Titre: Title:	Étude des méthodes et algorithmes pour gérer le problème de la fragmentation dans un réseau optique
Auteur: Author:	Louis Cadiou
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Cadiou, L. (2022). Étude des méthodes et algorithmes pour gérer le problème de la fragmentation dans un réseau optique [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10432/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Étude des méthodes et algorithmes pour gérer le problème de la fragmentation dans un réseau optique

LOUIS CADIOU
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Juillet 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Étude des méthodes et algorithmes pour gérer le problème de la fragmentation dans un réseau optique

présenté par **Louis CADIOU**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Sébastien LORANGER, président
Brunilde SANSO, membre et directrice de recherche
André GIRARD, membre

REMERCIEMENTS

Tout d'abord, je remercie Mme Brunilde Sansò pour m'avoir donné l'opportunité d'être son élève, pour m'avoir aidé, aiguillé et conseillé tout au long de cette maitrise.

Je remercie aussi CIENA pour m'avoir permis de travailler sur ce projet. Je souhaite aussi remercier plus particulièrement Scott Kohlert de CIENA pour sa disponibilité pour ce projet. Il a été d'une grande aide pour comprendre le fonctionnement des réseaux optiques et pour trouver des données liées à ceux-ci.

Ensuite, je remercie ma famille pour son support tout au long de mes études et notamment pour m'avoir aidé à préparer mon départ vers le Canada.

Enfin, je remercie Emilie Quenedey pour avoir relu ce mémoire et donné des commentaires précieux.

RÉSUMÉ

Les réseaux optiques sont en évolution permanente, proposant des structures de plus en plus flexibles. Le spectre est divisé en plusieurs tranches ou slots de fréquences et cette grille de slots de fréquences est passée d'une largeur de centaines de GHz à 6.25 GHz avec la dernière génération d'Elastic Optical Network (EON). Ceci permet d'améliorer les performances des réseaux, mais pose également de nouveaux problèmes. Le problème sur lequel nous allons principalement nous pencher est d'assigner les connexions dans un réseau.

La structure des réseaux optiques impose des contraintes de continuité et contigüité par rapport au spectre. Les demandes ne sont plus assignables sur n'importe quel slot libre dans le réseau. Les arrivées et départs de ces demandes entraînent l'apparition de slots difficilement accessibles sans violer une des deux contraintes. Dans ce cas, on parle de fragmentation du réseau.

L'état de fragmentation du réseau est intimement lié au blocage observé. Pour un même trafic, plus la fragmentation du réseau est considérée comme haute, plus le blocage doit être haut. Des métriques ont été construites pour quantifier la fragmentation du réseau et des algorithmes pour assigner les demandes ont été conçus pour minimiser cette fragmentation.

Afin de déterminer la qualité d'une métrique ou d'un algorithme, on a mis en place au sein de l'équipe de recherche un outil informatique pour simuler un tel réseau optique.

Pour améliorer le problème de la fragmentation dans un réseau optique, le but de ce mémoire est de proposer une description de l'ensemble des algorithmes et métriques présents dans la littérature. On discute aussi de certains aspects autour de ces métriques et algorithmes. Enfin, on propose un nouvel algorithme pour répondre au problème de l'assignation.

L'algorithme proposé se base sur l'idée de partition du spectre. Brièvement, on assigne à chaque taille de demandes possibles (granularité) une région du spectre. Pour évaluer notre algorithme, on a mis en place une méthodologie et un ensemble de tests. On évalue graduellement les performances de notre algorithme en partant d'un simple lien pour arriver jusqu'à un petit réseau. Au cours de cette méthode, on a mis en place des tests pour optimiser certains paramètres de cet algorithme afin d'avoir de bonnes performances tant sur le plan du blocage que des métriques de fragmentation. On compare les résultats de notre algorithme avec des algorithmes sans partition présents dans la littérature.

On observe qu'un algorithme partitionné possède de meilleures performances tant sur le blocage que sur les métriques de fragmentation qu'un algorithme non partitionné pour le cas

du lien et du chemin. Le passage au réseau cause des difficultés supplémentaires pour le calcul de nos partitions. On propose deux méthodes pour calculer les tranches de spectres à l'échelle d'un réseau. On évalue ces deux méthodes de slicing pour le cas d'un réseau. On observe pour ce cas qu'il est possible d'obtenir de meilleurs résultats de blocage et fragmentation. Cependant, on montre aussi que cette méthode possède certaines limites lorsque le trafic est plus hétérogène.

En conclusion de ce mémoire, on a étudié une méthode plutôt prometteuse. Elle pourrait être mise en place dans un réseau optique pour répondre au problème de l'assignation de demandes. L'introduction de l'idée de slicing pourrait aussi servir comme base pour d'autres algorithmes. On peut conserver le slicing tout en étudiant d'autres algorithmes d'assignation à l'intérieur de chacun des slices.

ABSTRACT

Optical networks are constantly evolving, offering more and more flexibility in their structure. Indeed, the frequency slot grid has decreased from hundreds of GHz to 6.25 GHz with the latest generation of EON. This improves the performances of these ones but it brings new problems. One of these is the problem of assigning the connections in a network.

The structure of optical networks imposes two main constraints : continuity and contiguity. Demands can no longer be assigned to any free slot in the network. The arrivals and departures of these demands cause the appearance of slots that are difficult to access without breaking one of the two constraints. This is called network fragmentation.

The state of fragmentation of the network is closely related to the observed blocking. For a given traffic, the higher the network fragmentation is, the higher the blocking should be. Metrics have been developed to quantify network fragmentation and algorithms for assigning demands have been designed to minimize this fragmentation.

In order to determine the quality of a metric or an algorithm, a simulator has been set up by the research team in order to simulate such an optical network.

In order to tackle the problem of fragmentation in an optical network, the purpose of this master thesis is to describe the different algorithms and metrics that exist in the literature. We discuss some issues related to these metrics and algorithms. Finally, we propose a new algorithm to deal with the assignment problem.

The proposed algorithm is based on the concept of spectrum partition. Briefly, we assign to each possible demand size (granularity) a region of the spectrum. To test our algorithm, we set up a methodology and a set of tests. We gradually evaluate the performance of our algorithm starting from a single link to a small network. During this methodology, we set up tests to optimize some parameters of this algorithm in order to have good performances both in terms of blocking and fragmentation metrics. We compare the results of our algorithm with non-partitioned algorithms present in the literature.

We observe that a partitioned algorithm has better blocking rate and fragmentation metrics performances than a non-partitioned algorithm for the link and path cases. We propose 2 methods to compute restrictions for the granularities at a network level. We observe that better results can be obtained for this case as well. However, we also show that this method has some limitations when we have more heterogeneous traffic.

To conclude, we have studied a rather promising method. It could be implemented in an optical network to respond to the problem of demand assignment. The introduction of the idea of slicing could also serve as a basis for other algorithms. Slicing can be conserved while studying other assignment algorithms inside each of the slices.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AWG	Array Wave Guide
DOC	Domain Optical Controller (controlleur optique de domaine)
DWDM	Dense Wavelength Division Multiplexing (multiplexage dense en longeur d'onde)
EON	Elastic Optical Network (réseau optique élastique)
ROADM	Reconfigurable Optical Add Drop Multiplexer (multiplexeur optique d'insertion-extraction reconfigurable)
RSA	Routing and Spectrum Assignment (routage et assignation de spectre)
RWA	Routing and Wavelength Assignment (routage et assignation de longueurs d'ondes)
SDH	Synchronous Digital Hierarchy (hiérarchie numérique synchrone)
SNR	Signal to Noise Ratio (Rapport signal sur bruit)
SONET	Synchronous Optical NETwork (réseau optique synchronisé)
WDM	Wavelength Division Multiplexing (multiplexage en longeur d'onde)
WSS	Wavelength Selective Switch (commutateur sélectif en longueur d'onde)

LISTE DES NOTATIONS MATHÉMATIQUES

G	Graphe
V	Ensemble des noeuds
L	Ensemble des liens
s	Slot ou tranche de spectre
S_i	Ensemble de slots du chemin i
d	Demande
D	Ensemble des demandes
c	Connexion
C	Ensemble des connexions
B	Ensemble des slots disponibles
f_i	Fragment de slots disponibles
P	Ensemble des chemins possibles
EF	External Fragmentation
SE	Shannon Entropy
HM	High slot Mark
$RMSF$	Root Mean Square Factor
ABP	Access Blocking Probability
WS	Wasted Slots
FF	First Fit
FA	Fragmentation Aware
ρ	Flux de demande
slice_value	numéro du slot frontière entre la partie partionnée du spectre et celle qui ne l'est pas
NS	Non-sliced
FS	Fully sliced
N	Nombre de canaux
E	blocage

CHAPITRE 1 INTRODUCTION

L'utilisation de l'optique pour communiquer remonte à de nombreux siècles. Les Indiens avaient notamment mis en place un système de signaux de fumée pour communiquer sur de longues distances. Ce genre d'idée a été repris par la suite par Claude Chappe pour mettre en place le télégraphe de Chappe en 1794. C'est un système de sémaphores récupérant un signal lumineux par le sémaphore précédent et transmettant ce même signal lumineux au sémaphore suivant. Ceci constitue des exemples de communications optiques à l'air libre. Avec le fort développement de l'électricité au cours du XIXe siècle, les communications optiques ont été laissées de côté durant de nombreuses années. Avec le développement des premières fibres optiques durant le XXe siècle et la mise en place des premiers lasers dans les années 60, les communications optiques regagnent en intérêt mais cette fois-ci en propagation guidée. La première démonstration expérimentale d'un système optique via une fibre et un laser est donnée en 1966 par Charles Kao, en collaboration avec Georges Hockman. Par la suite, le développement des technologies associées, la mise en place de protocoles de communication tels que SONET (Synchronous Optical NETwork) ou SDH (Synchronous Digital Hierarchy) ont largement installé les réseaux à fibres optiques comme incontestables pour des communications rapides, fiables et à longue distance. En effet, l'optique présente plusieurs avantages pour ce type de communication. L'atténuation dans une fibre optique atteint maintenant environ 0.2 dB par kilomètre alors que cela peut être plusieurs dizaines de dB pour un câble en cuivre. De plus, la communication avec des fréquences optiques se fait avec des signaux de quelques centaines de THz permettant une bande passante disponible bien plus grande.

Les réseaux optiques jouent maintenant un rôle fondamental dans la transmission d'information dans le monde entier. La dernière version de ces réseaux optiques se nomme Elastic Optical Network (EON).

La structure EON permet une grande flexibilité sur la mise en place et les opérations sur les réseaux. Cependant, ceci introduit un nouveau problème : celui de la fragmentation. Nous en donnerons une définition explicite au cours du chapitre 2.

Dans le chapitre 3, nous nous attarderons sur les métriques et algorithmes présents dans la littérature permettant de traiter ce problème d'assignation des demandes.

Dans le chapitre 4, nous présenterons brièvement le simulateur développé pour simuler le fonctionnement d'un réseau optique et dans le chapitre 5, nous introduirons un nouvel algorithme pour assigner les demandes dans un réseau optique. Nous comparerons les résultats de l'algorithme proposé avec des algorithmes existants dans la littérature. On commencera

par étudier les cas simples du lien et du chemin. Dans le chapitre 6, on introduira le cas du réseau et les difficultés que ce cas rajoute pour notre algorithme.

L'objectif de ce mémoire est de contribuer à l'état de l'art au niveau de la planification et l'assignation des demandes dans un EON en considérant le problème de la fragmentation.

CHAPITRE 2 LES TECHNOLOGIES D'UN RESEAU OPTIQUE

2.1 Bref fonctionnement d'une communication optique

On souhaite envoyer de l'information à travers une fibre. On va donc définir à l'émetteur la liste de symboles à transmettre. Les degrés de liberté optique pour encoder notre symbole sont l'amplitude de notre signal et la phase de celui-ci. Ce symbole va être généré par un laser. La lumière se propage dans la fibre et est reçue par une photodiode retransformant alors le signal optique en signal électrique décodable. De ce signal électrique, on va retrouver l'information initiale. Au cours de la propagation dans la fibre, le signal subit des effets linéaires et non-linéaires conduisant à une atténuation du signal voir à une distorsion de celui-ci. Le but de notre communication est d'être à la fois capable de décoder notre signal malgré cela, mais aussi de limiter ces effets.

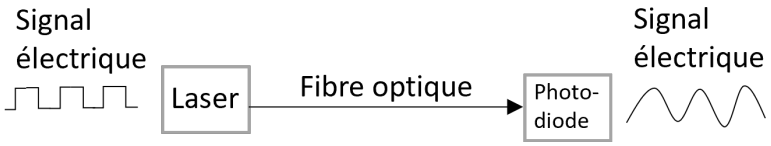

FIGURE 2.1 Schéma simple du fonctionnement d'une communication optique

La qualité d'un signal optique est notamment fonction du SNR (Signal to Noise Ratio). Lors de longue communication, le signal affaibli peut ne plus être suffisamment puissant pour être décodé. Il y a alors un amplificateur optique pour que celui-ci regagne en puissance. Cependant, l'amplification du signal n'est pas sans coût, elle dégrade le signal et augmente le SNR. Il n'est donc pas judicieux d'amplifier trop souvent notre signal. De plus, les effets non linéaires sont plus importants lorsque la puissance du signal est plus grande. Il faut donc aussi éviter d'amplifier trop notre signal. Classiquement, dans un réseau optique, un amplificateur est placé tous les 70 à 80 km. Lorsque le SNR atteint un niveau trop faible dû aux amplifications successives, le signal ne peut pas aller plus loin, il faut décoder ce signal et générer un nouveau signal si l'on veut atteindre des distances plus lointaines. Cependant, la regénération d'un signal est extrêmement coûteuse et est évitée autant que possible. Le SNR requis pour chaque transmission dépend des caractéristiques du récepteur et de l'émetteur. La propagation du signal dans une fibre ne se fait que dans un sens. Ce qui veut dire que dans un réseau optique, un lien considéré comme bidirectionnel se représente en fait comme

deux liens unidirectionnels. 2 demandes allant de A vers B et B vers A ne partagent donc pas les mêmes ressources.

Le taux de transmission en bits/s dans un réseau optique dépend de 2 facteurs :

— Le taux entre 2 symboles

— La constellation utilisée pour encoder ces symboles

Plus la constellation utilisée est complexe, plus 2 symboles différents sont proches dans celle-ci et donc plus il peut être difficile de décoder les symboles. Le bruit du système a plus de chance de créer une erreur. On voit alors que la complexité de la constellation est très sensible au bruit et n'est pas infiniment complexifiable. Pour ce qui est de le taux entre 2 symboles, chaque pulse possède une bande passante. Plus un laser envoie des pulses de manière fréquente en temps, plus la bande passante en fréquence de ces pulses augmente. Pour augmenter le taux de transmission d'un signal, il faut donc élargir la bande passante donnée à ce signal.

2.2 Les réseaux DWDM

Aujourd'hui, les réseaux optiques se positionnent principalement autour de 3 bandes :

— la bande O (1280-1325nm)

— la bande C (1528-1565nm)

— la bande L (1565-1620nm)

Les bandes C et L sont les plus communes et sont notamment les bandes principales des réseaux actuels. Afin d'augmenter la capacité des réseaux optiques, on a divisé le spectre optique en une multitude de slots de fréquence ou longueur d'onde sur lesquels on peut y mettre des signaux différents. Ceci représente le multiplexage en longueur d'onde (WDM). Chaque signal va être généré individuellement puis être rassemblé autour d'un même signal à l'aide d'un multiplexeur optique et va se propager en partageant la même fibre. Les signaux vont être séparés au récepteur via un démultiplexeur. Cette technologie représente la base des réseaux optiques DWDM (Dense Wavelength Division Multiplexing). Les ressources allouées par ces réseaux sont alors des tranches de fréquences que se partagent les usagers. La taille de ces tranches est de 50 ou 100 GHz pour des réseaux DWDM autour d'une fréquence de 193 THz (ou 1 550 nm). Un système DWDM est un prolongement de WDM. Il possède un espacement des canaux plus fins que DWDM. Le composant optique permettant d'effectuer ce multiplexage et démultiplexage des signaux est un réseau de guides d'onde (AWG) dans les réseaux DWDM.

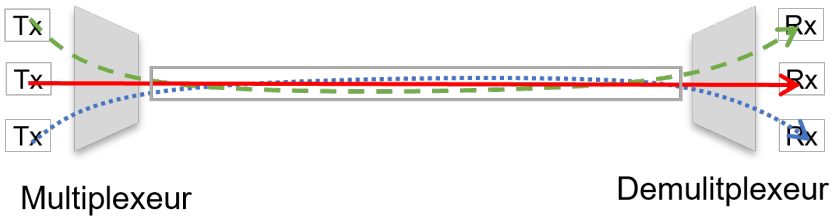

FIGURE 2.2 Multiplexage en longueur d'onde sur une transmission optique

Le routage dans le réseau se fait lors des passages dans des ROADM (Reconfigurable Optical Add Drop Mux). Sur ces nœuds, de nouveaux signaux peuvent être ajoutés à ceux arrivant (fonction add), d'autres peuvent le quitter (fonction drop). Chacun des signaux sera routé vers la bonne sortie pour atteindre sa destination finale [2].

Ceci paraît bien simple d'effectuer le bon routage de chacun des slots de fréquence à l'intérieur de notre signal WDM. Mais ceci est permis grâce à un composant optique à l'intérieur du ROADM : le commutateur sélectif en longueur d'onde WSS (Wavelength Selective Switch). Le WSS est un système composé d'un réseau de diffraction, de lentille, de miroir et de filtres permettant de séparer la lumière et alors d'ajouter de nouveaux ou signaux ou bien de séparer des signaux routés vers des endroits différents. La figure 2.3 montre un exemple de WSS qui ajoute une longueur d'onde dans une fibre.

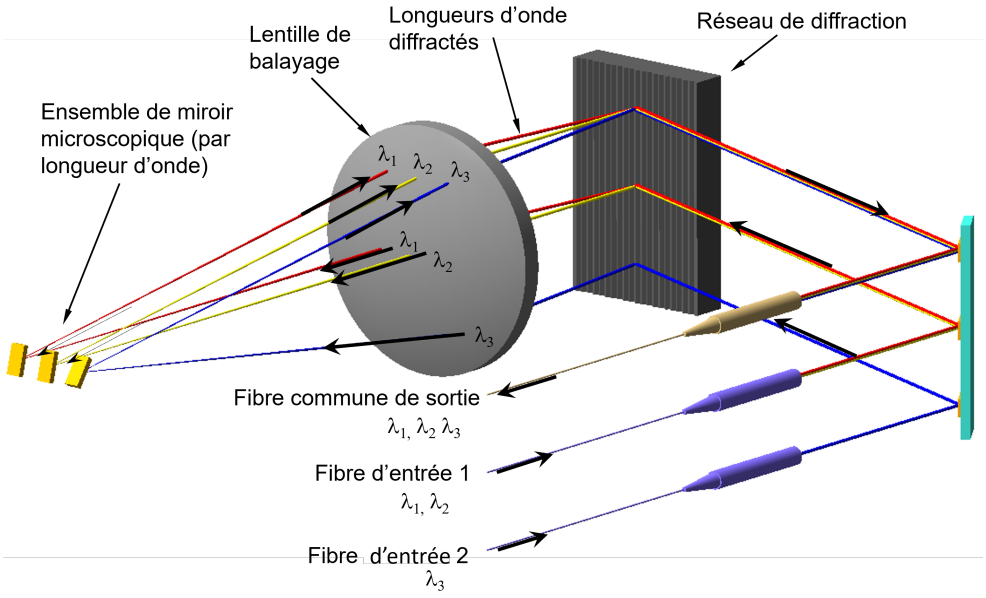

FIGURE 2.3 Exemple d'un module WSS [1]

Le contrôle général du réseau se fait au niveau des contrôleurs de domaines optiques DOC (Domain Optical Controller). Un ensemble de DOC récupère des informations sur le réseau telles que la qualité de chaque canal, l'utilisation ou non de ces canaux. De ces informations, le DOC est capable d'optimiser les canaux de chacun en modifiant les gains de chaque amplificateur afin de minimiser le SNR et les effets non linéaires. Il est aussi en charge de l'ajout ou de la suppression de demandes.

La majorité des réseaux DWDM fonctionne sur la bande C simplement, mais certains réseaux peuvent fonctionner avec les deux bandes en simultanée. En effet, les équipements optiques sont dimensionnés pour fonctionner sur une seule bande, il faut donc avoir des équipements pour chacune des bandes (par exemple, un amplificateur pour la bande L et un autre pour la bande C). La fibre en elle-même peut transporter les deux bandes, mais travailler avec la bande L dégrade les performances de la bande C. Ces difficultés font que les réseaux considérés par la suite fonctionneront avec la bande C.

Finalement, les composants principaux d'un réseau DWDM sont une source générant le signal, un récepteur optique, des amplificateurs optiques pour garder une puissance optique suffisante à la réception, des multiplexeurs et démultiplexeurs pour combiner et décombiner les signaux de chacun et des ROADMs assurant le bon routage de chacun des signaux. Ce réseau est géré depuis un ensemble de DOCs permettant de faire les choix pertinents pour assurer le bon fonctionnement de celui-ci.

2.3 Vers la structure EON

La conception des réseaux DWDM pose cependant encore quelques soucis. En effet, la grille de 50 GHz ou 100 GHz pose des problèmes de flexibilités. Une demande entrante ne pouvant prendre que des multiples de cette grille, il pouvait être parfois compliqué d'utiliser efficacement toute la bande passante sur ce slot. Un des enjeux est de faire évoluer cette grille fixe vers une grille plus flexible permettant alors de mieux ajuster la bande passante offerte pour satisfaire celle-ci. On quitte alors la structure DWDM pour aller vers une nouvelle structure EON (Elastic Optical Network). Celle-ci a la particularité d'avoir une grille bien plus fine, classiquement de 6.25 GHz ou 12.5 GHz. Ici, ce qu'on appelle slot représente un élément de cette grille, soit une tranche de fréquences de 12.5GHz ou 6.25GHz. Les demandes entrantes vont demander un certain débit pour transmettre ces informations. On va donc lui associer une paire modulation et taux de transmission. Cette paire a besoin d'un SNR minimal et d'une bande passante minimale comme discuté dans la section 2. La bande passante requise se traduit dans le réseau par un ensemble de slots formant un super-canal. Par exemple, si ma demande requiert 100GHz, on va chercher un ensemble de $\lceil \frac{100}{6.25} \rceil$ slots, soit 16 slots. Ce

passage entre ces 2 grilles permet d'améliorer grandement l'efficacité spectrale et de répondre à la demande croissante en données. Cependant, elle ajoute quelques nouveaux défis pour pouvoir profiter efficacement de cette nouvelle structure [3].

