes manières. Classiquement on s'intéresse au rapport signal sur bruit. Ce modèle étant trop compliqué pour l'analyse et la conception de protocoles d'ordonnancement d'appels, on utilise en pratique des modèles simplifiés. On associe au réseau un graphe de transmission où 2 sommets u et v sont reliés si v est dans la zone de transmission de u (c'est-à-dire que v peut recevoir un appel de u). On supposera dans la suite que les transmissions sont symétriques et on considérera un graphe non orienté. Un noeud u ne peut recevoir un appel d'un autre noeud v que si ces derniers ne sont pas dans la zone d'interférence d'un autre émetteur (ou récepteur). La zone d'interférence d'un noeud est l'ensemble de tous les noeuds à distance au plus d de lui. La distance étant prise dans le graphe de transmission.
Le modèle le plus simple ($d = 0$), appelé "primary node interference", consiste juste à supposer qu'un noeud ne peut émettre et recevoir en même temps. Un ensemble de calls compatibles correspond alors à un couplage dans le graphe associé au réseau. Un couplage est un ensemble d'arêtes 2 à 2 disjointes.
Dans un modèle plus raffiné, par exemple en prenant $d = 1$, un noeud v peut recevoir un message d'un noeud u que si aucun des voisins de u et de v ne transmet (ou ne reçoit) durant la même étape. Dans le graphe associé, les arêtes correspondantes aux calls d'une étape donnée, forment un "couplage induit".

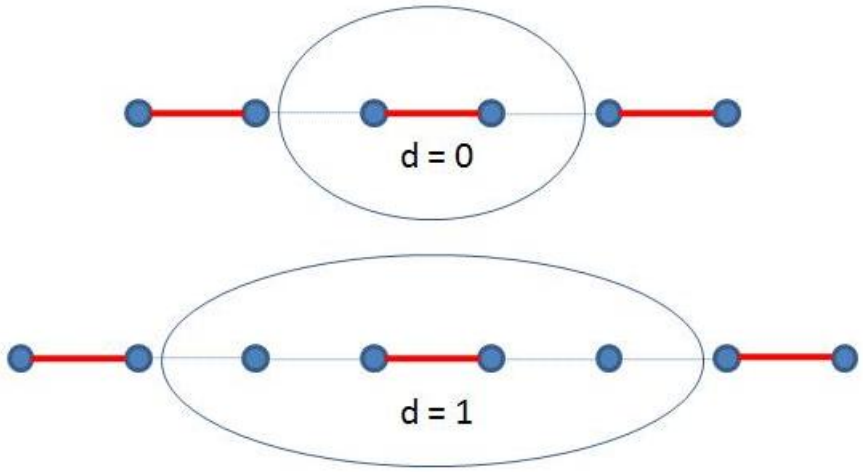
Exemple d'ordonnancements valides (les arêtes autorisées à envoyer des données sont celles représentées avec les plus gros traits (rouge)) :





L'objectif est de réaliser des communications avec une bonne qualité de service (assurant un bon débit ou un court délai par exemple). Ces communications peuvent être single hop ou multi hop (nécessitant le choix d'un routage). Un exemple de communication multi hop est le gathering où il s'agit de router des messages provenant d'une unique source (ou à destination d'un unique puits). La conception et l'analyse de ces protocoles sont compliquées car il y a dépendance entre les appels. Beaucoup d'auteurs se limitent au link scheduling, qui consiste à "optimiser" une étape. Dans ce cas, on cherche soit à maximiser le nombre d'appels effectués, soit à réaliser les appels qui permettent de diminuer les files d'attente en espérant obtenir un système stable. Ainsi, l'analyse de la stabilité des files d'attente est simplifiée car elles sont indépendantes les unes des autres. Nous avons supposé au cours du stage que le trafic était single-hop.
Comme l'ont fait les auteurs dans [9], nous avons émis l'hypothèse que l'intelligence se trouvait sur les arêtes du graphe associé. Par exemple, ce sont les arêtes qui prendront la décision d'envoyer un message. Ainsi la zone d'interférence, notée $\varepsilon(e)$, d'une arête $e = (u, v)$ est l'ensemble des arêtes ayant au moins un de ses deux noeuds dans la zone d'interférence (au sens de la zone d'interférence d'un noeud) de u ou de v.

Exemple d'ordonnancements valides (les arêtes autorisées à envoyer des données sont celles représentées avec les plus gros traits (rouge)). Les cercles représentent la zone d'interférence de l'arête du milieu pour chacun des modèles d'interférence.

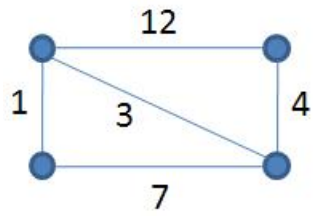


Ces problèmes ont été bien abordés dans un contexte centralisé où on connait les informations tant sur la topologie du réseau que sur l'état de toutes les files d'attente et sur la capacité des liens. On trouve deux grandes approches dans la littérature :

1) une plus optimisation combinatoire où on souhaite concevoir des protocoles efficaces (optimisant ou satisfaisant un certain critère). Par exemple dans MASCOTTE, le problème du gathering dans les réseaux radios avec interférence a été très étudié pour un trafic statique. Le but étant de minimiser le nombre d'étapes de transmission pour réaliser l'envoi de w(u) messages pour un noeud donné ou de maximiser le débit de chaque noeud. Ces problèmes sont très difficiles même sur des topologies très particulières comme le chemin ou la grille. En revanche, il existe de bons algorithmes d'approximation.

2) une autre approche, plus théorie des files d'attente, où on désire par exemple analyser la stabilité du système associé. C'est un des axes de recherche du projet MAESTRO, dont Philippe Nain est le responsable scientifique, qui a pour caractéristiques, entre autres, d'évaluer, d'optimiser et de contrôler les systèmes à évènements discrets. De ce point de vue là, l'objectif est d'analyser par exemple la stabilité ou le délai moyen d'envoi d'un paquet selon les algorithmes utilisés. Les files d'attente sont en général plaçées sur les liens (les arêtes du graphe associé). Nous avont fait cette hypothèse au cours du stage.

Exemple de graphe (valué) des transmissions avec les poids des arêtes correspondants aux tailles des files d'attente sur les liens.



Certains travaux mettent en exergue des algorithmes distribués efficaces, notamment dans [12], où un algorithme distribué avec un overhead constant est décrit. En d'autres termes, le nombre de time slots de contrôle est constant. Celui-ci ne demande pas de mise à jour de l'information locale détenue par un noeud avant une phase de contrôle. De plus l'overhead est constant et paramétrable en effectuant un compromis entre efficacité et nombre de slots de contrôle. Néanmoins, cet algorithme n'est en aucun cas généralisable à un modèle d'interférence quelconque et le nombre de slots nécessaires avant d'atteindre un état stable peut être long (cf simulations). L'objectif est d'établir des algorithmes distribués efficaces, en terme de stabilité notamment, ne demandant peu ou aucune information sur la zone d'interférence, et généralisable à un modèle d'interférence quelconque. Durant le stage, nous nous sommes limités au cas du "link scheduling". Nous proposons des algorithmes nouveaux et leurs études sont théoriques et empiriques, en prenant des topologies simples au départ (chemin, grille) puis des réseaux quelconques et différents modèles d'interférence comme le primary node par exemple.

C'est pourquoi nous effectuons tout d'abord une description des travaux existants, avant d'introduire quelques définitions. Nous proposons par la suite des algorithmes que nous analysons. Ensuite, nous effectuons une comparaison entre les algorithmes par le biais de simulations. Enfin, nous dressons une synthèse des principaux résultats et des enjeux futurs.

2 Travaux existants

Il s'agit dans cette partie de synthétiser les différents travaux réalisés dans ce domaine. Nous ne décrivons qu'une partie des articles qui nous ont aidé à mieux comprendre les problématiques. En effet, il est indispensable de comparer les différents modèles utilisés, les hypothèses effectuées, les principaux résultats de leurs études et les éventuelles limites afin d'établir les enjeux liés au problème d'ordonnancement dans les réseaux sans-fil.

2.1 Description détaillée

Décrivons, de façon non exhaustive, quelques travaux intéressants concernant les algorithmes permettant de router du trafic dans un réseau sans-fil.

2.1.1 Deux algorithmes distribués

Dans [9], les auteurs présentent deux algorithmes distribués permettant de définir un ensemble de calls compatibles dans un réseau sans fil.

Le premier algorithme, Q-SCHED, comporte deux phases : une phase de contrôle définissant un ensemble de calls compatibles et une phase d'envoi des données. Les files d'attente sont définies sur les liens du réseau et définissent ainsi le poids de chacune des arêtes. Autrement dit, l'intelligence se situe au niveau des arêtes. Le trafic est supposé single-hop, c'est-à-dire qu'un paquet envoyé, sort immédiatement du réseau. Le modèle d'interférence est quelconque. Nous pouvons considérer aussi bien le modèle d'interférence classique primary node par exemple ou un modèle d'interférence quelconque tel que si une arête e est active, toutes les arêtes situées dans sa zone d'interférence $\varepsilon(e)$ doivent être inactives. Les auteurs effectuent une autre hypothèse (pas la plus faible), en supposant qu'une arête e connaît les poids de l'ensemble $\varepsilon(e)$ des arêtes situées dans sa zone d'interférence ainsi que les poids des arêtes situées dans la zone d'interférence de chacune des arêtes appartenant à $\varepsilon(e)$.
La phase de contrôle est divisée en M mini-slots et chaque arête calcule aléatoirement un backoff en fonction des informations qu'elle possède et d'une fonction définie par les auteurs dans leurs travaux. Ainsi, elle tire un backoff dans l'ensemble $\{1,2,...,M\}$ ou elle tire $M+1$ si elle décide de ne pas être candidate à transmettre des données. Lorsque le backoff d'une arête e expire, elle décide de devenir active si aucune arête de $\varepsilon(e)$ n'est devenue active précédemment. En cas de collision entre plusieurs arêtes d'une même zone d'interférence, aucune ne devient active. Cet algorithme est distribué et très simple. Le principal problème est que les auteurs n'expliquent pas comment

calculer et mettre à jour les informations nécessaires au calcul du backoff. En effet, cette mise à jour demande un nombre important de messages de contrôle (même pour le modèle d'interférence primary node). Le défi majeur est justement de mettre à jour rapidement l'information que détient une arête e, avant chaque phase de contrôle. Pour une arête e, connaître le poids de chacune des arêtes de $\mathfrak{E}(e)$ et le poids de chacune des arêtes de $\mathfrak{E}(\mathfrak{E}(e))$ est en pratique inpensable car irréalisable, pour un modèle d'interférence quelconque. Cette hypothèse est beaucoup trop forte.

