THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**
Spécialité : Réseaux, télécom, système et architecture

Présentée et soutenue par **Thomas FERRANDIZ**
le **2 mars 2012**

Maîtrise des latences de communication dans les réseaux bord SpaceWire

---

JURY

M. Éric Fleury, président
M. Thierry Divoux, rapporteur
M. Christian Fraboul, directeur de thèse
M. Fabrice Frances, co-encadrant
M. David Jameux
M. Jean-Jacques Lesage, rapporteur
M. Gilles Moury
M. Xavier Olive

École doctorale : **Mathématiques, informatique et télécommunications de Toulouse**

Unité de recherche : **Équipe d'accueil ISAE-ONERA MOIS**

Directeur de thèse : **M. Christian Fraboul**

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Les réseaux bord satellite

Les satellites d'observation doivent répondre à des besoins de complexité croissante. Ceci se reflète dans leur architecture de traitement des données qui doit transporter des quantités de données en continuelle augmentation. Le trafic transporté par un réseau bord satellite peut être divisé en deux catégories de caractéristiques opposées. D'un côté, le trafic commande/contrôle, utilisé pour la gestion des équipements du satellite, génère un faible volume de données mais est soumis à des contraintes temporelles strictes dont le respect est nécessaire au bon fonctionnement du satellite. Par exemple, les messages envoyés par les capteurs déterminant l'altitude ou la position du satellite doivent être envoyés régulièrement et avec un délai borné au calculateur qui dirige le satellite. Autrement, celui-ci ne serait pas en mesure d'assurer le guidage du satellite. D'un autre côté, le trafic scientifique (ou charge utile) génère un volume de données très important mais ses données ne sont pas critiques. Elles ne sont donc pas soumises à des contraintes temporelles strictes. Par contre, le réseau doit être capable d'assurer un débit moyen suffisant aux différents flux de données scientifiques.

Un réseau bord satellite doit permettre de transporter ces deux types de données tout en respectant leurs contraintes propres. Dans le cas d'un satellite générant un trafic total faible, le bus le bus utilisé est le MIL-STD-1553B qui fournit des communications de type maître/esclave sur un bus série. Sur un bus de ce type, un contrôleur gère les connexions avec et entre des terminaux distants. Les terminaux ne peuvent émettre qu'en réponse à une commande du contrôleur ce qui assure un schéma de communication déterministe et simple à analyser. Le débit fourni par ce bus est de l'ordre de 1 Mbps et n'est plus suffisant pour assurer les transferts de données entre terminaux.

Des liens point-à-point à haut débit ont donc été ajoutés entre les terminaux pour assurer le transport de grandes quantités de données. On peut par exemple installer un lien point-à-point entre un capteur et une mémoire de masse ou entre cette mémoire et un émetteur air-sol. Le trafic commande/contrôle, qui ne nécessite qu'un faible débit, est quant à lui toujours transféré à l'aide d'un bus MIL-STD-1553B. Ce type de configuration reste facilement analysable car les deux types de trafic sont physiquement séparés. En revanche, lorsque le nombre de terminaux et donc de liaisons point-à-point augmente, l'architecture réseau tend à se rapprocher d'un réseau maillé et devient très complexe et coûteuse.

Une architecture semblable à celle utilisée dans les réseaux sol serait donc plus

appropriée pour accompagner l'évolution future des satellites. En effet, une telle architecture permet de réduire le nombre de liaisons et de simplifier la topologie du réseau en partageant les liens réseaux entre différents flux de données. Par contre, elle implique de transmettre les trafics charge utile et commande/contrôle sur les mêmes liens. Contrairement à une architecture basée sur un bus MIL-STD-1553B, une architecture routée ne garantit donc pas de délais de transmission déterministes pour le trafic commande/contrôle. Ces délais peuvent varier en fonction des conflits d'accès aux ressources réseau et des ralentissements qu'ils engendrent. Avant de pouvoir déployer une architecture routée dans un satellite, il est donc nécessaire de mettre au point un outil permettant aux concepteurs réseaux de contrôler que les contraintes temporelles des différents messages sont vérifiées.

1.2 Le standard SpaceWire

Le standard SpaceWire a été conçu par l'ESA pour servir de réseau bord dans ses futurs satellites. Il s'agit d'un standard de réseau embarqué basé sur des liens point-à-point, bidirectionnels, full-duplex à haut débit (jusqu'à 200 Mbps). Il comprend également des routeurs qui permettent d'interconnecter les équipements suivant une topologie arbitraire. De plus, SpaceWire a été conçu en fonction de contraintes spécifiques au contexte spatial, notamment en termes de résistance aux radiations. Enfin, un objectif majeur de la création de SpaceWire est de disposer d'une interface réseau standardisée qui permette de développer des composants génériques réutilisables pour plusieurs missions. Cela permettra de réduire les coûts de développement des satellites. Nous présenterons plus en détail le standard SpaceWire dans la première partie du chapitre 2.

Sa principale caractéristique est la technologie de routage retenue : le routage wormhole. Le principe est le suivant : au lieu de stocker des paquets complets comme dans un réseau de type Store and Forward, un routeur wormhole détermine le routage d'un paquet uniquement à partir de son premier caractère et le transmet au fur et à mesure de son arrivée sans jamais avoir à stocker plus d'un caractère du paquet.

Le routage wormhole présente plusieurs avantages dans le cadre d'un réseau bord satellite :

- sa consommation mémoire est faible. Il s'agit d'un critère majeur car les mémoires résistantes aux radiations sont extrêmement coûteuses ;
- il est simple à implémenter dans un FPGA ;
- grâce au routage wormhole, les routeurs transfèrent les paquets en très peu de temps ;
- il reste compatible avec les liens point-à-point préexistants.

Le routage wormhole présente un défaut majeur dans un contexte de transmission de trafic critique. En effet, lorsqu'un paquet utilise le port de sortie d'un routeur, celui-ci ne peut transmettre aucune autre donnée tant que le paquet n'a pas été intégralement transmis. Si un autre paquet a besoin d'utiliser ce même port, il devra attendre la fin de la transmission précédente. La progression de ce paquet

dans le réseau est ainsi stoppée. Ce paquet va à son tour bloquer les ports de sortie des routeurs traversés en amont et remplir les mémoires tampon (buffers) des ports d'entrée jusqu'à la fin de sa transmission.

Ainsi, les délais de blocage peuvent s'additionner de proche en proche jusqu'à devenir très importants. En revanche, lorsqu'un paquet ne subit pas de blocage, son délai de livraison est très faible. Ceci rend les délais de livraison des paquets très variables et donc très difficiles à prévoir. Dans notre contexte, ce problème est accentué par le fait que l'on souhaite transmettre sur un même réseau à la fois des messages critiques et des paquets charge utile. Ceux-ci sont généralement de taille importante ce qui conduit à des blocages encore plus longs.

SpaceWire n'a pas été conçu spécifiquement pour transmettre du trafic critique et ne fournit aucun des mécanismes existants qui permettent d'améliorer le déterminisme d'autres types de réseaux wormhole. Par exemple, certains réseaux utilisent un mécanisme dit de canaux virtuels pour réduire les variations de délais. Ce mécanisme repose sur l'existence de plusieurs buffers séparés, associés à chaque port d'entrée. Il est alors possible de transmettre plusieurs paquets en parallèle. Ainsi, lorsqu'un paquet est bloqué, d'autres peuvent continuer à utiliser le port de sortie. De plus, les canaux virtuels permettent d'implémenter un système de priorités préemptives qui permet aux routeurs d'interrompre la transmission d'un message non urgent pour laisser passer un message urgent. Ce type de mécanisme réduit les incertitudes pesant sur les délais de livraison et facilite donc l'analyse du réseau. Cependant, SpaceWire n'inclut pas de mécanismes de ce type pour le moment.

Une extension temps-réel du standard est en cours de développement à l'ESA mais, à ce jour, elle n'a pas encore été finalisée. Les équipements compatibles ne seront donc pas disponibles avant plusieurs années. Nous présenterons cette norme et ses évolutions successives dans la seconde partie du chapitre 2.

1.3 Méthodes d'analyse des délais de bout en bout

On voit ainsi la nécessité de concevoir une méthode d'analyse spécifique au SpaceWire qui permette de garantir le respect des contraintes temporelles pour tous les flux critiques. Une première piste est de créer un simulateur reproduisant le comportement du réseau et de mesurer les délais de livraison des différents paquets. Cependant, si elle permet de mieux comprendre le comportement du réseau, cette approche ne permet pas de garantir le respect des contraintes temporelles de façon générale. Sur un réseau complexe, il est en effet impossible d'être certain d'avoir observé le pire scénario possible. Cette approche, explorée par Thales Alenia Space hors du cadre de cette thèse, a mené à la réalisation du simulateur MOST. Ce simulateur, considéré comme représentatif du comportement du réseau par l'industriel, sera utilisé comme référence pour évaluer les méthodes de calcul proposées dans la thèse.

Une approche analytique nous a donc paru préférable. Etant donné les importantes variations possibles sur le délai de livraison d'un paquet individuel, il nous a

semblé pertinent de rechercher seulement une borne supérieure du délai de livraison pire cas d'un paquet. Si cette borne est inférieure au délai maximum de livraison acceptable, nous avons la certitude que tous les paquets seront livrés à temps sans avoir besoin de connaître leur délai de livraison réel.

Depuis la création des réseaux de type wormhole, de nombreuses études de leurs performances temporelles ont été publiées. Ces études ont été menées dans des contextes variés tels que les supercalculateurs ou les Networks-on-Chip. Globalement, comme nous le montrerons au chapitre 3 aucune de ces études n'est appropriée à l'analyse d'un réseau SpaceWire, en particulier le cas d'étude fourni par Thales Alenia Space. Nous avons donc décidé de concevoir une méthode d'analyse des délais pire-cas adaptée au contexte des réseaux SpaceWire. Au cours de cette thèse, nous avons adopté deux approches différentes dans la recherche d'une telle méthode analytique.

La première, présentée dans le chapitre 4, se base sur des hypothèses faibles sur le trafic d'entrée. Ainsi, il n'est pas nécessaire de connaître précisément la quantité de trafic que chaque terminal cherche à émettre. En particulier, nous ne faisons pas d'hypothèses sur la périodicité des flux de données injectés dans le réseau. Cette approche nous a permis de mettre au point deux méthodes de calcul distinctes avec des hypothèses complémentaires. La première méthode calcule récursivement le délai de transmission pire-cas d'un paquet en supposant que les paquets d'un même flux n'interfèrent pas entre eux. La seconde méthode utilise également un calcul récursif mais en faisant l'hypothèse qu'au début de la transmission du paquet étudié, tout le réseau est saturé de paquets émis auparavant par les autres flux. Cette méthode permet ainsi d'être certain d'obtenir une borne pire-cas absolue quel que soit le trafic injecté par les différents flux. Ces deux méthodes fournissent des résultats intéressants mais ne permettent pas l'étude complète du cas d'étude fourni par Thales Alenia Space.

Le chapitre 5 présente la seconde approche qui utilise des hypothèses plus fortes afin d'améliorer la borne calculée. Cette approche, basée sur la théorie du Calcul Réseau nécessite de caractériser de façon plus stricte le trafic injecté dans le réseau, par exemple en définissant la périodicité des flux. Par contre, cette méthode permet de couvrir complètement le réseau de référence de TAS et fournit, en général, des bornes plus serrées que les méthodes avec hypothèses faibles.

Nous présenterons au chapitre 6 une comparaison de ces deux approches et des résultats fournis par MOST sur le cas d'étude Thales Alenia Space. Puis nous comparerons nos trois méthodes de calcul sur plusieurs autres architectures réseau afin de déterminer dans quels cas chaque approche est la plus appropriée.

Enfin, au chapitre 7 nous conclurons et présenterons les perspectives de la thèse.

Chapitre 2

SpaceWire et SpaceWire-RT

Sommaire

Ce chapitre est organisée en trois parties. La première partie présente le standard SpaceWire de façon assez détaillée. La seconde décrit les limites de ce standard dans le contexte d'un réseau temps-réel. Enfin, la troisième partie présente SpaceWire-RT, une extension temps-réel de SpaceWire en cours de standardisation.

2.1 SpaceWire

SpaceWire [1],[2] est un réseau de communication conçu pour relier les différents équipements embarqués dans un satellite. Il fournit des liens série point à point bi-directionnel et full-duplex à haut débit (de 2 Mbps à 400 Mbps). SpaceWire définit également des routeurs permettant d'établir des topologies complexes interconnectant les différents noeuds. Il dérive du standard IEEE 1355 [3] mais a été adapté à une utilisation dans le cadre d'applications spatiales. SpaceWire est formé de six niveaux (voir Tableau 2.1) qui correspondent au trois premières couches OSI. Nous allons les présenter maintenant.

Réseau
Paquet
Echange
Caractère
Signal
Physique

TABLE 2.1 – Les six niveaux définis par le standard

2.1.1 Niveau physique et niveau signal

SpaceWire utilise le signal LVDS (Low Voltage Differential Signalling) pour transporter les données. LVDS permet d'avoir des transmissions à haute vitesse avec un faible voltage sur des distances pouvant atteindre 10m ce qui est suffisant pour connecter les équipements au sein d'un satellite. Étant différentiel, LVDS nécessite l'utilisation d'une paire de câbles pour chaque signal à transmettre.