2.4 Cas d'usage

On considère un réseau EON avec une grille de 6.25GHz sur la bande C. Ceci représente un ensemble de 768 slots de fréquences. Des demandes arrivent et partent de notre réseau optique. Chacune de ces demandes requiert une certaine bande passante se traduisant dans notre réseau par un nombre de slots. Ces demandes possèdent une origine et une destination. Un des objectifs est de trouver un chemin optique entre l'origine et la destination, avec suffisamment de slots pour répondre à la demande et l'introduire sur le réseau via le chemin optique trouvé. Pour cela, nous devons savoir comment trouver un tel chemin possible dans un réseau optique. S'il y a plusieurs chemins possibles, quel chemin devons-nous choisir ? La question sur la notion de fragmentation des réseaux optiques sera également abordé et on montrera que deux solutions possibles peuvent ne pas être égales de ce point de vue.

2.5 Mise en place du réseau

Notre réseau optique peut se représenter par un graphe G, avec V ses nœuds qui sont les ROADM et L ses liens. Chaque lien est considéré comme bidirectionnel, c'est-à-dire qu'il y a une fibre optique mise en place pour chaque sens comme discuté en 2. Le poids sur chaque lien est le nombre d'amplificateurs présents sur ce lien. Un exemple de réseau est donné sur la figure 2.4. Chaque lien possède un nombre de slots. Avec une grille de 6.25GHz dans la bande C, chaque lien sera composé de 768 slots. Un slot est une tranche de spectre noté s. s est égal à 1 si le slot est occupé et 0 sinon. On définit l'ensemble des slots du chemin i comme S_i. On note $d \in D$, une demande appartenant au set de demande. Chaque demande est composée d'une origine, d'une destination ainsi que d'un nombre de slots requis (ou une bande-passante requise). Cette demande devient une connexion $c \in C$, l'ensemble des connexions du réseau, lorsqu'on lui trouve une solution et qu'elle est insérée dans le réseau.

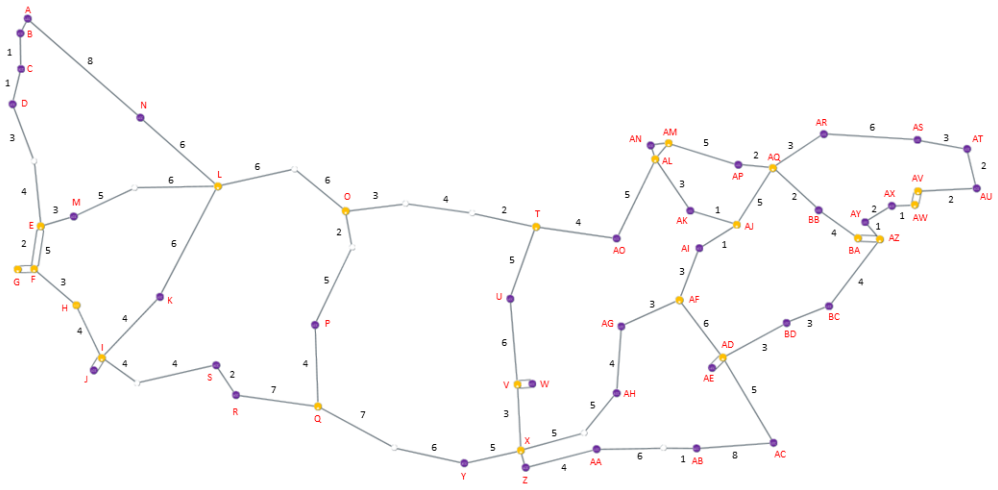

FIGURE 2.4 Exemple d'un réseau fourni par CIENA

2.6 Le problème de routage et de l'assignation

Pour l'assignation de demande, on nomme ce problème comme Routing and Spectrum Assignment (RSA). Cette terminologie est propre aux réseaux EON et est nommé plutôt Routing and Wavelength Assignment (RWA) dans le cas de réseau DWDM [2]. Ce problème complexe de l'assignation des demandes peut être subdivisé en 2 problèmes : le problème de routage et le problème d'assignation. On va définir au préalable les routes possibles. Par exemple, certaines routes sont impossibles à cause d'un trop grand nombre d'amplificateurs traversés (SNR trop haut). Par la suite on va faire l'assignation sur ces routes. On peut diviser le problème RSA en 2 variantes : le RSA dit offline et le RSA dit online [4]. Dans le cas du RSA offline, les demandes sont connues à l'avance et on les assigne de la manière la plus précise possible. Dans le cas du RSA online, on ne connait pas les demandes à l'avance. Le réseau s'adapte de manière dynamique en assignant les demandes au fur et à mesure de leur arrivée. Dans notre cas, on s'occupe plutôt du problème online. On présente dans les sections suivantes les contraintes sous-jacentes au problème ainsi que les éventuels défis et difficultés que cela représente. On verra comment les contraintes amènent au problème de fragmentation et comment ce problème affecte notre façon d'aborder le RSA.

2.7 Les contraintes des réseaux optiques

Nous souhaitons assigner à une demande un chemin allant de son origine à sa destination. Cependant, le choix de ce chemin est soumis à certaines contraintes invalidant des chemins

paraissant à première vue possibles. Ceci est dû à la conception des réseaux optiques ainsi que leurs possibilités technologiques. Cela représente principalement 3 contraintes [5] :

— la contrainte de capacité
— la contrainte de continuité
— la contrainte de contiguïté

Ces 3 principales contraintes sont utilisées dans l'ensemble des documents reliés à ce problème. D'autres contraintes plus spécifique comme par exemple sur un chemin de secours [6] peuvent aussi être considérées. Dans notre cas, on considère seulement ces 3 contraintes.

2.7.1 La contrainte de capacité

Deux connexions partageant le même lien ne peuvent pas partager les mêmes slots. On a donc que chaque slot s ne peut prendre la valeur que 0 ou 1. En reprenant les notations de la table 0.1, on peut formaliser cette contrainte via l'équation suivante :

$$\forall l \in L, \forall s \in S_l, s \leq 1 \tag{2.1}$$

Cette contrainte est assez simple et naturelle. Elle assure de ne pas avoir une capacité infinie sur chaque slot : deux demandes ne peuvent pas occuper le même slots sur le même lien.

2.7.2 La contrainte de continuité

Pour une demande d, la contrainte de continuité impose que lors d'un chemin, cette demande d doit garder la même fréquence tout au long de son chemin. Pour p un tel chemin et i, i'sont 2 liens de ce chemin, on peut formaliser cette idée :

$$S_i(d) = S_{i'}(d) \tag{2.2}$$

où $S_i(d)$ sont les slots de la demande d sur le chemin i

Cette contrainte est due au fait qu'une fois le signal généré, il est impossible de changer de manière peu coûteuse la longueur d'onde de ce signal optique. Les outils permettant de faire le passage entre signal en bande de base et un signal en radiofréquence, ne sont utilisables que pour traiter des signaux optiques à quelques centaines de THz. La seule méthode est de décoder le signal optique, puis regénérer un signal optique à partir d'un signal électrique. Méthode que l'on va éviter au possible.

La contrainte sur la continuité restreint les possibilités d'assignation d'une demande. On considère un réseau de 3 nœuds et 2 liens consécutifs A-B et B-C tel que sur la figure 2.5. On

représente l'occupation des slots sur ces liens avec slots grisés comme représenté par la figure 2.6. On souhaite insérer une demande de 1 slot entre A et C. La contrainte de continuité impose que certaines solutions sont impossibles. Notamment, le slot numéro 2 est libre sur le lien B-C, mais pas sur le lien A-B, il ne peut donc pas être utilisé par notre demande. En revanche, le slot 4 est une solution potentielle.

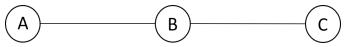

FIGURE 2.5 Simple réseau de 3 nœuds

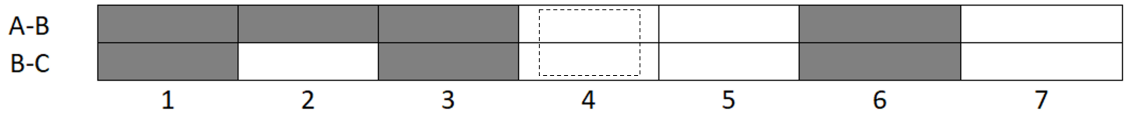

FIGURE 2.6 Illustration de la contrainte de continuité

2.7.3 La contrainte de contiguïté

On a évoqué dans la section 2 qu'augmenter le taux de transmission revient à augmenter la bande de fréquence nécessaire pour porter notre signal. Cette bande n'est logiquement pas divisible. Ainsi, si une demande requiert plusieurs slots, ces slots doivent être contigus. Ceci représente la contrainte de contiguïté. On considère cette fois la même disposition que dans la section 2.7.2. On souhaite insérer une demande de 2 slots sur B-C. On voit que les slots 2 et 7 sont disponibles, mais non contigus. Ils sont inutilisables pour satisfaire notre demande. En revanche, les slots 4 et 5 sont parfaitement utilisables pour répondre à notre demande.

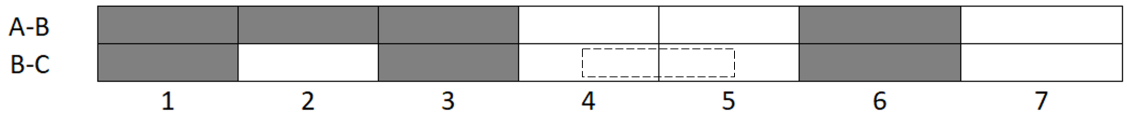

FIGURE 2.7 Illustration de la contrainte de contiguïté

2.8 Vers le problème de fragmentation

Ces 2 contraintes rendent certaines zones du spectre difficilement accessible de part soit un manque de slots consécutifs pour cette tranche de spectre, soit un manque de slots disponibles

avec la même longueur d'onde sur les slots voisins. En considérant le même réseau que dans les 2 sections précédentes, une demande de A vers C nécessitant 2 slots ne peut se placer que sur les slots 4-5. Le slots 2 est disponible sur le lien B-C, mais pas sur le lien A-B ce qui violerait la contrainte de continuité, et le slot 7 est disponible sur les 2 liens, mais ne possède pas de slots adjacents disponibles, ce qui violerait la contrainte de contiguïté.

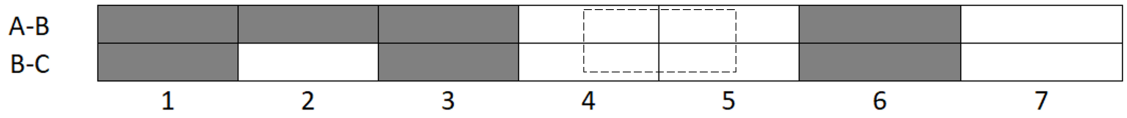

FIGURE 2.8 Illustration des 2 contraintes

Le problème de placement devient plus difficile avec ces contraintes et on remarque aussi que le placement des connexions précédentes affecte le placement des connexions futures. Si on place mal les connexions, on voit que des trous peuvent se créer dans notre spectre. Ces trous sont plus compliqués à utiliser : des connexions peuvent être rejetées, car il n'y aura pas de slots consécutifs ou continus malgré qu'il y ait sur chacun des liens suffisamment de ressources. On va perdre en efficacité spectrale. Pour illustrer cela, on a simulé l'allocation de demande dans un réseau et on a pris une capture de l'état de 2 liens dans la figure 2.9. Cette capture a été faite dans un simulateur mis en place et qui nous a permis de calculer différents résultats. Les détails sur ce simulateur seront donnés dans une partie suivante. On peut voir des trous fins sur la figure 2.9 compliqué à utiliser sans violer la contrainte de continuité et des mauvais alignements, par exemple le rectangle rouge, qui sont impossibles à utiliser pour une de demande passant par les liens A-B et B-C

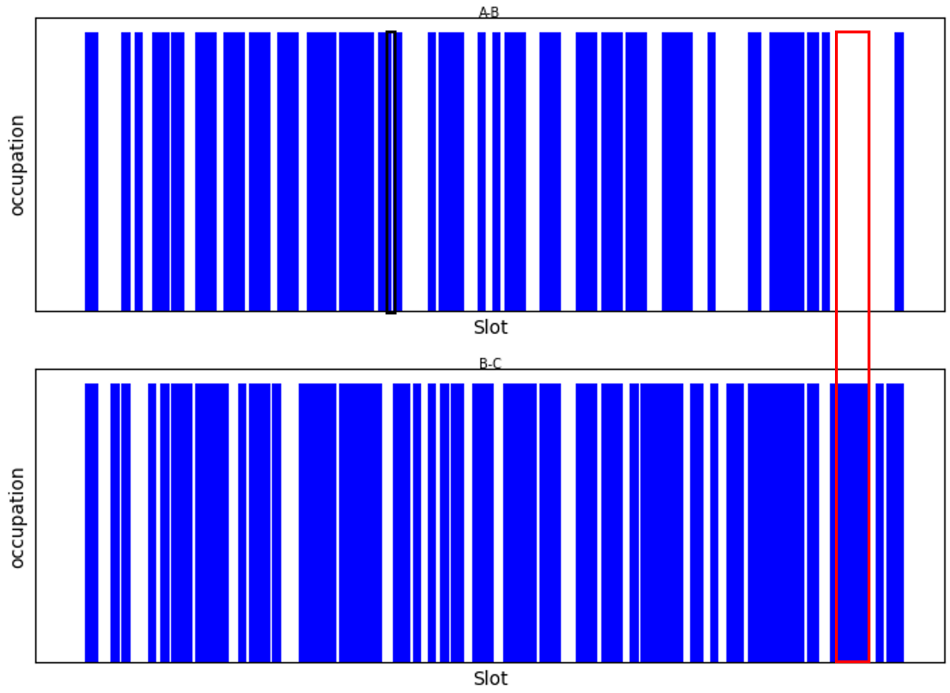

FIGURE 2.9 Capture de l'occupation de 2 liens

On parle dans la littérature de fragmentation du spectre optique [7,8]. En plus des contraintes imposées pour trouver une solution, il est primordial de considérer ce problème de fragmentation. Notre problème reste de trouver une solution pour satisfaire le plus de demande possible, cependant, il faut être méthodique dans notre assignation sous peine de fragmenter notre réseau, de perdre en efficacité spectrale, ce qui pourrait dans le futur nous entrainer un bloquage plus haut dû à ce manque d'efficacité. On considère qu'un réseau est fragmenté lorsque ces parties de spectres inutilisables ou difficilement utilisables sont nombreuses. Ce terme fragmentation n'est pas nouveau ou spécifique aux réseaux optiques. En effet, le terme fragmentation a déjà été utilisé pour caractériser un problème similaire de l'allocation de mémoire dans un ordinateur [9]. Une partie des algorithmes développés pour ce problème ont pu être récupéré et adapté au problème du RSA. Cependant, notre problème spécifique est plus élaboré que le problème d'allocation de mémoire. On a une fragmentation à deux dimensions dû à nos 2 contraintes. On a une fragmentation suivant les liens (contigüité) et suivant le chemin (continuité). On désigne ces 2 types de fragmentations comme fragmentation verticale et horizontale [2, 7, 8].

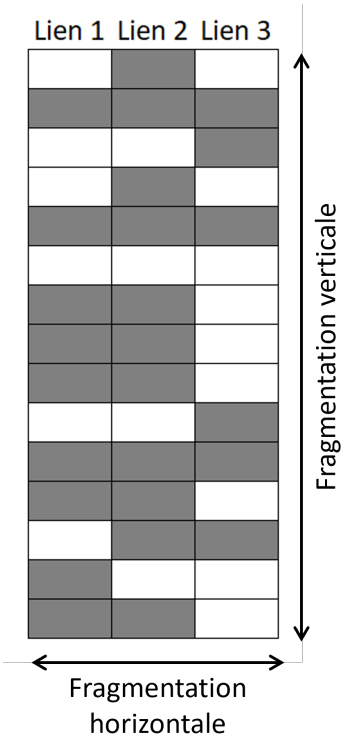

FIGURE 2.10 Illustration des 2 types de fragmentations dans un réseau optique

La fragmentation verticale est la fragmentation reliée à la disposition des slots sur un lien. Idéalement, on souhaite que les slots disponibles forment un seul même bloc.

La fragmentation horizontale est reliée à l'alignement des slots entre les liens. Via la figure 2.10, on voit que chaque lien peut accueillir individuellement une demande de 2 slots mais la demande sera rejetée par le système à cause du non-alignement de ces slots. Idéalement, pour minimiser la fragmentation verticale, on souhaite que les slots disponibles entre tous les liens soient à la même fréquence.

CHAPITRE 3 METRIQUES ET ALGORITHMES

3.1 Les métriques de fragmentation

On a défini la fragmentation de manière visuelle comme une mauvaise disposition de nos connexions. Cependant, cette définition n'est absolument pas quantitative. On peut dire qu'une disposition est moins bonne qu'une autre si elle est plus susceptible de bloquer les demandes qui arrivent. Cependant, regarder le blocage n'est pas une démarche proactive pour prévenir de la fragmentation : on souhaite savoir que notre réseau est fragmenté avant qu'il y ait un grand nombre de demandes bloquées. On définit alors des métriques de fragmentation comme des fonctions évaluant la fragmentation sur un lien. De nombreuses métriques ont été développées dans la littérature avec des avantages et désavantages. Le but de la métrique est de faire une corrélation entre le blocage à long terme et la valeur donné par la métrique : pour deux systèmes identiques, avec les mêmes demandes, si la métrique de fragmentation est moins grande sur un des deux systèmes, son blocage devrait être moins haut. Un objectif peut être la minimisation d'une métrique de fragmentation afin de minimiser la fragmentation dans le réseau et logiquement minimiser le blocage.

La majorité des métriques est définie pour un lien. La valeur de la métrique pour un réseau est la somme des valeurs de métrique pour chacun des liens. Idem pour un chemin, c'est la somme de chacun des liens. Il est possible aussi de mettre un poids pour chaque lien suivant son importance et de faire une moyenne pondérée des liens [10].

3.1.1 Le taux de blocage

Le but d'un réseau est de minimiser le nombre de connexions rejetées afin de satisfaire un maximum de clients. Une métrique utile pour estimer l'état d'un réseau est le taux de blocage, soit le pourcentage de connexions rejetées sur le nombre de connexions totales. Un fort de taux de blocage signifie que notre réseau est en surcharge et/ou fragmenté. Cependant, il est difficilement prédictible, la formule d'Erlang [11] n'est pas exacte pour les réseaux optiques à cause de la distribution contigüe des slots d'une demande (contrainte de contigüité) et il n'existe pas d'autre formule estimant le taux de blocage. Ce taux de blocage ne sera pas utilisé en tant que métrique dans le réseau, mais plutôt comme un des paramètres pour estimer la qualité d'un résultat obtenu.

3.1.2 La métrique de fragmentation externe (EF)

Ce type de métrique a tout d'abord été utilisé pour le problème de la fragmentation de mémoire [9] puis elle a été adaptée pour les réseaux optiques [7]. On peut la définir sur un lien avec la fonction F_{EF} suivante :

$$F_{EF} = 1 - \frac{A}{B} \tag{3.1}$$

avec A : nombre de slot max contigus et B : nombre de slots disponibles

Par exemple, en considérant la distribution sur lien de la figure 3.1, on peut calculer la fragmentation externe de ce lien. On a 7 slots libres d'où B = 7 et le plus gros bloc est de 3 d'où A = 3.

$F_{EF} = 1 - \frac{3}{7} \approx 0.57$

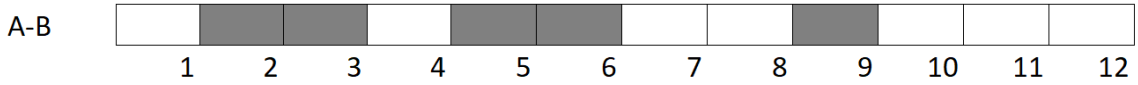

FIGURE 3.1 Exemple de distribution sur un lien

La métrique EF est entre 0 et 1. Dans le cas idéal, les slots disponibles forment un simple bloc et on n'a pas de fragmentation, la valeur est alors 0. Dans le pire cas, chaque slot disponible est un bloc de 1, le lien est très fragmenté et la valeur est 1-$\frac{1}{B}$. Cette métrique est très simple à calculer, mais pose quelques problèmes. En effet, sur la figure 3.2, on a représenté 2 liens qui ont la même valeur de fragmentation. Cependant, on voit visuellement que le deuxième lien devrait être moins fragmenté de par son nombre de fragments de 3 contre 4 sur le premier lien, de par la présence de 2 blocs de 3 slots contre 1 sur le premier lien et de par la présence d'un seul bloc de 1 slots contre 2 pour le premier lien.

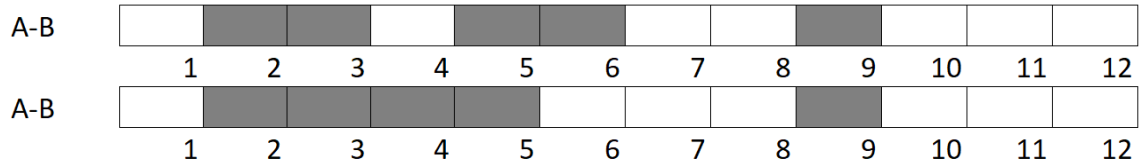

FIGURE 3.2 Problème de la métrique EF

3.1.3 L'entropie de Shannon SE

La formule d'entropie de Shannon est une des formules les plus connues dans les communications numériques pour déterminer la quantité d'information dans un message. De plus, la notion d'entropie est aussi utilisée en physique pour mesurer la désorganisation d'un système. Il parait assez naturel de mesurer la fragmentation ou le désordre des slots sur un lien avec une formule similaire. Wright *et al* [12] ont utilisé cette notion pour mettre en place une mesure de fragmentation. On peut définir la métrique SE :

$$F_{SE} = \sum_i \frac{f_i}{B} * \ln \frac{B}{f_i} \tag{3.2}$$

avec B : nombre de slot disponible et f_i la taille de chacun des fragments sur le lien

En reprenant le même exemple que précédemment (figure 3.1), on a toujours B = 7 :

$F_{SE} = \frac{1}{7} * \ln(7) + \frac{1}{7} * \ln(7) + \frac{2}{7} * \ln(\frac{7}{2}) + \frac{3}{7} * \ln(\frac{7}{3}) \approx 1.28$

On voit que dans le cas idéal d'un seul bloc libre on a bien la métrique qui tend vers 0. Le pire cas est bien d'avoir que des blocs de 1 slot et dans ce cas la fragmentation du réseau est log(B). Quelques petits slots agrandissent modérément la valeur de la métrique grâce à la pondération en $\frac{1}{B}$ d'un des termes. Mais un grand nombre de petits slots font énormément augmenter la valeur de la métrique SE.

De plus, par cette présence du nombre de slots disponibles, il ne parait pas judicieux de comparer 2 liens avec des taux de remplissages différents (soit un B différent). Cette métrique est plus pertinente pour comparer des liens ou réseaux avec une charge de trafic similaire.