Le second algorithme, BP-SIM, comporte également deux phases : une phase de contrôle divisée en $K.M$ mini-slots et une phase de transmission de données. L'hypothèse qu'une arête e connaisse la taille des files d'attente des arêtes voisines $\mathfrak{E}(e)$ et la taille des arêtes voisines de $\mathfrak{E}(e)$, $\mathfrak{E}(\mathfrak{E}(e))$, est relâchée. En revanche, l'algorithme est présenté uniquement pour le modèle d'interférence primary node. L'idée est de construire un ensemble de calls compatibles en K étapes (un couplage dans la graphe associé pour ce modèle d'interférence). Au fur et à mesure de ces K étapes, le couplage est amélioré. Il est vide initialement. Etant donné M_i le couplage après l'étape i, $1 \leq i < M$, l'étape $i+1$ améliore M_i en construisant un couplage M_{i+1} avec $M_i \subseteq M_{i+1}$.
A l'étape 1, chaque noeud décide d'être $left$ avec probabilité $\frac{1}{2}$, *right* sinon. Un noeud v_l, de type $left$, choisit au hasard et de façon uniforme un noeud voisin v_{lr}, qu'il contactera après expriation d'un backoff choisit également au hasard et uniformément. Si v_{lr} est de type *right* et n'a pas encore était contacté, alors l'arête (v_l, v_{lr}) est ajoutée au couplage M_1 si la taille de la file d'attente est plus grande ou égale à la capacité du lien. Si des collisions apparaissent, il y a neutralisation et aucun noeud ne prend de décision. Ce protocole est répété durant les $K-1$ autres étapes.
La complexité de BP-SIM dépend uniquement du degré nodal maximum d^*. Le résultat de stabilité est que la somme des taux d'arrivée dans une zone d'interférence divisée par la capacité du lien, doit être inférieure à κ, κ étant la somme maximale des taux d'arrivées de toutes les zones d'interférence. Cet algorithme est intéressant mais demande $K.M$ mini-slots de contrôle et est présenté seulement pour le modèle d'interférence primary node.

2.1.2 Un algorithme distribué avec overhead constant

Dans [12], les auteurs présentent un algorithme distribué pour définir un couplage dans un réseau sans-fil. Le trafic est supposé single-hop, les paquets envoyés quittent immédiatement le réseau. les files d'attente sont modélisées sur les liens, définissant ainsi le poids des arêtes. Le modèle d'interférence considéré est le modèle primary node.

L'algorithme présente quelques caractéristiques intéressantes :
— un algorithme distribué
— un overhead constant (nombre constant de slots de contrôle)
— une fraction aussi grande que souhaité de l'efficacité de l'algorithme centralisé optimal (compromis entre efficacité et nombre de slots de contrôle).

L'algorithme se décompose en deux phases : une phase de contrôle et une phase d'envoi des données. L'idée est de trouver un meilleur couplage à partir du précédent. La technique utilisée est celle des chaînes augmentantes. Il s'agit en fait de construire des chemins dans le graphe, alternant arêtes actives à l'étape précédente et arêtes inactives à l'étape précédente, dans le but de trouver un meilleur couplage.

Cet algorithme admet de nombreux avantages mais reste très fortement lié au modèle d'interférence primary node. En effet, en prenant un modèle d'interférence quelconque, l'adaptation de cet algorithme n'est pas du tout envisageable. Trouver des chaînes augmentantes demanderait beaucoup trop de slots de contrôle pour combattre les interférences.
De plus la stabilité, prouvée théoriquement, n'est pas très claire en pratique. Le nombre de time slots nécessaires avant d'obtenir un état stable, peut être important. Ceci est décrit en détails dans la partie simulations et résultats.
Enfin, l'algorithme débute par le choix aléatoire des noeuds actifs au début de la phase de contrôle. Ils correspondent en fait au point de départ des différentes chaînes augmentantes. En pratique, un noeud choisit d'être point de départ avec une probabilité p, identique pour tous les noeuds du réseau. Les auteurs ne donnent pas de critères quantitatifs quand au choix de la valeur de p. Libre au concepteur du réseau de trouver, avec un peu d'intuition (peut être même un peu de chance), la bonne valeur de p.

2.1.3 Un algorithme de gathering

Dans [11], les auteurs proposent un algorithme permettant de router des messages provenant d'une unique source (respectivement à destination d'un unique puits). Le modèle d'interférence est le primary node. L'activation des liens se fait selon un mécanisme $pair - impair$, c'est-à-dire que les noeuds pairs (respectivement impairs) sont autorisés à transmettre lors des slots pairs (respectivement impairs). Un programme linéaire est établi dans le but de résoudre le problème de la détermination de ce mécanisme. Ils prouvent également que ce problème est NP-complet et proposent dans le même temps quelques heuristiques afin de résoudre ce problème. Une des heuristiques proposée est de construire un arbre des plus courts chemins routé à la source (respectivement au puits). La politique de scheduling permet d'obtenir des délais deux fois supérieurs à ceux rencontrés dans des réseaux similaires filaires (cette perte est liée à l'alternance des slots *pairs* et des slots *impairs*). Le problème est que cette technique ne se généralise pas à un trafic multi-hop quelconque même si on peut utiliser une gateway de référence et ensuite rerouter le trafic vers la véritable gateway. Le problème est qu'il peut y avoir de la surcharge et des délais. Ces limites ne sont clairement pas explicitées dans l'article. Enfin, cette technique est totalement dépendante du modèle d'interférence choisi : le modèle primary node.

2.2 Points forts, limites et enjeux

Les nombreux travaux existants décrivent des algorithmes d'ordonnancement intéressants. Nous nous intéressons particulièrement aux algorithmes distribués (locaux) et la description précédente montre que de nombreuses recherches ont porté sur l'élaboration de tels algorithmes.

L'algorithme Q-SCHED, présenté dans [9], a l'avantage d'être valide quelque soit le modèle. Cependant, il n'est pas clairement réaliste et les auteurs passent totalement sous silence le coût de la mise à jour des informations requises par chaque arête dans la recherche d'un ensemble de calls compatibles. En effet, une arête e doit connaître le poids des arêtes situées dans $\varepsilon(e)$ et dans $\varepsilon(\varepsilon(e))$. Les hypothèses sont trop fortes.

En revanche, en considérant un modèle d'interférence simple, comme le primary node, des al-

gorithmes efficaces sont décrits. L'exemple de l'algorithme proposé dans [12] est sûrement le plus représentatif. En effet, le nombre de messages de contrôle est constant et l'efficacité est aussi grande que souhaité. Aucune mise à jour d'information n'est faite par les arêtes. En fait, le concepteur du réseau doit effectuer un compromis entre la constante représentant la taille de l'overhead et l'efficacité souhaitée.
Cependant, empiriquement, l'algorithme converge assez lentement par rapport à d'autres (cf simulations). De plus, les auteurs donnent des arguments uniquement qualitatifs pour le choix de la probabilité p. p est la probabilité qu'un noeud du réseau soit le point de départ d'une (éventuelle) chaîne augmentante. La stabilité et l'efficacité de l'algorithme dépendent directement de cette probabilité et aucune information concrète n'est donnée. De plus si la mobilité des noeuds est possible, p devra nécessairement être variable. Comme expliqué précédemment, cet algorithme est décrit pour le modèle d'interférence primary node et n'est pas clairement adaptable à un modèle quelconque.

L'article [12] sert de référence notamment concernant le modèle que nous utilisons et les hypothèses effectuées. Nous proposons des algorithmes nouveaux indépendants du modèle d'interférence, en gardant la propriété de constance de l'overhead et bien évidemment le caractère local de l'algorithme. Le trafic est supposé single-hop, comme dans [12].

L'objectif était de décrire des algorithmes distribués simples permettant de palier les limites décrites précédemment, tout en sauvegardant les points positifs. Précisément, notre travail décrit des **algorithmes distribués (ou locaux), avec un overhead constant, valides quelque soit le modèle d'interférence et voire sans aucune mise à jour d'information comme pour l'algorithme log présenté plus bas.**

3 Définitions

Tout au long du stage, nous avons considéré un réseau sans-fil modélisé par un graphe non orienté et valué $G = (V, E)$. Le trafic est supposé single-hop (même si une généralisation à un trafic multi-hop est envisageable). Un paquet qui arrive sur le réseau par un sommet $u \in V$ et qui est transmis à un sommet $v \in V$ tel que $(u, v) \in E$, quitte le réseau immédiatement après. Par hypothèse, comme dans l'article de référence [12], les files d'attente sont situées sur les arêtes du graphe. Le poids $q(e)$ d'une arête $e \in E$ représente ainsi la taille de sa file d'attente. Sans perte de généralité, l'intelligence est située sur les arêtes de G. Enfin le modèle d'interférence est quelconque, même si certaines analyses et simulations sont effectuées pour le modèle primary node.

Definition 1 *Une arête $e \in E$ est i-dominée si et seulement si $|\{e' \in \varepsilon(e); q(e') > q(e)\}| = i$*
En d'autres termes, une arête i-dominée a exactement i arêtes ayant un poids strictement plus grand dans sa zone d'interférence. Une arête 0-dominée est dite maximum local strict.

Definition 2 *Une arête $e \in E$ est dite **substituable** si et seulement si elle est maximum local strict et s'il existe deux arêtes $e_1 \in \varepsilon(e)$ et $e_2 \in \varepsilon(e)$, telles que $e_1 \notin \varepsilon(e_2)$, $q(e_1) < q(e)$, $q(e_2) < q(e)$ et $q(e_1) + q(e_2) > \sqrt{3}.q(e)$.*

Definition 3 *Au cours de la phase de contrôle, une arête $e \in E$ est soit active, soit inactive, soit indeterminée.*
— active signifie que e a été choisie comme arête autorisée à transmettre des données lors de la prochaine phase d'envoi de données.
— inactive traduit le fait que e ne transmettra pas de données lors de la prochaine phase d'envoi.
— indéterminée veut dire que e n'est pas encore fixée sur son rôle lors de la prochaine phase d'envoi de données.

4 Algorithmes

L'objectif est d'établir des algorithmes permettant de router le trafic de façon stable avec un overhead constant et un modèle d'interférence quelconque. Nos algorithmes distribués sont simples et se décomposent en trois phases :
— Une phase de contrôle, parfois divisée en sous-phases, dans laquelle il s'agit de déterminer un ensemble de calls compatibles, c'est-à-dire les arêtes actives, qui seront autorisées à transmettre des données lors de la phase d'envoi. Cette phase de contrôle a pour but de garantir l'inexistence d'interférence lors de la phase d'envoi de données.
— Une phase d'envoi dans laquelle les arêtes actives envoient un certain nombre de paquets situés dans leurs files d'attente.
— Eventuellement une phase de mise à jour de la connaissance que détient une arête $e \in E$ sur $\varepsilon(e)$. Il sera décrit précisément par la suite, les informations que doit avoir éventuellement e avant chaque phase de contrôle. L'algorithme log ou l'algorithme backoff décroissant simple entre autres, n'effectuent pas cette dernière phase.

Dans la plupart des algorithmes présentés ci-dessous, la seule hypothèse est que les files d'attente (les poids des arêtes), sont bornées par un entier K, même si cette hypothèse n'est pas très contraignante.

Nous commençerons par décrire rapidement l'algorithme greedy centralisé et l'amélioration que nous apportons en utilisant le principe de substitution, avant de décrire les algorithmes distribués que nous proposons.

4.1 Algorithmes greedy

4.1.1 Algorithme greedy centralisé

Le principe de l'algorithme greedy centralisé est le suivant :
Initialement, les arêtes sont indéterminées.
On choisit l'arête de poids maximum encore indéterminée. On la rend active et les arêtes de sa zone d'interférence deviennent inactives. On fait cela tant qu'il existe des arêtes indéterminées.

Cet algorithme assure une efficacité de 0.5 de l'optimal pour le modèle d'interférence primary node.