De plus, SpaceWire utilise l'encodage Data Strobe qui fournit une bonne tolérance au décalage d'horloge. Cet encodage utilise deux signaux (voir Figure 2.1) : le signal D transmet les données directement tandis que le signal S change de valeur lorsque le signal D reste constant sur deux bits consécutifs. Il suffit ensuite de calculer le XOR des deux signaux pour obtenir l'horloge du lien. Un lien SpaceWire utilise ainsi 2 paires de fils dans chaque direction soit 4 paires au total puisque les liens sont full-duplex.

Le niveau physique définit aussi les exigences de résistance électromagnétique que doivent satisfaire les câbles SpaceWire.

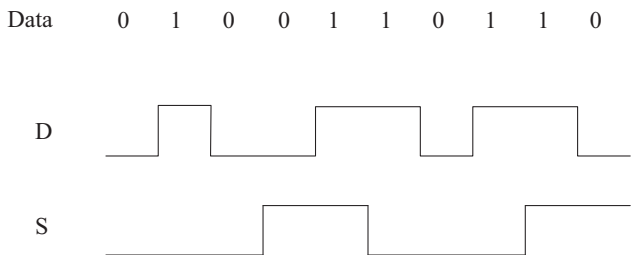


FIGURE 2.1 – Chronogramme de l'encodage utilisé par SpaceWire

2.1.2 Niveau caractère

Les caractères SpaceWire se répartissent en deux catégories : données et contrôle.

Un caractère de données comprend 10 bits : un bit de parité, un flag données/contrôle (mis à 0) et 8 bits de données.

Un caractère de contrôle comprend 4 bits : un bit de parité, un flag données/contrôle (mis à 1) et 2 bits de données. Les quatre caractères de contrôle sont présentés dans le Tableau 2.2.

Il existe aussi deux codes de contrôle, formés d'un caractère ESC et d'un autre caractère de données ou de contrôle. Le code NULL (ESC+FCT) est utilisé pour maintenir une connexion lorsqu'il n'y a pas de données à transmettre. Ainsi, lorsqu'une interface n'a pas de données à émettre, elle émet des caractères NULL à intervalles réguliers pour maintenir la connexion ouverte. Les Time-Codes, formé d'un caractère ESC et d'un caractère de données permettent de synchroniser toutes les horloges du réseau automatiquement comme nous le verrons dans la section suivante.

P	1	0	0	**FCT** Flow Control Token
P	1	0	1	**EOP** Normal End Of Packet
P	1	1	0	**EEP** Error End of Packet
P	1	1	1	**ESC** Échappement

TABLE 2.2 – Les différents codes de contrôle SpaceWire

2.1.3 Niveau Échange

Le niveau Échange est chargé de l'établissement d'une connexion sur un lien SpaceWire point-à-point ainsi que de sa gestion. Pour cela, les interfaces SpaceWire utilisent certains caractères de contrôle : FCT, ESC, NULL ainsi que les Time-Codes. Les autres caractères (données et marqueurs de fin de paquet) sont transmis par les interfaces aux systèmes hôtes.

2.1.3.1 Initialisation des liens

La première fonction assurée au niveau Échange est l'initialisation des liens SpaceWire. Un lien peut être désactivé soit après une erreur sur ce lien soit parce qu'un des hôtes a désactivé son interface. Au début de l'initialisation, chaque interface désactive son récepteur et son émetteur. Au bout de 6,4 μs, elles activent leur récepteur puis 12,8 μs après, leur émetteur. Les interfaces sont alors prêtes à être activées par leurs systèmes hôtes. Lorsqu'une interface est activée, elle émet un caractère NULL et attend d'en recevoir un de l'autre interface. Si elle le reçoit dans un délai de 12,8 μs, elle émet ensuite un caractère FCT et attend à nouveau 12,8 μs d'en recevoir un. Si elle le reçoit, c'est le signe que l'autre interface est également activée et l'interface est alors prête à émettre les données transmises par l'hôte. Si l'un des deux caractères n'est pas reçu avant la fin du time-out, l'interface rebascule dans l'état *Reset*. Il est à noter que pendant toute l'initialisation, l'interface fonctionne à sa vitesse minimale (2 Mbps). Ce n'est qu'une fois que l'interface est active que l'hôte peut modifier sa vitesse.

2.1.3.2 Contrôle de flux

Les interfaces SpaceWire assurent également un contrôle de flux propre à chaque lien afin d'éviter des pertes de données si la mémoire tampon du récepteur venait à déborder. Le contrôle de flux SpaceWire est basé sur l'envoi d'autorisation d'émission par le récepteur à l'émetteur. Ainsi, à chaque fois qu'il dispose de la mémoire nécessaire pour stocker 8 octets de données, le récepteur peut envoyer un caractère FCT à l'émetteur pour autoriser celui-ci à lui envoyer 8 nouveaux caractères. Un maximum de 56 octets (soit 7 FCT) peuvent ainsi être autorisés simultanément par le récepteur. Ce mécanisme de contrôle permet de s'assurer que la mémoire du récepteur ne débordera pas au prix d'un léger encombrement supplémentaire du lien en sens inverse de la communication.

Pour que ce mécanisme fonctionne, il est nécessaire que les caractères FCT puissent être émis n'importe quand par l'interface, même au cours de la transmission d'un paquet de données. Ceci est possible puisque les caractères de contrôle se différencient des caractères de données grâce à leur flag. De plus, les caractères sont émis par l'interface selon des règles de priorité, qui sont détaillées dans le Tableau 2.3.

1	Time-Code, plus haute priorité
2	FCT
3	Données et fin de paquet
4	NULL, plus basse priorité

Table 2.3 – Les priorités respectives des différents caractères

Ces priorités étant appliquées à chaque émission d'un caractère, les Time-Codes et les FCT peuvent être émis pendant l'émission d'un paquet de données. Ceci leur

permet d'avoir une latence faible lors de leur émission. Par ailleurs, lorsqu'une interface n'a aucun caractère à émettre, elle émet des caractères NULL pour maintenir la connexion. En effet, lorsqu'une interface ne reçoit aucun caractère pendant une durée supérieure à 850 ns, cela est considéré comme une erreur du lien et celui-ci est réinitialisé.

2.1.3.3 Les Time-Codes

L'interface SpaceWire gère également la réception et la distribution des Time-Codes. Le mécanisme des Time-Codes permet de disposer d'une horloge commune à tous les nœuds en synchronisant périodiquement dans chaque nœud et chaque routeur un compteur sur 6 bits avec celui d'un nœud maître. Comme on l'a vu à la section précédente, un Time-Code contient un caractère ESC et un caractère de données. Dans ce caractère, les deux bits de poids fort sont utilisés pour transmettre un code de contrôle à tous les nœuds tandis que les 6 bits de poids faible contiennent la nouvelle valeur du compteur qui va être diffusée à tous les nœuds. Pour l'instant, une seule valeur du code de contrôle est utilisée, 00, et représente un Time-Code standard. Les autres sont réservées pour une utilisation future.

La propagation d'un Time-Code se fait de la façon suivante. Une seule interface SpaceWire (l'interface maître) est à l'origine de la propagation des nouvelles valeurs. L'hôte reliée à cette interface incrémente périodiquement son compteur puis émet la nouvelle valeur dans un Time-Code. Lorsqu'une interface esclave reçoit un Time-Code, elle prévient le système hôte et lui transmet la valeur du Time-Code. Il y a alors plusieurs cas possibles.

Si la nouvelle valeur est égale à l'ancienne plus un (modulo 64), les deux systèmes (maître et esclave) sont synchrones. Le système met alors à jour son compteur et émet un Time-Code avec la nouvelle valeur sur toutes ses autres interfaces (notamment lorsqu'il s'agit d'un routeur). Pour éviter que les Time-Codes ne se propagent indéfiniment en boucle, lorsqu'un hôte reçoit un Time-Code contenant la même valeur que son propre compteur, il le considère comme redondant et ne le répercute pas sur les autres interfaces. Enfin, si un hôte reçoit un Time-Code avec une valeur incorrecte, c'est-à-dire qui ne soit ni égale à son compteur ni immédiatement supérieure, on considère qu'un ou plusieurs Time-Codes ont été perdu avant la réception de celui-ci. Dans ce cas, l'hôte met à jour son compteur mais ne retransmet pas le Time-Code.

SpaceWire fournit donc un moyen de synchroniser facilement toutes les horloges d'un réseau afin, par exemple, d'établir des communications synchrones. De plus, ce mécanisme n'induit qu'un faible décalage entre les différentes horloges car les Time-Codes ont la priorité la plus élevée et peuvent être entrelacés avec les paquets de données.

FIGURE 2.2 – Format d'un paquet SpaceWire

2.1.4 Niveaux Paquet et Réseau

2.1.4.1 Format des paquets

Les paquets SpaceWire ont un format extrêmement simple (voir Figure 2.2). Un paquet contient d'abord une adresse qui peut être de longueur quelconque (les différents modes d'adressage sont décrits un peu plus loin), puis l'ensemble des données à transporter qui peuvent là encore être de taille quelconque puis un marqueur de fin de paquet (EOP ou EEP). Dans le cas d'une connexion point-à-point, il est donc possible d'employer un paquet de taille infinie pour transmettre un flux de données entre deux équipements. Il est à noter que bien que les paquets SpaceWire aient un format très simple, les protocoles de niveau supérieur comme RMAP [4] définissent des formats plus détaillés pour les paquets.

2.1.4.2 Modes d'adressage

Avant de transmettre un paquet, le routeur doit déterminer quel port il doit utiliser. Il existe pour cela deux modes d'adressage dans SpaceWire.

Le premier est l'adressage physique. Dans ce cas, le routage est effectué par la source. Pour chaque routeur traversé, celle-ci fournit le numéro du port à utiliser pour transmettre le paquet. Le champ d'adresse du paquet SpaceWire est donc une suite d'octets correspondant au numéro de port à utiliser dans les routeurs successifs traversés par le paquet. Chaque routeur traversé lit le premier octet du paquet pour savoir par où le transmettre puis supprime cet octet. Ainsi, de proche en proche, chaque routeur utilise le premier octet du paquet qu'il reçoit pour effectuer le routage.

Le second mode d'adressage présent dans SpaceWire est l'adressage logique. Dans ce mode, chaque nœud se voit attribuer une (ou plusieurs) adresse(s) codée(s) sur un octet. Chaque routeur dispose ensuite d'une table de routage qui associe chaque adresse logique au port de sortie à utiliser pour la rejoindre. Pour réduire la taille de la table d'adressage, il est également possible d'associer un intervalle d'adresses à un port de sortie.

2.1.4.3 Le Wormhole Routing

Dans les cas où des connexions point-à-point se révèlent insuffisantes, SpaceWire permet d'établir des topologies plus complexes grâce à des routeurs interconnectant les différents systèmes hôtes. La méthode de routage utilisée par les routeurs SpaceWire est le Wormhole Routing. Cette technique présente plusieurs avantages pour une utilisation dans un réseau bord satellite. Premièrement, elle permet d'avoir des

latences de communication faibles la plupart du temps et est implémentable de façon simple et peu coûteuse dans un FPGA. Deuxièmement, elle est compatible avec les liens point-à-point SpaceWire déjà déployés et avec le fonctionnement du contrôle de flux. Enfin, elle ne nécessite que très peu de mémoire dans les routeurs. En général, les routeurs SpaceWire n'utilisent que 64 octets de mémoire par port d'entrée et aucune mémoire pour les ports de sortie.

Le routage wormhole fonctionne de la façon suivante. Lorsque le premier octet d'un paquet arrive sur une interface du routeur, celui-ci détermine le port de sortie à utiliser puis regarde l'état de ce port. Si celui-ci est libre, le routeur le marque comme occupé puis commence à transmettre les caractères du paquet vers ce port au fur et à mesure de leur arrivée. Tant que le transfert du paquet n'est pas terminé, aucun autre paquet ne peut être envoyé sur ce port. Une fois le paquet complètement transféré, le port est libéré et peut être utilisé pour un autre paquet.

En revanche, si le port de sortie est déjà occupé par le transfert d'un autre paquet, le nouveau paquet doit attendre la fin du transfert du paquet en cours et reste bloqué. Le paquet ne peut pas être stocké en mémoire par le routeur mais grâce au mécanisme de contrôle de flux présenté à la Section 2.1.3, cela ne pose pas de problèmes. En effet, si les caractères ne peuvent pas sortir du routeur, le buffer de réception du port d'entrée va se remplir très rapidement et le port va cesser d'émettre des jetons FCT. La source va ainsi cesser d'émettre automatiquement. S'il s'agissait d'un routeur, il va lui-même bloquer son port d'entrée et ainsi de suite jusqu'à la source du paquet.

Ainsi, les caractères d'un paquet peuvent être répartis dans les buffers d'entrée de plusieurs routeurs successifs. Le paquet bloque également l'accès aux ports de sortie dans chaque routeur qu'il utilise pendant toute la durée du blocage. Ce paquet peut à son tour bloquer d'autres routeurs qui vont eux même bloquer des ports de sortie et éventuellement d'autres paquets... Comme on le voit, si trop de paquets sont émis en même temps, on risque d'aboutir à un effondrement complet du réseau.