Un autre souci est que cette métrique ne considère pas l'index du dernier slot alloué. Ce facteur est intéressant, car on sait qu'après le dernier slot alloué, les slots sont libres. Or si le dernier slot alloué est le même sur tous les liens (et le plus faible possible), on a un ensemble de slots libres sur tous les liens et notamment pour la contrainte de continuité, cela est fort souhaitable.

3.1.4 La métrique de l'indice maximal HM (High Slot Mark)

Cette métrique est évoquée dans [13]. Elle est assez simple et consiste à prendre la valeur du slot d'indice maximal alloué. Le but de cette métrique est de rassembler les demandes sur un côté du spectre pour laisser complètement libre l'autre côté du spectre. Ceci permet de créer un gros bloc consécutif sur la fin du spectre (avantageux pour le problème de contiguïté) et

aussi de laisser le même côté du spectre libre dans tous les liens (bon pour le problème de continuité). En considérant le lien A-B de la figure 3.1, la valeur de la métrique est 9.

Cette métrique n'est cependant pas bonne dans certaines situations. Comme le montre la figure 3.3, on voit que la valeur de la métrique HM est la même sur les 2 situations (5 liens 1 et 2) mais le réseau semble plus fragmenté dans la situation 1 à cause du trou sur le slot 3 du lien A-B et du non-alignement du slot libre 3 sur A-B et slots 1-2 sur B-C. En revanche, dans la situation où les demandes forment un seul bloc et on a un alignement sur le slot 1 des liens A-B et B-C [14].

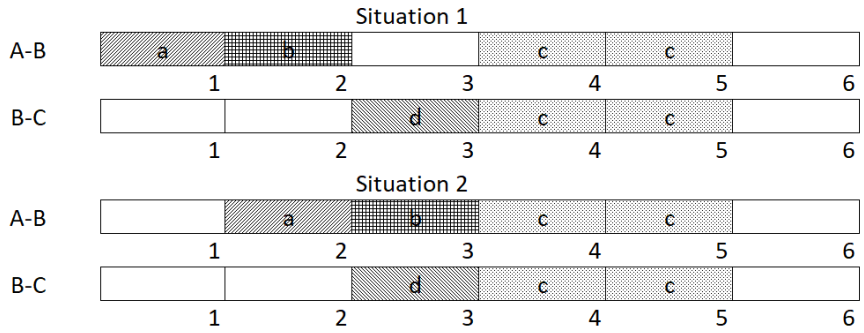

FIGURE 3.3 Problème de la métrique HM

Cette métrique est assez simple, elle n'est pas si efficace seule, mais elle peut très bien être utilisée comme facteur dans d'autres métriques.

3.1.5 La métrique Root Mean Square Factor (RMSF)

La métrique RMSF est présentée dans [15]. C'est une métrique qui ressemble à la métrique SE, de par la présence de rapport entre la taille des fragments et la taille de tous les slots libres. Cependant, elle est plus élaborée en prenant en compte l'index du dernier slot alloué (métrique HM). On peut définir la métrique RMSF avec la formule suivante :

$$F_{RMSF} = \frac{s_{max} * B}{\sqrt{\frac{\sum_i f_i^2}{B}}} \tag{3.3}$$

avec B et f_i tels que définis précédemment (3.1.3) et s_{max} l'indice maximal d'un slot alloué sur le lien

On reprend une nouvelle fois l'exemple de la figure 3.1. On a B = 7 et les f_i qui prennent les valeurs 1, 1, 2, 3.s_{max} vaut 9. D'où :

$F_{RMSF} = \frac{9*7}{\sqrt{\frac{1+1+2^2+3^2}{7}}} = \frac{63}{\sqrt{\frac{15}{7}}} \approx 43.04$

Cette métrique prend des valeurs entre 0 (si le lien est vide, $s_{max} = 0$) et $+\infty$. Elle donne de plus faibles valeurs lorsque les ressources sont allouées en bas du spectre, s'il y a un faible nombre de segments libres et si il y a un gros segment principal avec une taille proche de B. L'avantage de cette métrique est qu'elle prend en compte la taille de tous les fragments libres, le nombre de fragments libre et l'index maximal d'un slot alloué. Cependant, elle ne prend pas en compte la granularité des demandes tout comme les métriques précédentes [16].

3.1.6 La métrique Access blocking probability (ABP)

La métrique ABP est présentée dans [8] et est une métrique qui contrairement aux métriques RMSF ou SE prend en compte les granularités possibles. Le nombre de slots d'une demande dépend de le taux de transmission. Cependant, les différents taux de transmissions possibles peuvent être connues à l'avance et le nombre de slots des demandes peut en être déduit. Dans certains réseaux, il est possible de savoir que la majorité des demandes ne va prendre qu'un faible nombre de granularités possibles. Il peut être intéressant de considérer ces granularités. En effet, avec des granularités possibles de 4 et 5, avoir un fragment de 5,6 ou 7 ne change rien au nombre de connexions que l'on va pouvoir mettre sur ce fragment alors que 8 cela va changer (on peut insérer 2 demandes de 4 slots). Ces nuances ne sont pas visibles avec des métriques de fragmentations telles que la RMSF ou la SE.

On définit B et f_i tel que précédemment, G représente l'ensemble des granularités possibles dans le réseau et DIV(a,b) est le quotient de la division euclidienne de a par b. La métrique ABP s'écrit :

$$F_{abp} = 1 - \frac{\sum_i \sum_k DIV(f_i, G_k)}{\sum_k DIV(B, G_k)} \tag{3.4}$$

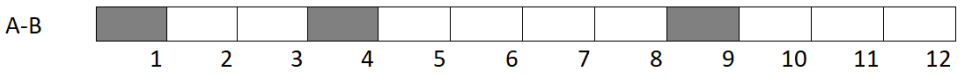

FIGURE 3.4 Exemple de distribution sur un lien

On a cette fois B = 9, $f_1 = 2$, $f_2 = 4$, $f_3 = 3$. On considère que les granularités possibles sont 2 et 3. On a alors :

$F_{ABP} = 1 - \frac{(2:2+4:2+3:2)+(2:3+4:3+3:3)}{9:2+9:3} = 1 - \frac{(1+2+1)+(0+1+1)}{4+3} = \frac{1}{7}$

Cette métrique donne une valeur entre 0 et 1. Elle vaut 0 lorsque le réseau n'est pas fragmenté : les fragments sont globalement tous des multiples de chacune des granularités. Elle vaut 1 lorsque le réseau est fragmenté : les fragments sont tous plus petits que la plus petite des granularités possibles.

Le gros point fort est donc qu'il prend en compte les granularités, mais cela en fait une métrique avec une plus grosse complexité puisque l'on parcourt l'ensemble des tailles de fragments pour chaque granularité contrairement à un simple parcours des tailles de fragments pour la SE ou la RMSF. Son second point faible est que cette métrique ne prend aucunement en compte l'alignement des slots entre les liens.

3.1.7 La métrique Wasted Slots (WS)

Jusqu'à présent, nous avons défini peu de métriques prenant en compte la fragmentation horizontale. En effet, seule l'introduction d'un indice s_{max} pour la métrique RMSF prend légèrement en compte l'alignement des slots. Plusieurs types de métrique de fragmentation prennent en compte l'alignement des slots. Comme exemple, nous avons choisi celle présentée dans [10]. Son principe est de compter les slots non alignés entre un lien cible et ses voisins. Pour un lien, on va parcourir les slots du spectre entre ce lien et chacun de ses voisins. Si les liens ne sont pas du même état (occupé-occupé ou libre-libre), on ajoute 1 à la valeur de la métrique. Lien voisin signifie que ce lien partage un sommet avec le lien cible. Ceci peut se formaliser pour un lien l sous la forme :

$$F_{WS}(l) = \sum_{l' \in N(l)} \sum_{s \in S} \|s_l - s_{l'}\| \tag{3.5}$$

avec $N(l)$: les voisins de l, s est un indice de slot et donc s_l est la valeur du slots de cet indice sur le lien l.

Un exemple montre comment calculer cette métrique dans la figure 3.5. On voit 2 liens avec des occupations différentes. Un slot blanc est libre, un slot grisé est occupé, un slot quadrillé est libre sur un lien, mais occupé sur l'autre. La métrique WS revient à trouver les slots quadrillés et les compter. Dans notre exemple, la valeur de la métrique est 15.

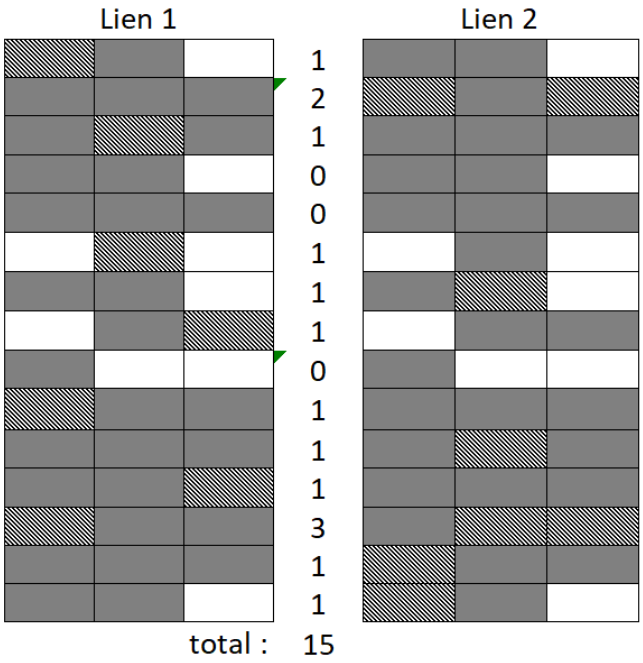

FIGURE 3.5 Exemple pour la métrique WS

Cette métrique est très utile, car elle considère l'alignement des slots. Elle peut cependant être associée avec une métrique s'attaquant plus au problème de fragmentation verticale afin d'être plus complète.

3.1.8 Combiner les métriques

On a présenté un certain nombre de métriques ayant des avantages et inconvénients. Il est aussi possible d'utiliser les métriques existantes et de les combiner pour gommer certains soucis et essayer d'en faire une métrique plus complète. Par exemple, on a vu que le RMSF prenait peu en compte l'alignement sur les liens des slots mais est une métrique assez efficace pour caractériser la disposition des slots sur le lien (la contiguïté des slots). En revanche, la métrique WS est bonne pour l'alignement des slots entre voisins, mais ne prend pas du tout en compte la disposition des slots sur le lien. Il semble efficace d'utiliser ces 2 métriques complémentaires. Lechowicz *et al* [10] ont mis en place une métrique WS-RMSF en la définissant comme produit des deux métriques :

$$F_{WS-RMSF}(l) = F_{WS}.F_{RMSF} \tag{3.6}$$

Il existe de nombreuses métriques dans la littérature et de nombreuses métriques seront sûrement mises en place. Il peut être efficace d'en combiner pour en obtenir des nouvelles.

3.2 Les algorithmes d'assignation

Le problème d'assignation des connexions est central dans le problème du RSA. On cherche des algorithmes dont la politique d'assignation permettra d'obtenir des résultats satisfaisants en termes de blocage, mais aussi de limiter au maximum la fragmentation dans le réseau pour éviter un problème futur dû à cette fragmentation trop haute. On a vu que le terme fragmentation a déjà été utilisé pour le problème d'assignation de mémoire dans un ordinateur. Certaines métriques telles que la métrique EF ont été adaptées depuis ce problème. Il en est de même pour les algorithmes d'assignation.

De manière générale, on fixe un réseau avec des liens, nœuds et des slots. Une partie de ces slots sont potentiellement occupés par des connexions existantes. Des demandes arrivent dans le réseau et nous souhaitons les placer. L'algorithme d'assignation est un algorithme définissant si la demande entrante peut être placé dans le réseau, et si oui, l'algorithme trouve l'emplacement de cette demande dans le réseau.

On va distinguer 2 types de politique d'assignation : celle se basant sur une métrique et celle ne se basant sur aucune métrique.

3.2.1 Politique ne se basant sur aucune métrique

L'algorithme first fit

Le first-fit (FF) est un algorithme d'assignation très simple utilisé dans le problème d'allocation de mémoire [9]. Il a ensuite été adapté pour les réseaux optiques [17]. Le principe du first fit est de parcourir l'ensemble des chemins possibles, du plus court au plus long pour la demande, des slots de plus bas indices vers les slots de plus hauts indices et d'assigner la demande au premier endroit pouvant contenir la demande en respectant les contraintes de notre problème. C'est un algorithme assez simple et qui se résume avec l'algorithme 1.

Algorithm 1 First fit

Require: Graphe G(V,L) avec V les noeuds, L les liens,d connexion arrivante, P l'ensemble des chemins possibles de d

```
1: Found = True
2: n_c = demande en slot contigu de d
3: for p ∈ P do
4:     n_i = 0 ; n_f = n_c
5:     while n_f < indmaxslot (p) do
6:         libre = 0
7:         for l ∈ p do
8:             if S_l(n_i), ..., S_l(n_f) = 0 then
9:                 libre += 1
10:            end if
11:        end for
12:        if libre = len(p) then
13:            Assigner d aux slots entre n_i et n_f
14:            Found = True
15:            Break While
16:        end if
17:        n_i += 1 ; n_c += 1
18:    end while
19: end for
20: Retourner Found, n_i, n_f
```

La variable *libre* permet de compter le nombre de liens sur lesquels ces slots sont libres. Dans la ligne 12, on s'assure que les slots sont bien libres sur chacun des liens.

Prenons un exemple pour simuler l'assignation d'une demande via le FF. On considère le réseau circulaire de 3 liens tel que sur la figure 3.6. On considère la distribution des demandes de la figure 3.7. Ceci représente l'état de notre réseau à cet instant.

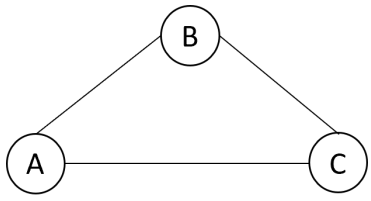

FIGURE 3.6 Exemple d'un réseau 3 liens

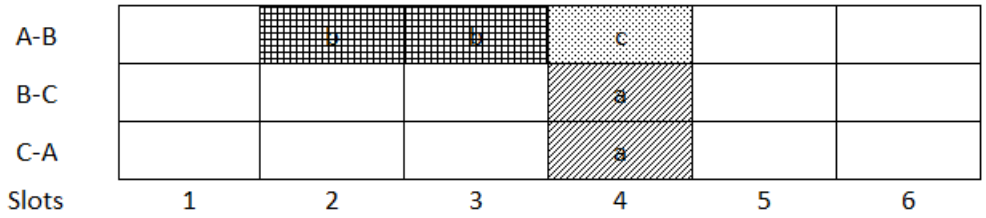

FIGURE 3.7 Situation initiale

On souhaite insérer une demande de 2 slots entre les nœuds B et A. Le chemin B-A est impossible (pour l'exemple), on doit utiliser le chemin B-C-A. On parcourt les slots du spectre de 1 à 6. On voit que les slots 1 et 2 sont libres, on va placer la demande sur les slots 1 et 2. Ceci donne le spectre de la figure 3.8 qui représente l'état de notre réseau après allocation.

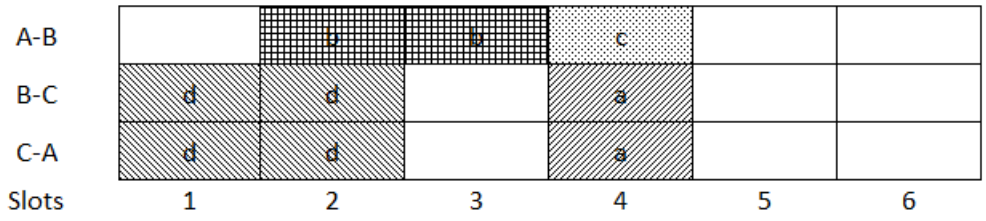

FIGURE 3.8 Etat du réseau après allocation d'une demande de taille 2 entre B et A en utilisant un algorithme first fit

Le first fit est un algorithme très simple, mais qui marche relativement bien au vu de sa simplicité. Il va avoir tendance à placer les demandes sur le plus court chemin, ce qui est souhaitable, et dans le début du spectre. Il va laisser l'autre partie du spectre relativement libre comme on peut le voir sur la figure 3.9.

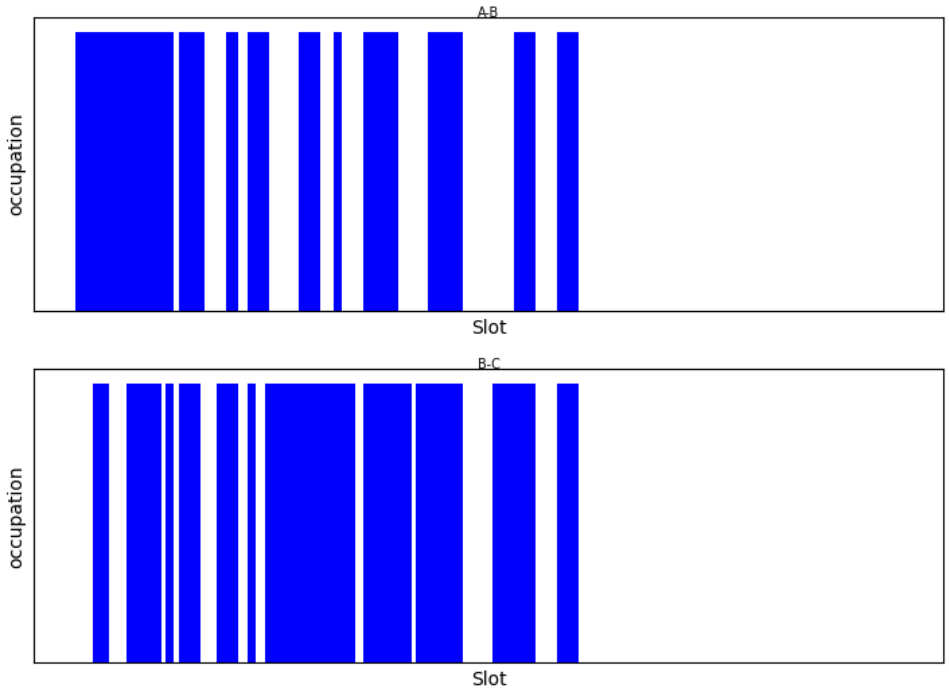

FIGURE 3.9 Etat des liens après une simulation en utilisant l'algorithme first fit

L'algorithme last fit

Du first fit découlent un certain nombre d'algorithmes. Le last fit [7] utilise le même parcourt et la même politique d'assignation que le first fit. La différence se trouve sur le parcourt des slots du chemin. On parcourt les slots dans le sens décroissant (des indices hauts vers les indices bas). En prenant le même exemple que pour le first fit, on assigne la demande d sur les slots 5-6 des liens B-C et C-A.

L'algorithme exact fit

L'algorithme exact fit [17] utilise le même parcours du spectre que le first fit. La différence se trouve dans la solution trouvée. L'algorithme cherche un fragment faisant exactement la taille de la demande. Pour une demande de 2 slots, l'algorithme cherche un slot s_1 tel que s_1 et $s_1 + 1$ soit libre sur le chemin et $s_1 - 1$ et $s_1 + 2$ soient occupés sur au moins 1 des liens du chemin. Si un tel fragment n'est pas trouvé, l'algorithme essaie d'assigner la demande via un first fit classique. On peut résumer la méthode d'assignation avec l'algorithme suivant.

Algorithm 2 Algorithme first-exact fit

Require: Graphe G(V,L) avec V les noeuds, L les liens,d connexion arrivante, P l'ensemble des chemins possibles de d

```
    Found = True
 2: n_c = demande en slot contigu de d
    for p ∈ P do
 4:     n_i = 0 ; n_f = n_c
        while n_f < indmaxslot (p) do
 6:         libre = 0
            side = 0
 8:         for l ∈ p do
                if S_l(n_i), ..., S_l(n_f) = 0 then
10:                 libre += 1
                end if
12:             if S_l(n_i − 1) = 1 et S_l(n_f + 1) = 1 then
                    side += 1
14:             end if
            end for
16:         if libre = len(p) et side>0 then
                Assigner d aux slots entre n_i et n_f
18:             Found = True
                Break While
20:         end if
            n_i += 1 ; n_c += 1
22:     end while
    end for
24: if not Found then
        Essayer une assignation first fit
26: end if
    Retourner Found, n_i, n_f
```

On utilise la même variable libre que pour le first fit et on ajoute la variable side qui regarde l'état des slots adjacents. On souhaite qu'il y ait au moins un lien avec exactement la bonne taille, d'où la condition sur size ligne 16. Pour gérer les bordures, on considère que le slot 0 et n+1 sont toujours occupés. À noter que l'on peut garder en mémoire la première solution trouvée d'un point de vue first fit pour ne pas reparcourir le spectre à la fin.

On récupère le même exemple que pour l'algorithme first fit avec la figure 3.7. On souhaite encore assigner une demande de 2 slots sur les liens B-C-A. On commence par essayer les slots 1-2. On voit qu'ils sont libres, que le slot 1 est une bordure, mais que le slot 3 est libre, donc la solution n'est pas bonne pour notre algorithme. On essaie les slots 2-3, ils sont bien libres, mais le slot 4 n'est pas libre. Les slots 3-4, 4-5 ne sont pas libres. On essaye les slots

5-6, ils sont bien libres, le slot 4 est occupé, le slot 6 est une bordure donc le slot adjacent est occupé. La solution est satisfaisante : on insère notre demande sur les slots 5-6.

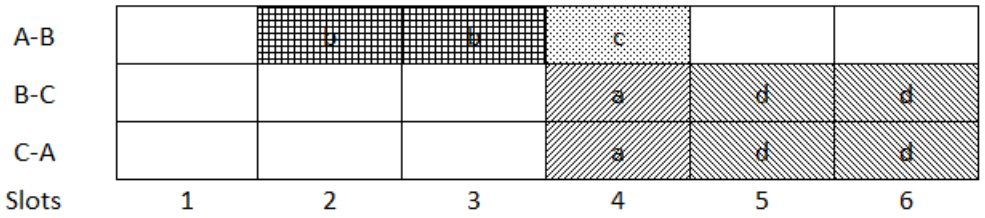

FIGURE 3.10 Etat du réseau après allocation d'une demande de taille 2 entre B et A en utilisant un algorithme exact fit

L'algorithme first-last fit

On peut utiliser les algorithmes précédents pour construire de nouveaux algorithmes. Le first-last fit algorithme est un algorithme utilisant à la fois une politique d'assignation first-fit pour une partie des demandes et une politique d'assignation last-fit pour l'autre partie. On partitionne les demandes selon un critère. Le choix de ce critère est particulièrement important. S'il est mal choisi, les performances peuvent être dégradées [18].