4.1.2 Algorithme greedy centralisé avec substitution

Le principe est le suivant :
Initialement, les arêtes sont indéterminées. On crée une file d'arêtes indéterminées telle que si deux arêtes e_1 et e_2 de poids $q(e_1), q(e_2)$ avec $q(e_1) < q(e_2)$, alors e_1 est après e_2 dans la file.
Tant que la file n'est pas vide, l'arête en première position est retirée de la file et :
— si elle n'est pas substituable, l'arête devient active et les arêtes de sa zone d'interférence deviennent inactives.
— si elle substituable, l'arête reste indéterminée et est insérée en dernière position dans la file des arêtes indéterminées.

4.1.3 Analyse de l'algorithme greedy centralisé avec substitution

L'analyse est réalisée pour un réseau de type cycle avec le modèle d'interférence primary node.

Soit l'arête e_0 de poids $q(e_0)$. Soient les arêtes e_{-1} et e_1 les deux voisines de e_0 respectivement de poids $q(e_{-1})$ et $q(e_1)$. Soient les arêtes e_{-2} et e_2 respectivement l'arête voisine de e_{-1} de poids $q(e_{-2})$ et l'arête voisine de e_1 de poids $q(e_2)$. Par hypothèse, nous avons $q(e_i) \leq q(e_0)$ avec $i \in \{-2;-1;1;2\}$.

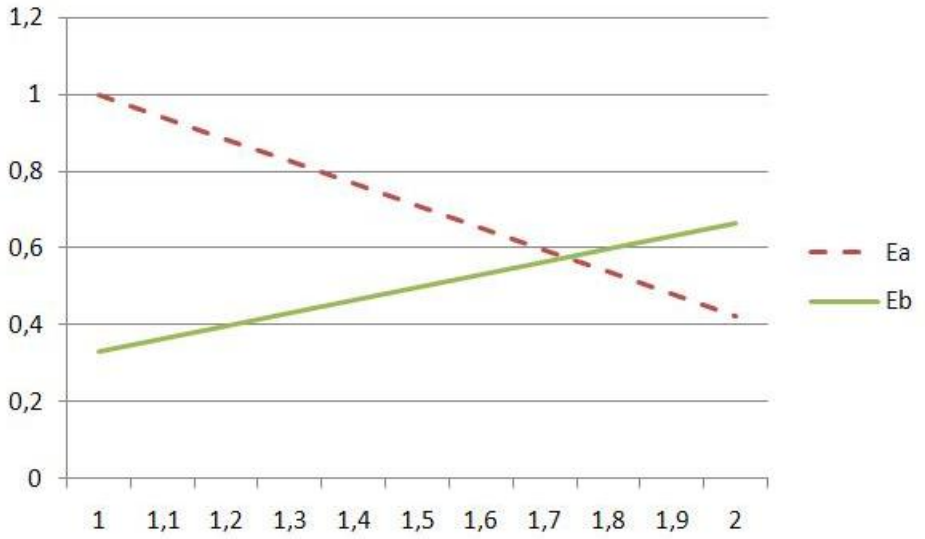
Analyse du pire des cas

Considérons les deux situations représentant le pire des cas :

— Choisir e_{-2}, e_0 et e_2 comme arêtes actives alors que le choix e_{-1} et e_1 était meilleur. Dans ce cas, l'efficacité E_a de ce choix est la fraction $\frac{q(e_{-2})+q(e_0)+q(e_2)}{q(e_{-1})+q(e_1)}$. Dans le pire des cas, nous avons $E_a = \frac{q(e_0)}{q(e_{-1})+q(e_1)}$ en prenant $q(e_{-2}) = q(e_2) = 0$ avec $q(e_0) \leq q(e_{-1}) + q(e_1)$.

— Choisir e_{-1} et e_1 comme arêtes actives alors que le choix e_{-2}, e_0 et e_2 était meilleur. Dans ce cas, l'efficacité E_b de ce choix est la fraction $\frac{q(e_{-1})+q(e_1)}{q(e_{-2})+q(e_0)+q(e_2)}$. Dans le pire des cas, nous avons $E_b = \frac{q(e_{-1})+q(e_1)}{3q(e_0)}$ en prenant $q(e_{-2}) = q(e_2) = q(e_0)$, $q(e_{-1}) + q(e_1) \leq 3q(e_0)$ est toujours vrai par hypothèse.

Le graphique ci-dessous représente l'efficacité E_a et E_b en fonction de $\frac{q(e_{-1})+q(e_1)}{q(e_0)}$.



Nous remarquons que seule la somme $q(e_{-1})+q(e_1)$ est importante. A $q(e_0)$ fixé, nous remarquons que E_a est une fonction décroissante de $q(e_{-1})+q(e_1)$ et que E_b est une fonction croissante de cette même somme. Dans le but de maximiser l'efficacité dans le pire des cas, il est possible de calculer le point d'intersection de ces deux fonctions, qui représentera la valeur limite de l'efficacité globale. Soit $c' = q(e_{-1})+q(e_1)$. En résolvant l'équation $E_a = E_b$, nous obtenons $\bar{c'} = \sqrt{3}.q(e_0)$, en considérant uniquement la racine positive. En synthétisant, le pire des cas $E = max(E_a; E_b)$ est une fonction de c'. Elle est décroissante jusqu'à $\bar{c'}$ et croissante ensuite. Le pire des cas est donc atteint lorsque $c' = \sqrt{3}.q(e_0)$. L'efficacité dans le pire des cas est donc $\frac{1}{\sqrt{3}} > 0.57$.

Analyse en moyenne du pire des cas

Soit $X = \frac{q(e_{-1})+q(e_1)}{q(e_0)}$ avec $X \in [1;2]$ (si $X < 1$, on choisit trivialement e_0 comme arête active, en étant certain de faire le meilleur choix).
Le pire des cas est représenté par la fonction affine par morceaux $y(X)$ définie comme suit :
$y(X) = \frac{-1}{\sqrt{3}}X + \frac{\sqrt{3}+1}{\sqrt{3}}$ si $X \in \left[1;\sqrt{3}\right]$
$y(X) = \frac{1}{3}X$ si $X \in \left[\sqrt{3};2\right]$
C'est en fait le maximum des deux fonctions traçées précédemment (enveloppe supérieure).

Si $X \in [1;2]$, alors $P(X < x) = F(x) = -x^2 + 4x - 3$ avec $x \in [1;2]$ si nous supposons que $q(e_{-1})$ et $q(e_1)$ prennent des valeurs uniformément réparties entre 0 et $q(e_0)$.
Alors en moyenne l'efficacité est $\int_1^2 f(x)y(x)dx = \frac{10}{9}(\sqrt{3}-1) > 0.81$, avec $f(x) = F'(x)$.

Analyse en moyenne

Sous les hypothèses fixées précédemment, il est facile de remarquer qu'avec une probabilité 0.5, l'algorithme peut faire un choix non optimal. Ainsi, l'efficacité est $\frac{10(\sqrt{3}-1)+9}{18} > 0.9$ en moyenne.

4.2 Algorithme aléatoire

Le choix des arêtes actives, celles autorisées à transmettre des données dans la phase d'envoi, se fait de façon totalement aléatoire sans tenir compte des poids des arêtes. Cet algorithme (un des plus simples) sert à écarter éventuellement des algorithmes qui seraient moins performants, en terme de poids total des arêtes actives, ou en terme de stabilité notamment.

Phase de contrôle

Soient une arête $e \in E$.

a) e choisit aléatoirement et uniformément un backoff $b(e)$ dans l'intervalle $[1, T_{alea}]$.
$P(b(e) = i) = \frac{1}{T_{alea}}$ pour $i = 1..T_{alea}$.

b) Lorsque le backoff de e expire, plusieurs configurations sont possibles :

— Si e est inactive, e ne fait rien et n'envoie aucun message aux arêtes de $\varepsilon(e)$.
— Sinon e envoie un message à $\varepsilon(e)$.
  – Si e n'a reçu aucun message d'une arête de $\varepsilon(e)$, alors
  e devient active et envoie un message aux arêtes de $\varepsilon(e)$: $\forall e' \in \varepsilon(e)$, e' devient inactive.
  – Si e a reçu au moins un message de $\varepsilon(e)$, e devient inactive.

Phase d'envoi de données

Chacune des arêtes actives envoie certains paquets situés dans sa file d'attente (selon une politique FIFO par exemple). Remarquons qu'aucune phase de mise à jour des informations n'est requise ici.

Nombre de time slots de contrôle

Le nombre de time slots de contrôle est $T = 2T_{alea}$ car pour chaque valeur de backoff, 2 time slots sont nécessaires.

4.3 Algorithme backoff décroissant simple

Dans cet algorithme, le choix des arêtes actives est réalisé de façon aléatoire en donnant priorité aux arêtes de poids importants.

Phase de contrôle

Soit une arête $e \in E$ de poids $q(e)$. Si e était inactive lors de la précédente phase de contrôle (à cause d'une interférence par exemple), e augmente son $\sigma(e)$ de 1, sinon e diminue son $\sigma(e)$ de 1 ou reste égal à 0. Initialement (lors de la toute première phase de contrôle), $\sigma(e) = 0$.

a) e choisit un backoff $b(e) \in [1; T_{decr}]$ avec $b(e) = f(q(e), \sigma(e))$.
L'intervalle $[0; K]$ est partagé en T_{decr} sous-intervalles (de longueurs quasi identiques) : $t_1 = \left[0; \frac{K}{T_{decr}}\right]$, $t_2 = \left[\frac{K}{T_{decr}} + 1; \frac{2K}{T_{decr}}\right]$, ..., $t_{T_{decr}} = \left[\frac{(T_{decr}-1)K}{T_{decr}} + 1; K\right]$.
$f(x,y)$ prend une valeur aléatoire et uniformément distribué dans l'intervalle
$[T_{decr} - i - y + 1, T_{decr} - i + y + 1]$, avec i tel que $x \in t_i$ ou $i = T_{decr}$ si $x > K$.
Si $T_{decr} - i - y + 1 < 1$, $f(x,y) = 1$ et si $T_{decr} - i + y + 1 > T_{decr}$, $f(x,y) = T_{decr}$.

Une arête choisit un backoff selon une fonction décroissante de son poids. L'idée est de donner la priorité aux arêtes de poids importants. $\sigma(e)$ sert à dilater éventuellement l'intervalle selon le nombre de collisions lors des phases de contrôle précédentes.

b) Lorsque le backoff d'une arête e expire, plusieurs configurations sont possibles :

— Si e est inactive, e ne fait rien et n'envoie aucun message aux arêtes de $\varepsilon(e)$.
— Sinon e envoie un message à $\varepsilon(e)$.
  – Si e n'a reçu aucun message de $\varepsilon(e)$, alors
  e devient active et envoie un message aux arêtes de $\varepsilon(e)$: $\forall e' \in \varepsilon(e)$, e' devient inactive.
  – Si e a reçu au moins un message de $\varepsilon(e)$, e devient inactive.

Phase d'envoi de données

Chacune des arêtes actives envoie certains paquets situés dans sa file d'attente. Il n'y a pas de phase de mise à jour des informations détenues par une arête car l'algorithme fonctionne en supposant connu le poids maximum K d'une file d'attente. Cette hypothèse peut être levée facilement en donnant 1 comme valeur de backoff pour les arêtes de poids supérieur à K. C'est la technique que nous utilisons dans nos simulations (cf chapitre Simulations).