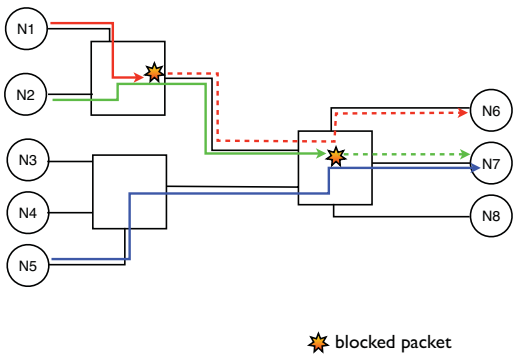


FIGURE 2.3 – Exemple de blocages dans un réseau wormhole

2.1.4.4 Mécanismes d'arbitrage

Comme on l'a vu, il est possible qu'un paquet reste bloqué lorsque le port de sortie est déjà utilisé par un autre paquet. Lorsque plusieurs paquets se retrouvent bloqués sur le même port de sortie, il devient nécessaire d'utiliser un mécanisme d'arbitrage pour déterminer l'ordre dans lequel ils pourront être transmis par ce port.

Le premier mécanisme mis en place dans SpaceWire est un système de priorités entre les paquets basé sur leurs adresses de destination. Ce mécanisme nécessite donc l'utilisation des adresses logiques. Dans la table de routage, chaque adresse est associée à un numéro de port et à une priorité. Il n'existe que deux priorités sur les adresses : haute et basse. La priorité ne dépendant que de l'adresse, il est possible d'attribuer deux adresses à un nœud, une de priorité haute et une de priorité basse, afin de mieux gérer les différents types de trafic. Ce système de priorité n'est pas préemptif : lorsqu'un paquet de priorité basse utilise un port, il ne peut pas être interrompu, même par un paquet de priorité haute. Les priorités s'appliquent donc uniquement lorsqu'un paquet de priorité basse et un paquet de priorité haute attendent tous deux la libération d'un même port, en cours d'utilisation par un troisième paquet. Dans ce cas, le routeur transmettra toujours le paquet de priorité haute avant celui de priorité basse (priorités statiques).

Il reste à résoudre le cas où deux paquets de même priorité attendent tous les deux la libération d'un même port. Le standard ECSS-E50-12-C [2] spécifie trois politiques d'arbitrage possible : avec priorités (voir ci-dessus), round-robin et aléatoire. La politique d'arbitrage la plus répandue semblant être le round-robin, nous nous y intéresserons plus particulièrement dans notre étude. Cette politique ne correspond pas exactement à une politique de round-robin au sens habituel. Il s'agit en réalité d'une version simplifiée qui permet d'économiser la mémoire utilisée et qui fonctionne de la façon suivante. Chaque port de sortie garde en mémoire le numéro du dernier port d'entrée qui l'a utilisé. Lors de l'arbitrage suivant, le routeur considérera le port de numéro immédiatement supérieur (modulo le nombre de ports) comme le plus prioritaire. Cette politique permet d'assurer que chaque port d'entrée du routeur reçoit un accès équitable à chaque port de sortie du routeur en terme de nombre de paquets transférés. En revanche, étant donné que la transmission d'un paquet ne peut être interrompue quelle que soit sa durée, l'arbitrage ne garantit pas un accès équitable aux ports de sortie en terme de durée d'accès.

SpaceWire fournit une dernière fonctionnalité qui peut être utile pour réduire les problèmes de blocage. Il s'agit du Routage Adaptatif de Groupe (*Group Adaptative Routing* ou GAR). Cette fonctionnalité permet de remplacer un lien entre deux routeurs par un groupe de liens. Une fois ces liens déclarés comme un groupe dans la table de routage, le routeur est capable de répartir automatiquement les paquets sur les différents liens du groupe. Cette fonctionnalité doit permettre de mieux utiliser la bande passante disponible au prix d'une augmentation du nombre de câbles réseau.

2.2 SpaceWire et temps réel

Plusieurs satellites utilisent déjà des liens point-à-point SpaceWire pour transporter du trafic charge utile. Par exemple, les missions Swift et Lunar Reconnaissance Orbiter utilisent SpaceWire ([5]). Cependant, le but de l'Agence Spatiale Européenne est de pouvoir utiliser un réseau SpaceWire comme réseau bord unique dans un satellite. Un même réseau serait donc utilisé à la fois pour le trafic charge utile et pour le trafic commande/contrôle.

Le trafic charge utile, ou trafic scientifique, est constitué par les flux de données générés par les instruments d'observation du satellite. Il peut s'agir par exemple d'un flux entre le capteur et une mémoire de masse, ou entre la mémoire et un émetteur vers la Terre. Ce type de trafic se caractérise par des paquets de grande taille, un débit élevé mais peu de contraintes sur la latence des paquets.

Le trafic commande/contrôle quant à lui est constitué par les messages de gestion du satellite échangés par les différents équipements. Il peut s'agir par exemple des commandes transmises par un contrôleur central ou des messages périodiques de monitoring envoyés par les instruments d'observation. Ce trafic présente des caractéristiques opposées à celle du trafic charge utile. En général, les messages sont de faible taille et ne requièrent qu'un débit réduit. En revanche, ils doivent absolument être livrés en temps et en heure pour assurer le bon fonctionnement du satellite.

A l'heure actuelle, la bande passante offerte par les liens SpaceWire est suffisante pour transmettre tout le trafic d'un satellite sur un seul réseau. Le problème est donc de pouvoir garantir le respect des contraintes temporelles des messages commande/contrôle qui transitent par le réseau. En effet, SpaceWire a été conçu pour minimiser la quantité de mémoire utilisée dans les satellites mais pas pour fournir des garanties sur les délais de transmission.

D'une part, le routage wormhole peut entraîner des blocages récursifs au sein du réseau qui peuvent créer d'importantes variations sur les délais de transmission. Deux paquets d'un même flux peuvent ainsi avoir des délais de livraison variant d'un facteur 10 ou plus suivant les conflits subis par chacun d'eux dans le réseau. D'autre part, le fait de multiplexer sur un même lien les deux types de trafic renforce l'importance des variations car un message urgent de faible taille peut être bloqué par un paquet non-urgent de grande taille, lui même bloqué plus loin par un autre grand paquet.

En outre, SpaceWire n'a pas été conçu spécifiquement pour transmettre du trafic critique et ne fournit aucun des mécanismes qui permettent d'améliorer le déterminisme d'autres types de réseaux wormhole. Par exemple, certains réseaux utilisent un mécanisme dit de canaux virtuels [6] pour réduire les variations de délais. Ce mécanisme repose sur l'existence de plusieurs buffers séparés associés à chaque port d'entrée. Un lien doté de plusieurs canaux virtuels fonctionne de façon similaire à une route à plusieurs voies. Lorsque l'une d'entre elle est bloquée, les automobilistes peuvent continuer à avancer sur une voie parallèle. De même, grâce aux canaux virtuels, lorsqu'un paquet est bloqué, il ne bloque pas complètement tous les ports qu'il a traversé en amont car les autres buffers disponibles permettent à d'autres

paquets de continuer à avancer.

De plus, les canaux virtuels permettent d'implémenter un système de priorités préemptives qui permet aux routeurs d'interrompre la transmission d'un message non-urgent pour laisser passer un message urgent. Ce type de mécanisme réduit les incertitudes pesant sur les délais de livraison. Cependant, SpaceWire n'inclut pas de mécanismes de ce type pour le moment. Il est possible que les prochaines versions du standard en incluent mais elles ne sont pas finalisées et aucune implémentation matérielle ne sera disponible dans les prochaines années.

Le mécanisme de priorités non-préemptives implémenté par les routeurs SpaceWire ne permet pas de compenser l'absence de canaux virtuels. Il peut certes réduire dans certains cas le délai causé à un paquet urgent mais, comme un paquet peut toujours bloquer le port de sortie pendant qu'il est lui-même bloqué plus loin, des blocages longs risquent quand même de se produire. De même, le mécanisme de Group Adaptative Routing peut être utilisé pour fluidifier le trafic mais seulement au prix d'une augmentation du nombre de câbles SpaceWire, donc de la complexité du réseau et de son coût.

Exemple de calcul de délai

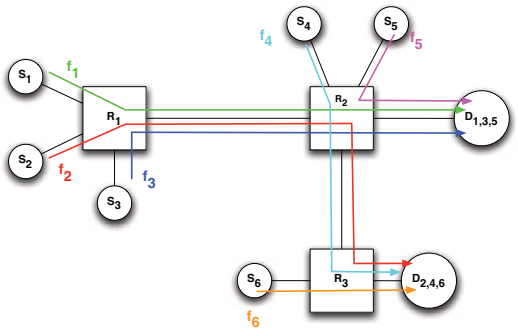


FIGURE 2.4 – Réseau SpaceWire

Considérons le réseau de la Figure 2.4. Ce réseau comprend 8 terminaux, 3 routeurs et 6 flux. Les liens ont tous la même capacité C. Chaque flux f_i émet des paquets de taille L_i. Afin de mieux appréhender les variations possibles sur le délai de livraison d'un flux, nous allons chercher le délai de livraison du flux f_1 dans le meilleur et le pire cas.

Dans le meilleur cas, un paquet p_1 du flux f_1 ne subit aucun conflit avec des paquets d'autres flux. Dans ce cas, étant donné que les routeurs transmettent les caractères au fur et à mesure de leur arrivée, le délai sera $d_1 = \frac{L_1}{C}$.

Dans le pire cas, p_1 ne peut pas accéder au lien entre R_1 et R_2 à cause d'autres paquets. Le pire scénario est qu'un paquet de f_2 et un paquet de f_3 passent avant p_1. Le paquet de f_2 risque lui-même d'être bloqué par un paquet de f_4 dans R_2. Ce paquet de f_4 va lui aussi être bloqué par un paquet de f_6 dans R_3. Le paquet de f_2 peut ensuite avancer jusqu'à R_3 où il peut être bloqué par un autre paquet de f_6.

Ensuite, le paquet de f_3 qui bloque p_1 dans R_1 peut être bloqué par un paquet de f_5 dans R_2 avant d'être livré. Quand tous ces paquets ont été livrés, f_1 peut avancer jusqu'à R_2. Le délai maximum subi dans R_1 est donc $\frac{L_2+L_3+L_4+L_5+2.L_6}{C}$. Dans R_2, p_1 peut être bloqué par un autre paquet de p_5 puis il est livré à $D_{1,3,5}$.

Son délai maximum au total est donc

$$D_1 = \frac{L_1 + L_2 + L_3 + L_4 + 2.L_5 + 2.L_6}{C}.$$

Prenons l'application numérique suivante :

Paramètre	Valeur
L_1, L_3, L_5	200 caractères
L_2, L_4, L_6	4000 caractères
C	20 Mbps

On a alors $d_1 = 100\ \mu s$ et $D_1 = 8400\ \mu s$ soit $D_1 = 84.d_1$. Les variations de délai peuvent donc être très importantes. De plus, tous les blocages détaillés ci-dessus ne vont pas se produire systématiquement pour chaque paquet de p_1. On voit donc également qu'il est très difficile de calculer le délai précis subi par un paquet donné d'un flux.

2.3 Ajout d'une surcouche temps-réel

Une première possibilité pour réduire l'incertitude sur les délais de livraison d'un paquet est de faire évoluer le standard pour lui ajouter les fonctionnalités nécessaires. Cette évolution a un coût important puisqu'elle oblige les industriels à modifier les équipements et à les tester à nouveau une fois le nouveau standard élaboré.

L'ESA a donc plutôt choisi de créer un nouveau standard permettant de garantir une certaine Qualité de Service aux différents flux sans remettre en cause le standard SpaceWire existant. Ce nouveau standard prend la forme d'une couche protocolaire supplémentaire qui s'intercale entre la couche SpaceWire de base et la couche applicative.

La première version de ce nouveau standard, baptisé SpaceWire-RT (pour Reliability & Timeliness) devait respecter plusieurs contraintes essentielles. En premier lieu, elle devait définir des classes de trafic permettant de garantir le respect de contraintes temporelles strictes (Timeliness). En second lieu, elle devait fournir toute une série de mécanismes assurant la fiabilité du réseau du point de vue de la livraison des paquets (Reliability). Enfin, cette surcouche ne devait impliquer aucune modification des routeurs SpaceWire existants. SpaceWire-RT ne devait être implémenté qu'au niveau des terminaux et ceci sans modification de l'interface applicative. En d'autres termes, une application conçue pour utiliser un port SpaceWire pour communiquer devait pouvoir utiliser la surcouche SpaceWire-RT sans aucune modification.

Nous allons maintenant présenter les principaux aspects de cette norme en nous basant sur le Draft A Issue 2.0 en date du 8 septembre 2008.

	Asynchrone	Synchrone
Non Fiable	Best Effort	Reserved
Fiable	Assured	Guaranteed

TABLE 2.4 – Les quatre classes de trafic de SpaceWire-RT

2.3.1 SpaceWire-RT

SpaceWire-RT organise les transferts de données sous forme de connexions virtuelles point-à-point entre applications distantes, nommées canaux virtuels. Les canaux virtuels définis par SpaceWire-RT sont différents des canaux virtuels habituels présentés précédemment. Plus précisément, les canaux virtuels de SpaceWire-RT implémentent le même principe que les canaux virtuels classiques mais debout en bout entre deux terminaux plutôt qu'entre deux routeurs sur un lien point-à-point.