3.2.2 Politique se basant sur des métriques

On a présenté des algorithmes se basant sur une manière ordonnée d'assigner les connections, mais sans prendre en compte réellement le problème de fragmentation dans le sens où l'impact sur la fragmentation de l'assignation d'une demande n'est pas pris en compte. Il existe un certain nombre d'algorithmes prenant en compte des métriques

L'algorithme de Fragmentation Aware (FA)

L'algorithme FA est un algorithme très populaire dans la littérature. Il s'agit plus d'un type d'algorithme applicable pour diverses situations. Dans un algorithme FA, on va définir une fonction ou métrique m. On va chercher l'ensemble des candidats possibles pouvant accueillir la demande que l'on cherche à assigner. Pour chacun des candidats, on va calculer le résultat de m en considérant l'assignation de la demande sur ce candidat. Le candidat choisi est le candidat avec la plus petite valeur de fonction. Le but est de minimiser la probabilité de blocage en minimisant une métrique ou fonction. Ce type d'algorithme est notamment utilisé dans [16, 19, 20].

Dans [16], on décrit la comparaison entre plusieurs algorithmes de fragmentation aware avec des métriques différentes. On observe notamment que les métriques ABP et RMSF sont les plus performants pour un algorithme FA avec les métriques présentées dans l'article.

Dans [20], on utilise une approche combinant un algorithme d'exact fit et de fragmentation aware. On assigne en priorité selon la polique exact fit et si on ne trouve pas de fragment qui fait exactement la taille de notre demande, on choisit le fragment minimisant la métrique EF. Cette approche produit de meilleurs résultats qu'un simple first fit.

Enfin dans [19], on utilise une métrique d'assignation s'attaquant à la continuité des slots présents. Cette métrique est proche de la métrique WS et les résultats sont de nouveau comparés au first fit.

On a présenté un certain nombre de métriques dans la section 3.1. L'utilisation de ces métriques peut notamment se faire à travers un algorithme FA. On remarque que les métriques ne donnent pas toutes les mêmes résultats et que certaines métriques sont plus efficaces que d'autres pour minimiser le blocage via ce type d'algorithme [16].

Algorithm 3 Algorithme fragmentation aware [10]

Require: m métrique pour le calcul de la fragmentation, d demande, B les slots disponibles par liens, ϕ Objet calculant la fragmentation du réseau, G(V,L) état du réseau (graphe, nœuds, liens), P l'ensemble des chemins possibles pour d

```
initialiser φ avec le reseau actuel G(V,L)
F_r = φ.frag(m)
c_n = ∅
F_b = ∞
for p ∈ P do
    Slots candidats dans C
    if C ≠ ∅ then
        for c ∈ C do
            F_n = φ.fragafteralloc(m)
            if F_n < F_b then
                F_b = F_n
                c_n = c
            end if
        end for
    end if
end for
if c_n ≠ ∅ then
    allouer c_n à d
    F_r = F_b
else
    d est bloqué
end if
```

Ce type d'algorithme requiert un temps de calcul plus long que ceux ne nécessitant pas de calcul d'une métrique, justement à cause de ce calcul. On doit chercher tous les candidats, ce qui n'est pas forcément le cas du first-fit vu que l'on s'arrête au premier trouvé. Par la suite, on doit calculer la valeur de la fonction sur le chemin pour chaque candidat. Ceci rajoute du temps de calcul. C'est l'algorithme le plus commun quand on considère des métriques. De plus, cet algorithme s'applique bien au problème du RSA online : une demande arrive et on la place au meilleur endroit pour elle du point de vue d'une certaine métrique.

Des heuristiques plus élaborées

Pour résoudre le problème de la fragmentation, des algorithmes suivant des heuristiques plus complexes ont aussi été développés. Dans [21–23], on a la mise en place d'un algorithme Tabu pour traiter le problème RSA. Dans [23], on a une liste de demande à insérer dans le réseau et l'objectif est de maximiser le nombre de connexions de cette liste que l'on va insérer dans le

réseau. Dans [21,22], les fonctions objectif de leur algorithme tabu sont assez proches. Le but est de minimiser le nombre de fréquences utilisées dans le le réseau. Ceci, revient à minimiser le nombre de slots occupés dans l'ensemble du réseau.

Cependant, ces méthodes requièrent un grand nombre de demandes à placer simultanément et un temps de calcul conséquent. Ils sont plus difficiles à mettre en place dans le cas du problème RSA online où les demandes arrivent une à une, de manière aléatoire et il faut les placer le plus rapidement et efficacement possible. Pour ces raisons, nous n'avons pas considéré ce genre d'algorithmes. Il pourrait cependant être utile pour un problème proche au RSA, celui de la défragmentation. La défragmentation est l'action de réarranger les connexions existantes pour minimiser la fragmentation du réseau. Bien que, nous évoquerons sûrement ce problème vers la fin de ce mémoire, mais ce n'est pas le problème sur lequel nous nous sommes penchés.

3.3 Discussion autour des politiques d'assignation

Par la diversité des chemins possibles et des slots disponibles, il est très probable qu'il existe plusieurs solutions pour assigner une même demande. Il vient alors la question sur le choix de cette solution. Par exemple, pour le first fit, on choisit la première qui vienne. Pour le fragmentation aware, on choisit la meilleure place du point de vue d'une métrique. Plusieurs questions peuvent se poser. Y a-t-il des métriques plus efficaces que d'autres ? Cette meilleure place ne peut-elle pas être plus bénéfique à une demande arrivant juste après ?

Pour cela, nous avons mis en place un test pour simuler les résultats en termes de blocage. Ces résultats sont obtenus à l'aide d'un outil informatique dont nous présenterons les détails dans le chapitre suivant. Nous allons comparer des algorithmes de fragmentation aware pour différentes métriques ainsi que des algorithmes choisissant aléatoirement une solution parmi celles possibles. Ces solutions ont des probabilités qui sont fonction des métriques. On va faire varier le degré d'aléa via un facteur k. Plus précisément k est un facteur utilisé dans le calcul des probabilités. Plus k est grand, plus il est probable de choisir la meilleure solution du point de vue de la métrique choisie.

On résume les résultats de ces tests dans la figure 3.11. On note les algorithmes de fragmentation aware comme FA-X où X est la métrique et les algorithmes aléatoires comme RANDOM-X. On choisit de comparer les métriques RMSF, ABP et SE. Tout d'abord, on voit bien que les algorithmes de FA ne dépendent pas de k. En effet, il n'y a pas d'aléa dans ces algorithmes. Ensuite, les performances sont dépendantes de la métrique utilisée. En comparant les courbes FA, on voit que la métrique SE est bien moins bonne dans un algorithme

FA que les métriques RMSF et ABP. Par ailleurs, la métrique ABP est légèrement meilleure que la métrique RMSF.

En comparant les courbes dans le cas random et dans le cas FA, on observe que les performances du cas FA sont toujours meilleures que le cas random. Les performances de la courbe random tendent à rejoindre les performances du cas FA lorsqu'on augmente k, mais sans les battre. Comme augmenter k équivaut à augmenter la probabilité de choisir la meilleure solution, on peut en conclure que l'introduction d'aléas dans la politique d'assignation n'est pas bénéfique pour les performances.

Cependant, il existe des algorithmes efficaces comme le tabu search introduisant une part d'aléas. Ces algorithmes sont utilisés dans le cas offline ou pour la défragmentation. Dans ces 2 cas, le nombre de demandes que l'on traite en simultané est plus grand, ce qui réduit la part de l'aléatoire. Ce type d'algorithme n'est peut-être simplement pas efficace pour le cas du RSA online avec des arrivées et départs de demandes.

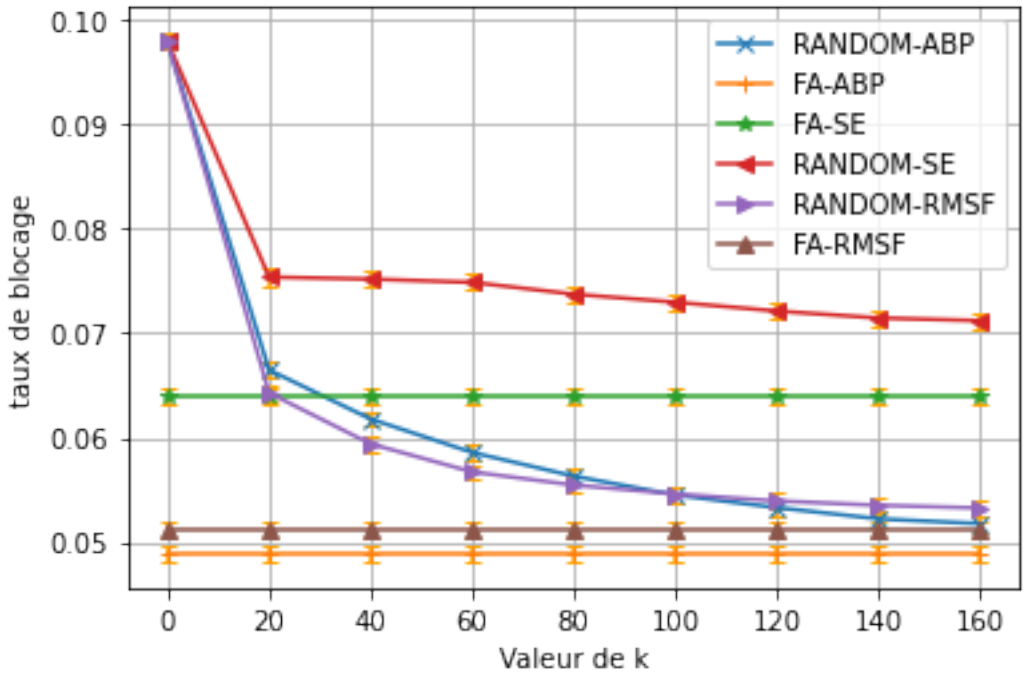

FIGURE 3.11 Comparaison des taux de blocage pour diverses méthodes

En conclusion, via ce test, on a appris que toutes les métriques n'étaient pas égales. Certaines obtiennent de meilleurs résultats que d'autres. On a aussi essayé d'introduire un facteur aléatoire dans l'assignation des demandes. On se demandait si la diversification qu'une telle politique pouvait procurer amènerait des bons résultats. Il faut alors que notre politique

d'assignation soit bien structurée. C'est via de telles observations que nous avons eues l'idée d'ordonner le spectre. On introduira cette idée durant le chapitre 5.

CHAPITRE 4 MODELISATION INFORMATIQUE ET SIMULATION

Dans le cadre global du projet, un simulateur informatique a été mis en place pour pouvoir simuler le fonctionnement d'un réseau optique. C'est un simulateur qui a été mis en place par plusieurs membres de l'équipe de recherche. J'ai contribué à la mise en place de ce simulateur, notamment sur les parties reliées à l'assignation des demandes. Nous allons maintenant passer en revue brièvement les différentes caractéristiques de ce simulateur.

4.1 Les classes

L'ensemble des éléments présentés pour la conception des réseaux optiques est traduit dans le simulateur de manière informatique. Les nœuds, les liens, les demandes et les slots sont réprésentés par des classes python permettant de définir informatiquement notre réseau. D'autres classes existent aussi pour gérer l'allocation et le retrait des demandes. L'ensemble de ces outils permettent de simuler au mieux un réseau optique. On va présenter quelques points du simulateur qui sont utiles pour aborder le problème d'assignation via ce simulateur informatique

4.2 Quelques méthodes

Le simulateur possède un certains nombres de méthodes nécessaires au fonctionnement de celui-ci. Certaines peuvent être pour assigner ou réassigner des attributs aux classes. Par exemple, la classe demande possède des méthodes pour déclarer le chemin assigné ou les slots qu'elle occupe une fois assignée. D'autres méthodes permettent le calcul des métriques des différents liens et une méthode plus générale appartenant à la classe réseau permet de calculer la fragmentation du réseau ou sur un chemin du réseau. L'assignation des demandes se fait via un dossier allocation contenant un ensemble de fonction pour allouer les demandes. Ces fonctions ont été présentées dans la section 3.2. Le travail autour de notre problème vise principalement à effectuer des tests autour d'algorithmes différents et de comparer les résultats de ces tests.

4.3 Initialisation et tests

Pour ces tests, on a des méthodes et des fichiers d'initialisations pour définir un réseau spécifique. Dans ce fichier d'initialisation, il est possible de choisir les paramètres pour notre

simulation future. Le listing ci-dessous montre un bout d'un fichier d'initialisation. On y voit notamment un certain nombre de paramètres permettant de gérer la durée de simulation ou bien le nombre de slots sur chacun des liens. On remarque aussi qu'il est possible de définir un type de test (ligne 5). Les différents tests possibles font varier divers paramètres et sont regroupés dans un dossier. Certains tests seront développés dans les chapitres suivants, car ils peuvent avoir été designés pour étudier un paramètre spécial pour une méthode spécifique.

Les résultats sont stockés dans un fichier .csv. Classiquement, les résultats sont le bloquage moyen, le bloquage par type de connexion ou bien une métrique de fragmentation. Il est aussi possible d'afficher les graphiques directements après la simulation comme l'occupation des liens (figure 2.9). Le stockage des résultats permet aussi de les réutiliser a posteriori pour les retravailler et éventuellement les croiser avec le résultat d'autres simulations.

4.4 Simulation

La simulation se fait pour un test spécifique. On définit au cours des paramètres une durée de simulation et un nombre de répétitions. En effet, à cause de l'aléa autour de la génération et l'arrivée de demande, il est nécessaire d'être capable d'effectuer une simulation un certain nombre de fois ainsi que de fixer l'aléa au sein de la simulation (en déclarant une amorce pour cet aléa). Ceci permet d'avoir des résultats reproductibles ainsi que d'avoir la même distribution de demandes pour nos algorithmes dans deux simulations différentes.

La simulation se fait temps par temps. À chaque instant de temps, des évènements ont lieu. Ceci peuvent être des connexions entrantes ou expirantes à gérer, des mesures de la fragmentation... À chaque instant de temps, le simulateur met à jour les différentes variables pour chacun des évènements ayant lieu. Par exemple, si une demande expire, il faut changer le statut des slots et des liens qui étaient utilisés par cette demande. Une fois l'ensemble des évènements traités, le simulateur met à jour les variables utiles pour générer les résultats. Par exemple, il met à jour son compteur de demandes bloquées ou l'état de fragmentation général du réseau. À la fin d'une simulation, ces variables sont ajoutées à un fichier csv pour pouvoir les réutiliser ultérieurement comme discuté dans la partie précedente.

Dans une simulation, il est aussi possible de préremplir le réseau. Si nous ne souhaitons commencer la simulation avec un réseau entièrement vide, il est possible de définir dans notre fichier d'initialisation les paramètres pour préremplir le réseau. Ce pré-remplissage fonctionne comme une simulation classique, seulement les résultats de pré-remplissage ne sont pas ajoutés au csv de résultats. La simulation prend le réseau dans l'état à la fin de la

```
{"NAME OF FILE" :                    "ciena_network",
 "Parameters":
    {
        "type of test":                      "Slice size test",
        "plot only":                         true,
        "number of seeds":                   1,
        "seed to generate seeds":            2500,
        "confidence interval":               0.95,

        "type of granularities": [8,12,19],
        "number of k_shortest_paths":        5,
        "metric for k_shortest_paths":       "n_EDFA",
        "number of slots per link":          768,
        "size of slots" : 6.25,
        "demand option":                     "custom",
        "custom demands path":               "simulateur/init/
          demands_excel_ciena.xlsx",
        "bordering SuperChannels for fragmentation aware": true
          ,
        "reuse demands if available": true ,
        "network name" :                     "
          ciena_network_small",
        "network topology path":               "simulateur/init/
          network_topology/ciena_network.csv",
        "result plot":                      [4],
        "plot yscale":                      "linear",
        "result folder explicit name":      "ff_parti_full_net",
        "result iteration folder":          "
          test_slice_size_full_net",
        "default node type": "CD",

        "default slot model":
        {
            "bandwidth":    6.25,
            "modulation":   "QPSK",
            "capacity":     50},
        "virtual slot model":
        {
            "bandwidth":    6.25,
            "modulation":   "QPSK",
            "capacity":     "inf"},
```

FIGURE 4.1 Exemple d'une partie d'un fichier d'initialisation

simulation. On peut notamment voir dans le listing que ces paramètres sont définis dans les sections "filling" et "simulation".

4.5 Génération des demandes

Le simulateur est également doté de méthodes pour générer les demandes. Il y a deux manières de générer nos demandes. Il est possible de les générer très spécifiquement à partir d'un fichier, ce qui permet de controler de manière plus précise les demandes dans le réseau. Il est aussi possible de les générer de manière générale dans le réseau à partir d'un ensemble d'entrée défini dans le fichier d'initialisation. Dans les deux cas, il est possible de controler le nombre global de demandes créées via le flux de demande ρ. Le flux moyen de demande est représenté par ρ et représente la durée moyenne d'une demande divisée par le temps moyen entre 2 générations. C'est notamment via ce paramètre que l'on va contrôler le niveau de congestion du réseau.

4.6 Utilisation du simulateur

Notre travail se porte sur l'implémentation de méthodes pour gérer l'allocation des demandes. Par la suite, on va utiliser ce simulateur pour étudier chacune de ces méthodes. On va créer des tests permettant de faire varier certains paramètres du simulateur et ensuite récupérer les résultats de cette simulation pour les analyser.

CHAPITRE 5 LE SLICING DANS DES CAS SIMPLES

Dans ce chapitre, nous allons introduire une nouvelle politique d'allocation pour le problème du RSA. On va commencer par introduire l'idée et le concept de cet algorithme. Ce concept est nouveau et n'a pas été étudié dans la littérature. On va ensuite présenter des résultats pour un lien et un chemin.

5.1 Concept du slicing

Le problème de fragmentation vient des arrivées et départs de connexions de différentes tailles laissant alors des trous pouvant ne pas être comblés par une autre demande. Prenons l'exemple de la figure 5.1. On considère un chemin A-B-C-D, une demande de 3 slots occupe le lien A-B, une demande de 2 slots occupe le lien B-C et une demande de 4 slots occupe le lien C-D. On souhaite introduire une demande de 2 slots de A vers D passant par ces 3 liens. On va naturellement l'introduire sur les slots 5 et 6 (slots quadrillés). Ceci va laisser les slots 3-4 du chemin B-C et 4 du chemin A-B difficilement accessibles pour d'autres connexions.

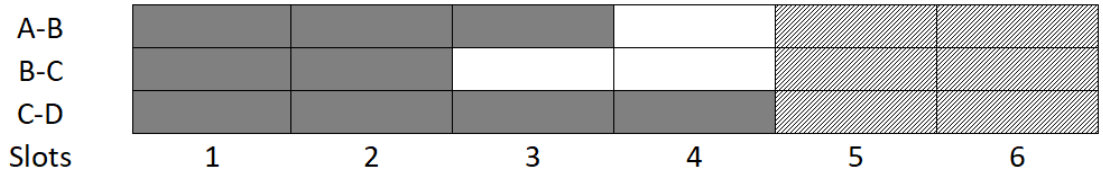

FIGURE 5.1 Occupation sur un chemin A-B-C-D

Comme on en a discuté dans la section 3.1.6, les taux de transmissions possibles à l'émetteur sont standardisées et les largeurs de bandes possibles sont restreintes. Classiquement, pour un réseau donné par CIENA, les largeurs possibles sont 50GHz, 75GHz et 118.75GHz. Avec une grille de 6.25GHz, on a des demandes possibles de 8, 12 et 19 slots. Lors de l'introduction de nouveaux modems dans le réseau (remplacement des anciens), ces largeurs peuvent changer. Lors de la transition, on va se retrouver avec un assez grand nombre de granularités possibles. À terme, les nouveaux modems auront remplacé les anciens et on retrouvera un petit nombre de granularités possibles.

L'idée du slicing est de rassembler les connexions par type de granularité. On va partitionner le spectre en assignant à chacune des granularités une tranche de spectre (slice). À l'intérieur de cette tranche, toutes les connexions ont la même granularité et donc une connexion expirante

laisse un trou faisant sa taille et ce trou pourra être utilisé par une autre connexion de même taille. On s'affranchit en quelque sorte de la contrainte de contigüité puisque tous les fragments ont pour taille un multiple de la granularité originelle. La seule raison pour laquelle une connexion ne trouverait pas de place dans son slice alors qu'il y a assez de place libre serait à cause de la contrainte de continuité.

Cependant, cette idée pose un certain nombre de problèmes à régler. Tout d'abord, il faut bien dimensionner la taille de chacunes des slices. Vu que les slices ne peuvent accueillir qu'un seul type de granularité, une demande n'a pas accès à tout le spectre disponible, mais seulement à une partie du spectre. Surdimensionner un slice par rapport au nombre de demandes de ce type réduit la place pour les autres slices qui peuvent se retrouver sous-dimensionnés.

Bien dimensionner les slices sur le long terme peut ne pas être suffisant pour gérer les pics ponctuels de fortes arrivées d'une demande. En plus des slices pour chacune des granularités possibles, on va introduire un slice pour gérer le surplus ponctuel d'un type de connexions et les éventuelles connexions avec une granularité différente de celles que l'on considère dans nos slices. On appelle un tel slice le slice commun. Il faut bien choisir la taille du slice commun. Plus le slice commun est grand, plus l'autre partie du spectre contenant des slices réservés pour chacune des granularités est petite et donc plus on perd l'intérêt de partitionner notre spectre. Il faut donc choisir de manière minutieuse la taille de la portion du spectre non partitionnée et partitionnée

Finalement, on peut résumer le concept de slicing avec la liste suivante :

- Dimensionner la taille du spectre partionné et du slice commun
- Pour le spectre partionné, dimensionner pour chacune des granularités
- Assigner les demandes en priorité sur le slice correspondant à sa granularité
- Si cela est impossible, assigner sur le slice commun

Le trafic peut ne pas être égal en tout point du réseau. Il faut alors réfléchir comment gérer ces inégalités pour gérer la taille de nos slices.

On va commencer par présenter l'algorithme exact pour l'assignation d'une demande.