Nombre de time slots de contrôle

Le nombre de time slots de contrôle est $T = 2T_{decr}$ car pour chaque valeur de backoff, 2 time slots sont nécessaires.

4.4 Algorithme backoff décroissant évolué

Cet algorithme consiste en une phase de contrôle (divisée en deux sous-phases A et B), une phase d'envoi des données et une phase permettant une mise à jour des informations détenues par une arête $e \in E$ sur sa zone d'interférence $\varepsilon(e)$, pour prendre en compte notamment les arrivées de nouveaux paquets sur le réseau. Initialement, une arête $e \in E$ a une connaissance partielle (et parfois légérement erronée) des poids des arêtes de $\varepsilon(e)$. Elle saît notamment, avec une certaine probabilité, si elle est *subsituable* et/ou $0-dominee$. Cette probabilité sera quantifiée par la suite dans la partie analyse.

Phase de contrôle A

Soit une arête $e \in E$ de poids $q(e)$.

a) e choisit un backoff $b_1(e) \in [1, T_1]$.
— Si e est $0-dominee$, e tire aléatoirement et de façon uniforme, un backoff $b_1(e) \in [1, T_1']$. $T_1' < T_1$. Pour un réseau de type anneau, $T_1' \approx \frac{1}{4} T_1$.
— Sinon e tire un backoff $b_1(e) \in [T_1' + 1, T_1]$ avec $b_1(e) = f(q(e))$.
$f(q(e))$ est défini comme suit : l'intervalle $[0, K[$ est divisé en $T_1 - T_1'$ intervalles de longueurs quasi identiques. e choisit alors l'indice de l'intervalle correspondant à son poids de la même façon que dans l'algorithme backoff décroissant simple.

b) Lorsque le backoff d'une arête expire, plusieurs configurations sont possibles :
— Si e est inactive, e ne fait rien et n'envoie aucun message aux arêtes de $\varepsilon(e)$.
— Sinon (e nécessairement indéterminée) :
  – Si e est *substituable*, alors e reste indéterminée et n'envoie aucun message à $\varepsilon(e)$.
  – Sinon e envoie un message à $\varepsilon(e)$:
    - Si e n'a reçu aucun message de $\varepsilon(e)$, alors
    e devient active et envoie un message aux arêtes de $\varepsilon(e)$: $\forall e' \in \varepsilon(e)$, e' devient inactive.
    - Si e a reçu au moins un messagee de $\varepsilon(e)$, e devient inactive.

Phase de contrôle B

Dans cette seconde phase de contrôle (plus courte que la précédente), le but est de rendre actives certaines des arêtes restées indéterminées dans la phase de contrôle A. On applique en fait l'algorithme aléatoire, où une arête indéterminée $e \in E$ choisit un backoff $b_2(e)$ de façon uniforme dans l'intervalle $[1, T_2]$.

Phase d'envoi de données

Chacune des arêtes actives envoie certains paquets situés dans sa file d'attente.

Phase d'approximation de la connaissance du voisinage par une arête

Cette phase permet à chacune des arêtes d'obtenir une bonne approximation de la connaissance de $\varepsilon(e)$. Soit une arête $e \in E$ de poids $q(e)$. e est par défaut $0-dominee$ mais pas *substituable*. Cette phase permet notamment à e de savoir si elle est *substituable* et/ou $0-dominee$ avec une certaine

probabilité,. Cette probabilité sera décrite précisément par la suite.

a) e choisit un backoff $b_3(e) \in [1, T_3]$.

Chaque time slot $t \in [1, T_3]$ est divisé en deux mini time slots : t_1 et t_2.
e tire un backoff $b_3(e) \in [1, T_3]$ avec $b_3(e) = f_3(q(e))$. $f_3(x)$ est similaire à la fonction utilisée dans l'algorithme backoff décroissant simple.

b) A un time slot t donné, plusieurs cas de figure se présentent :
— Si $b_3(e) = t$, $(b1)$ décrit ce que fait e.
— Sinon, $(b2)$ décrit ce que fait e.
$(b1)$ représente en fait le protocole d'envoi et $(b2)$ le protocole de réception.

b1) e envoie un message à son voisinage $\varepsilon(e)$ lors du mini time slot t_1.
— Si au moins une arête de $\varepsilon(e)$ a envoyé un message lors de t_1, alors e n'envoie rien lors du mini time slot t_2.
— Sinon, e envoie à nouveau un message à $\varepsilon(e)$ durant t_2.

b2) Plusieurs cas de figure se présentent :
— Si $t \in [1, b_3(e) - 1]$ et e a reçu un message ou plusieurs messages qui se sont interférés entre eux, alors e n'est pas $0-dominee$.
— Si $t \in \left[f_3(q(e)-1), f_3(\frac{\sqrt{3}}{2}.q(e) - \varepsilon) \right]$, alors si e a reçu plusieurs messages lors du mini time slot t_2, e devient *substituable*.
— Si $t \in \left[f_3(q(e)-1), f_3(\frac{\sqrt{3}}{2}.q(e) - \varepsilon) \right]$, alors si e a reçu un unique message provenant d'une arête de $\varepsilon(e)$ lors du mini time slot t_2, alors elle enregistre l'idendité de cette arête. Lorsque t sera tel que $t < f_3(q(e))$ et que e n'est pas *substituable*, alors s'il existe au moins 2 arêtes $e_1 \in \varepsilon(e)$, $e_2 \in \varepsilon(e)$ dont les identités sont enregistrées et telles que $e_1 \notin \varepsilon(e_2)$, e devient substituable.

Conclusion

Prendre en compte la notion de substitution peut augmenter significativement l'efficacité de l'algorithme. L'objectif est de le faire en distribué. Nous avons implémenté l'algorithme greedy centralisé avec substitution dans le but de vérifier empiriquement l'apport de la substitution.

4.5 Algorithme log

Le coût de la mise à jour des informations détenues par chacune des arêtes (même partielles) peut être important, notamment dans le cas d'un modèle d'interférence quelconque, même si dans l'algorithme proposé dans [12], les arêtes n'ont pas d'information sur les arêtes voisines mais l'algorithme ne se généralise pas à un modèle d'interférence quelconque.
C'est pour cela que nous avons voulu ériger un **algorithme indépendant du modèle d'interférence** et ne demandant aucune information (même locale). La seule hypothèse effectuée est d'avoir une borne sur la taille des files d'attentes même si cette hypothèse peut être levée facilement, en donnant aux arêtes de poids supérieur à K, une valeur de backoff réservée pour ce type d'arêtes ou par défaut 1.

Supposons donc pour le moment que $\forall e \in E, q(e) \leq K$.

4.5.1 Description

La phase de contrôle est décomposée en deux sous phases (A et B). Elle a pour but de définir les arêtes actives, qui transmettront des données lors de la phase d'envoi.

Phase de contrôle A

k+1 mini slots sont nécessaires dans cette première phase de contrôle avec $k = \lceil log_2(K) \rceil$.
On va déterminer pour chaque arête un vecteur de longueur k, correspondant à son poids écrit en binaire.
Au slot t, les arêtes qui ont un 1 en t-ième position émettent un message (si elles ne sont pas inactives) et à l'instant k+1, les arêtes qui envoient un message sont celles qui ont un 0 en k-ième position (si elles ne sont pas inactives).

le tableau suivant donne pour $K = 7$ en fonction du poids et de son écriture en binaire, les slots d'émission d'une arête :

Poids	Ecriture binaire	Slots d'émission
0	000	4
1	001	3
2	010	2,4
3	011	2,3
4	100	1,4
5	101	1,3
6	110	1,2,4
7	111	1,2,3

En procédant ainsi, les arêtes ayant un poids plus grand que $\frac{K}{2}$ émettent lors du slot 1. Celles ayant un poids plus grand que $\frac{3K}{4}$ émettent aussi au slot 2 et ainsi de suite.
Pour 2 arêtes de poids différents, l'arête de plus grand poids émettra pendant un certain nombre de slots (éventuellement zéro) comme l'autre, mais aura ensuite un slot où elle émettra et pas l'autre. Par exemple, une arête de poids 7 émet, comme une arête de poids 6, aux slots 1 et 2 mais lors du

slot 3, la première émet alors que la seconde non.

Le protocole se déroule ainsi pour une arête e indeterminée au cours d'un time slot t quelconque :
— cas 1 : e n'émet pas mais reçoit un message : e devient inactive
— cas 2 : e n'émet pas et ne reçoit pas de message : e reste indéterminée
— cas 3 : e émet et ne reçoit pas de message : e devient active (et par le cas 1 ses voisines deviennent inactives)
— cas 4 : e émet mais reçoit aussi un message : e reste indéterminée (mais les arêtes de sa zone d'interférence n'ayant pas émis deviennent inactives)

Phase de contrôle B

3 slots de contrôle sont nécessaires dans cette seconde phase de contrôle.

a) Les arêtes redeviennent indeterminées sauf celles étant actives. Chaque arête active e envoie un message à $\varepsilon(e)$. Les arêtes recevant au moins un message deviennent inactives. Cela nécessite 1 slot de contrôle.

b) Chaque arête indéterminée choisit un backoff de façon uniforme dans l'intervalle $[0;2]$. On applique l'algorithme aléatoire (0 correspond au fait que l'arête restera inactive). Cela nécessite 2 slots de contrôle.

Cette phase de contrôle B, très courte, sert à palier d'éventuels "trous" dans le réseau. En effet, certaines arêtes se neutralisent entre elles et peuvent devenir inactives inutilement. Nous palions cela par un choix d'arêtes arbitraire afin de compléter l'ensemble de calls compatibles.

Lemma 1 *Après la phase de contrôle A, un sous-graphe connexe formé d'arêtes inactives n'ayant aucune arête active dans leur zone d'interférence respective, a un diamètre au plus* $2\lceil log_2(K)\rceil + 3$.

Phase d'envoi de données

Chacune des arêtes actives envoie certains paquets situés dans sa file d'attente.

Nombre de time slots de contrôle

Le nombre T de time slots nécessaires au total est $k+1+3 = \lceil log_2(K)\rceil + 4$. Le tableau ci-dessous décrit les valeurs de T selon K.