Les canaux disposent de plusieurs paramètres réglables suivant le niveau de fiabilité que l'on souhaite obtenir (demande d'acquittements, réémission automatique en cas d'échec...) et l'importance de la communication (priorités). Un canal est constitué d'un buffer d'émission sur le nœud source, d'une suite de liens reliant le nœud source au nœud destination et d'un buffer de réception sur ce dernier. Les différents canaux sont identifiés par leur adresse source et un numéro de canal codé sur un octet. Il est possible d'avoir plusieurs canaux partant ou arrivant sur un nœud, y compris plusieurs canaux entre le même couple de noeuds. Cela permet de différencier les trafics puisque la classe de service est un paramètre associé à chaque canal.

L'unique interaction entre les applications et SpaceWire-RT a lieu au niveau des buffers. Une application source souhaitant émettre des données écrit dans le buffer source d'un canal (lorsqu'il n'est pas plein) et SpaceWire-RT se charge de transférer les données jusqu'au buffer destination où l'application distante pourra les récupérer.

SpaceWire-RT dispose de deux modes de fonctionnement : un mode asynchrone, qui ne fournit pas de garanties temporelles strictes mais utilise un système de priorités pour réguler le trafic ; et un mode synchrone dans lequel le respect strict de bornes temporelles est garanti.

Au total, le trafic est réparti en quatre classes de qualité de service, présentées dans le Tableau 2.4. Cette section est organisée de façon à présenter les différentes classes de service de la plus simple (Basic) à la plus complexe (Guaranteed) en introduisant au fur et à mesure les fonctionnalités nécessaires, fournies dans SpaceWire-RT.

2.3.1.1 La classe de trafic Best Effort

Cette classe de service est non fiable et sans garanties temporelles strictes. Cependant, la classe Best Effort garantit quand même que les données seront délivrées dans l'ordre et sans duplication. Par rapport au SpaceWire simple, elle ajoute plusieurs

mécanismes qui permettent de réduire l'incertitude sur les délais de transmission, sans toutefois fournir de garanties strictes. Ces mécanismes sont les suivants : un système de priorités entre les canaux de même source, la fragmentation des paquets en segments de 256 octets et un mécanisme de contrôle de flux de bout en bout. Nous allons les détailler un par un.

Le mécanisme de priorité entre les canaux partant d'un même nœud est indépendant des priorités SpaceWire associées aux adresses logiques mais peut être utilisé en conjonction avec celles-ci. Il s'agit de priorités statiques non-préemptives associées aux numéros des canaux. Ce mécanisme peut en particulier être utilisé pour établir plusieurs connexions entre deux nœuds avec une priorité différente pour chacune.

En second lieu, toutes les données envoyées par les applications à SpaceWire-RT sont fragmentées en UDS (User Data Segment) de taille réduite afin de permettre un meilleur multiplexage des paquets par les routeurs. Les UDS sont encapsulés dans des paquets SpaceWire nommés Data Packet (DP). Un UDS contient au maximum 256 octets de données. Les applications peuvent émettre des données de taille quelconque et les écrire dans le buffer source. Dès que le buffer contient 256 octets, le canal peut émettre un DP (si aucun canal plus prioritaire n'a de données à émettre, bien entendu). Si une application écrit un paquet complet de taille inférieure à 256 octets, il peut être émis immédiatement, sans attendre d'autres données pour compléter l'UDS.

Par rapport à SpaceWire, SpaceWire-RT nécessite un en-tête spécifique, ajouté à chaque DP. Il contient notamment l'adresse de la source et un numéro de canal ainsi qu'un numéro de séquence utilisé par les classes de trafic fiables.

Le dernier mécanisme utilisé en mode Best Effort est un contrôle de flux de bout en bout similaire au contrôle de flux qui a lieu sur chaque lien SpaceWire. En effet, le buffer source d'un canal ne peut émettre un DP que lorsque le buffer destination l'y a autorisé à l'aide d'un code de contrôle BFCT (pour Buffer Flow Control Token). Réciproquement, le buffer destination n'émet un code BFCT que lorsqu'il dispose de la place mémoire nécessaire pour stocker un DP entier.

Pour éviter un blocage en cas de perte d'un code BFCT, chaque code doit être acquitté par le buffer source à l'aide d'un code BACK. Les codes BFCT et BACK sont identifiés par leur numéro de séquence et il est donc possible d'envoyer plusieurs BFCT d'avance si la destination dispose d'assez de mémoire pour recevoir plusieurs DP.

Ce mécanisme permet ainsi d'éviter qu'un paquet ne soit bloqué à l'entrée du nœud destination par le contrôle de flux point-à-point standard de SpaceWire lorsque la destination n'est pas prête à le recevoir. Ainsi, il ne bloque en même temps les routeurs intermédiaires. En revanche, il ajoute un surcoût non négligeable aux communications.

Pour conclure, la classe de trafic Best Effort ne fournit de garanties ni sur la livraison des messages , ni sur le respect de bornes temporelles mais apporte tout de même des fonctionnalités non négligeables par rapport au SpaceWire de base. Ces fonctionnalités permettent de réduire l'incertitude sur les délais de livraison en réduisant la probabilité des blocages et leur durée.

2.3.1.2 La classe de trafic Assured

Par rapport à la classe Best Effort, la classe Assured ajoute la livraison fiable des messages mais pas de garanties temporelles. La fiabilité des livraisons est assurée par trois mécanismes complémentaires : l'acquittement des DP, leur réémission automatique et la redondance des routes reliant les nœuds. Si malgré tout un DP ne peut être délivré, SpaceWire-RT prévient alors l'application source au lieu d'échouer silencieusement comme pour les canaux de classe Best Effort.

Les acquittements utilisent le numéro de séquence présent dans les DP. Chaque DP émis par la source doit être acquitté par un code ACK portant le même numéro de séquence. Contrairement aux codes BFCT, les DP doivent tous être acquittés un par un. Cependant, le nombre de DP que la source peut émettre sans avoir reçu leurs acquittements est paramétrable.

Lorsque la réémission automatique est activée, les DP non acquittés sont réémis à l'identique avec le même numéro de séquence. Si aucune tentative de réémission sur le chemin principal ne fonctionne, SpaceWire-RT essaie alors d'émettre les DP en utilisant la route alternative pour le nœud destination (si elle est définie). Si, à nouveau, l'émission échoue, SpaceWire-RT rapporte l'échec de l'émission à l'application source.

La fonction de redondance comporte plusieurs variantes. La première, présentée plus haut, utilise une route principale et une route secondaire en cas d'échec sur la première route. Une seconde permet de définir deux routes vers la même destination sans préciser laquelle est prioritaire. Enfin, la dernière variante consiste à envoyer chaque DP sur les deux routes simultanément.

La classe Assured utilise également les priorités et le contrôle de flux de bout en bout de la même façon que la classe Best Effort. Les priorités sont d'ailleurs appliquées en même temps sur les canaux des deux classes.

Cette classe permet au final d'avoir des communications fiabilisées sans que les applications aient quoi que ce soit à gérer en dehors de l'échec complet de la communication.

2.3.1.3 La classe de trafic Reserved

Les classes de trafic Reserved et Guaranteed permettent de faire fonctionner tout un réseau SpaceWire-RT en mode synchrone. La classe Reserved fournit des garanties temporelles strictes mais pas de communications fiables.

En mode synchrone, SpaceWire-RT utilise un mécanisme TDMA (Time Division Multiple Access) pour gérer les trafics concurrents sur le réseau. La bande passante du réseau est alors divisée en slots temporels qui sont alloués aux différents canaux de façon à empêcher les conflits d'accès aux liens SpaceWire. De plus, les mécanismes de contrôle de flux et d'acquittements sont différents de ceux du mode asynchrone pour tenir compte des différences de fonctionnement.

Les slots temporels sont délimités grâce aux Time-Codes SpaceWire présentés à la section 2.1.3. Le temps est ainsi divisé en cycles de 64 slots chacun. L'utilisation des Time-Codes permet d'avoir un réseau complet synchronisé sur les mêmes cycles.

Dans SpaceWire-RT, l'allocation des slots est faite statiquement et chaque nœud contient une table d'allocation lui indiquant, pour chaque slot, quels canaux ont le droit d'émettre. L'allocation des liens est faite de manière à ce qu'aucun lien ne soit utilisé par plus d'un canal au cours d'un slot. Si le Routage Adaptatif de Groupe est utilisé, n canaux peuvent passer par un groupe de n liens au cours d'un slot. De plus, les liens SpaceWire sont full-duplex et peuvent donc être utilisé simultanément par deux canaux traversant le lien en sens contraires.

Au cours d'un slot, chaque canal ayant le droit d'émettre peut envoyer un ou plusieurs DP du moment que l'émission est terminée avant la fin du slot. Il est également possible d'utiliser les priorités en même temps que le TDMA. Un slot peut alors être alloué à plusieurs canaux ayant la même source et la même destination. Au début du slot, l'autorisation d'émettre est alors attribuée de la même façon qu'en mode asynchrone (priorités statiques). Puis, si le slot n'est pas fini, le canal suivant par ordre de priorité peut émettre à son tour et ainsi de suite.

En mode synchrone, il n'est pas possible de laisser les nœuds émettre librement des codes BFCT et BACK quand ils le veulent sous peine de perturber le trafic synchrone. Les codes BFCT vont donc être émis uniquement au début de chaque slot. De plus, comme ils sont émis de façon régulière, il n'est plus nécessaire de les acquitter. Enfin, pour réduire le trafic, la destination va regrouper tous les BFCT à destination d'une même source pour le slot qui commence dans un seul code Scheduled BFCT.

Le corps du code Scheduled BFCT est une suite de codes sur 2 bits décrivant le statut de chaque buffer de réception. Chaque code donne le nombre de DP que le buffer pourra recevoir au cours du slot. Un maximum de trois DP pourra donc être transféré sur un canal au cours d'un slot.

La classe de trafic Reserved permet donc d'obtenir des bornes sur les temps de temps de transmission en isolant totalement les différents canaux dans des slots temporels disjoints. Il ne peut donc plus se produire de blocages dans les routeurs SpaceWire ce qui permet d'avoir des délais de livraison déterministes.

2.3.1.4 La classe de trafic Guaranteed

Pour finir, la classe Guaranteed apporte à la fois des garanties temporelles et des communications fiables. Elle combine donc les mécanismes des classes Reserved et Assured.

Comme pour les jetons de contrôle de flux, il n'est pas possible de laisser les acquittements être émis n'importe quand. Pour tous les canaux entre une source et une destination, ils vont donc être regroupés dans un unique code Scheduled ACK qui sera envoyé au début du slot suivant l'émission. Ce code, en plus de l'en-tête habituel, est formé d'une suite de paires d'octets. Dans chaque paire, le premier octet code le numéro du canal et le second le numéro de séquence du dernier DP reçu.

Il est à noter que l'émission de l'acquittement a lieu au début du slot suivant quels que soient les canaux auxquels il a été attribué. Ceci est possible uniquement

si un délai est laissé au début de chaque slot pour l'émission des acquittements.

Si un DP n'est pas acquitté, il sera réémis au prochain slot attribué à son canal ainsi que tous les DP émis après lui dans le slot. Cette approche n'engendre pas un surcoût très important dans la mesure où un canal ne peut pas de toute façon pas émettre plus de trois DP par slot. Si les DP ne sont toujours pas délivrés après la deuxième tentative, une route redondante pourra être utilisée si elle existe (à nouveau, au slot suivant).

Structure d'un slot temporel Pour que les acquittements puissent être émis en dehors du slot attribué aux canaux qu'ils concernent, il est nécessaire d'avoir une plage de temps prévue au début de chaque slot pour la transmission des acquittements. Un délai est aussi nécessaire au début du slot pour laisser le temps aux sources du slots précédents de détruire les paquets encore en cours d'émission à la fin du slot. La destruction des DP se traduit par l'émission d'un caractère EEP (Error End of Packet) qui indique que le paquet se termine de façon prématurée. Le début du slot prévoit aussi une plage réservée à l'émission des codes Scheduled BFCT. Après ces trois phases de maintenance, la transmission des DP peut prendre place.

Un slot temporel a donc le format donné en Figure 2.5. Cette Figure donne également les durées recommandées pour les différents périodes du slot. La durée recommandée par le standard pour un time-slot est de 100 μs, soit un total de 6,4 ms pour un cycle complet. Avec le découpage illustré sur la Figure 2.5, la plage d'émission proprement dite dure 85 μs ce qui laisse le temps d'émettre six DP si la vitesse des liens est de 200 Mbps. Toutefois, les durées ne sont pas imposées et peuvent être modifiées en changeant le débit d'émission des Time-Codes.