5.2 Algorithme slice first fit

On a défini au préalable les restrictions sur chacun de nos liens. Via le simulateur présenté dans le chapitre 4, les restrictions sont un attribut de notre classe lien. Cet attribut est un dictionnaire dont les clés sont les granularités possibles et les valeurs sont un tuple contenant le slot du début et le slot de fin du slice. Dans chacun des slices ainsi que dans le slice commun, on effectue une assignation first fit des connexions. Il est éventuellement possible

de choisir une autre politique d'assignation, mais ce n'est pas notre but pour l'instant. On peut résumer notre concept avec le pseudo algorithme 4 suivant :

Algorithm 4 Algorithme slicing first fit

```
Require: d demande, G(V,E) état du réseau (graphe, nœuds, liens), P l'ensemble des chemins possibles pour d
    trouve = Faux
    for p ∈ P do
3:      restriction-min = 0, restriction-max = 768
        for l ∈ p do
            link-min,link-max = l.restriction[d.granularity]
6:          restriction-min = max(restriction-min, link-min)
            restriction-max = min(restriction-max,link-max)
        end for
9:      F = get-fragment(p, (restriction-min,restriction-max))
        for f_i ∈ F do
            if f_i suffisament grand then
12:             trouve = Vrai
                Allouer d entre f_i(0) et f_i(0) + d.granularity
            end if
15:     end for
    end for
    if trouvé = faux then
18:     Utiliser une politique FF sur le slice commun
    end if
    if trouvé = faux then
21:     d est bloqué
    end if
```

Les lignes 4-8 permettent de définir la restriction sur le chemin en fonction des restrictions sur les liens. À noter que si on choisit de mettre la même restriction sur tous les liens, on peut simplement définir notre restriction via celle du premier lien. La fonction get-fragment de la ligne 9 permet d'obtenir l'ensemble des tranches de slots libres sur tous les liens. La boucle for entre les lignes 10-15 cherche un fragment de taille suffisante. Si elle n'en trouve pas, elle essaye d'en trouver un sur le slice commun (ligne 17-19). Enfin, si aucun fragment n'est trouvé, la demande est bloquée.

Prenons un exemple pour expliquer comment se fait l'assignation. On considère un simple lien A-B avec l'occupation suivante :

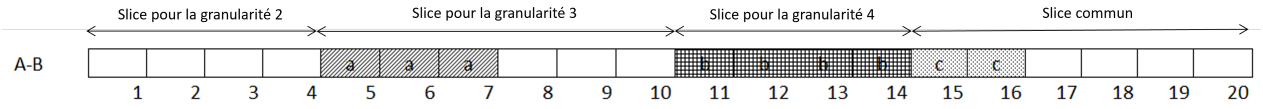

FIGURE 5.2 Exemple d'occupation sur un lien A-B

On a défini au préalable les restrictions pour des granularités de 2,3 et 4. Entre les slots 1 et 4, on a le slice pour les demandes de taille 2. Entre les slots 5 et 10, on a le slice pour les demandes de taille 3. Entre les slots 11 et 14, on a le slice pour les demandes de taille 4. Entre les slots 15 et 20, on a le slice commun. On va considérer 2 cas : un premier cas où on souhaite introduire une demande de 2 slots et un second cas où on souhaite introduire une demande de 4 slots.

Dans le cas 1, on commence par regarder le slice correspondant, soit les slots 1 à 4. On utilise un politique first fit sur ce slice. On commence alors par regarder les slots 1 et 2 qui sont libres. On assigne la demande sur les slots 1 et 2.

Dans le cas 2, on considère une demande de 4 slots. On regarde les slots 11 à 14. Ils sont utilisés et on ne peut pas assigner cette demande sur ce slice. On regarde maintenant le slice commun (slots 15 à 20). On voit que les slots 15 et 16 sont déjà utilisés. On voit que le slot 17 est libre et les slots 18-19-20. On peut finalement allouer cette demande sur les slots 17 à 20. La figure 5.3 résume ces deux cas.

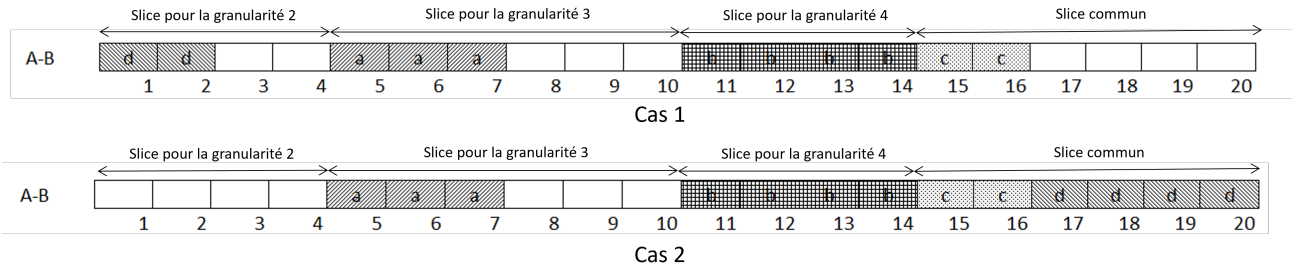

FIGURE 5.3 Exemple d'occupation sur un lien A-B après assignation

5.3 Détermination des restrictions

L'algorithme 4 montre comment nous effectuons l'assignation des connexions une fois les restrictions définies. Cependant, nous n'avons pas encore expliqué comment nous calculons ces restrictions.

Notons qu'il y a 2 principales restrictions à définir. Une première marquant la tranche du spectre qui va être partitionnée et celle qui ne va pas l'être (ce qui correspond au slice commun). On démarque cette limite à l'aide d'une variable slice_value. On considère que le spectre entre 0 et slice_value est partitionné : chaque granularité possède sa tranche de spectre où elle ne sera en compétition qu'avec des demandes de même taille. Entre slice_value et 767, nous avons le slice commun. On trouve cette valeur empiriquement à l'aide d'une étude que l'on développera dans les parties suivantes. La seconde restriction se situe au niveau du partitionnement pour chacune des granularités. À l'intérieur de cette tranche, il faut définir quelle granularité hérite de quelle tranche de spectre pour elle seule.

Pour définir les restrictions de ce slice partitionné, nous nous basons sur l'inverse de la formule d'Erlang. On a vu que dans le cas général d'un EON, elle n'est pas exacte, cependant avec le système de slicing, à l'intérieur d'un slice, les demandes sont toutes de la même taille. De plus, elle permet d'obtenir un ordre de grandeur pour assigner les slices. Cette formule trouve un blocage E en considérant un trafic ρ et un nombre de canaux N. Pour une demande de k slots, un canal est une tranche contigüe de k slots. On énonce alors la formule :

$$\frac{1}{E(n)} = \frac{\rho^n/n! + \sum_{j=0}^{n-1} \rho^j/j!}{\rho^n/n!} \tag{5.1}$$

Cette formule assez complexe peut aussi s'écrire sous une forme plus facile à manipuler avec une récurrence :

$$\frac{1}{E(n)} = 1 + \frac{n}{\rho}.\frac{1}{E(n-1)} \tag{5.2}$$

Avec $E(0) = 1$

Pour nos besoins, on utilise la deuxième forme qui est plus simple à calculer. On cherche dans notre cas N. Pour cela, on va définir un blocage maximal E_{max} considérant un trafic ρ pour obtenir un nombre de canaux N. Plus précisément, on cherche N tel que :

$$\frac{1}{E_n} \geq \frac{1}{E_{max}} \tag{5.3}$$

Ce trafic d'entrée peut être déterminé à l'avance en ayant connaissance du pourcentage moyen de chaque type de demande. On peut aussi utiliser une mémoire des dernières demandes, ou bien des algorithmes de prédiction de trafic. Dans nos calculs, on utilise comme taux de blocage pour la formule 1% car on ne souhaite pas avoir un blocage qui dépasse les 1 à 5%. Le nombre de blocs devient un nombre de slots en multipliant par la granularité.

Prenons un exemple. On a un trafic A de 0.33 et un blocage requis E_{max} de 1%. Soit on a $\frac{1}{E_{max}} = 100$. On va alors chercher N tel que :

$$\frac{1}{E_N} \geq 100$$

On va utiliser la formule de récurrence pour trouver ce N :

$$\begin{gathered}\frac{1}{E(0)} = 1 \\ \frac{1}{E(1)} = 1 + \frac{1}{0.33} \cdot \frac{1}{E(0)} = 1 + 3.1 = 4 \\ \frac{1}{E(2)} = 1 + \frac{2}{0.33} \cdot \frac{1}{E(1)} = 1 + 6.4 = 24 \\ \frac{1}{E(3)} = 1 + \frac{3}{0.33} \cdot \frac{1}{E(2)} = 1 + 9.24 = 217\end{gathered}$$

Pour ce cas, on va trouver $N = 3$. Si notre demande requiert 5 slots, on va alors dimensionner pour 15 slots.

On effectue le calcul de ce nombre pour chacun des slices. Ensuite, on souhaite utiliser l'ensemble de la place disponible, c'est-à-dire un nombre de slots proche de la taille du slice partitionné (défini par la variable slice_value). Par exemple, pour 3 de granularité, on requiert 12 slots, pour 4 de granularité, on requiert 16 slots et pour 5 de granularité, on requiert 15 slots. La somme de ces slots fait 43 slots. Si notre slice partitionné doit faire autour de 80 slots, il faut augmenter de manière proportionnelle chacune des tranches données pour nos granularités afin d'occuper la place donnée à notre slice partitionné. Au contraire, si notre slice partionné doit faire autour de 30 slots, il faut réduire les tranches de chacun. Lors de ce passage à l'échelle, il faut faire attention de garder un nombre de slots multiple de la granularité. Ces slots supplémentaires seraient inatteignables.

Nous pouvons réutiliser les valeurs de 12, 16 et 15 pour les granularités 3,4 et 5 avec une taille de 80 pour le slice partitionné. Le passage à l'échelle se ferait comme suivant : on va donner au slice pour les demandes de 3 slots le multiple de 3 proche de $\frac{12}{12+16+15} * 80$ soit 21 slots. Pour celles de 4 slots, on donne le multiple de 4 proche de $\frac{12}{12+16+15} * 80$, soit 28 slots. Enfin, pour celles de 5 slots, on donne le multiple de 5 proche de $\frac{15}{12+16+15} * 80$, soit 25 slots.

À noter que l'on n'atteint pas nécessairement la taille exacte fixée slice_value en cumulant le nombre de slots. Ceci n'est pas dérangeant, car ces slots ne sont pas perdus et appartiendront par la suite au slice commun.

5.4 Cas lien simple

5.4.1 Présentation du test pour optimiser la taille du slice commun

On va maintenant évaluer notre algorithme dans le cas d'un lien simple A-B. On considère des granularités de 8, 12 et 19 slots comme définit au début du chapitre. Dans un premier temps, on connait d'avance le pourcentage moyen de chacune des granularités possibles et aucune autre demande de taille différente des 3 granularités majoritaires. De plus, on considère que le trafic moyen reste le même dans le temps et donc qu'il n'y a pas besoin de changer pour chacune des granularités au cours de la simulation. De ce trafic moyen, on est capable d'avoir un ordre de grandeur du nombre de slots requis pour chacune des granularités en utilisant la formule d'Erlang comme discuté dans la section précédente.

Le premier but est de savoir quelle est la taille optimale de notre slice commun pour cela.

Pour cela, on va étudier l'évolution du blocage en fonction de la taille du slice commun. On va faire varier plus exactement une variable slice_value entre le début du spectre et la fin du spectre. On considère qu'entre 0 et slice_value, le réseau est partitionné avec les restrictions pour chacune des granularités et entre slice_value et 767, on a le slice commun à tous.

Quelques cas particuliers sont à noter. Dans le cas où slice_value est égal à 0, le slice commun représente tout le spectre. Comme on considère une assignation first fit sur le slice commun, notre algorithme dans ce cas particulier équivaut à un first fit classique sans slice. Par la suite, on note ce cas extrême comme NS (Non-Sliced). Cas extrême inverse, si la variable slice_value est égale à 767, le réseau est divisé dans son entièreté et le slice commun est nul. On se retrouve alors avec un réseau entièrement divisé. On note ce cas particulier comme FS (Fully Sliced).

Chaque simulation ayant sa part d'aléas dans la génération et l'arrivée de demandes, on va répéter la simulation un certain nombre de fois afin d'avoir un blocage moyen sur ces simulations et des intervalles de confiance à 95% pour s'assurer de la fiabilité des résultats. On effectue dans ce test 500 fois la simulation. On fixe notre flux de demande ρ tel que le taux de blocage soit autour de 5% (en se basant sur un first fit). L'idée est de se baser sur un tel taux de blocage pour dimensionner notre slice commun comme taux critique. Un taux plus haut semblerait peu réaliste pour la majorité des réseaux. On va comparer les résultats de notre algorithme au first fit (cas NS) car c'est un algorithme comparable en termes de complexité (aucun calcul de métrique interne à la fonction) et ce dernier est l'algorithme le plus répandu et utilisé dans le cas d'algorithme simple.

5.4.2 Résultats du test

On représente dans la figure 5.4 les résultats pour le test décrit précédemment. On observe tout d'abord que les cas particuliers extrêmes sont moins bons que les résultats avec une slice_value autour de 400. Ceci montre qu'utiliser un algorithme avec un réseau complètement divisé est mauvais. On a un blocage de plus de 6% dans ce cas-là contre moins de 5% pour le cas extrême NS. Ceci est dû au fait que l'on ne change pas les restrictions au cours de la simulation. Même si on connait le taux de chaque granularité à long terme, on n'est pas capable de gérer les pics ponctuels d'arrivés d'une granularité spécifique, excédant la capacité du réseau et provoquant un blocage. Ceci montre alors la nécessité dans notre cas d'utiliser un slice commun pour gérer ces pics. Ensuite, on remarque un minimum dans notre courbe pour une slice_value de 400. On a bien des meilleures performances que le cas NS pour cette valeur (cas slice_value = 0). Les valeurs de la courbe sont données dans le tableau 5.1. On voit que dans le cas NS le blocage est de 4.617% tandis que dans le cas optimal (400), il est de 3.802%. Ceci représente un gain relatif de 17% pour le cas optimal. Pour cette valeur particulière de flux de demande entrant, on obtient des meilleurs résultats que le cas NS. On va maintenant confirmer nos résultats en fixant la valeur de slice_value à 400 et en faisant varier le flux de demande ρ.

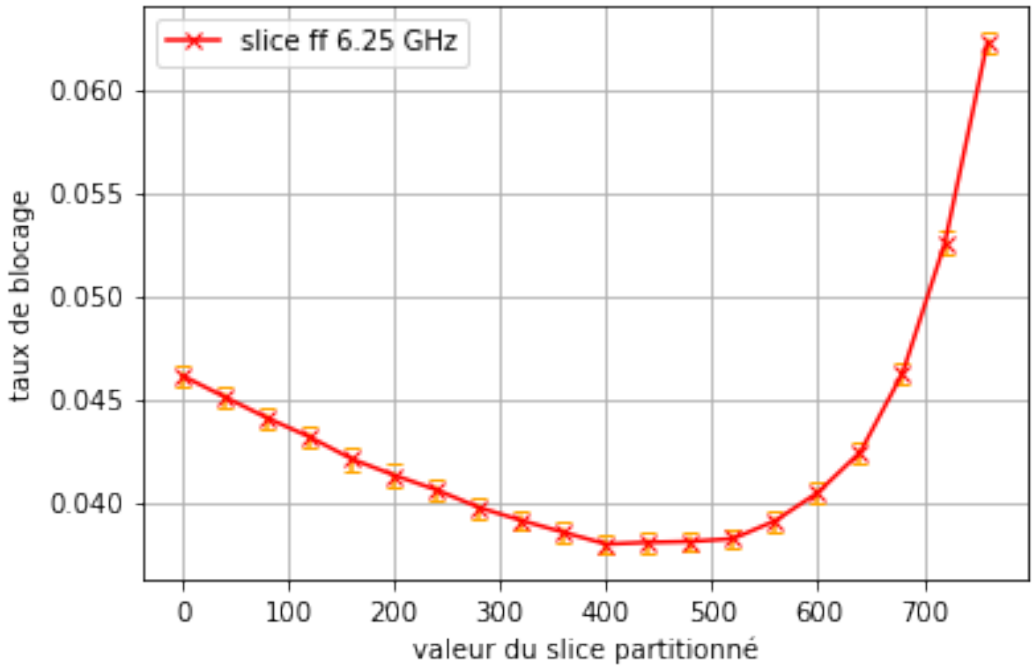

FIGURE 5.4 Taux de blocage pour différentes valeurs du slice partitionné dans le cas 1 lien

Tableau 5.1 Table de résultats du test de variation du slice partitionné dans le cas 1 lien

slice_value	blocage
0	4.617%
40	4.514%
80	4.413%
120	4.321%
160	4.214%
200	4.135%
240	4.064%
280	3.979%
320	3.916%
360	3.858%
400	3.802%
440	3.810%
480	3.814%
520	3.828%
560	3.916%
600	4.052%
640	4.248%
680	4.630%
720	5.262%
760	6.232%

5.4.3 Confirmation des résultats

On garde le même réseau que précédemment. On va faire varier le flux de demande entrant ρ pour les algorithmes de first fit (cas NS) et slice first fit avec un slice_value de 400. Les résultats sont visibles dans la figure 5.5. On remarque que bien que l'on ait dimensionné pour avoir un blocage minimal à 5%, le slice ff reste meilleur que le ff classique pour tout ρ. On voit aussi que plus ρ augmente, plus l'avantage relatif entre les 2 algorithmes diminue. Par exemple, le dernier point donne un blocage de 13.910% dans le cas NS contre 13.008% dans le cas optimal. Ce gain de 0.9% ne représente qu'un gain relatif de 6% alors que pour un ρ de 8 comme utilisé dans le test précédent représente un gain de 0.8% mais un gain relatif de 17%. De manière générale, en diminuant ρ, on augmente le gain relatif entre notre slice ff et le cas NS comme indiqué dans la dernière colonne du tableau 5.5. On obtient même des gains relatifs de plus de 20% pour des ρ inférieurs à 7.6 soit des blocages inférieurs à 3%, ce qui représente une diminution de 6% contre les 21% pour un blocage autour de 5%.

Ces tests montrent que sur un simple lien, il y a un avantage certain à utiliser cette technique. On va maintenant voir si l'intérêt reste lorsqu'on étend l'étude à un chemin puis un réseau.

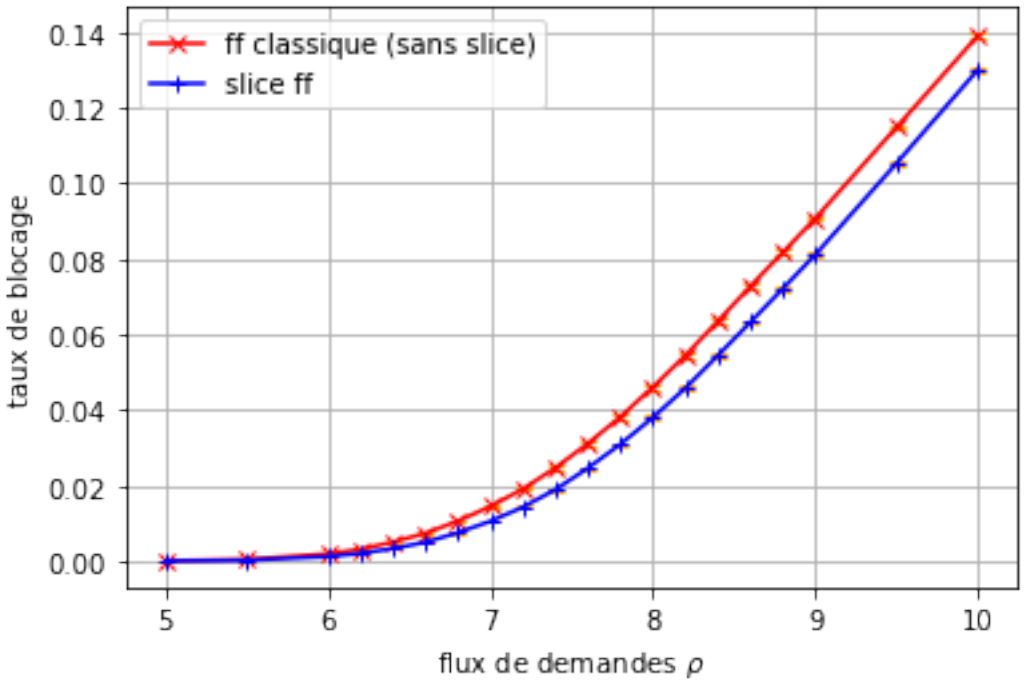

FIGURE 5.5 Taux de blocage en fonction de ρ pour le cas NS et le slice first fit dans le cas 1 lien

TABLEAU 5.2 Table de résultats du test de variation de ρ dans le cas 1 lien

ρ	blocage slice ff	blocage non slice ff	$blocageS-FF/blocageNS-FF$
5	4.7e-3%	8.1e-3%	0.576
5.5	0.023%	0.045%	0.521
6	0.119%	0.191%	0.624
6.2	0.204%	0.313%	0.652
6.4	0.332%	0.505%	0.657
6.6	0.499%	0.738%	0.676
6.8	0.754%	1.062%	0.709
7	1.055%	1.449%	0.729
7.2	1.429%	1.912%	0.748
7.4	1.893%	2.470%	0.766
7.6	2.463%	3.114%	0.791
7.8	3.095%	3.821%	0.810
8	3.802%	4.608%	0.825
8.2	4.581%	5.447%	0.841
8.4	5.444%	6.353%	0.857
8.6	6.325%	7.278%	0.869
8.8	7.216%	8.176%	0.883
9	8.114%	9.076%	0.894
9.5	10.536%	11.490%	0.912
10	13.008%	13.910%	0.935

5.5 Cas du chemin

On passe désormais au cas d'un chemin. On considère un réseau de 4 liens comme représenté sur la figure 5.6. On garde les mêmes paramètres pour effectuer les tests (mêmes granularités, connaissance à l'avance du trafic moyen...). Dans le cas 1 lien, on s'affranchissait de la contrainte de continuité à cause du choix d'un lien simple. Cette fois-ci, sur le chemin, des demandes peuvent être créées de n'importe quel lien vers n'importe quel autre, la contrainte de continuité jouera donc un rôle important. On cherche à savoir si notre algorithme d'assignation reste toujours meilleur dans ce cas plus complexe que le premier.

FIGURE 5.6 Réseau de 4 liens

5.5.1 Test d'optimisation du slice commun

On effectue le même test que précédemment mais avec un réseau différent : ici on traite le cas d'un chemin de 4 liens. Les résultats sont résumés dans la figure 5.7 et la table 5.3. On observe un comportement similaire que dans le cas 1 lien. On obtient la même valeur optimale de 400 avec des gains relatifs en blocage similaires dans ce cas optimal (17%). Le passage au cas 4 liens semble indiquer que le gain effectué par rapport au cas NS sur 1 lien se conserve sur un chemin.

Une différence notable entre les résultats des 2 cas se trouve sur le cas FS. Les résultats restent beaucoup moins bons, mais progressent entre les 2 cas (de 6.2% à 5.2%) alors que les résultats du cas NS et du cas optimal n'ont que sensiblement bougé.

On va maintenant confirmer le gain obtenu sur un chemin en faisant varier le flux de demande ρ pour une variable slice_value de 400.