K	100	1 000	10 000	100 000
T	11	14	18	21

4.5.2 Analyse

Lemma 2 *Une arête $e \in E$ maximum local strict est active à la fin de la phase de contrôle A.*

Proof. Soit $e \in E$ une arête maximum local strict. Alors $\forall e_i \in \varepsilon(e)$, e_i n'a pas un backoff strictement plus petit que e. Dans le cas contraire, cela voudrait dire que $\exists e_i \in E, q(e_i) > q(e)$. Ce n'est pas possible car e est maximum local strict. Si e est la seule arête parmi les arêtes de $\varepsilon(e)$ à avoir ce backoff, alors elle est active.
Sinon, certaines arêtes de $\varepsilon(e)$ peuvent avoir un backoff identique. Dans ce cas, ces dernières et e se neutralisent à cet instant de l'algorithme. Les arêtes choisissent à nouveau un backoff. Cela est possible car sinon cela voudrait dire que ces arêtes ont le même poids. Ce n'est pas le cas car e est maximum local strict. Les arêtes choisissent à nouveau un backoff de la même manière. Ainsi, l'ordre n'est pas changé. Nous pouvons procéder par induction pour montrer qu'à la fin de la phase de contrôle, l'arête e maximum local strict est active. □

Lemma 3 *l'algorithme log a en moyenne une efficacité de $\frac{d}{d+1}$ du greedy centralisé.*

Proof. Soit un graphe $G = (V,E)$ avec des poids distribués aléatoirement et uniformément entre 0 et K. $\forall e \in E, |\varepsilon(e)| = d$.
Probabilité p qu'une arête soit maximum local strict dans G : $p = \frac{1}{d+1}$

L'algorithme greedy centralisé peut rendre active au plus $\frac{m}{d}$ arêtes. L'algorithme log rend actives toutes les arêtes maxima locales, soit en moyenne $\frac{m}{d+1}$ maxima locales. De plus les $\frac{m}{d+1}$ appartiennent toutes aux $\frac{m}{d}$ de l'algorithme greedy. Ainsi, l'algorithme log a en moyenne une efficacité de $\frac{d}{d+1}$ du greedy centralisé. □

Conjecture 1 *Soit $G = (V,E)$ un cycle de longueur n suffisamment grand. $\forall e \in E, q(e) \leq K$ et $\forall i, 0 \leq i \leq K, P(q(e) = i) = \frac{1}{K+1}$, avec K suffisament grand. $\forall e \in E, \forall e' \in \varepsilon(e), q(e) \neq q(e')$. La probabilité qu'une arête soit active après la phase de contrôle A est supérieure à* 0.38.

Lemma 4 *Cela revient à résoudre la récurrence suivante :*
$R(k_1,k_2) = \frac{1}{8}\sum_{i=0}^{k_1-1}\sum_{j=0}^{k_2-1}\frac{1}{2^{i+j}}R(i,j) + \frac{1}{2^{k_1+2}}\sum_{j=0}^{k_2-1}\frac{1}{2^j}R(k_1,j) + \frac{1}{2^{k_2+2}}\sum_{i=0}^{k_1-1}\frac{1}{2^i}R(i,k_2) + \frac{1}{2^{k_1+k_2}}R(k_1,k_2)$
$+\frac{1}{32}\sum_{i=0}^{k_1-2}\sum_{j=0}^{k_2-2}\frac{1}{2^{i+j}}R(i,j) + \frac{1}{2^{k_1+3}}\sum_{j=0}^{k_2-2}\frac{1}{2^j}R(k_1,j) + \frac{1}{2^{k_2+3}}\sum_{i=0}^{k_1-2}\frac{1}{2^i}R(i,k_2) + \frac{1}{2^{k_1+k_2}}R(k_1,k_2)$
$R(0,0) = 1$

Par la suite, nous essaierons de montrer que $lim_{k_i \to \infty} R(k_1,k_2) > 0.38$.
Cette récurrence, difficile à résoudre, converge néanmoins très rapidement. En calculant les valeurs pour $k_1 = k_2 = 5$, nous observons que la valeur de R est très proche de 0.38. Par la suite, nous essayerons de prouver analytiquement ce résultat.

Proof. Nous pouvons caractériser facilement le poids d'une arête e_0 par une suite de 0 et de 1 : $b_0, b_1, b_2, ..., b_{n-1}, b_n$. Si $b_i = 1$, cela signifie que e_0 enverra un message à $\varepsilon(e_0)$ au time slot i (si elle n'est pas inactive évidemment). Par hypothèse, $P(b_i = 1) = P(b_i = 0) = \frac{1}{2}$, $0 \leq i \leq n$.
Pour calculer la probabilité qu'une arête e_0 soit active, il est possible de regarder tous les cas favorables.

— Soit une arête devient active au premier time slot.
— Soit une arête reste indeterminée au premier time slot.
  – Soit une arête devient active lors du second time slot.
  – Soit une arête reste indeterminée lors du second time slot.
    - Soit une arête devient active lors du troisième time slot.
    - Soit une arête reste indeterminée lors du troisième time slot.
    ...

En énumérant tous les cas favorables (tous les cas où e devient active), on obtient la récurrence ci-dessus où k_1 et k_2 représentent respectivement le nombre d'arêtes à gauche qui peuvent encore influencer e et le nombre d'arêtes à droite qui peuvent encore influencer e. □

Exemples :

Exemple où e_0 devient active lors du second time slot :

Les arêtes sont ordonnées dans l'ordre du cycle. e_0 a 2 arêtes voisines à gauche et 2 arêtes voisines à droite.

time slots / arêtes	e_{-2}	e_{-1}	e_0	e_1	e_2
t_1	0	1	1	1	1
t_2	.	0	1	0	1

Le point signifie que l'arête e_{-2} n'influence plus e_0.

Exemple où e_0 devient active au troisième time slot :

e_0 a 4 arêtes voisines à gauche et 3 arêtes voisines à droite.

time slots / arêtes	e_{-4}	e_{-3}	e_{-2}	e_{-1}	e_0	e_1	e_2	e_3
t_1	1	0	0	0	0	0	1	1
t_2	.	.	0	1	1	.	.	.
t_3	.	.	.	0	1	.	.	.

Nous pouvons comprendre cela autrement en introduisant une chaîne de Markov à temps discret et à espace d'états discret. Le couple (i, j) représente un état du système (avec i et j des entiers non négatifs), où i (respectivement j) est le nombre d'arêtes à gauche (respectivement à droite) qui peuvent encore influencer e. La récurrence précédente revient à calculer la probabilité (partant d'un état quelconque) de se retrouver à l'état $(0,0)$. En fait $(0,0)$ est un état absorbant. Nous ajoutons artificiellement l'état $(-1,-1)$ qui correspond à l'inactivité de l'arête alors que $(0,0)$ représente l'activité d'une arête. Il y a donc 2 états absorbants : $(0,0)$ et $(-1,-1)$.
Soit X_t la variable représentant l'état du système au temps t. Nous avons clairement $lim_{t\to\infty}P(X_t = (-1,-1)) = 1 - lim_{t\to\infty}P(X_t = (0,0))$
Nous pouvons calculer la matrice de transition, pour $(i, j) \neq (-1,-1)$ et $(i, j) \neq (0,0)$:
$\forall t > 0, P(X_{t+1} = (i', j')/X_t = (i, j)) = 1_{i' \leq i, j' \leq j} \cdot (\frac{1}{2^{max(i'+1;i)+max(j'+1;j)+1}} + 1_{i' \neq i-1, j' \neq j-1} \cdot \frac{1}{2^{max(i'+2;i)+max(j'+2;j)+1}})$
Nous essayerons là aussi d'utiliser les techniques propres aux chaînes de Markov absorbantes pour résoudre analytiquement le problème.

Conclusion

L'algorithme log est un des algoritmes les plus intéressants parmi ceux proposés. Il admet un overhead constant (comme celui de Srikant dans [12]) et est adaptable à n'importe quel modèle d'interférence (ce n'est en revanche pas le cas pour l'algorithme de Srikant). De plus les conditions de stabilité mises en exergue dans les simulations semblent très encourageantes alors que le temps mis par l'algorithme de Srikant pour se stabiliser est décevant (cf simulations). L'étude de la stabilité sera approfondie dans la seconde partie du stage ingénieur.
En définitive, l'algorithme log vérifie toutes les caractéristiques que nous voulions et que nous avons défini dans les parties précédentes.

5 Simulations

Dans cette partie, il s'agit de comparer les différents aglorithmes distribués que nous avons proposé précédemment, l'algorithme distribué présenté dans [12], certains algorithmes centralisés et l'algorithme exhaustif (lorsque cela est possible). Nous expliquerons tout d'abord la méthodologie employée et enfin nous mettrons en exergue les principaux résultats.

5.1 Méthodologie

Les simulations sont réalisées en **JAVA** et sont exécutables en ligne via le lien
http ://www-sop.inria.fr/members/Dorian.Mazauric/simul.html
L'intégralité du travail réalisé se trouve sur la page
http ://www-sop.inria.fr/members/Dorian.Mazauric/wireless.html

5.1.1 Les algorithmes implémentés

— L'algorithme greedy centralisé qui consiste à choisir l'arête e encore indéterminée de poids maximum, de la rendre active et de rendre les arêtes de sa zone d'interférence $\varepsilon(e)$ inactives, et de faire cela tant qu'il reste au moins une arête indéterminée. L'efficacité de cet algorithme est au pire 0.5 de l'optimal dans le cas du modèle d'interférence primary node.
— Ce même algorithme greedy centralisé avec prise en compte de la notion de substitution.
— L'algorithme distribué aléatoire : les arêtes sont choisies aléatoirement.
— L'algorithme distribué backoff décroissant simple, utilisant un backoff décroissant en fonction du poids de l'arête.
— Sa version améliorée, l'algorithme distribué backoff décroissant évolué, sera implémentée dans la deuxième partie du stage ingénieur.
— L'algorithme distribué log.
— L'algorithme exhaustif, qui trouve le meilleur ordonnancement possible en testant toutes les configurations possibles. Il sera exéctuté uniquement pour des topologies particulières (chemin de moins de 30 sommets).
— L'algorithme distribué Srikant, développé dans [12], avec possibilité de changer la probabilité p et de changer la taille maximale d'une chaîne augmentante.

5.1.2 Les topologies implémentées

— Le chemin : l'utilisateur choisit à l'exécution le nombre de sommets.
— La grille : l'utilisateur choisit également à l'exécution les dimensions de la grille.
— Un graphe aléatoire.

5.1.3 Les modèles d'interférence implémentés

— Le modèle d'interférence primary node.
— Les modèles d'interférence, avec $d = 1$ puis d quelconque, seront implémentés par la suite.

5.1.4 L'application JAVA

Le programme développé en JAVA, permet de prendre en compte deux types de critère d'évaluation. Le premier permet de tester l'efficacité sur une étape des algorithmes implémentés en effectuant un nombre de tests choisi par l'utilisateur à l'exécution (mais sans arrivée ni départ de paquets). Le second permet de prendre en compte le départ et l'arrivée de paquets et d'étudier ainsi la stabilité des algorithmes. Le nombre de slots est choisi à l'excécution.

Critère d'évaluation 1 : poids total de l'ensemble des arêtes actives pour une étape

L'utilisateur choisit la topologie souhaitée, le nombre de tests à effectuer, les algorithmes à comparer et éventuellement de faire cela étape par étape (en déroulant les algorithmes à chaque clic de l'utilisateur). Durant un test, les poids sur les arêtes sont tirés aléatoirement et uniformément. Chaque algorithme exécute une phase de contrôle (en partant des poids originaux identiques évidemment). Ensuite la somme des poids des arêtes actives est divisée par la somme des poids des arêtes actives de l'algorithme exhaustif (si c'est un chemin de moins de 30 sommets). Sinon on le compare avec l'algorithme greedy centralisé. On obtient un ratio pour chaque algorithme que l'utilisateur a voulu tester (l'utilisateur coche au début les algorithmes qu'il veut). On fait cela pour le nombre de tests souhaité. A la fin, un graphique est affiché avec une courbe pour chaque algorithme (tous les ratios obtenus durant les n tests par un algorithme donné forme la courbe de cet algorithme). Nous pouvons ainsi visualiser l'efficacité, sur une étape, des algorithmes (sans prise en compte du temps, des départs et des arrivées de paquets). En fait le nombre de time slots est 1 dans chacun des tests. **Il est indispensable de souligner que le nombre de time slots de contrôle est équivalent pour chacun des algorithmes distribués. Dans le cas contraire, les comparaisons n'auraient aucun sens**.