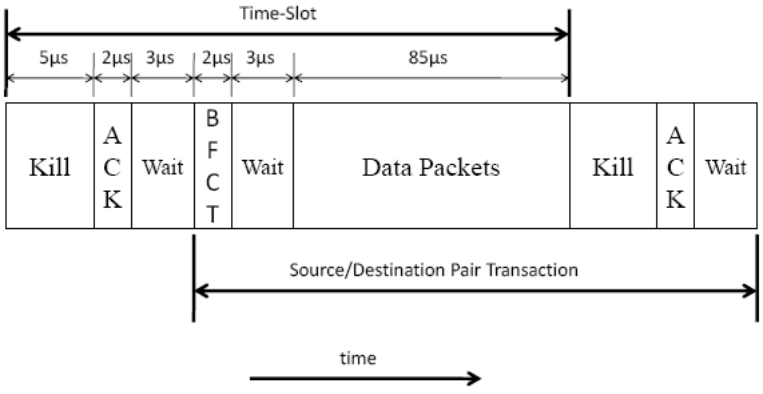


FIGURE 2.5 – Structure d'un slot temporel

2.3.1.5 Bilan sur SpaceWire-RT

– SpaceWire-RT apporte plusieurs fonctionnalités permettant de répondre au problème exposé en section 2.2. Tout d'abord, la fragmentation des paquets

en DP et l'ajout de priorités entre canaux permet d'améliorer le multiplexage des différents trafics et de réduire la durée de blocage pour l'accès à un lien utilisé par plusieurs flux. Le contrôle de flux de bout en bout permet également de réduire les blocages en s'assurant qu'aucun DP ne reste bloqué parce que sa destination ne peut pas le traiter. Ces différents éléments devraient faciliter l'utilisation d'un réseau unique avec plusieurs classes de trafics puisque les paquets commande/contrôle urgents pourront s'insérer à la source entre deux DP d'un paquet scientifique de grande taille. Ils devraient également simplifier le calcul de bornes sur les latences de communication. En outre, SpaceWire-RT rend possible une séparation complète des deux types de trafic grâce au mode synchrone. Dans ce cas, on dispose de délais de livraison facilement calculables pour tous les canaux.

SpaceWire-RT fournit également les mécanismes nécessaires pour fiabiliser les communications entre nœuds : acquittements, réémission et redondance automatique. Cependant, ces fonctions ne concernent pas l'aspect temps-réel des communications et ne nous intéressent que dans la mesure où elles peuvent influer sur les délais de livraisons des messages. En particulier, en mode asynchrone, le mécanisme des acquittements peut faire augmenter les latences réseau en obligeant la source à attendre les codes ACK de la destination avant de continuer à émettre. La redondance automatique peut également avoir un impact si deux routes utilisées pour une même destination n'entraînent pas les mêmes délais.

– Néanmoins, l'utilisation de SpaceWire-RT dans son état actuel pose un certain nombre de problèmes. Tout d'abord, l'utilisation de la couche de fragmentation implique d'avoir des buffers de réception de taille conséquente, notamment sur les nœuds recevant des données de plusieurs sources. En effet, chaque canal arrivant sur un nœud doit disposer d'un buffer de réception capable de recevoir un DP complet, c'est-à-dire 256 octets. Il en est de même pour les buffers sources des canaux qui doivent pouvoir contenir 256 octets de données. De plus, dans le cas d'une mémoire de masse, étant donné que plusieurs paquets peuvent être reçus simultanément et en plusieurs parties, il risque d'être impossible d'avoir des transactions atomiques. Ceci peut remettre en cause la compatibilité de SpaceWire-RT avec RMAP.

  Par ailleurs, le contrôle de flux de bout en bout permet d'éviter d'envoyer des paquets inutiles (car ne pouvant pas être traités sur le réseau) mais il alourdit aussi considérablement les communications sur un canal puisqu'il oblige à la fois la source et la destination à émettre un code de contrôle pour chaque DP émis. Pour réduire le nombre de codes de contrôle nécessaires, il faudrait que la destination autorise la source à émettre plusieurs DP d'un coup mais elle devrait pour cela disposer d'un buffer capable de tous les accueillir. Cela augmenterait encore significativement l'espace mémoire nécessaire aux buffers de réception.

  En mode synchrone, ce mécanisme entraîne moins de trafic puisqu'il est regroupé au début de chaque slot mais il implique d'avoir de la place pour trois

DP dans le buffer de réception si l'on veut utiliser au maximum la bande passante du slot disponible pour le canal. Réciproquement, il rend impossible d'utiliser intégralement la bande passante d'un slot en limitant à trois le nombre de DP qu'un canal peut émettre dans un slot.

– Le mode synchrone souffre également d'autres limitations. D'une part, il oblige à synchroniser l'intégralité du réseau sur le même découpage temporel quels que soient les besoins des différentes applications en terme de débit et de latences. D'autre part, étant donné que l'allocation des slots est fixée une fois pour toute et que la longueur de slot recommandée est faible, il est nécessaire d'attribuer une bande passante disproportionnée aux canaux de trafic commande/contrôle, en leur attribuant des slots plus fréquemment que leur débit ne le justifie.

  L'allocation statique des slots pose également problème dans la mesure où les besoins en bande passante des équipements évoluent en fonction des phases de la mission. Il serait donc souhaitable de disposer d'un mécanisme permettant de basculer les tables d'allocation entre plusieurs configurations voire d'allouer les slots dynamiquement à chaque cycle suivant les besoins courants des différents équipements.

  SpaceWire-RT est en outre conçu pour être utilisé avec des slots extrêmement courts. Le fait que les Time-Codes disposent de la plus haute priorité de transmission au niveau caractère devrait permettre de synchroniser les équipements même sur des périodes aussi faibles. Toutefois, il reste à savoir si les applications peuvent gérer leur trafic aussi finement. De plus, le mécanisme de contrôle de flux choisi ne permettant pas de transmettre plus de trois DP par slot et par canal, il paraît inutile d'augmenter la durée des slots pour faciliter la synchronisation puisque cette durée est déjà suffisante pour transmettre trois DP.

– Une dernière limitation de SpaceWire-RT est qu'il n'existe à l'heure actuelle aucun équipement compatible. Dans la mesure où SpaceWire-RT impose aux équipements d'être compatibles avec le mécanismes des Time-Codes, d'être capables d'une synchronisation fine et d'utiliser une interface compatible SpaceWire, les modifications nécessaires risquent d'être longues à mettre en place. Par ailleurs, en cas de panne, SpaceWire-RT ne fournit aucun mécanisme pour isoler ou stopper un équipement qui ne respecterait plus ses slots d'émission.

2.3.2 SpaceWire-D

Comme on peut le constater, le projet de norme SpaceWire-RT est d'une grande complexité et engendre nombre de nouveaux problèmes. En 2009, l'ESA a donc décidé de simplifier la future norme en supprimant en grande partie les classes Assured et Guaranteed qui fournissaient des communications fiables.

La norme a donc été renommée SpaceWire-D (pour Determinism) puisque seul l'aspect temporel est conservé. La norme ne comporte donc plus que les classes Best Effort et Reserved ce qui simplifie déjà énormément le standard. Seuls les

acquittements sont préservés sous forme de fonctionnalité optionnelle pouvant être activée canal par canal.

En 2010, SpaceWire-D a été de nouveau remaniée et n'inclut plus qu'une sous-couche du protocole RMAP, chargée d'assurer le déterminisme des transactions. Ce protocole RMAP pour *Remote Memory Access Protocol* permet d'implémenter une mémoire distribuée par dessus un réseau SpaceWire. Il se base sur trois transactions : Read, Write et Read-Write. Dans cette nouvelle version de la norme, SpaceWire-D ne comprend plus que le mode synchrone qui permet d'assurer un déterminisme strict des transactions RMAP.

En 2011, il semble que la portée de la future norme ait été encore réduite et qu'elle n'inclut plus certaines fonctionnalités secondaires comme la segmentation des données. De plus, les routeurs seraient chargés de faire respecter les durées des time-slots en interrompant les transmissions non-terminées à la fin d'un time-slot.

Comme on le voit, la création d'une norme de surcouche temps-réel pour SpaceWire est un processus complexe qui nécessite des compromis entre de nombreux acteurs aux intérêts contradictoires. En attendant l'émergence de cette norme, il est donc nécessaire de disposer d'outils pour l'étude de réseau à base d'équipements SpaceWire standard afin de répondre aux exigences des missions à plus court terme.

Chapitre 3

Etat de l'art

Sommaire

Ce chapitre présente d'abord un bref historique du routage wormhole ainsi qu'une comparaison avec les autres modes de commutation existant. Il présente ensuite les différentes méthodes conçues pour étudier les délais de livraison dans un réseau wormhole et leur applicabilité à un réseau SpaceWire.

3.1 Historique du routage Wormhole

L'invention du routage Wormhole est liée à l'évolution des ordinateurs massivement parallèles. Un ordinateur massivement parallèle est une machine multiprocesseurs dont chaque nœud contient un processeur et une mémoire locale. Les nœuds contiennent un également un routeur et communiquent entre eux par l'envoi de message sur le réseau qui les interconnecte. Ce réseau est dit *réseau direct* car il connecte directement les nœuds entre eux sans routeurs intermédiaires.

Pour maximiser les performances des programmes parallèles exécutés sur une telle machine, il est essentiel que le réseau transmette les messages inter-processeurs avec des latences les plus faibles possibles. A cette fin, plusieurs modes de commutation ont été successivement considérés pour une utilisation dans un réseau direct [7].

- Le premier mode de commutation est le *Store-and-Forward* ou *packet switching* qui a été importé depuis les réseaux d'ordinateurs traditionnels. Dans ce mode, lorsqu'un paquet atteint un nœud, il est d'abord entièrement stocké dans son buffer. Ensuite, le paquet est routé vers le port de sortie adéquat.
  Ce mode de commutation présente deux défauts majeurs dans un réseau direct. Premièrement, chaque routeur doit contenir un buffer suffisamment grand pour pouvoir stocker un ou plusieurs paquets intégralement. Deuxièmement, il mène à des latences réseaux importantes. En effet, comme le paquet est stocké entièrement à chaque nœud traversé, sa latence est proportionnelle à la longueur de son trajet : si on note L la longueur du paquet, h le nombre de nœuds à traverser pour atteindre sa destination et C la capacité d'un lien, la latence du paquet est de l'ordre de $\frac{L \times h}{C}$.
- Le second mode possible est la commutation de circuit. Dans ce mode, un circuit physique est établi entre la source et la destination d'un paquet lors d'une première phase. Dans une seconde phase, le paquet est transmis le long de ce circuit qui lui est réservé tout le temps de la transmission. Une fois le paquet transmis, le circuit est supprimé et les liens peuvent être utilisés pour en créer un autre.
  Dans ce mode, une fois le circuit établi, la latence d'un paquet est de l'ordre de $\frac{L}{C}$ et la longueur du chemin parcouru par le paquet n'a pas d'impact sur la latence. Par contre, si un lien nécessaire au circuit est occupé et que le circuit ne peut pas être immédiatement établi, le paquet subira des délais supplémentaires de blocage.
- Le troisième mode de commutation est nommé *virtual cut-through* [8]. Ce mode est une sorte d'hybride des deux précédents. Ainsi, dans ce mode, lorsqu'un

paquet arrive dans un nœud, il peut être transmis immédiatement si le port de sortie est libre. En revanche, si le port est occupé, le paquet est stocké dans un buffer jusqu'à ce que le port soit disponible, tout comme dans le mode *Store-and-Forward*.
Si aucun conflit ne se produit, ce mode fournit une latence similaire à celle de la commutation de circuits puisque le paquet avance sans délais de nœud en nœud. Son principal défaut est qu'il nécessite toujours l'existence de buffers importants dans les routeurs au cas où des blocages se produiraient.

– Enfin, le *wormhole routing*, introduit par Dally et Seitz en 1986 [9], est une évolution du *virtual cut-through*, conçue pour consommer moins de mémoire dans chaque nœud. Le *wormhole routing* fonctionne de façon identique au *virtual cut-through* mais avec des buffers de petite taille. En général, un nœud ne peut pas stocker un paquet entier dans son buffer. Par conséquent, lorsqu'un paquet ne peut pas être transféré immédiatement, il reste bloqué dans le réseau et s'étale dans les buffers de plusieurs nœuds successifs ainsi que dans le buffer d'émission du nœud source.
En l'absence de conflits, la latence réseau est de l'ordre de $\frac{L}{C}$ comme pour le *virtual cut-through* mais il n'est plus nécessaire de disposer de buffers importants dans chacun des nœuds.
Le *wormhole routing* pose par contre d'autres problèmes. D'une part, comme dans le cas de la commutation de circuits, des blocages peuvent se produire et augmenter les délais de transmission. Si l'algorithme de routage est mal choisi, les blocages peuvent même devenir des interblocages et bloquer complètement le réseau. D'autre part, l'accumulation des blocages rend les latences difficiles à analyser.

La Figure 3.1 illustre les différences de latence entre trois des modes de commutation citées ci-dessus. Elle montre le délai de livraison d'un paquet traversant trois liens successifs pour chacun des trois modes de commutation. Cette figure est valable dans le cas où aucun conflit ne se produit et montre que pour la commutation de circuit et le *wormhole routing*, le délai de transmission d'un paquet dépend très peu du nombre de liens traversés. A l'inverse, dans un réseau *Store-and-Forward*, ce délai est proportionnel au nombre de liens. Le mode *Virtual cut-through* n'est pas illustré car, en l'absence de conflits, il se comporte de la même façon que la commutation de circuits.

3.2 Méthodes d'étude des latences réseaux

Les travaux portant sur l'étude des latences réseaux dans les réseaux Wormhole peuvent être globalement répartis en trois catégories. Premièrement, de nombreux travaux étudient le réseau wormhole interne d'une machine massivement parallèle d'un point de vue statistique et cherchent à déterminer un délai de bout en bout moyen plutôt qu'un délai pire-cas.