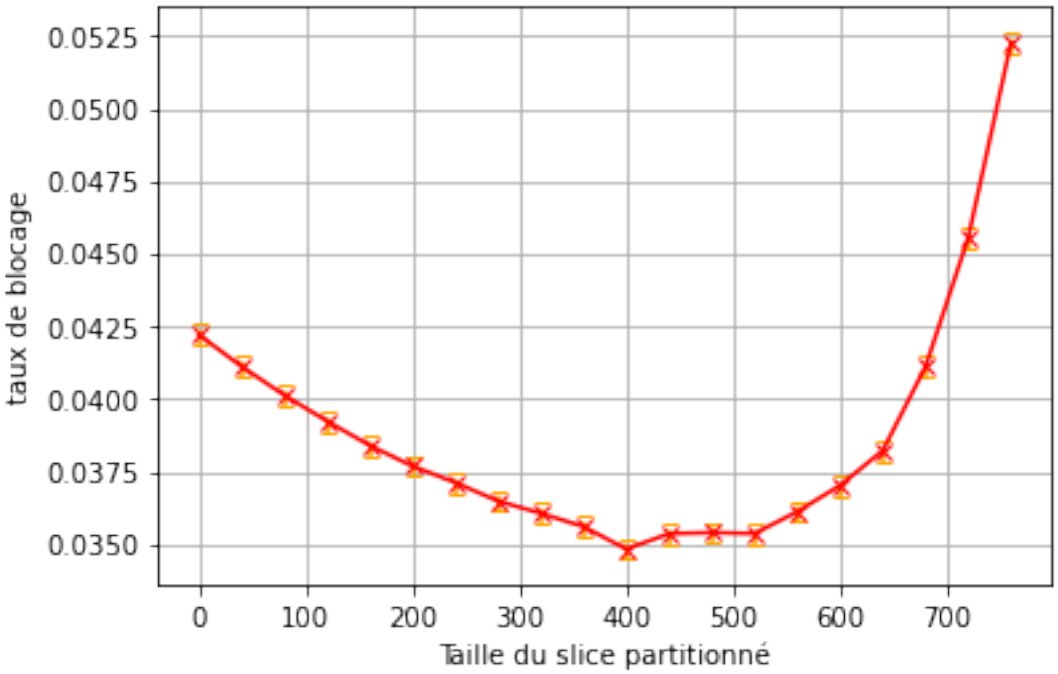

FIGURE 5.7 Taux de blocage pour différentes valeurs de slice_value dans le cas 4 liens

Tableau 5.3 Table de résultats du test de variation du slice partitionné dans le cas 4 liens

slice_value	blocage
0	4.222%
40	4.113%
80	4.014%
120	3.925%
160	3.841%
200	3.770%
240	3.713%
280	3.649%
320	3.607%
360	3.562%
400	3.484%
440	3.537%
480	3.541%
520	3.537%
560	3.613%
600	3.703%
640	3.823%
680	4.111%
720	4.556%
760	5.227%

5.5.2 Confirmation des résultats

On effectue le même test que dans le cas 1 lien et les résultats sont visibles sur la figure 5.8 et la table 5.4. On observe des résultats plutôt similaires au cas du lien simple. On a toujours un gain relatif de 5-6% pour un blocage de 13%, un gain relatif de 17% pour un blocage autour de 4% et un gain relatif de plus de 20% pour des blocages de moins de 4%. Le concept de slicing possède donc un avantage certain à être utilisé au vu de ses résultats sur un lien simple. On va ensuite voir si ces résultats observés sur le blocage sont aussi valables pour les métriques.

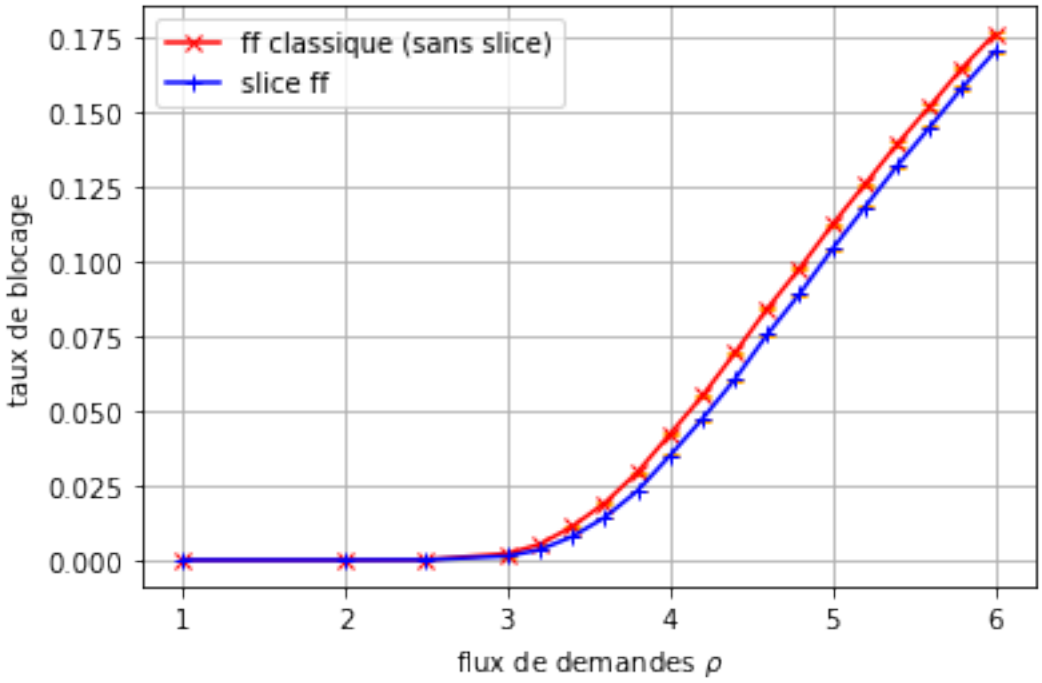

FIGURE 5.8 Taux de blocage en fonction de ρ pour le cas NS et le slice first fit dans le cas 4 liens

TABLEAU 5.4 Table de résultats du test de variation de ρ dans le cas 4 liens

ρ	blocage slice ff	blocage non slice ff	$blocageS-FF/blocageNS-FF$
1	0%	0%	non défini
2	0%	2.9e-4%	0
2.5	5.3e-3%	8.4e-3%	0.625
3	0.132%	0.214%	0.616
3.2	0.342%	0.527%	0.649
3.4	0.788%	1.123%	0.701
3.6	1.429%	1.912%	0.748
3.8	2.315%	2.939%	0.788
4	3.494%	4.215%	0.829
4.2	4.727%	5.495%	0.860
4.4	6.053%	6.938%	0.872
4.6	7.560%	8.427%	0.897
4.8	8.923%	9.782%	0.912
5	10.413%	11.236%	0.927
5.2	11.815%	12.583%	0.939
5.4	13.206%	13.942%	0.947
5.6	14.501%	15.198%	0.954
5.8	15.826%	16.491%	0.960
6	17.035%	17.622%	0.967

5.5.3 Test sur les métriques de fragmentation

On a présenté des meilleurs résultats que le cas NS (first fit) pour l'algorithme de slicing. On a discuté d'un certain nombre de métriques et de leur importance pour déterminer si un réseau est fragmenté ou non. On va alors comparer l'évolution de différentes métriques de fragmentation au cours du temps dans une simulation pour l'algorithme de slicing et pour l'algorithme de first fit sans slicing.

On va tester la métrique RMSF, Shannon entropy (SE) et Access blocking probability (ABP). On a deux métriques intéressantes pour évaluer la présence de fragments sur un lien (RMSF et SE) et une métrique intéressante pour évaluer si ces fragments sont bien dimensionnés pour les métriques considérées. Pour les tests, on va faire 500 simulations pour différents flux de trafic ρ durant un temps de simulation de 4500 en partant d'un réseau vide. Toutes les 10 unités de temps, on calcule la valeur de la métrique de fragmentation en question sur le chemin.

On s'attend à avoir de très bons résultats pour la métrique ABP car chacune des slices est dimensionnée pour qu'il n'y ait aucune perte pour un certain type de granularité. Au contraire, on s'attend à des résultats moins bons pour les métriques SE et RMSF car le slicing introduit des fragments naturels non nécessaires.

Resultats pour Shannon entropy et Root Mean Square Factor

On présente dans les figures 5.9 et 5.10 les résultats respectifs pour les métriques RMSF et SE. On remarque tout d'abord que pour les flux de trafics considérés, le slice first fit obtient de meilleurs résultats que le cas NS (first fit) pour tous les ρ considérés. On voit aussi avec la métrique SE que plus ρ augmente, plus la différence entre les 2 augmente. Ceci parait logique, car plus ρ augmente, plus le remplissage moyen du réseau est grand, et donc plus nos slices sont pleins et les blocs créés par les slices non pleins disparaissent. On s'attendait à avoir des résultats moins bons, mais finalement ce n'est pas le cas. Le gain dans notre manière d'assigner dépasse assez rapidement les blocs créés par nos slices et qui devraient dégrader la fragmentation pour ces deux métriques.

On voit aussi que la fragmentation varie au cours du temps, mais atteint assez vite un équilibre où en moyenne elle ne varie plus. Les fluctuations évoluent autour de cette valeur.

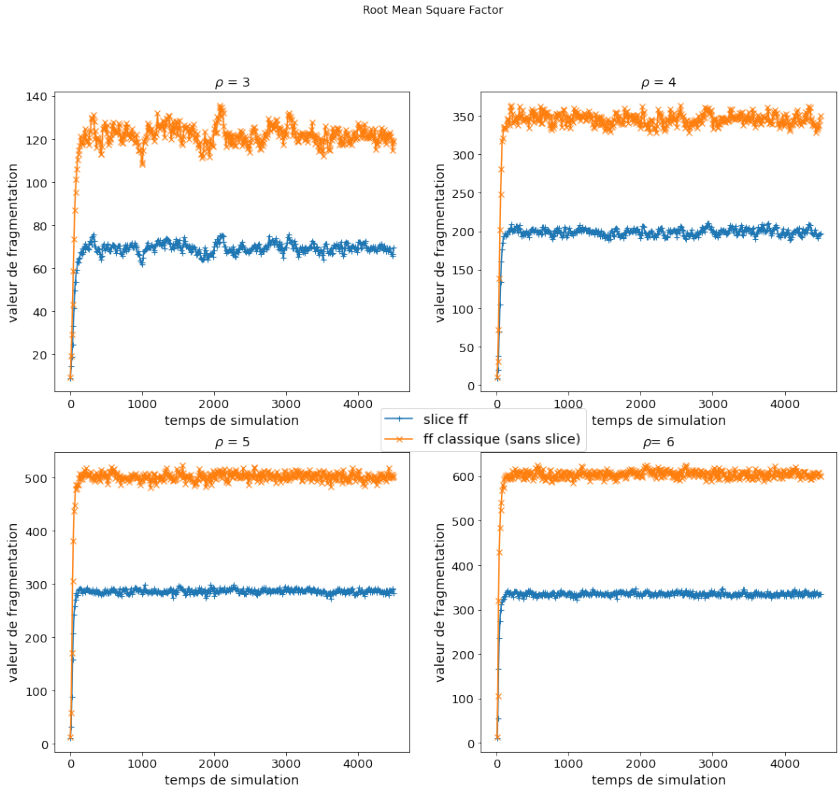

FIGURE 5.9 Evolution de la mesure de fragmentation RMSF en fonction du temps pour différents ρ

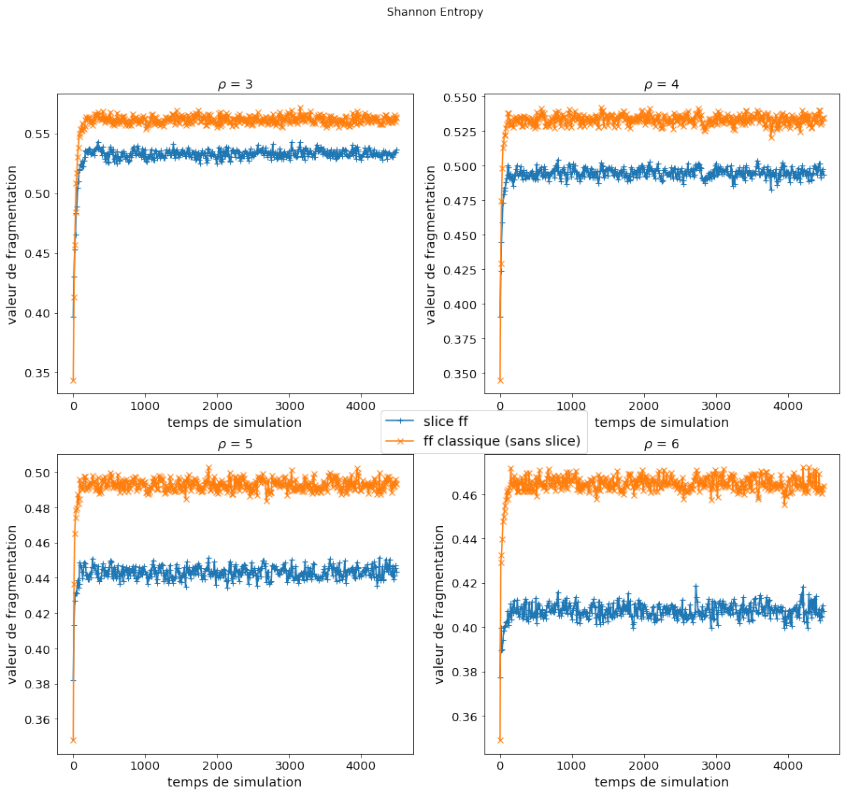

FIGURE 5.10 Evolution de la mesure de fragmentation SE en fonction du temps pour différents ρ

Resultats pour l'Access Blocking Probability

La figure 5.11 présente les résultats pour la métrique ABP. Comme attendu, les résultats sont meilleurs que dans le cas NS. La courbe représentant le slice first fit est en dessous de celle représentant le cas sans slicing. On a bien une fragmentation plus faible pour le slice first fit. Ceci est dû à la conception des slices comme multiple d'une granularité et n'acceptant que les demandes de cette granularité qui permet de garantir que les fragments sur ces slices seront multiples aussi de la granularité. Dans le calcul de notre métrique, on va alors n'avoir aucun slot perdu pour cette granularité sur ce slice.

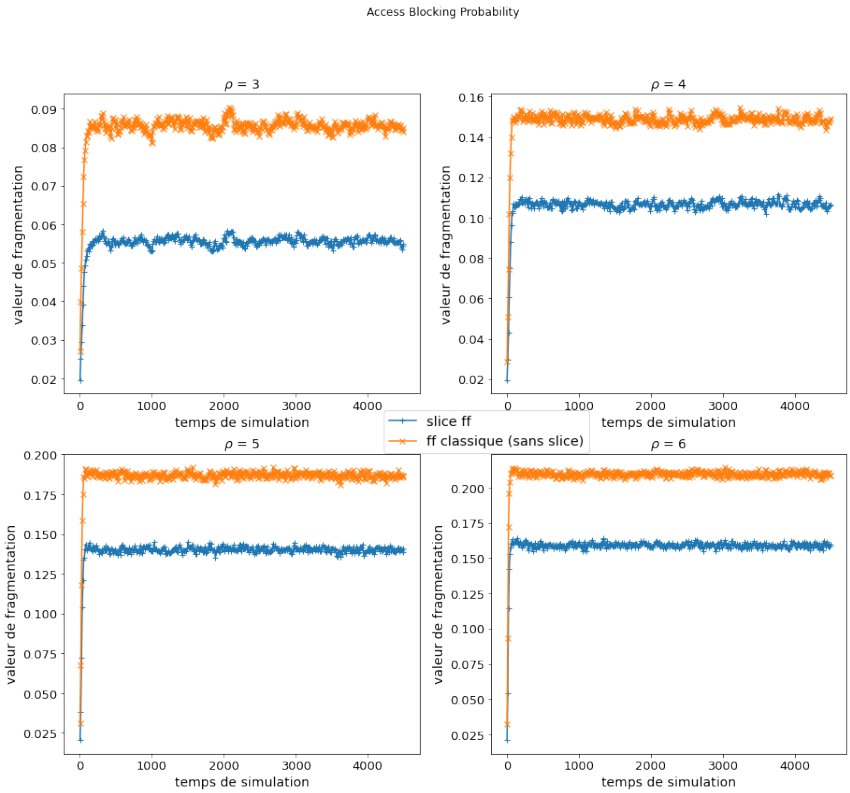

FIGURE 5.11 Evolution de la mesure de fragmentation ABP en fonction du temps pour différents ρ

Observations générales sur les métriques

On a finalement observé que les gains en taux de blocage se traduisent aussi en gain sur les métriques de fragmentation. Il y a bien une certaine corrélation entre blocage et métrique. La méthode proposée sur un chemin permet d'obtenir un réseau moins fragmenté et avec de meilleures performances. Il faut maintenant voir dans quelle mesure cet algorithme s'applique dans un réseau plus complexe qu'un simple chemin. En effet, la notion de trafic estimé pour définir nos slices pourrait être un souci dans un réseau à cause des disparités entre chaque lien et chaque chemin sur le trafic estimé.

CHAPITRE 6 LE SLICING DANS LE CAS D'UN RESEAU

On a introduit le concept de slicing dans le chapitre précédent. On a aussi traité les cas très simples du lien et du chemin. Dans ce chapitre, on va étudier le cas d'un réseau entier. Ce cas pose certaines difficultés, notamment sur la définition des restrictions. Nous expliquerons dans ce chapitre comment on a procédé à ce passage vers un réseau.

6.1 Introduction au cas du petit réseau

Considérons maintenant un réseau complet et non plus un simple chemin. Une demande peut maintenant avoir plusieurs chemins possibles pour une même origine-destination. Nous souhaitons au maximum éviter d'allouer une demande sur le slice commun. En conséquence, si sur le chemin principal, la seule solution se trouve sur le slice commun alors que sur un chemin secondaire une solution sur le slice spécifique à la granularité de cette demande existe, nous choisirons la solution sur le chemin secondaire. Ceci donne une nouvelle manière de gérer le surplus en déchargeant le chemin principal vers les chemins secondaires. Le réseau pose des questions sur la gestion de nos restrictions. Dans le cas du lien et du chemin, on a considéré qu'elles étaient les mêmes sur tous les liens. Ceci peut ne pas être valable pour certains réseaux où il pourrait y avoir des profils de trafics très différents. Cependant, considérer des restrictions différentes sur les liens pose des soucis au niveau de la contrainte de continuité. En effet, si pour 2 liens d'un chemin, les restrictions ne sont pas alignées, certains slots seront inutilisables par les demandes empruntant ces liens bien que libres, car ces slots appartiennent à deux slices dédiés à deux granularités différentes. On a alors développé 2 heuristiques différentes pour calculer ces restrictions. Comme pour les exemples précédents, le trafic est connu à l'avance et les restrictions sont fixées en fonction de ce trafic au début de chaque simulation.

Le réseau considéré est la partie encadrée en rouge sur la figure 6.1 représentant un réseau classique américain fourni par CIENA. On choisit de ne pas considérer le réseau entier pour deux raisons. La première est qu'avant de faire les tests sur un grand réseau américain, nous souhaitons étudier le comportement de notre algorithme sur un plus petit réseau. La seconde est que le simulateur n'est pas encore assez rapide pour faire fonctionner un aussi grand réseau durant des milliers d'unités de temps pour des centaines de seeds et de valeurs à faire varier en un temps raisonnable. Nous nous restraignons donc au petit réseau rouge.

FIGURE 6.1 Exemple d'un réseau américain fournit par CIENA

6.2 Restriction identique pour tous les liens du réseau

Nous connaissons le trafic global dans le réseau. On définit nos restrictions de la même manière que précedemment. On essaye de définir une taille du slice commun et on partitionne le slice spécifique avec la formule d'Erlang (5.2). Il s'agit du même fonctionnnement que pour le lien et le chemin mais cette fois-ci appliquée à un réseau entier. Cette heuristique devrait se montrer efficace dans le cas d'un trafic plus ou moins similaire sur un réseau, mais devrait être moins efficace dans le cas de trafic plus hétérogène. La première étape pour cette heuristique est comme précédemment de trouver la taille optimale du slice commun.

6.3 Restriction spécifique pour chaque lien

On définit cette fois nos restrictions dans chacun des liens. Comme nous connaissons en avance les demandes entrantes, nous allons regarder pour chacune d'elles leur chemin le plus court. Nous récupérons les liens de ce chemin et ajoutons 1 à un compteur pour la granularité de la demande considérée. En étudiant les demandes une par une, nous avons ensuite un trafic par demande pour chacun des liens. Par la suite, nous utilisons la formule d'Erlang (5.2) sur chacun des liens et obtenons les restrictions des liens. On ne change rien cependant pour la définition du slice commun. Il sera le même sur tous les liens du réseau. Par conséquent, on commence par le même test que précédemment pour définir la taille optimale du slice commun.

6.4 Test pour trouver la taille optimale du slice commun

On utilise le même test pour le lien et le chemin. Cependant, pour des raisons de temps de calcul, certains choix ont été faits sur certains paramètres. On va faire varier la taille de notre slice partitionné de 0 à 750 avec 1 point tous les 50 (contre 0 à 760 avec 1 point tous les 40 avant). De plus, le temps de simulation choisi est de 1000 (contre 4500 avant). Un temps plus long augmente considérablement le temps de simulation et un temps plus faible ne permet pas à notre réseau de se stabiliser, le réseau commençant vide. On considère toujours des granularités possibles de 8, 12 et 19 slots. La simulation est effectuée 200 fois et on garde un blocage autour de 4-5%. On génère les demandes via l'outil dévéloppé dans le simulateur.

6.5 Résultats pour la restriction identique pour tous les liens

On fait varier la taille du slice partitionné slice_value entre 0 et 750 pour le cas d'une restriction globale au réseau. Les résultats sont résumés dans la figure 6.2 et la table **??**. Le profil de courbe reste le même que dans le cas du lien avec une valeur optimale entre les cas NS et FS. Cependant, la valeur optimale ici semble être aux alentours de 650 contre 400 auparavant. De plus, le cas FS a des performances similaires voir légèrement meilleures que le cas NS. On peut l'expliquer par la capacité d'un réseau à gérer le surplus en basculant les demandes sur des chemins secondaires. Cependant, il paraît toujours nécessaire d'avoir un slice commun. Pour la valeur de 650, on obtient un blocage 1.3% plus bas que dans le cas NS. Ceci représente un gain relatif de 30% pour notre solution. Les gains sont alors encore plus grand que dans le cas chemin (gain relatif aux alentours de 17%). Le principal problème du slicing est de bien dimensionner les slices et de gérer les pics ponctuels. La diversité des chemins possibles dans un réseau permet d'obtenir plus de flexibilité, ce qui peut expliquer que le gain soit encore plus grand.