Critère d'évaluation 2 : étude de la stabilité

Ce type de simulation ressemble fortement à la précédente, sauf que l'utilisateur choisit en plus le nombre de paquets quittant le réseau lorsqu'une arête est active (trafic single-hop) et le nombre de paquets arrivant sur le réseau. Ces valeurs sont constantes dans les simulations effectuées mais le caractère aléatoire a été également implémenté. Cela permet d'obtenir de bonnes indications sur les conditions de stabilité. Il choisit également le nombre de time slots (la durée de la simulation). A la fin, une courbe pour chaque algorithme représentant la somme totale des poids des arêtes divisée par la somme totale initiale, en fonction du temps, est traçée. Cela permet de visualiser la stabilité ou non des algorithmes.

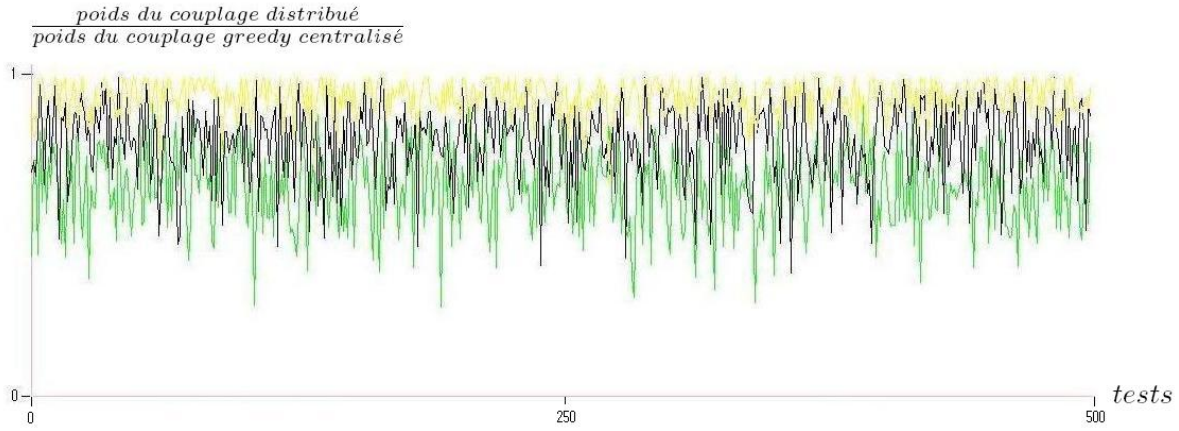
5.2 Principaux résultats

Analysons quelques simulations effectuées pour le modèle d'interférence primary node.

Critère d'évaluation 1 : poids total de l'ensemble des arêtes actives pour une étape

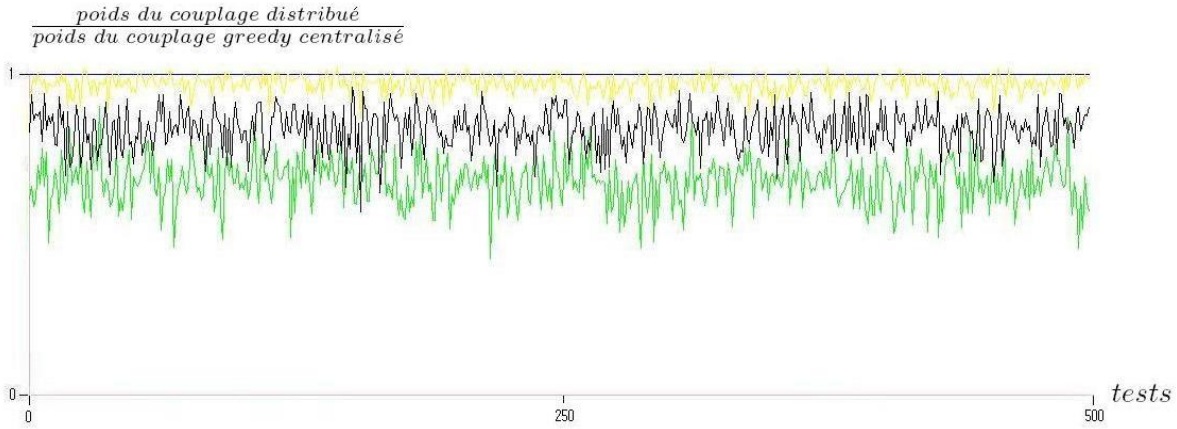
Notons ici que l'efficacité sur une étape de l'algorithme de Srikant n'est pas pertinente. En effet, l'algorithme repose sur la notion de la recherche de chaînes augmentantes au fur et à mesure des phases de contrôle. Dans cette partie, nous analysons uniquement le couplage sur un slot d'envoi de données et pour le premier slot, un couplage aléatoire est donné pour l'algorithme de Srikant. Il sera intéressant de simuler l'algorithme de Srikant en prenant en compte le temps et le départ et l'arrivée de paquets (cf étude de la stabilité).

La figure ci-dessous montre le résultat d'une simulation effectuée pour un chemin de 30 sommets pour 500 tests. Les algorithmes algorithme aléatoire, algorithme backoff décroissant simple et algorithme log, sont comparés à l'algorithme exhaustif (possible car 30 sommets est la limite pour l'utilisaiton de cet algorithme). L'efficacité de l'algorithme log est clairement visible notamment par rapport à l'algorithme aléatoire. L'algorithme backoff décroissant simple est moins bon que l'algorithme log mais dans une moindre mesure.



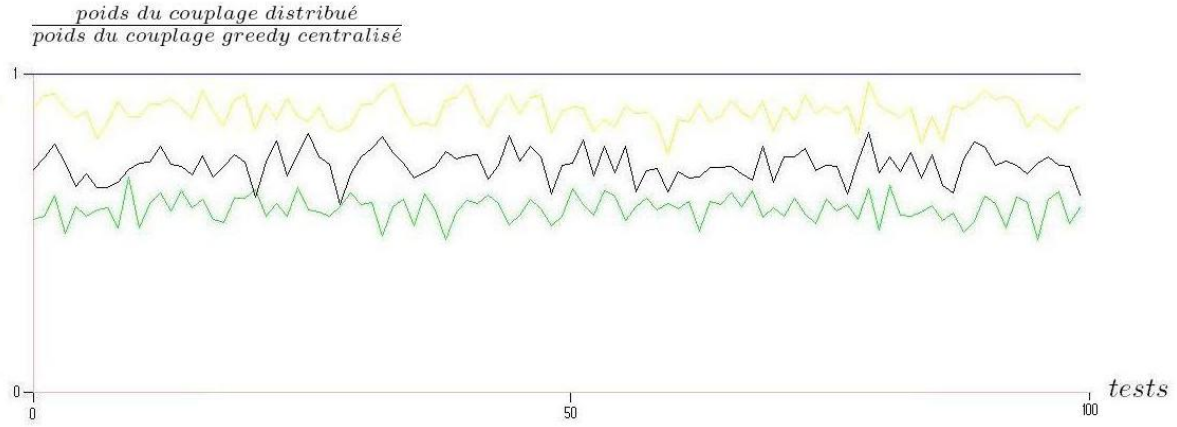
En **abscisse**, les tests effectués.
En **ordonnée**, le rapport poids du couplage des algorithmes aléatoire, backoff décroissant et algorithme log sur poids du couplage de l'algorithme exhaustif.

La figure ci-dessous montre le résultat d'une simulation effectuée pour un chemin de 100 sommets pour 500 tests. Les algorithmes algorithme aléatoire, algorithme backoff décroissant simple et algorithme log, sont comparés à l'algorithme greedy centralisé car le nombre de sommets est trop grand pour effectuer l'algorithme exhaustif. Nous observons que l'algorithme log est encore clairement meilleur, parfois même plus efficace que l'algorithme greedy centralisé.



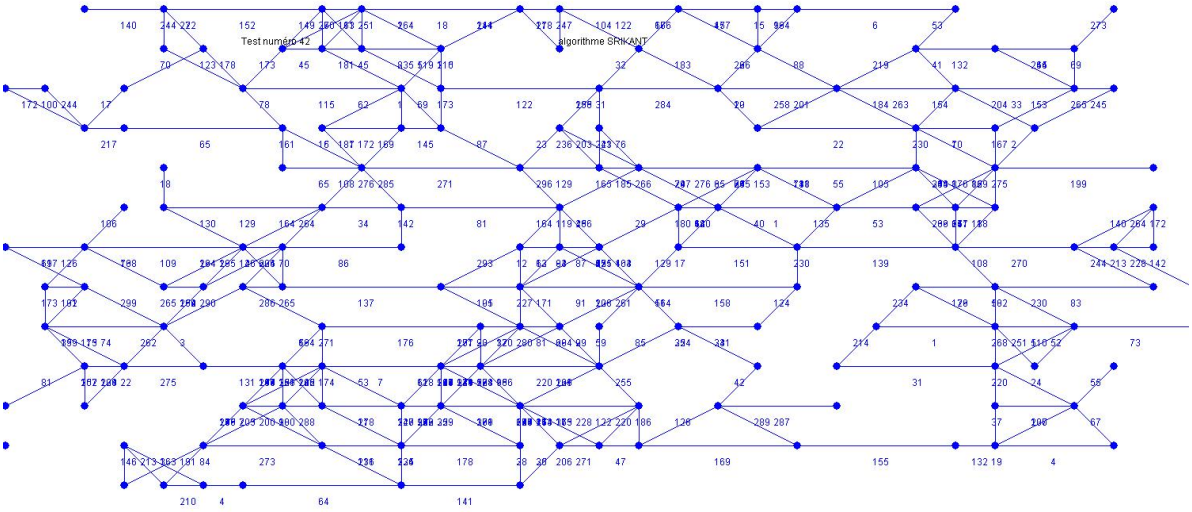
En **abscisse**, les tests effectués.
En **ordonnée**, le rapport poids du couplage des algorithmes aléatoire, backoff décroissant et algorithme log sur poids du couplage de l'algorithme greedy centralisé.

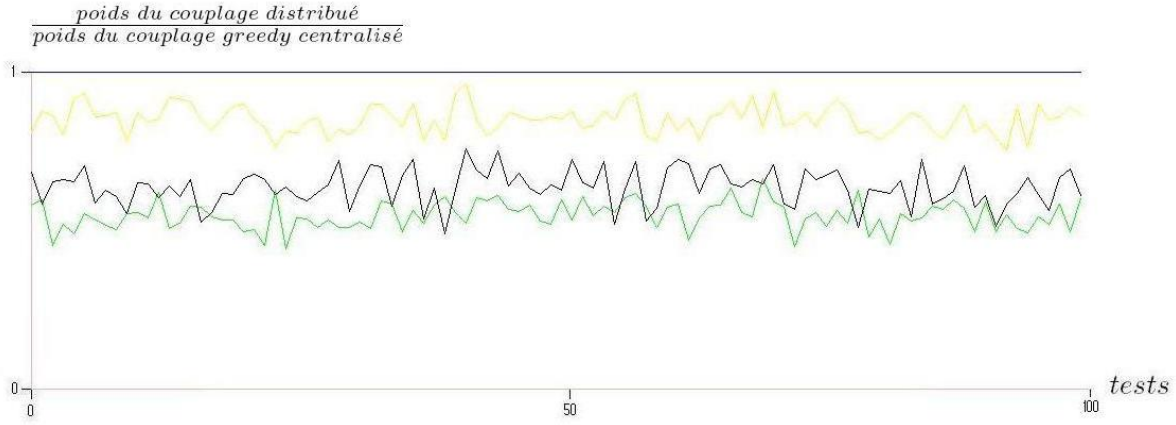
Nous avons ensuite effectué une simulation pour une grille composée de 15x15 sommets pour 100 tests. Les algorithmes algorithme aléatoire, algorithme backoff décroissant simple et algorithme log, sont comparés à l'algorithme greedy centralisé.