Une deuxième catégorie de travaux est consacrée à l'étude des délais pire-cas

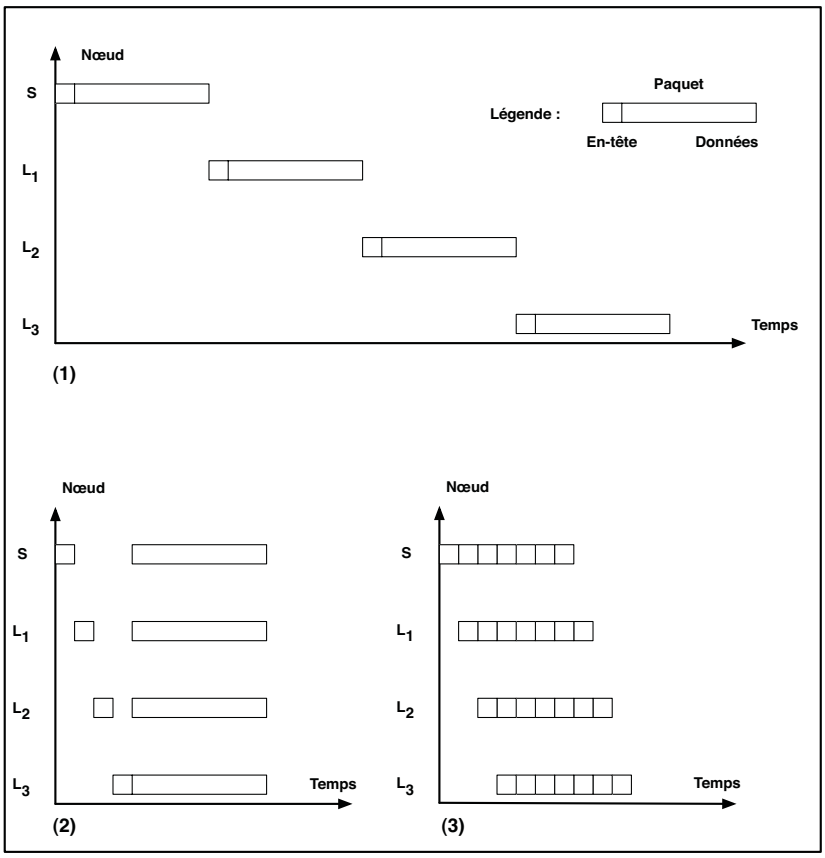


FIGURE 3.1 – Comparaison de différentes techniques de commutation : (1) *Store-and-Forward*, (2) Commutation de circuit, (3) *wormhole routing*

dans un réseau wormhole temps-réel. Ces études portent sur des réseaux dotés de mécanismes temps-réel dédiés ou cherchent à obtenir des variantes du wormhole routing plus adaptées aux communications temps-réel.

Enfin, la dernière catégorie de travaux, plus récente, étudie des réseaux wormhole de type Best Effort, utilisés comme réseau temps-réel. Ces réseaux sont souvent semblables au SpaceWire en ce qu'ils n'utilisent pas de mécanismes temps-réel dédiés qui augmentent la quantité de mémoire nécessaire dans les routeurs.

3.2.1 Méthodes d'étude statistique d'un réseau wormhole

Au cours des années 90, plusieurs études ont été menées pour déterminer comment minimiser la latence moyenne et maximiser le débit moyen au sein d'un supercalculateur ou d'une puce électronique.

Le but ici n'est pas de déterminer si les réseaux étudiés respectent certaines contraintes temporelles mais de maximiser la capacité de traitement globale du supercalculateur ou de la puce. Nous avons retenu quatre articles qui illustrent cette catégorie d'articles.

Dans [10], l'auteur étudie une catégorie générale de réseaux qu'il nomme "*k-ary n-cubes*". Ces réseaux ont une topologie en cube à n dimensions et comportent k nœuds dans chaque dimension. L'auteur étudie leur utilisation dans le contexte des puces VLSI (Very Large Scale Integration). Le but du papier est de maximiser les performances de la puce pour une densité de liens constante. Ainsi, l'auteur considère soit des réseaux avec un faible nombre de liens de bande passante élevée, soit des réseaux avec un grand nombre de liens de faible passante. L'auteur suppose également qu'un message n'est bloqué que pendant la durée de livraison d'un seul autre message à chaque nœud. Il s'agit d'une approximation très restrictive qui n'est valable que pour des réseaux de faible taille.

L'auteur fournit une expression analytique de la latence d'un message entre deux nœuds en fonction des paramètres k et n. Il démontre que la latence moyenne est minimale pour des réseaux de faible dimension (paramètre n) avec des liens de bande passante élevée.

[11] présente un modèle plus général de réseau wormhole à base de chaînes de Markov. Les messages sont générés par des processus de Poisson. Les auteurs calculent la probabilité qu'un message soit bloqué ainsi que la durée de blocage. Ils peuvent ainsi calculer le délai moyen de commutation pour un paquet dans un switch donné. Les auteurs montrent que leur résultat est cohérent avec des résultats obtenus par simulation mais notent que la complexité du modèle est élevée et qu'il ne peut donc être utilisé que sur des réseaux de faible taille.

[12] présente un modèle complet de réseau wormhole basé sur la théorie des files d'attente. Le modèle est présenté pour un réseau de type *k-ary n-cubes* avec deux canaux virtuels par lien physique mais peut être étendu à d'autres topologies. Il prend en compte l'interdépendance entre les switches due au routage wormhole et lève l'hypothèse restrictive de [10] sur le nombre de conflits lors de l'accès à un port de sortie. En revanche, il fait l'hypothèse que chaque source peut envoyer des messages à toutes les destinations aléatoirement. Les messages de chaque source sont générés par des processus de Poisson.

[12] permet de calculer le temps moyen d'attente pour l'accès à un port de sortie et la latence moyenne d'un message. La précision du modèle est validée grâce à des simulations. Ce modèle permet aux auteurs de montrer qu'augmenter la bande passante d'une partie des liens n'améliore en général pas la latence des messages.

Enfin, [13] compare différentes variantes de gestion du trafic dans un réseau wormhole. Les auteurs considèrent un réseau wormhole doté de canaux virtuels et étudient différentes façons d'allouer des priorités aux messages et d'arbitrer l'accès au port de sortie des routeurs. L'étude est faite à l'aide d'un simulateur au niveau caractère et utilise comme critère le pourcentage de messages qui ne sont pas livrés à temps. Les auteurs établissent ainsi une classification des différentes méthodes de gestion du trafic en fonction de leur performance et du coût supplémentaire qu'elles engendrent au niveau hardware.

Comme on le voit, les performances des réseaux wormhole ont été étudiées depuis longtemps mais seulement d'un point de vue statistique. Même dans le cas d'une utilisation temps-réel ([13]), la latence pire-cas des messages n'est pas utilisée

comme métrique. Ce type d'étude est utile pour déterminer le coût d'une infrastructure réseau et étudier ses performances globales mais ne permet pas de fournir des garanties suffisantes quant au respect des contraintes temporelles.

3.2.2 Méthodes d'étude des réseaux wormhole temps-réel

Cette seconde catégorie d'études portent sur l'utilisation de réseaux wormhole conçus pour les communications temps-réel. Le réseau étudié n'est donc pas un simple réseau wormhole mais utilise des mécanismes temps-réel dédiés. Certains proposent également des variantes du routage wormhole destinées à améliorer les performances temps-réel du réseau.

Le mécanisme le plus important dans un réseau wormhole temps-réel est l'utilisation de canaux virtuels (voir 2.2). Il réduit fortement les blocages car il permet à un paquet avançant dans le réseau de doubler un autre paquet bloqué en aval. De plus, couplé à un mécanisme de priorités préemptives, il permet à un message d'interrompre provisoirement la transmission d'un autre message moins prioritaire. Ceci permet de réduire largement la latence des messages les plus urgents. En l'absence de canaux virtuels, il faudrait soit bloquer le message urgent jusqu'à la fin de la transmission du message moins prioritaire, soit détruire le paquet moins prioritaire ce qui gaspille des ressources réseau.

[14] présente un test de faisabilité permettant de déterminer si un ensemble de messages peut être transmis par un réseau donné en respectant les deadlines de tous les messages. Ce test s'applique sur un réseau wormhole utilisant des canaux virtuels et des priorités statiques préemptives. Chaque message est périodique et a un niveau de priorité différent, du plus prioritaire au moins prioritaire. Chaque lien physique comporte un canal virtuel pour chaque message qui l'utilise.

Le principe du calcul pour un message M_i est le suivant. On considère l'ensemble S_i des messages plus urgents que M_i et qui partagent au moins un lien avec M_i. Comme on a supposé l'existence d'un canal virtuel pour chaque message, seuls ces messages plus prioritaires peuvent bloquer M_i. De plus, comme les messages sont périodiques, plusieurs occurrences d'un même message peuvent bloquer M_i. Le délai de transmission pire-cas de M_i à l'instant t est donc donné par :

$$R_i(t) = \frac{L_i}{C} + \sum_{j=1}^{i-1} \frac{L_j}{C} \cdot \lceil \frac{t}{p_j} \rceil, M_j \in S_i \tag{3.1}$$

où C est la capacité d'un lien, p_j est la période de M_j et L_i la taille du message M_i.

M_i est ordonnançable si $R_i(t) \leq t$ pour $t \leq d_i$ où d_i est la deadline de M_i. (3.1) peut être résolue itérativement et fournit un test d'ordonnançabilité. Si (3.1) converge pour $t \leq d_i$, M_i est ordonnançable. Les auteurs nomment ce modèle *lumped link* (lien regroupé) car il revient à traiter tous les liens utilisés par M_i comme un seul lien qui doit être libre de tout message de plus haute priorité pour que M_i puisse être transmis.

Ce test fournit une condition suffisante mais pas nécessaire car il peut être pessimiste. En effet, il ne tient pas compte du fait que des messages plus prioritaires

peuvent être transmis simultanément s'ils n'ont pas de liens en commun. De plus, le modèle dépend fortement de l'existence des canaux virtuels et des priorités que ne fournit pas SpaceWire. Enfin, l'algorithme utilisé ne nous semble pas correct dans tous les cas car il ne tient pas compte des conflits indirects que peut subir M_i.

Dans [15], les auteurs proposent une modification du routage wormhole plus appropriée aux communications temps-réel et utilisent le même modèle que [14] pour étudier ses performances. L'algorithme de routage a pour but de lever l'hypothèse restrictive d'existence d'un canal virtuel pour chaque message sur chaque lien, utilisée dans [14]. Ceci requiert en effet des ressources mémoires importantes.

L'algorithme proposé est nommé *Preemptive Pipelined Circuit Switching for Real-Time messages (PPCS-RT)*. L'idée est de décomposer la livraison d'un message en deux phases : *Path Establishment (PE)* et *Data Delivery (DD)*. Pendant la phase PE, l'en-tête d'un message avance seul dans le réseau et réserve un canal virtuel sur chaque lien traversé. Tant que cette phase n'est pas terminée, le message peut être préempté par un message de priorité supérieure. Quand l'en-tête atteint la destination, un acquittement est transmis à la source du message et la phase DD commence. Durant cette phase, le paquet ne peut plus être préempté et est transmis intégralement.

Dans un réseau wormhole temps-réel classique, lorsqu'un lien ne fournit pas un canal virtuel par message, une inversion de priorité peut se produire. Lorsque tous les canaux virtuels sont occupés par des messages de priorité basse et qu'un message de priorité haute arrive, il reste bloqué derrière ces messages, faute de canal disponible pour les préempter. PPCS-RT permet de réduire ce problème car les messages de priorité basse peuvent être préemptés pendant la phase PE. Des inversions de priorité peuvent toujours se produire mais leur durée est bornée par le temps de transmission du message de priorité basse (phase DD).

Dans [15], les auteurs proposent également une adaptation du modèle *lumped link* qui prend en compte correctement les conflits indirects dus aux messages de priorité haute. Ceci augmente le pessimisme de l'algorithme mais le rend correct. Cependant, l'effet des inversions de priorité ne semblent pas pris en compte dans le nouvel algorithme.

[16] propose un autre algorithme pour calculer une borne supérieure sur le délai de livraison d'un message. L'algorithme semble avoir été développé de façon indépendante mais il s'agit essentiellement d'une optimisation de l'algorithme *lumped link*. Les auteurs supposent toujours l'existence de canaux virtuels et de priorités préemptives.

Par rapport au modèle *lumped link*, les auteurs établissent un graphe des blocages qui permet de trier les conflits indirects. On peut ainsi savoir si un message M_k interfère directement avec un message M_i ou s'il interfère via un message M_j. Lorsque la taille de la fenêtre temporelle augmente (par résolution itérative), un message en conflit indirect n'interfèrera avec le message étudié qu'à condition que tous les messages intermédiaires dans le graphe des blocages soient aussi émis périodiquement au même moment. Autrement, ce message ne peut interférer et son impact n'est pas compté dans la borne supérieure calculée. Cet algorithme permet

d'améliorer la borne supérieure mais n'est toujours pas adapté à un réseau wormhole sans mécanismes temps-réel.

[17] propose une autre amélioration du modèle *lumped link* de [15]. Les hypothèses sont les mêmes : le réseau étudié fournit des canaux virtuels et des priorités statiques préemptives sur chaque lien. Le défaut du modèle *lumped link* est qu'il agrège tous les liens utilisés par un message ainsi que les liens utilisés par les flux qui interfère avec lui en une seule ressource qui doit être intégralement réservée pour que le message soit transmis. Or, si deux flux en conflit avec le flux étudié ne se croisent pas, ils peuvent être transférés simultanément.