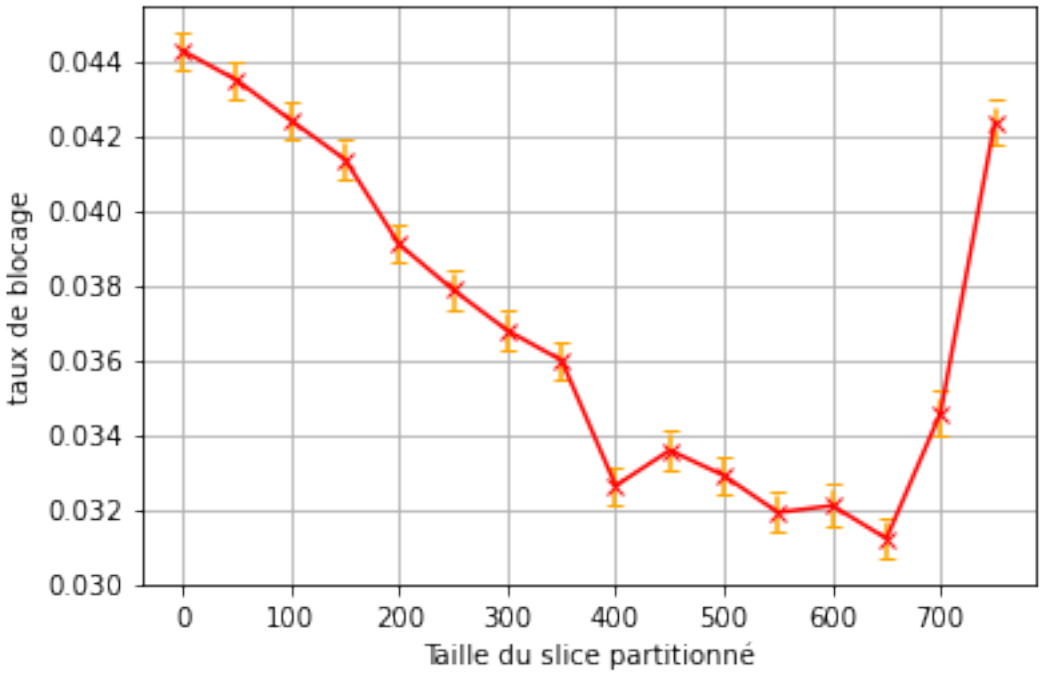

FIGURE 6.2 Taux de blocage pour différentes valeurs de slice_value dans le cas du petit réseau avec l'heuristique des restrictions indentiques pour tous les liens

TABLEAU 6.1 Table de résultats du test de variation de slice_value dans le cas du petit réseau avec restriction indentiques pour tous les liens

slice_value	blocage
0	4.429%
50	4.351%
100	4.243%
150	4.138%
200	3.911%
250	3.789%
300	3.681%
350	3.601%
400	3.266%
450	3.360%
500	3.293%
550	3.194%
600	3.212%
650	3.125%
700	3.461%
750	4.238%

6.6 Résultats pour la restriction spécifique pour chaque lien

On montre les résultats pour le même test avec l'heuristique par lien présentés dans la section 6.3. Ces résultats sont résumés dans la figure 6.3 et la table 6.2. Ils ne sont pas aussi bons que pour l'heuristique globale. On n'obtient pas de gain significatif par rapport au cas NS. De plus, le cas FS est bien pire dans ce cas qu'avec l'heuristique identique pour tous les liens. Il semblerait que le potentiel non-alignement entre 2 liens joue un rôle néfaste sur les performances en termes de blocage. On a superposé les deux courbes sur la figure 6.4. Il y a un très net avantage pour la courbe avec l'heuristique globale. On a bien le point 0 qui coïncide entre les deux courbes. Ceci est conforme à nos attentes puisqu'on est dans le cas NS qui revient à un algorithme first fit.

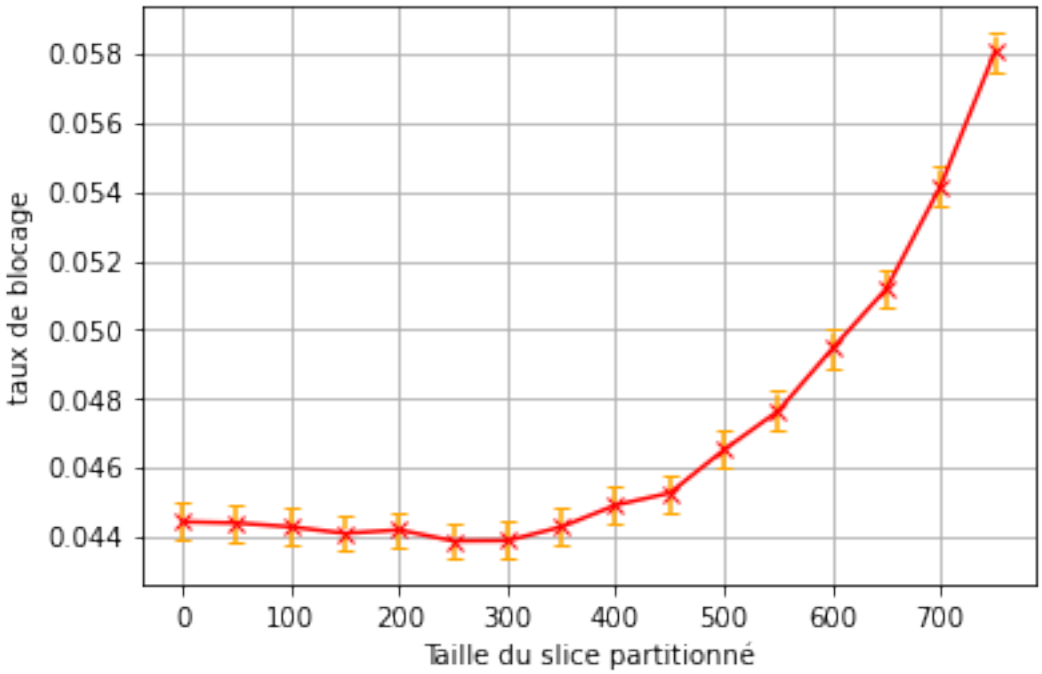

FIGURE 6.3 Taux de blocage pour différentes valeurs de slice_value dans le cas du petit réseau avec l'heuristique des restrictions par liens

TABLEAU 6.2 Table de résultats du test de variation de slice_value dans le cas du petit réseau avec restriction par lien

slice_value	blocage
0	4.443%
50	4.4.439%
100	4.428%
150	4.409%
200	4.419%
250	4.387%
300	4.388%
350	4.428%
400	4.491%
450	4.526%
500	4.652%
550	4.764%
600	4.945%
650	5.119%
700	5.412%
750	5.805%

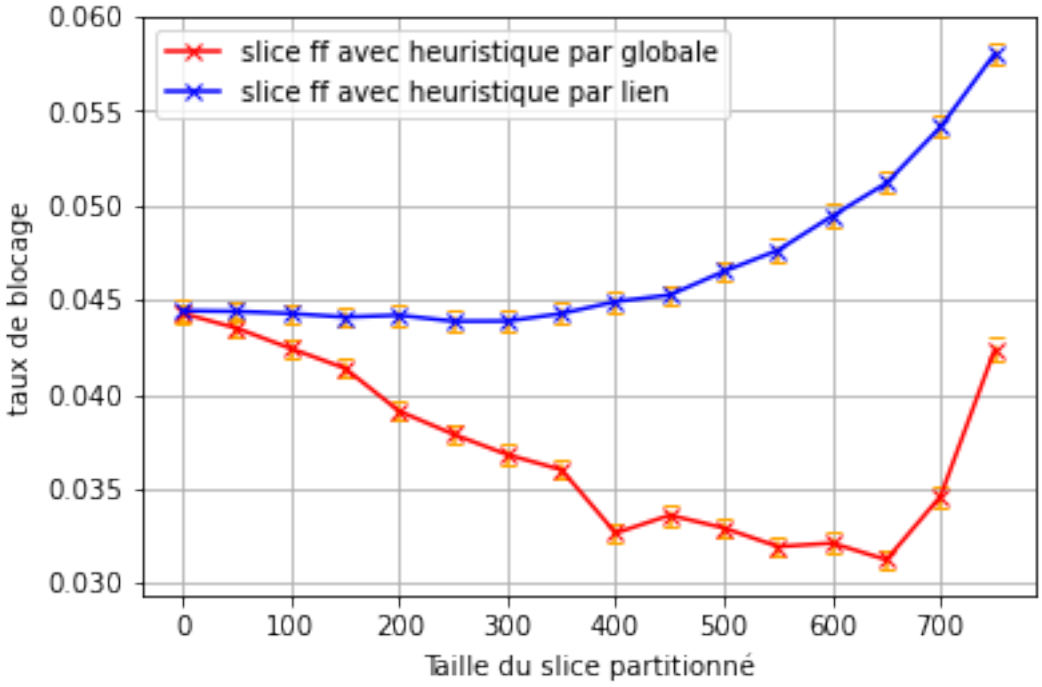

FIGURE 6.4 Taux de blocage pour différentes valeurs de slice_value dans le cas du petit réseau avec les deux heuristiques

6.7 Confirmation des résultats pour l'heuristique des restrictions identiques pour tous les liens

On va maintenant confirmer le gain observé dans le cas cette heuristique en faisant varier ρ et en effectuant 100 simulations. Pour cela, on garde les mêmes paramètres de simulation en faisant varier ρ et on va choisir 3 politiques d'assignations à comparer :

- un first fit (cas NS)
- un slice first fit avec restriction identique pour tous les liens et avec une slice_value de 650 (cas optimal observé dans 6.5)
- un autre slice first fit avec restriction identique pour tous les liens et avec une slice_value de 400 (ancienne valeur optimale avec des résultats toujours bons)

Les résultats sont exposés dans la figure 6.5 et la table 6.3. Tout d'abord, notre algorithme de slicing dans les 2 cas possède des meilleurs résultats que l'algorithme sans slicing jusqu'à des blocages de 15%. Le concept est alors capable d'améliorer les performances par rapport à un simple first fit en des temps de calcul comparables et même légèrement plus rapides : en réduisant les tranches de spectres allouables possibles, on réduit aussi le nombre de solutions possibles à parcourir et donc le temps pour trouver une solution. Cependant, le choix de la

taille du spectre partitionné est important et très sensible. Le cas 650 est optimal autour de 5% mais en augmentant encore ρ, on voit que les performances se dégradent et deviennent même moins bonnes que le cas NS autour de 15% alors que le cas 400 garde un avantage de 0.7% pour ce même blocage. On obtient un gain relatif de plus de 20% dans les deux cas pour des ρ inférieur à 2.2. Ceci représente pour le cas slice ff 650 un blocage 1.3% plus bas pour un ρ de 2.

Finalement, l'algorithme proposé est aussi capable d'obtenir de bons résultats lorsqu'on utilise cette méthode sur un réseau. Ces gains sont même encore plus que dans les cas simples du chemin et du lien étudié précédemment. On va maintenant voir quels sont les répercussions sur les métriques.

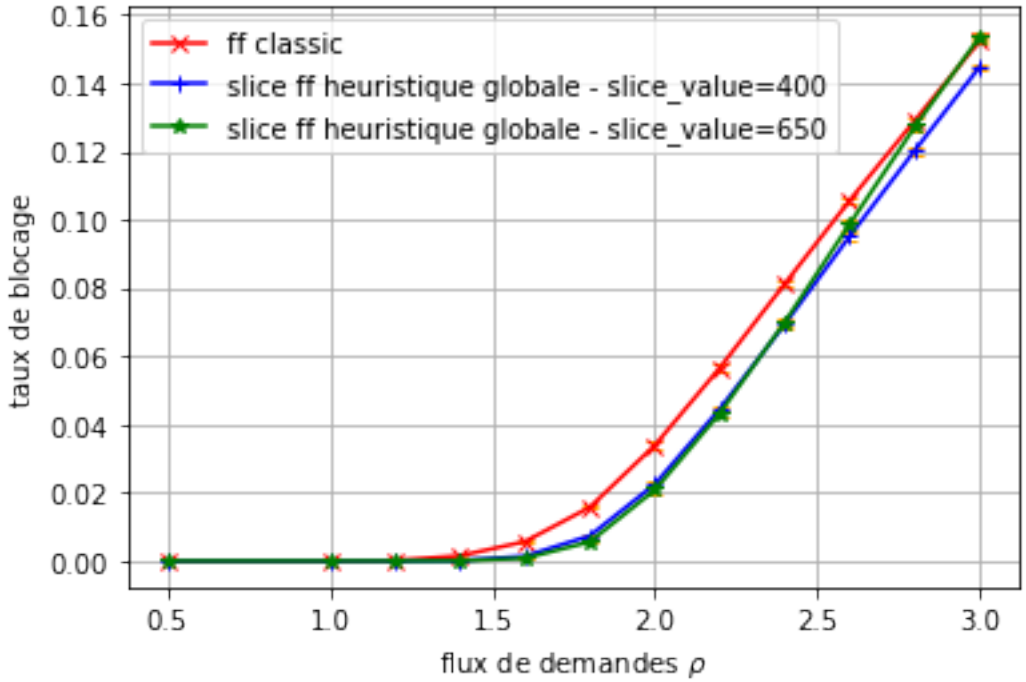

FIGURE 6.5 Taux de blocage pour différentes valeurs du flux de demandes ρ dans le cas du petit réseau

TABLEAU 6.3 Table de résultats du test de variation du flux de demandes ρ dans le cas du petit réseau

ρ	blocage non slice ff	blocage slice ff 400	blocage slice ff 650	gain relatif slice ff 400	gain relatif slice ff 650
0.5	0%	0%	0%	non défini	non défini
1	1.03e-3%	0%	0%	0	0
1.2	0.015%	4.6e-4%	4.2e-4%	0.030	0.028
1.4	0.149%	0.011%	5.4e-3%	0.074	0.036
1.6	0.569%	0.136%	0.079%	0.240	0.139
1.8	1.572%	0.731%	0.568%	0.465	0.361
2	3.405%	2.252%	2.083%	0.661	0.612
2.2	5.632%	4.410%	4.349%	0.783	0.772
2.4	8.119%	6.935%	7.061%	0.854	0.870
2.6	10.573%	9.482%	9.852%	0.897	0.932
2.8	12.872%	11.977%	12.648%	0.930	0.983
3	15.271%	14.503%	15.415%	0.950	1.009

6.8 Cas des métriques de fragmentation

Comme pour le cas du chemin, nous montrons les résultats pour les métriques de fragmentation. Nous choisissons les métriques ABP, RMSF et WS. Par rapport au cas du chemin, on choisit de ne pas considérer la métrique SE, car les résultats et le fonctionnement sont proches de la métrique RMSF. Comme précedemment, nous regardons les métriques RMSF et ABP pour les mêmes raisons. Nous regardons aussi les résultats pour la métrique WS. C'est une métrique qui regarde plus la fragmentation horizontale contrairement aux deux autres. Ça avait peu du sens de regarder cette métrique dans le cas du chemin, car chaque lien n'avait que peu de voisin et le réseau était très simple.

Comme pour les tests précédents sur le petit réseau, on considère un temps de simulation de 1000 unités de temps et des ρ de 1.8, 2, 2.2 et 2.4. On va calculer la fragmentation globale du réseau pour les 10 unités de temps. On compare le first fit classique (sans slice), notre algorithme slice FF avec une slice_value de 400 et notre algorithme slice FF avec une slice_value de 650. On s'attend à avoir globalement des résultats bien meilleurs pour les deux slice FF que pour le FF. Ainsi que des résultats légèrement meilleurs pour le slice FF 650 que le slice FF 400 pour des ρ de 1.8 et 2. Mais environ égal pour des ρ de 2.2 et 2.4 car comme on peut voir dans la table 6.3 les résultats de blocage sont similaires dans les deux cas et même légèrement meilleurs dans le cas slice FF 400.

6.8.1 Cas de la métrique RMSF

Les résultats pour le cas de la métrique RMSF sont dans la figure 6.6. Tout d'abord, on observe bien une nette amélioration de cette métrique dans le cas des algorithmes de slice FF comparé au cas NS (simple first fit). Par exemple, pour un ρ de 2, le slice FF 400 a une fragmentation autour de 430 et le slice FF 650 a une fragmentation autour de 330 tandis que dans le cas NS, la fragmentation se situe autour des 700. On a donc près d'un rapport de 2 entre le cas NS et nos algorithmes pour cette métrique de fragmentation. De plus, on observe bien une légère amélioration dans le slice FF 650 par rapport au slice FF 400 ce qui peut expliquer le taux de blocage plus bas de notre algorithme dans le cas d'une slice_value de 650 par rapport à une slice_value de 400. Cependant, de manière surprenante, pour les ρ de 2.2 et 2.4, on observe toujours un gain dans la valeur de la fragmentation alors que les blocages entre ces méthodes sont similaires voir légèrement meilleur dans le cas slice FF 400 (cf table 6.3).

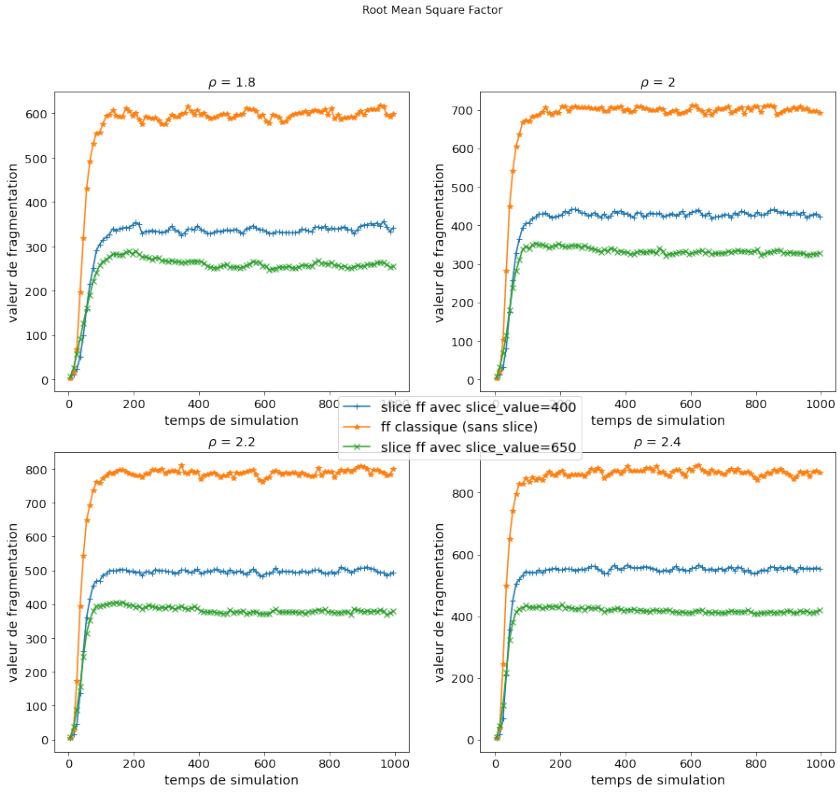

FIGURE 6.6 Evolution de la mesure de fragmentation RMSF en fonction du temps pour différents ρ sur le petit réseau

6.8.2 Cas de la métrique ABP

Les résultats pour le cas de la métrique ABP sont dans la figure 6.7. La même observation en gain est faite entre le cas NS et les algorithmes slice FF 400 et slice FF 650. Pour un ρ de 2, on varie autour de 0.26 pour le cas NS alors que l'on varie plutôt autour de 0.19 dans le cas slice FF 400 et 0.16 dans le cas slice FF 650. On observe aussi une légère amélioration dans le cas slice FF 650 par rapport au cas slice FF 400, et ce, pour les 4 ρ considérés.

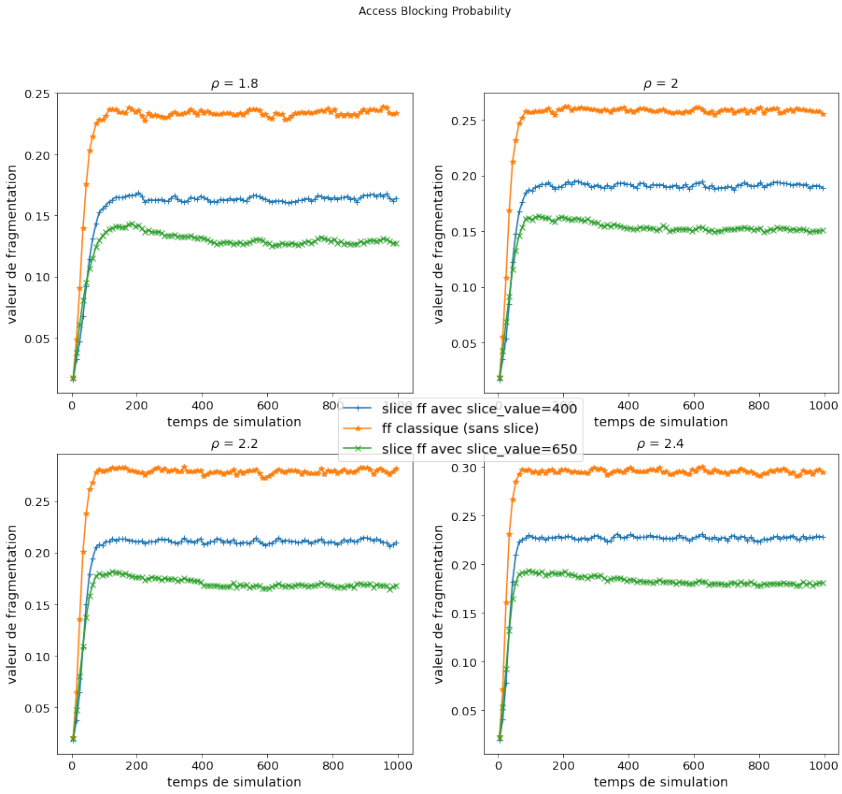

FIGURE 6.7 Evolution de la mesure de fragmentation ABP en fonction du temps pour différents ρ sur le petit réseau

6.8.3 Cas de la métrique WS

Les résultats pour le cas de la métrique ABP sont dans la figure 6.8. Pour cette métrique observant plus la fragmentation horizontale, on n'observe pas de réel gain de la part d'une des 3 méthodes. Elles ont des valeurs très proches, légèrement au-dessus de 500, et ce, pour les 4 ρ considérés. Il semble alors que notre méthode proposée représente principalement un gain au niveau de la fragmentation verticale et moins au niveau de la fragmentation horizontale.