En **abscisse**, les tests effectués.
En **ordonnée**, le rapport poids du couplage des algorithmes aléatoire, backoff décroissant et algorithme log sur poids du couplage de l'algorithme greedy centralisé.

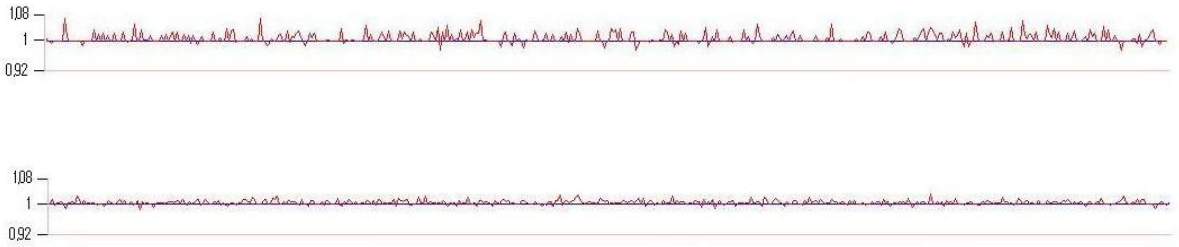
Une simulation sur un graphe aléatoire quelconque a été effectué (100 tests). L'algorithme log est encore nettement meilleur que les autres.





En **abscisse**, les tests effectués.
En **ordonnée**, le rapport poids du couplage des algorithmes aléatoire, backoff décroissant et algorithme log sur poids du couplage de l'algorithme greedy centralisé.

Enfin, nous pouvons voir sur les 2 captures ci-dessous (chemin de 100 sommets, grille de 15x15 sommets) que le principe de substitution améliore l'efficacité. Cela corroborre l'analyse théorique précédente. Les simulations sont réalisées avec les 2 algorithmes centralisés :
— l'algorithme greedy centralisé (bleu)
— l'algorithme greedy centralisé avec prise en compte du principe de substitution (rouge)

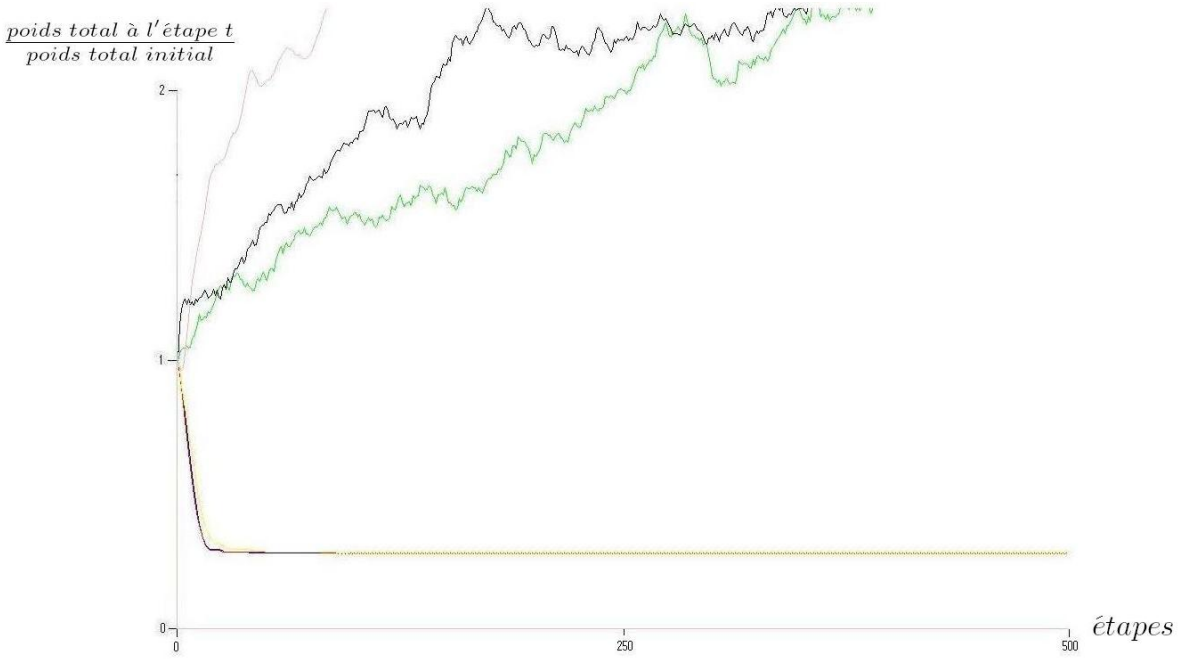


En **abscisse**, les tests effectués.
En **ordonnée**, le rapport poids du couplage de l'algorithme greedy centralisé avec substitution et sur poids du couplage de l'algorithme greedy centralisé.

Critère d'évaluation 2 : étude de la stabilité

Nous avons testé les algorithmes de façon plus réaliste en enchaînant les slots de contrôle et d'envoi de données (avec des départs et des arrivées de paquets). Dans un premier temps nous avons pris des valeurs constantes pour ces deux dernières valeurs. Nous avons comparé l'évolution des différents algorithmes pour différentes topologies et différentes valeurs (dans ce rapport, seules les simulations réalisées pour le chemin sont présentées).

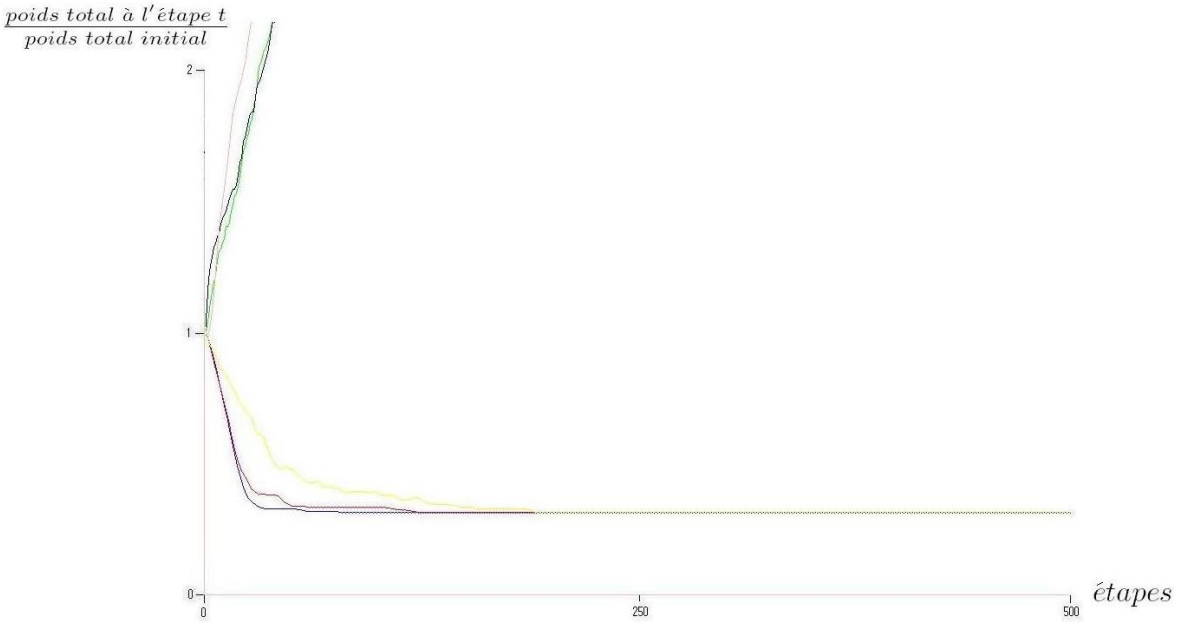
Pour un chemin de 100 sommets avec :
— 30 paquets qui partent lorsqu'une arête est active
— 10 paquets qui arrivent à chaque time slot



En **abscisse**, les 500 slots.
En **ordonnée**, le rapport poids total des arêtes après le slot t, pour les algorithmes aléatoire, backoff décroissant, algorithme log, algorithme de Srikant, greedy centralisé avec substitution et greedy centralisé sur poids total initial des arêtes.

Nous remarquons que l'algorithme log et l'algorithme greedy centralisé se comportent quasiment de la même façon. Ils convergent très rapidement vers un état stable. D'ailleurs la courbe bleue se confond avec la courbe jaune. L'algorithme aléatoire et l'algorithme backoff décroissant simple et l'algorithme de Srikant ne semblent pas converger immédiatement. Concernant l'algorithme de Srikant, le nombre de slots nécessaires avant stabilisation dépend fortement du choix de p.

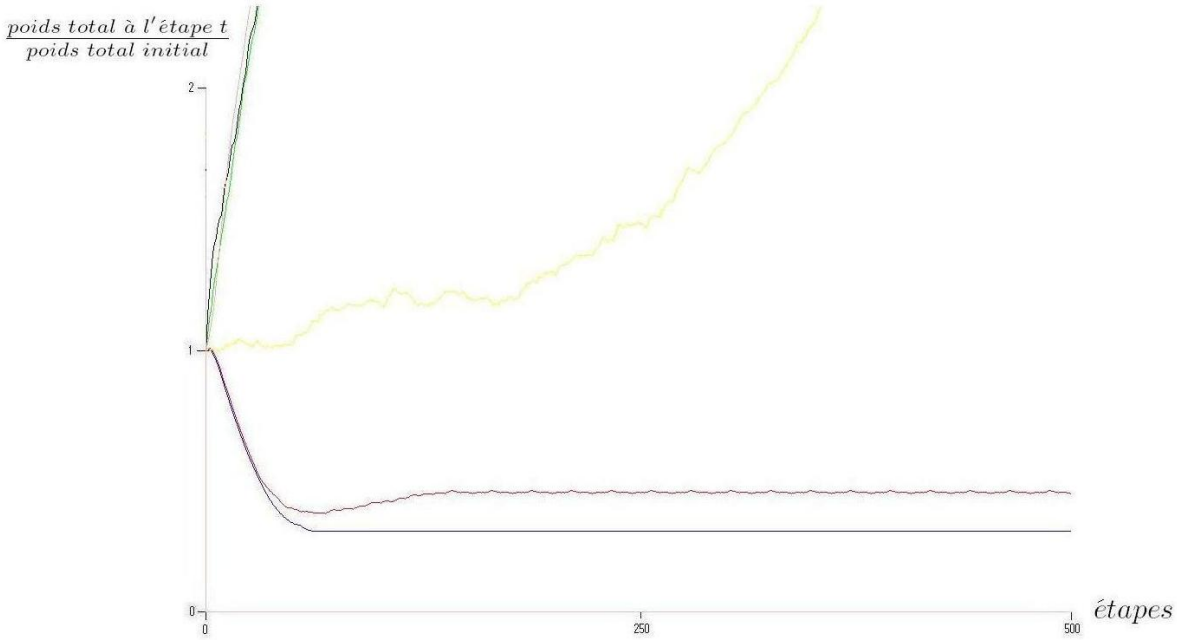
Pour un chemin de 100 sommets avec :
— **25** paquets qui partent lorsqu'une arête est active
— 10 paquets qui arrivent à chaque time slot



En **abscisse**, les 500 slots.
En **ordonnée**, le rapport poids total des arêtes après le slot t, pour les algorithmes aléatoire, backoff décroissant, algorithme log, algorithme de Srikant, greedy centralisé avec substitution et greedy centralisé sur poids total initial des arêtes.