Les auteurs proposent donc de construire un *contention tree* qui permet de repérer ces transmission parallèles et donc de réduire le pessimisme du calcul. Les nœuds de l'arbre sont constitués par les messages. Les messages sont numérotés de 1 à n, par ordre de priorité décroissante. Une arête $E_{i,j}, i < j$ entre M_i et M_j représente un conflit direct entre les messages M_i et M_j. Un chemin entre M_i et M_j représente un conflit indirect entre ces deux messages. Les auteurs illustrent l'utilisation de leur arbre pour calculer le délai pire-cas d'un paquet sur un exemple mais ne donne pas d'algorithme général. De plus, ils ne fournissent pas d'estimation numérique de l'amélioration apportée par leur méthode.

[18] se base sur les travaux précédents et propose une autre méthode pour déterminer l'ordonnançabilité d'un ensemble de messages. Le réseau étudié est toujours un réseau wormhole doté de canaux virtuels et de priorités préemptives. [18] reprend la distinction faite par [17] entre interférences directes et indirectes mais montre que la méthode présentée dans [17] peut être fausse car certaines transmissions sont considérées comme simultanées alors qu'elles peuvent ne pas l'être. Le calcul proposé par [18] se base sur l'hypothèse qu'un message subit son délai pire-cas s'il est émis en même temps que tous les messages de priorité plus élevée. Les auteurs fournissent ensuite une méthode itérative et calculent séparément les délais dus aux interférences directes et indirectes. Ils montrent également que leur méthode fournit des bornes plus serrées que [15] et [14] mais qu'elle n'est pas optimale lorsque des interférences simultanées existent. Les auteurs sous-entendent cependant qu'il n'est pas possible d'obtenir une borne exacte dans ce cas.

[19] est une évolution de ce modèle destinée à réduire le nombre de niveaux de priorité nécessaires au bon fonctionnement du réseau. En effet, les canaux virtuels sont coûteux en terme de mémoire et de consommation d'énergie. Les auteurs proposent donc de partager un niveau de priorité donné entre plusieurs messages. Ils fournissent également un modèle pour calculer le délai de transmission dans ce nouveau mode de fonctionnement.

Le principe du calcul est le suivant. A chaque niveau de priorité, on traite tous les messages de ce niveau comme un unique message qui doit réserver tous les liens utilisés par les messages de même niveau pour pouvoir être transmis. Le délai pour ce "message agrégé" est ensuite calculé suivant la méthode présentée dans [18]. Le délai complet pour un message est donné par la somme du délai pour le "message agrégé" et du délai dans le terminal source, propre au message. Lorsqu'on l'emploie avec un seul niveau de priorité, ce modèle correspond au fonctionnement de Spa-

ceWire. Cependant, si on considère comme un seul "message agrégé" les paquets urgents de taille faible et les paquets charge utile de grande taille, le résultat va être extrêmement pessimiste. Il nous faut donc trouver une méthode plus précise qui différencie les différents flux, même lorsqu'ils ont le même niveau de priorité.

3.2.3 Méthodes conçues pour l'étude des réseaux wormhole Best Effort

A l'opposé d'un réseau wormhole temps-réel, un réseau wormhole Best Effort ne fournit pas de mécanismes dédiés pour faciliter le respect des contraintes temporelles. Un réseau wormhole Best Effort ne dispose donc pas de canaux virtuels, ni de mécanisme de priorités.

La méthode WCFC

La première étude que nous présentons ici ne correspond pas tout à fait à ces critères. En effet, il s'agit de l'étude d'un réseau wormhole disposant de canaux virtuels mais sans priorités préemptives. Il est donc possible d'utiliser cette étude dans le cas particulier où un seul canal est utilisé pour chaque lien. Cela revient à ne pas utiliser de canaux virtuels et correspond donc au fonctionnement d'un réseau SpaceWire. Ainsi, [20] présente un algorithme appelé WFCF (*Wormhole Channel Feasibility Checking*) qui permet de calculer une borne sur le délai pire-cas d'un flux et de vérifier que la deadline du flux est respectée. Ce calcul est basé sur une formule récursive qui prend en compte les blocages dans les routeurs traversés successivement par un paquet ainsi que les blocages subis en aval par les paquets qui le bloquent. Ce calcul est applicable à un réseau SpaceWire et similaire à celui que nous présentons au chapitre 4.1 avec un formalisme différent. Cependant, notre méthode de calcul a été développé indépendamment et fournit des bornes moins pessimistes que WCFC. En effet, WCFC ne suppose pas l'existence d'un mécanisme Round-Robin comme celui utilisé dans les routeurs SpaceWire et donne donc des bornes plus pessimistes que celles de notre méthode qui, elle, tient compte de ce mécanisme.

Les deux autres articles présentés ici étudient des réseaux de type Network-on-Chips (NoC). Le but des NoCs est d'intégrer un réseau dans un système de type System-on-Chip (SoC) afin de faciliter l'augmentation du nombre de composants connectés. D'une manière générale, les Networks-on-Chip sont soumis aux mêmes contraintes que SpaceWire. Leur utilisation mémoire doit donc être réduite pour limiter le coût et l'encombrement du système.

Les méthodes RTB-LL et RTB-HL

[21] présente deux méthodes de calcul de délai pire-cas dans un réseau wormhole Best-Effort. La première, nommée *RTB-LL* (*Real-Time Bound for Low Latency traffic*), est une optimisation de la méthode WCFC de [20]. Elle est similaire à la méthode de calcul que nous présentons en Section 4.1 et a été publiée simultanément. Cette méthode suppose que l'intervalle entre deux paquets successifs d'un

même flux est suffisamment important pour que les paquets n'interfèrent pas entre eux.

La seconde méthode présentée dans [21], nommée *RTB-HB* pour *Real-Time Bound for High Bandwidth traffic* est similaire dans le principe mais se base sur une hypothèse de trafic différente. Ici, on suppose que les paquets successifs d'un flux sont émis de façon continue, du moment que le réseau peut les absorber. De plus, on fait l'hypothèse que, au début du calcul, le réseau est déjà complètement saturé de paquets. Cette méthode mène à une borne plus élevée que *RTB-LL* mais est plus générale car elle ne nécessite pas d'hypothèse sur la périodicité des paquets ou leur intervalle minimum. [21] introduit cette méthode pour un réseau wormhole dans lequel tous les liens et tous les terminaux fonctionnent de façon synchronisée. Ainsi, à chaque cycle, chaque routeur et chaque terminal reçoit et émet un caractère sur chacune de ces interfaces réseau. Les auteurs présentent leur calcul dans le cas où tous les paquets ont la même taille que le buffer d'entrée des routeurs et affirment que le modèle se généralise directement aux autres cas.

Le calcul d'une borne pire-cas se fait récursivement de la façon suivante pour un paquet p_i. Le réseau étant supposé saturé, lorsque p_i doit avancer du buffer d'émission de son terminal source vers le premier routeur, il faut que le paquet présent dans le buffer du routeur en aval avance jusqu'au routeur suivant pour que p_i puisse avancer. Ce paquet est lui-même bloqué par un autre qui doit avancer à son tour et ainsi de suite. De plus, dans le cas d'un routeur, d'autres paquets provenant des autres ports d'entrée du routeur peuvent doubler p_i. Leur délai, calculé récursivement lui aussi, s'ajoute au délai de transfert de p_i. Lorsque le paquet atteint sa destination, il est lu par la destination en L_i cycles ce qui constitue le cas de base de la récursion.

Cette méthode fournit un pire cas absolu, indépendant du schéma de trafic utilisé et fonctionne bien dans le contexte pour lequel elle a été créée. De plus, elle permet de calculer l'intervalle minimal à respecter entre deux paquets pour qu'il n'y ait pas de pertes de paquets. Nous verrons en Section 4.2 qu'elle devient trop pessimiste pour être utilisable sur un réseau dont les liens ont des vitesses hétérogènes.

Une méthode basée sur le Calcul Réseau

Enfin, [22] présente un modèle de réseau wormhole basé sur la théorie du Calcul Réseau [23]. Les auteurs considèrent que les ressources partagées par les flux dans le réseau sont de trois types : les crédits du contrôle de flux, les liens et les buffers d'entrée des routeurs. Ils analysent successivement ces trois partages pour aboutir à une représentation du réseau sous forme de courbe de service sur laquelle le Calcul Réseau peut être utilisée pour déterminer la courbe de service de bout en bout offerte à chaque flux.

Les auteurs construisent ensuite un *contention tree* qui permet de déterminer dans quel ordre calculer les courbes d'arrivée des différents flux. Dans cet arbre, chaque nœud représente un buffer partagé. Le "tronc" de l'arbre représente le chemin du flux étudié de sa source à sa destination. Les branches greffées dessus sont les

chemins suivis par les flux qui interfèrent avec le flux étudié, de leur source au point où ils interfèrent avec le flux étudié. Sur chacune des branches, d'autres branches peuvent se greffer pour prendre en compte les conflits subis par ces flux.

Enfin, pour analyser les conflits le long d'une branche de l'arbre, les auteurs présentent trois schémas d'interférence entre trois flux et montrent que, lorsqu'on sait résoudre ces trois schémas d'interférence, l'analyse peut être complétée pour un nombre quelconque de flux. On aboutit ainsi à la courbe de service de bout en bout du flux étudié.

Cet article présente une approche très intéressante mais deux hypothèses nous paraissent trop simplificatrices pour que ce modèle soit utilisable en pratique. En premier lieu, pour modéliser le service offert par un routeur à un flux, les auteurs supposent que le temps qu'un caractère d'un paquet peut passer dans un routeur est borné. Ceci se traduit dans leur modèle par le choix d'un modèle de type *Weighted Round-Robin* pour gérer l'accès aux ports de sortie. Le problème ici est que pour déterminer les poids attribués par le Round-Robin aux différents flux, il nous faut connaître le délai passé par le paquet à l'intérieur du routeur. Or, c'est précisément ce que nous cherchons à déterminer. En réalité, ce modèle n'est réaliste qu'à condition de supposer que les paquets sont de taille suffisamment faible pour être stockés intégralement dans le buffer d'entrée d'un routeur. Cette hypothèse, bien que restrictive, peut être acceptable dans le cas d'un NoC mais pas pour un réseau SpaceWire typique. En second lieu, les auteurs emploient un modèle de démultiplexeur très pessimiste qui conduit à sous-estimer de façon importante la bande passante offerte à un flux démultiplexé (voir la Section 5.2.5.1 pour plus de détails).

En revanche, l'utilisation des schémas d'interférence pour déterminer la courbe de service de bout en bout des flux nous paraît très intéressante. Nous verrons ce mécanisme plus en détail en Section 5.3.2.

3.2.4 Conclusion

Au total, un faible nombre d'études a été consacré à l'études des réseaux wormhole best-effort tels que SpaceWire. La plupart des études du comportement temporel des réseaux wormholes cherchent à mesurer les performances moyennes d'un réseau ou traitent de réseaux dotés de mécanismes temps-réel dédiés. Ces deux catégories d'études ne sont pas utilisables dans notre contexte et ne fournissent pas d'angle pertinent pour aborder l'études des réseaux SpaceWire.

Deux études ont été consacrées au système de type Networks-on-Chip ces dernières années. L'une utilise une méthode récursive pour calculer le délai pire-cas subi par un paquet traversant un réseau synchrone saturé. L'autre élabore un modèle qui se base sur la théorie du Calcul Réseau pour obtenir une borne sur le délai pire-cas de bout en bout d'un paquet. Ces deux études ne sont pas applicables de façon immédiate à un réseau SpaceWire car leurs hypothèses de travail ne sont pas identiques aux notres mais elles peuvent servir d'inspiration dans la mise au point d'une méthode d'étude des réseaux SpaceWire.

Chapitre 4

Première approche : hypothèses faibles sur le trafic en entrée

Sommaire

La première approche que nous présentons pour le calcul d'une borne supérieure sur le délai pire cas d'un paquet traversant un réseau SpaceWire se base sur des hypothèses faibles sur le trafic en entrée du réseau.

En effet, les deux méthodes de calcul présentées dans ce chapitre ne nécessitent pas de caractériser précisément ce trafic. En particulier, il n'est pas nécessaire de savoir si les flux sont périodiques ou de connaître leur loi d'arrivée.

4.1 Calcul récursif

Le principe de cette méthode de calcul est de parcourir le chemin suivi par un flux et d'additionner, pour chaque routeur, les délais causés par chacun des flux qui sont en compétition avec le flux étudié. Ces délais sont calculés récursivement de la même façon puisque les paquets bloquants peuvent eux-mêmes être bloqués en aval. On aboutit ainsi à une formule récursive qui permet de calculer le délai de livraison pire cas d'un paquet à partir de n'importe quel point de son chemin.