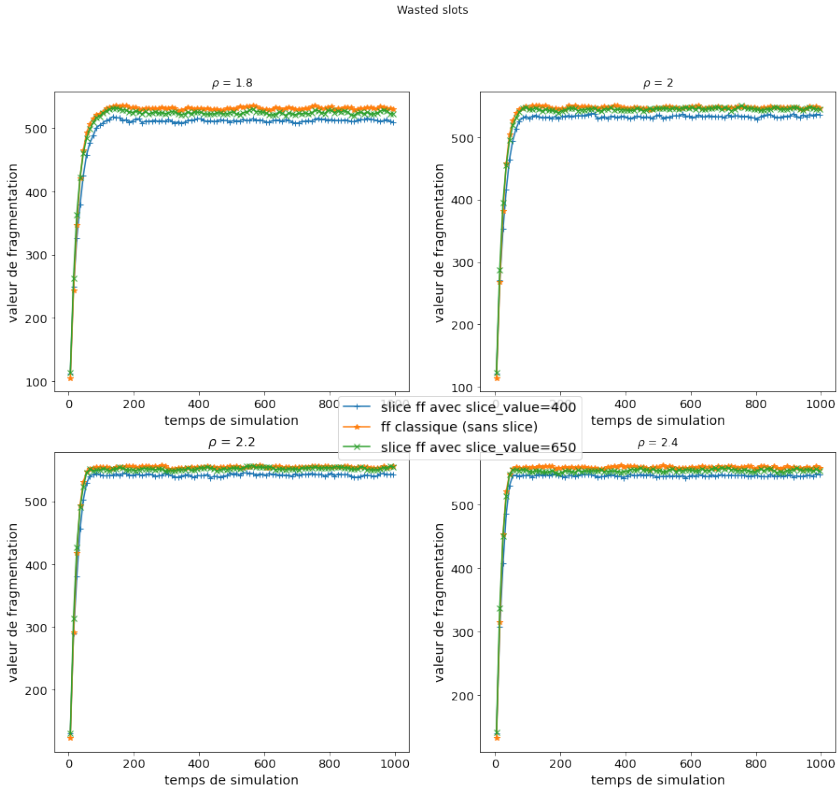

FIGURE 6.8 Evolution de la mesure de fragmentation WS en fonction du temps pour différents ρ sur le petit réseau

6.8.4 Observations générales sur les métriques

Finalement, on voit bien que le gain en blocage peut s'expliquer par un gain via nos métriques de fragmentation. Cependant, ce gain se situe principalement au niveau de la fragmentation horizontale. De plus, on a aussi pu voir que métrique de fragmentation et blocage ne sont pas exactement pareils. On a vu un cas où la métrique de fragmentation est meilleure alors que le blocage ne l'est pas. Il faut alors voir la métrique de fragmentation comme un des paramètres donnant une information sur le blocage. Notre hypothèse sur cela est que sous fort trafic, le surplus ponctuel d'un type de connexion est plus fort et alors le gain en fragmentation n'est pas compensé par la perte dû à un surplus non géré. Dans le cas slice FF 400, le slice commun est plus grand et est donc plus capable de gérer ces pics ponctuels malgré une fragmentation plus grande.

6.9 Limites et travail futur

Dans l'ensemble des tests effectués, on avait connaissance en avance du trafic arrivant. Dans la pratique, c'est généralement faux. Il est possible d'avoir une planification à long terme sur certains réseaux, mais on ne peut pas se permettre de considérer ce critère comme systématiquement vrai. Il faut donc adapter cette idée en prédisant dans sa globalité le trafic arrivant. Une mémoire des dernières connexions ou une utilisation du machine learning pour de la prédiction de trafic sont des pistes pour réussir cela. Aussi, on utilise un trafic moyen invariant au cours d'une simulation, ce qui n'est pas forcément juste en pratique. En effet, le réseau évolue et on peut imaginer que le pourcentage de chacune des granularités présentes dans le réseau change. Dans ce cas, il sera nécessaire de changer les restrictions sur nos slices entrainant certains problèmes. Une demande présente dans le réseau peut être dans le bon slice avant le changement des restrictions, mais ne plus l'être après ce changement. Que faire de cette demande ? Quand effectuer ce changement ? Il faut peut-être déterminer un seuil à partir duquel la différence est trop grande et n'effectuer ces changements que lorsque le seuil est trop grand. On peut aussi effectuer une action de défragmentation pour bouger les demandes posant un problème lors de ce changement. La défragmentation est l'action de réarranger les connexions existantes pour réduire la fragmentation dans le réseau lorsque celle-ci devient trop haute.

Une limite observée se trouve sur les granularités possibles. Nous n'avons pas regardé comment se comportent les résultats si jamais on augmente ou diminue le nombre de granularités possibles ou si on modifie les valeurs de ces granularités. Effectuer le même test d'optimisation du slice sur un chemin, mais avec des granularités de 8,12,20 donne des résultats différents. La valeur optimale se trouve plutôt autour de 300 et le gain n'est plus que de 6% entre le cas optimal et le cas NS. Notre hypothèse est que la fragmentation est moins grande dans ce cas-ci, car toutes les demandes sont de tailles paires et donc les trous laissés sont aussi de tailles paires alors que dans le cas que l'on a étudié un trou impair comblé par une demande paire laissera toujours un résidu de slots potentiellement perdu.

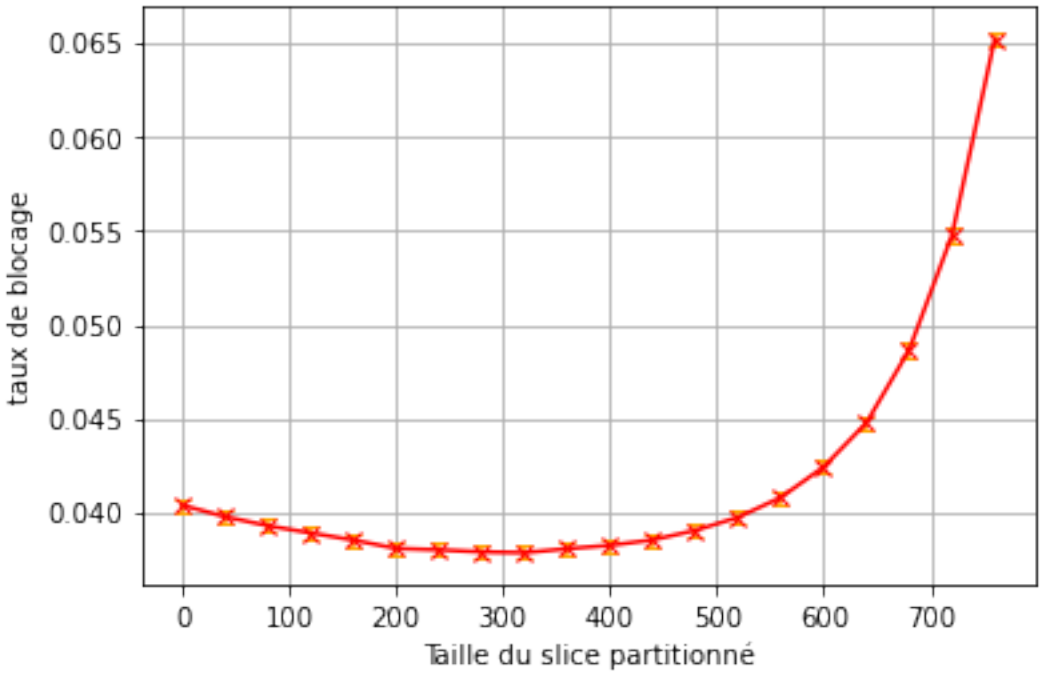

FIGURE 6.9 Taux de blocage pour différentes valeurs de slice_value dans le cas du chemin avec des granularités de 8,12 et 20

Enfin, une dernière limite est si le trafic dans le réseau est plus hétérogène, notamment au niveau des granularités. Par exemple, si pour un lien le trafic accueille 50% de granularité 8, 30% de granularité 12 et 20% de granularité 20 alors qu'un autre lien accueille 20% de granularité 8, 30% de granularité 12 et 50% de granularité 20, il est difficile de dimensionner des restrictions à la fois pour un lien et pour l'autre dans le cas de restrictions identiques pour chacun des liens. La figure 6.10 montre des résultats dans un tel cas. On voit notamment que les blocages explosent pour un réseau entièrement partitionné avec l'heuristique des restrictions identiques pour tous les liens. On voit d'ailleurs que l'heuristique des restrictions par lien s'en sort mieux. Dans les deux cas, on conserve tout de même des performances similaires au cas NS jusqu'à une valeur de slice_value de 400.

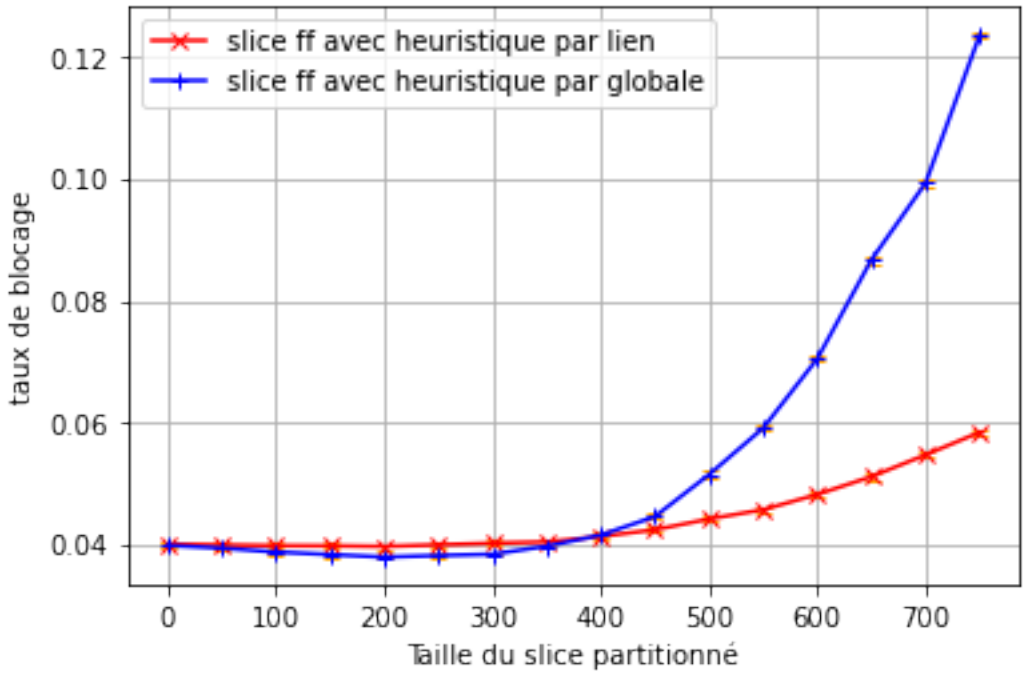

FIGURE 6.10 Taux de blocage pour différentes valeurs de slice_value avec un autre trafic pour les deux heuristiques

6.10 Discussion autour des grilles de fréquence

La taille de la grille de fréquence a fortement diminué avec les EON. Aujourd'hui, il y a 2 tailles de grille qui ressortent : 12.5 et 6.25 GHz. On se demande s'il y a un fort intérêt ou non à utiliser la plus petite grille. Pour cela, on prend un simple lien avec des demandes possibles de 50GHz, 75GHz et 118.75GHz. On simule les résultats avec une grille de 6.25GHz et 12.5GHz. On effectue un test d'optimisation de la taille d'un slice avec ces deux grilles. On affiche les résultats des deux cas dans la figure 6.11 en utilisant en abscisse la grille de 6.25GHz. À noter qu'il y a 2 fois moins de slots dans le cas 12.5GHz. Afin de pouvoir superposer les courbes, on considère que le point pour slice_value = 20 pour la grille 12.5GHz équivaut au point slice_value = 40 pour la grille de 6.25GHz.

On remarque que pour le cas NS, la différence entre les 2 grilles n'est pas frappante. Cependant, pour le cas optimal de notre test, la différence est visible : on passe d'un blocage de 3.8% à 4.2%. Pour ce cas, on a donc un clair avantage à utiliser une grille plus fine. La réduction de grille semble alors être d'autant plus bénéfique pour un algorithme utilisant le slicing.

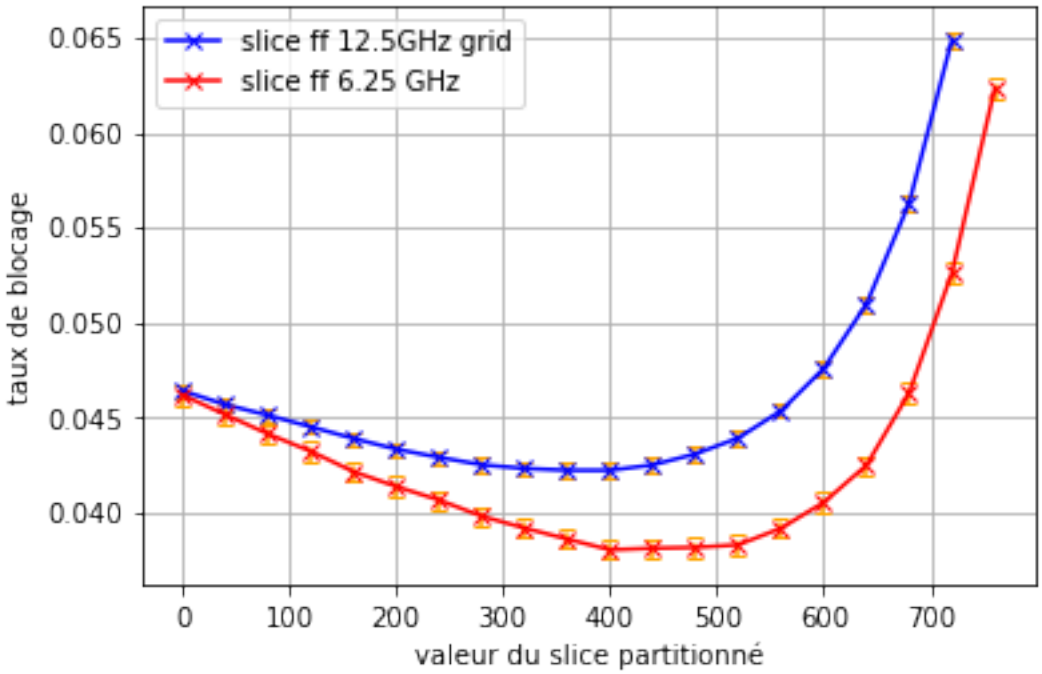

FIGURE 6.11 Taux de blocage pour différentes valeurs du slice partitionné avec deux grilles

CHAPITRE 7 CONCLUSION

7.1 Synthèse des travaux

L'évolution permanente des réseaux optiques rend la grille de fréquences de plus en plus fine. On est passé de plusieurs centaines de GHz à moins de 10GHz avec la nouvelle génération de réseaux EON. Cela permet de mieux adapter la bande passante offerte à une certaine demande pour éviter que cette dernière n'utilise qu'une partie de cette bande passante offerte. Cependant, cela pose un certain nombre de défis pour pouvoir utiliser ces nouveaux réseaux avec une efficacité maximale. La fragmentation est un problème crucial à résoudre si l'on souhaite arriver à cette efficacité. Pour cela, il est important de mettre en place des politiques d'assignation innovantes.

Au cours de ce mémoire, nous avons comparé quelques algorithmes de fragmentation aware. Certaines métriques sont plus efficaces que d'autres pour quantifier le problème de fragmentation. On a essayé d'introduire de l'aléa dans le choix des solutions dans un algorithme de fragmentation aware et on a finalement vu que cela n'était pas bénéfique. Les performances de blocages restaient moins bonnes que le cas déterministe consistant à choisir la meilleure solution par rapport à une métrique. Structurer notre méthode d'assignation est primordial. Pour cela, nous choisissons de partitionner le spectre entre les différentes tailles de demandes. Via cette idée de partition, nous espérons proposer une méthode nouvelle et efficace pour traiter ce problème.

Nous avons vu que le First Fit classique n'est pas un mauvais algorithme. Il possède des performances plutôt bonnes au vu de la simplicité d'un tel algorithme. Mais il est possible d'avoir des algorithmes bien plus performants. Nous avons montré qu'un algorithme utilisant une partition du spectre peut se montrer très efficace s'il est bien adapté et se trouve dans les bonnes conditions. Pour ce faire, nous avons montré les différents résultats d'un tel algorithme pour des situations de plus en plus complexes allant du simple lien à un réseau réel. Ces résultats montrent de meilleures performances à la fois pour le blocage et pour les métriques de fragmentation. On a vu que ces gains se situaient autour de 20% pour des blocages autour de 5% par rapport à ce même first fit. Cet algorithme pourrait alors être développé pour gérer l'assignation de demande à l'intérieur d'un réseau optique. Cependant, il y a certains axes de développement à résoudre avant qu'une telle version puisse en effet être utilisée dans un réseau optique réel.

7.2 Limites et améliorations futures

Au cours de nos différentes simulations, nous nous sommes positionnés en oracle en connaissant à l'avance les demandes entrante pour déterminer le trafic moyen futur. Ceci n'est pas toujours possible dans un réseau classique. La suite est donc de développer des outils pour déterminer ce trafic. Il faut aussi penser comment changer les restrictions sans trop perturber le réseau. Enfin, le slicing est une idée générique développable. À l'intérieur de chacun des slices, on utilise un simple first fit. On peut imaginer un développement des algorithmes utilisé dans chacun des slices, notamment pour le slice commun. Certains de ces algorithmes auront sûrement un apport, que ce soit en termes de blocage ou de temps de calcul.

Tous ces points sont cruciaux à résoudre avant d'éventuellement implémenter cette idée dans un réseau optique. Enfin, malgré nos efforts pour limiter au plus le phénomène de fragmentation, il se manifestera toujours. Dans ce cas, il sera important d'étudier le problème proche de la défragmentation afin de réoptimiser le réseau.

Enfin, le slicing est une idée générique développable. À l'intérieur de chacun des slices, on utilise un simple first fit. On peut imaginer un développement des algorithmes utilisé dans chacun des slices, notamment pour le slice commun. Certains de ces algorithmes auront sûrement un apport, que ce soit en termes de blocage ou de temps de calcul.

RÉFÉRENCES

[1] S. Kohlert, juin 2021, présentation des réseaux optiques de CIENA.

[2] S. N. K. Sunny, "Spectrum defragmentation in elastic optical networks," Thèse de doctorat, Politecnico di Torino, 2018.

[3] D. Sharma et S. Kumar, "An overview of elastic optical networks and its enabling technologies," *International Journal of Engineering and Technology*, vol. 9, p. 1643–1649, 06 2017.

[4] L. Delvalle, E. Alfonzo et D. P. P. Roa, "Eons : An online rsa simulator for elastic optical networks," dans *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, 2016, p. 1–12.

[5] S. Talebi, F. Alam, I. Katib, M. Khamis, R. Khalifah et G. Rouskas, "Spectrum management techniques for elastic optical networks : A survey," *Optical Switching and Networking*, vol. 13, 07 2014.

[6] M. Klinkowski, "An evolutionary algorithm approach for dedicated path protection problem in elastic optical networks," *Cybernetics and Systems*, vol. 44, p. 589–605, 09 2013.

[7] B. C. Chatterjee, S. Ba et E. Oki, "Fragmentation problems and management approaches in elastic optical networks : A survey," *IEEE Communications Surveys Tutorials*, vol. 20, n°. 1, p. 183–210, 2018.

[8] D. Amar, E. Rouzic, N. Brochier, J.-L. Auge, C. Lepers et N. Perrot, "Spectrum fragmentation issue in flexible optical networks : analysis and good practices," *Photonic Network Communications*, vol. 29, 06 2015.

[9] M. Johnstone et P. Wilson, "The memory fragmentation problem : Solved ?" *ACM SIGPLAN Notices*, vol. 34, 01 1998.

[10] P. Lechowicz, "Regression-based fragmentation metric and fragmentation-aware algorithm in spectrally-spatially flexible optical networks," *Computer Communications*, vol. 175, p. 156–176, 2021. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S0140366421001961

[11] E. Chromy, M. Tibor et K. Matej, "Erlang c formula and its use in the call centers," *Advances in Electrical and Electronic Engineering*, vol. 9, 03 2011.

[12] P. Wright, M. C. Parker et A. Lord, "Simulation results of shannon entropy based flexgrid routing and spectrum assignment on a real network topology," dans *39th European Conference and Exhibition on Optical Communication (ECOC 2013)*, 2013, p. 1–3.

[13] A. Ghallaj, R. Romero Reyes, M. Ermel et T. Bauschert, “Optimizing spectrum allocation in flex-grid optical networks,” dans *Photonic Networks ; 18. ITG-Symposium*, 2017, p. 1–8.

[14] N. T. Khi, R. Romero Reyes et T. Bauschert, “Spectrum defragmentation with improved lightpath migration scheme in flex-grid networks,” dans *2021 International Conference on Optical Network Design and Modeling (ONDM)*, 2021, p. 1–6.

[15] Z. Ye, A. N. Patel, P. N. Ji et C. Qiao, “Root mean square (rms) factor for assessing spectral fragmentation in flexible grid optical networks,” *2014 OptoElectronics and Communication Conference and Australian Conference on Optical Fibre Technology*, p. 357–358, 2014.

[16] P. Lechowicz, M. Tornatore, A. Wlodarczyk et K. Walkowiak, “Fragmentation metrics and fragmentation-aware algorithm for spectrally/spatially flexible optical networks,” *IEEE/OSA Journal of Optical Communications and Networking*, vol. 12, n°. 5, p. 133–145, 2020.

[17] A. Rosa, C. Cavdar, S. Carvalho, J. Costa et L. Wosinska, “Spectrum allocation policy modeling for elastic optical networks,” *2012 9th International Conference on High Capacity Optical Networks and Enabling Technologies, HONET 2012*, p. 242–246, 12 2012.

[18] W. Fadini, B. C. Chatterjee et E. Oki, “A subcarrier-slot partition scheme with first-last fit spectrum allocation for elastic optical networks,” vol. 91, n°. C, 2015. [En ligne]. Disponible : https://doi.org/10.1016/j.comnet.2015.08.048

[19] L. Liu, Z. Zhu et S. J. Ben Yoo, “3d elastic optical networks in temporal, spectral, and spatial domains with fragmentation-aware rssma algorithms,” dans *2014 The European Conference on Optical Communication (ECOC)*, 2014, p. 1–3.

[20] J. Sócrates-Dantas, R. M. Silveira, D. Careglio, J. R. Amazonas, J. Solè-Pareta et W. V. Ruggiero, “A study in current dynamic fragmentation-aware rsa algorithms,” dans *2014 16th International Conference on Transparent Optical Networks (ICTON)*, 2014, p. 1–4.

[21] Y. Wang, T. H. Cheng et M. H. Lim, “A tabu search algorithm for static routing and wavelength assignment problem,” *IEEE Communications Letters*, vol. 9, n°. 9, p. 841–843, 2005.

[22] X. Wu, S. Yan, X. Wan et Z. Lü, “Multi-neighborhood based iterated tabu search for routing and wavelength assignment problem,” *Journal of Combinatorial Optimization*, vol. 32, 2016.

[23] C. Dzongang, P. Galinier et S. Pierre, “A tabu search heuristic for the routing and wavelength assignment problem in optical networks,” *IEEE Communications Letters*, vol. 9, n°. 5, p. 426–428, 2005.