Les deux algorithmes distribués ainsi que l'algorithme log sont stables et atteignent très rapidement leurs états stables. En revanche, les trois autres algorithmes distribués (algorithme aléatoire, algorithme backoff décroissant simple et algorithme Srikant) divergent clairement.

Pour un chemin de 100 sommets avec :
— **23** paquets qui partent lorsqu'une arête est active
— 10 paquets qui arrivent à chaque time slot



En **abscisse**, les 500 slots.
En **ordonnée**, le rapport poids total des arêtes après le slot t, pour les algorithmes aléatoire, backoff décroissant, algorithme log, algorithme de Srikant, greedy centralisé avec substitution et greedy centralisé sur poids total initial des arêtes.

Les algorithmes centralisés sont stables alors que l'algorithme log semble diverger. Les trois autres algorithmes distribués divergent encore plus rapidement.

6 Conclusion

En conclusion, les résultats sont encourageants mais d'autres simulations seront nécessaires pour valider ou non certains algorithmes. Il faudra notamment implémenter un modèle d'interférence quelconque et pouvoir faire varier tous les paramètres. De plus, des analyses plus théoriques seront effectuées notamment en ce qui concerne les efficacités des algorithmes distribués proposés et leurs conditions de stabilité.
En définitive, le bilan est positif car nous avons pu concevoir quelques algorithmes intérssants pour router le trafic dans un réseau sans-fil. Nous avons proposé notamment l'algorithme log, permettant de trouver un ensemble de calls compatibles dans un réseau sans-fil avec un overhead constant, valable quelque soit le modèle d'interférence et ne demandant aucune mise à jour d'informations. Certaines preuves théoriques sont encore incomplètes même si les simulations corroborrent les conjectures. Les premiers résultats palient les limites mises en exergue dans l'analyse des travaux existants.

En conclusion, les résultats sont encourageants et me motivent à poursuivre dans cette voie. L'application JAVA et des captures d'écran se trouvent en ligne :
(http ://www-sop.inria.fr/members/Dorian.Mazauric).

J'ai également poursuivi mon travail effectué lors du stage précédent, dans le domaine des jeux de capture. Cela s'est concrétisé par trois publications aux conférences **AlgoTel'2008** [3], **DISC'08** [4] et **OPODIS 2008** [5]. Un article est également soumis à **ICC 2009** [6]. Une partie en annexe est consacrée à ces travaux réalisés en parallèle avec David Coudert et Florian Huc.

Bibliographie

[1] R. Bar-Yehuda and A. Israeli. Multiple communication in multi-hop radio networks. In *PODC '89 : Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 329–338, New York, NY, USA, 1989. ACM.

[2] J-C. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Hardness and approximation of gathering in static radio networks. *Parallel Processing Letters*, 16(2) :165–183, 2006.

[3] D. Coudert, F. Huc, and D. Mazauric. Algorithme générique pour les jeux de capture dans les arbres. In *10ème Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications – AlgoTel'08*, Saint-Malo, May 2008.

[4] D. Coudert, F. Huc, and D. Mazauric. A distributed algorithm for computing and updating the process number of a forest (brief announcement). In G. Taubenfeld, editor, *22nd International Symposium on Distributed Computing (DISC)*, volume 5218 of *Lecture Notes in Computer Science*, pages 500–501, Arcachon, France, September 2008. Springer.

[5] D. Coudert, F. Huc, and D. Mazauric. Computing and updating the process number in trees (short paper). In T. Baker and S. Tixeuil, editors, *12th International Conference On Principles Of DIstributed Systems (OPODIS)*, volume x of *Lecture Notes in Computer Science*, Luxor, Egypt, December 2008, to appear. Springer.

[6] D. Coudert and D. Mazauric. Network reconfiguration using cops-and-robber games. Technical Report inria-00315568, INRIA, August 2008.

[7] D. Coudert, S. Perennes, Q-C. Pham, and J-S. Sereni. Rerouting requests in wdm networks. In *AlgoTel'05*, pages 17–20, Presqu'île de Giens, France, 2005. The algorithm for trees (section 3) proposed in this paper in false. A correct one is given in [CHM08].

[8] D. Coudert and J-S. Sereni. Characterization of graphs and digraphs with small process number. Research Report 6285, INRIA, September 2007.

[9] A. Gupta, Xiaojun Lin, and R. Srikant. Low-complexity distributed scheduling algorithms for wireless networks. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1631–1639, May 2007.

[10] X. Lin, N. B. Shroff, and R. Srikant. A tutorial on cross-layer optimization in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(8) :1452–1463, 2006.

[11] G. Narlikar, G. Wilfong, and L. Zhang. Designing multihop wireless backhaul networks with delay guarantees. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, April 2006.

[12] Sujay Sanghavi, Loc Bui, and R. Srikant. Distributed link scheduling with constant overhead. *SIGMETRICS Perform. Eval. Rev.*, 35(1) :313–324, 2007.

[13] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *Decision and Control, 1990., Proceedings of the 29th IEEE Conference on*, pages 2130–2132 vol.4, Dec 1990.

[14] X. Wu, R. Srikant, and J.R. Perkins. Scheduling efficiency of distributed greedy scheduling algorithms in wireless networks. *IEEE Transactions on Mobile Computing*, 6(6) :595–605, 2007.

7 Annexe

J'ai effectué un stage de trois mois en 2007, dans le cadre de ma deuxième année ingénieur, au sein de l'équipe-projet MASCOTTE. Sous la responsabilité de David Coudert, nous avons continué certains travaux dans le domaine du routage et de la reconfiguration dans un réseau WDM (Wavelength Division Multiplexing).
Décrivons précisément le domaine du stage. Dans un réseau optique WDM reconfigurable, il est possible de déplacer une connexion (canal de communication optique entre 2 noeuds du réseau, généralement à plus de 2.5Gbit/s) vers une nouvelle route. L'évolution du trafic au court du temps (ajout et suppression de connexions) entraîne une mauvaise utilisation des ressources du réseau. Une reconfiguration du réseau consiste alors à déplacer des connexions pour retrouver une utilisation optimale des ressources. Toutefois, le déplacement d'une connexion comporte des risques : arrêt du trafic ou perte d'informations par exemple. Aussi il faut élaborer des algorithmes de reconfiguration minimisant par exemple le nombre de déplacements.

Le travail à réaliser était le suivant : lors du déploiement d'un réseau de communication de type WDM, les opérateurs dimensionnent les liens de communications (câbles de plusieurs fibres optiques) en fonction d'une connaissance du trafic à l'instant t et d'une estimation de sa croissance. L'évolution du trafic est due à deux phénomènes, d'une part une évolution lente due à l'apparition et à la disparition de clients (réseaux d'entreprises) et, d'autre part, une évolution rapide des besoins en bande passante des clients présents. Aussi est-il intéressant pour l'opérateur de savoir modifier la configuration de son réseau selon les besoins et à moindre coût. Dans les réseaux que nous considérons, une demande de connexion entre 2 noeuds du réseau correspond à l'affectation d'un chemin et d'une longueur d'onde, chaque connexion servant une partie des besoins en bande passante d'un client. Etant donné un réseau et un ensemble I de requêtes, le problème d'affecter un chemin optique (route et longueur d'onde) à chaque requête est NP-difficile mais des approximations performantes existent. Dans ce contexte, une évolution du trafic correspond à l'ajout ou au retrait de connexions. Chaque requête retirée libère des ressources qui pourront être utilisées pour l'ajout de nouvelles requêtes. Lors de l'ajout de nouvelles connexions, il s'agit de trouver une route et une longueur d'onde dans le réseau sans modifier les connexions déjà établies. Dans certaines configurations, il est impossible d'établir une connexion pour une requête alors que le problème de trouver un routage optique pour l'ensemble des requêtes admet une solution. Pour contourner cette difficulté, il faut développer des stratégies de reconfiguration du réseau (déplacement de connexions sur d'autres routes) permettant une meilleure utilisation des ressources.

Au cours de ce stage, nous avons abordé le problème suivant : Etant donné un réseau WDM et une instance I de routage R obtenus a la suite d'une succession d'ajouts et suppressions de requêtes.

Soit R' un routage optimal de l'instance I dans le réseau. Trouver la stratégie de reconfiguration la plus performante pour passer de R à R'. Nous avons travaillé plus particulièrement sur l'élaboration d'un algorithme de reconfiguration d'arbres.

Ce problème est équivalent à ceux rencontrés dans le domaine des jeux de captures dans les graphes. En effet, le process number d'un graphe orienté (digraphe) est un invariant qui a été introduit dans [7] et [8] dans le cadre de l'étude de la reconfiguration du routage à l'intérieur d'un réseau WDM. Les modifications à apporter sont modélisées par un digraphe, qui peut être vu comme un digraphe de conflits.
Passer d'un routage R_1 à un routage R_2 revient à supprimer tous les sommets du digraphe des conflits en respectant les règles suivantes :
— On peut à tout moment mettre un sommet dans une mémoire temporaire s'il n'y est pas déjà. Ce faisant, on supprime tous les arcs entrant dans ce sommet.
— On peut supprimer un sommet qui n'a pas d'arc sortant. Si de plus le sommet était dans une mémoire temporaire, il la libère.
Une suite d'actions permettant de supprimer tous les sommets d'un digraphe D est appelée stratégie. Le nombre maximum de mémoires temporaires simultanément utilisées est appelé coût de la stratégie. Le minimum des coûts des stratégies permettant de supprimer tous les sommets est le process number du digraphe, noté pn(D). Le process number d'un graphe (non orienté) G est le process number du graphe orienté symétrique associé D.

Nous avons développé un algorithme distribué simple calculant le process number des arbres en n étapes, avec un nombre total d'opérations en $O(nlog(n))$ et un total de $O(nlog(n))$ bits échangés. De plus cet algorithme est facilement adaptable pour calculer d'autres paramètres sur l'arbre, dont le node search number.
Nous tentons maintenant d'adapter cet algorithme pour le calcul du process number des graphes planaires extérieurs (outerplanars).

Nous avons cette année poursuivi et amélioré le travail réalisé l'an dernier. Nous avons publié trois articles sur ce sujet (avec David Coudert et Florian Huc) aux conférences **AlgoTel'2008** [3], **DISC'08** [4] et **OPODIS'2008** [5]. De plus, un autre article est à l'heure actuelle **soumis à ICC 2009** [6].
Les articles et l'application JAVA associée, se trouvent en ligne :
http ://www-sop.inria.fr/members/Dorian.Mazauric/jeux.html