4.1.1 Notations et définitions

Nous représentons un réseau SpaceWire par un graphe orienté connexe $\mathcal{G}(N, L)$. L'ensemble N des nœuds du graphe représente les terminaux et les routeurs SpaceWire. L'ensemble L des arcs représente les liens SpaceWire. Nous utilisons deux arcs de directions opposées pour représenter un lien SpaceWire bidirectionnel. Etant donné que deux routeurs peuvent être connectés par de multiples liens en parallèle, $\mathcal{G}(N, L)$ peut être un multigraphe. Par contre, il ne peut jamais y avoir plusieurs liens parallèles entre deux terminaux ou entre un terminal et un routeur. Le cas échéant, on remplacerait un terminal équipé de plusieurs interfaces par plusieurs terminaux. Cette hypothèse ne réduit pas la généralité du calcul car le Group Adaptative Routing (GAR) ne peut être utilisé qu'entre deux routeurs.

Une communication entre une source et une destination est modélisée par un flux unidirectionnel f. Chaque flux f est caractérisé par une taille maximale de paquet notée T_f et passe par un ensemble de liens formant un chemin dans $\mathcal{G}(N, L)$. On note F l'ensemble des flux. Nous n'avons pas besoin de connaître le débit de chaque flux mais nous faisons l'hypothèse que la capacité de chaque lien est suffisante pour transmettre toutes les données transitant par ce lien. Autrement, le délai pire-cas serait clairement infini.

Nous supposons également qu'aucun paquet ne peut être stocké intégralement dans les buffers des routeurs qu'il traverse. Par conséquent, chaque paquet doit être de longueur supérieure à la somme des tailles des buffers de réceptions entre sa source et sa destination. Cela revient à supposer que lorsque l'en-tête du paquet atteint sa destination, la fin du paquet est toujours dans le buffer du terminal source.

Notons $liens(f)$ la liste ordonnée des liens traversées par le flux f et $premier(f)$ la fonction qui renvoie le premier lien de $liens(f)$. Lorsque deux routeurs sont reliés par un groupe de plusieurs liens, on considère qu'un lien du groupe est le lien

canonique utilisé par tous les flux. Le routeur se charge ensuite de répartir dynamiquement les paquets sur les liens du groupe. Notons enfin $suiv(f, l)$ la fonction qui, étant donnés un flux f et un lien l de $liens(f)$ renvoie le lien suivant l dans $liens(f)$ et $prec(f, l)$ la fonction qui renvoie le lien précédent l dans $liens(f)$. Si l est le dernier lien de $liens(f)$, $suiv(f, l)$ renvoie $null$.

Nous faisons également les hypothèses suivantes :

- le routage est statique (c'est le cas sur les routeurs SpaceWire actuels) ;
- tous les liens ont la même capacité C ;
- on suppose que la politique d'arbitrage round-robin est utilisée par tous les routeurs ;
- tous les flux traversant un groupe utilisent le GAR avec tous les liens du groupe ;
- nous négligeons les délais causés par les Time-Codes et les Flow Control Token ;
- les paquets sont traités dès qu'ils arrivent à destination (la destination ne cause donc aucun délai) ;
- lorsqu'une source a commencé à émettre un paquet, elle le transmet à la vitesse maximale (la source ne cause donc aucun délai).

4.1.2 La méthode de calcul

Notre méthode de calcul se base sur l'idée que, dans un réseau SpaceWire, la livraison d'un paquet a lieu en deux phases. Pendant la première phase, le premier caractère (l'en-tête) du paquet est routé jusqu'à sa destination et crée un "circuit virtuel" entre la source et la destination du paquet. Pendant la seconde phase, le corps du paquet est transféré sur ce "circuit virtuel" comme sur un lien point-à-point. Une fois le paquet transféré, les liens sont libérés et le circuit disparaît.

Calculons maintenant une borne supérieure $d(f, l)$ sur le délai pire-cas subi par un paquet p du flux f cherchant à accéder à un lien l. Nous examinerons successivement les différents cas possibles.

4.1.2.1 Cas où $l = null$

Il s'agit du cas de base de la récursion et du cas le plus simple. Lorsque $l = null$, cela signifie que l'en-tête du paquet est arrivé à destination. Ainsi, il n'y a pas de délais causés par un routeur. Le seul délai restant est le temps nécessaire à la transmission du corps du paquet sur le circuit virtuel créé par l'en-tête (seconde phase du transfert). Ce délai est tout simplement :

$$d(f, null) = \frac{T_f}{C}. \tag{4.1}$$

4.1.2.2 $l \neq null$ et $l \neq premier(f)$

Lorsque $l \neq null$ et $l \neq premier(f)$, étant donné que la source et la destination ne causent aucun délai au paquet, le seul délai subi par p est causé par le routeur traversé pour accéder à l. Ce délai a deux causes possibles. La première est le délai

de commutation du routeur, que nous noterons d_C par la suite. Ce délai est identique pour tous les routeurs et tous les paquets.

Le second délai apparaît lorsque le port de sortie (ou le groupe de ports) est déjà utilisé par un autre paquet. Supposons dans un premier temps qu'il s'agit d'un lien simple.

Si l est un lien simple Avec le routage Wormhole, p doit attendre jusqu'à ce que le paquet bloquant le port ait été complètement transféré. Il est également possible que des paquets entrant par d'autres ports soient déjà en train d'attendre la libération du même port que p et passent avant lui. Notons que seuls les paquets sortant par le même port que p peuvent le retarder puisque les ports des routeurs SpaceWire sont indépendants.

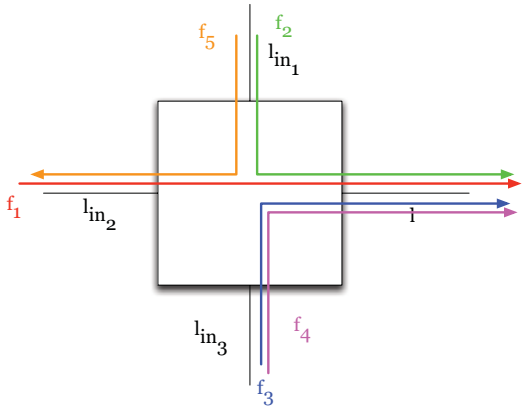


FIGURE 4.1 – Illustration du calcul de $d(f, l)$ sur un exemple

Soit $A_{f,l} = \{l_{in} \in L,\ l_{in} \neq prec(f, l),\ pour\ lesquels\ \exists f_{in} \in F | suiv(f_{in}, l_{in}) = l\}$ (i.e. $A_{f,l}$ est l'ensemble des liens qui sont utilisés immédiatement avant l par au moins un flux f_{in} et qui ne sont pas utilisés par f). Le délai causé par les paquets provenant d'autres ports d'entrée peut alors s'écrire sous la forme

$$d_{A_{f,l}} = \sum_{l_{in} \in A_{f,l}} d_{l_{in}} \tag{4.2}$$

Par exemple, sur le réseau de la figure 4.1, on a $A_{f_1,l} = \{l_{in_1}, l_{in_3}\}$.

De plus, SpaceWire utilise un mécanisme Round Robin pour sélectionner le port d'entrée qui aura accès au port de sortie. Ainsi dans le pire des cas, au plus un paquet pour chaque port d'entrée autre que celui de p pourra passer avant p. Etant donné que, pour chacun de ces ports d'entrée, plusieurs flux peuvent entrer dans le routeur, il nous faut donc calculer une borne sur le délai maximal que chacun de ces flux pourraient produire et prendre le maximum de ces délais. Remarquons que du fait des blocages possibles en aval, il ne s'agit pas forcément du flux dont les paquets sont les plus longs.

On a ainsi

$$d_{l_{in}} = [\max_{f_{in} \in U_{l_{in}}} d_{f_{in}}] \tag{4.3}$$

avec $U_{l_{in}} = \{f_{in} \in F | l_{in} \in links(f_{in})$ et $l \in links(f_{in}\}$ (i.e. $U_{l_{in}}$ est l'ensemble des flux qui utilisent le lien l_{in} puis le lien l).

Sur le même exemple, $U_{l_{in_1}} = \{f_2\}$ et $U_{l_{in_3}} = \{f_3, f_4\}$.

Enfin, le délai maximum qu'un paquet q peut causer à p est la durée nécessaire pour transférer q à partir du routeur courant si q était seul dans ce routeur, c'est-à-dire si q ne subissait aucun délai dans ce routeur. Cependant, q peut lui aussi être bloqué dans les routeurs qu'il va ensuite traverser. Par suite,

$$d_{f_{in}} = d(f_{in}, suiv(f_{in}, l)) + d_C \tag{4.4}$$

Enfin, une fois qu'il a obtenu l'accès au port de sortie, l'en-tête du paquet p est transféré au routeur suivant et peut avoir à nouveau à attendre pour obtenir l'accès au port de sortie de ce routeur.

Nous pouvons donc maintenant énoncer une définition récursive d'une borne supérieure $d(f, l)$ sur le délai nécessaire pour transférer un paquet du flux f à partir du moment où il tente d'accéder à un lien l tel que $l \neq null$ et $l \neq premier(f)$:

$$d(f, l) = \sum_{l_{in} \in A_{f,l}} [\max_{f_{in} \in U_{l_{in}}} d(f_{in}, suiv(f_{in}, l)) + d_C] + d(f, suiv(f, l)) + d_C \tag{4.5}$$

Sur notre exemple, on a au final

$$d(f_1, l) = d(f_1, suiv(f_1, l) + d(f_2, suiv(f_2, l) + max(d(f_3, suiv(f_3, l), d(f_4, suiv(f_4, l)).$$

Si l fait partie d'un groupe Supposons maintenant que le Group Adaptative Routing soit utilisé et que l fasse partie d'un groupe d'au moins deux liens. On notera n_l le nombre de liens du groupe auquel appartient le lien l, y compris l.

Dans ce cas, les paquets entrant par des ports de $A_{f,l}$ ne sont pas transmis séquentiellement sur l mais répartis sur les liens du groupe. Ainsi, dès qu'un lien du groupe se libère, le routeur envoie immédiatement un des paquets en attente sur ce lien.

Par conséquent, dans le pire des cas, un paquet du flux f est transmis dès qu'un lien est libre et qu'il ne reste plus aucun autre paquet en attente. Le délai subi est donc égal au temps d'occupation du lien le moins utilisé parmi les n_l liens du groupe. Ceci nous permet de calculer le délai pour une distribution donnée des paquets sur les n_l liens du groupe.

Notons $(A^k_{f,l})_{k \in \{1,...n_l\}}$ une partition de $A_{f,l}$ en n_l parties. Pour rappel, une partition d'un ensemble E est un ensemble P de sous-parties non-vides de E telles que tous les éléments de E appartiennent à exactement un élément de P.

Si $prec(f, l)$ fait partie d'un groupe, $A_{f,l}$ contient tous les liens de ce groupe sauf celui emprunté par le paquet considéré. Le délai causé par les paquets traversant l avant un paquet du flux f est alors :

$$D((A^k_{f,l})) = \min_{k \in \{1,...n_l\}} \sum_{l_{in} \in A^k_{f,l}} [\max_{f_{in} \in U_{l_{in}}} d(f_{in}, suiv(f_{in}, l)) + d_C] \tag{4.6}$$

Pour obtenir le délai pire-cas subi par un paquet de f, il ne nous reste plus qu'à calculer (4.6) pour chaque partitionnement possible de $A_{f,l}$ en n_l parties. Les partionnements tels que certaines parties soient vides sont valides. Au final, $d(f,l)$ s'exprimera donc ainsi :

$$\max_{((A^k_{f,l})_k)} D((A^k_{f,l})) + d(f, suiv(f,l)) + d_C \tag{4.7}$$

Essayer tous les partitionnements possibles de $A_{f,l}$ nous oblige à prendre en compte un certain nombre de situations qui ne peuvent se produire en réalité. Par exemple, le cas où un lien du groupe est laissé entièrement libre par les autres flux est envisagé. Cela n'est pas un problème car ces situations mènent à des délais meilleurs que la réalité. Elles n'ont donc pas d'influence sur le délai pire-cas puisque nous prenons le maximum des délais engendrés.

Notons que (4.7) est valable également pour $n_l = 1$ et cohérente avec notre première formulation du délai : si $n_l = 1$, il n'y a qu'une seule partition possible de $A_{f,l}$ si bien que (4.7) devient :

$$d(f,l) = \sum_{l_{in} \in A_{f,l}} [\max_{f_{in} \in F_{l_{in}}} [d(f_{in}, suiv(f_{in}, l))] + d_C] + d(f, suiv(f,l)) + d_C$$

qui est bien notre première définition de $d(f,l)$.

4.1.2.3 Cas où $l = premier(f)$

Enfin, si $l = premier(f)$, cela signifie que le premier nœud de l est la source de f. Il s'agit donc d'un terminal et il ne peut y avoir aucun flux entrant par d'autres ports et créant des conflits d'accès à l. Par contre, ce nœud peut être source d'autres flux que f. Leurs paquets peuvent alors utiliser l avant que p n'y ait accès. Le délai maximum qu'un paquet q peut causer à p est la durée nécessaire pour transférer q à partir de la source si q était seul, c'est-à-dire si q ne subissait aucun délai au démarrage. Cependant, q peut lui aussi être bloqué dans les routeurs qu'il va ensuite traverser.

Pour modéliser de façon simple les délais subis dans le terminal source, on peut représenter celui-ci comme un routeur virtuel sur lequel viennent se connecter des sources virtuelles représentant les différents flux (voir Figure 4.2). Chaque source virtuelle est la source d'un seul des flux et le délai d'accès au routeur virtuel pour f est équivalent au délai subi par f dans le terminal source.

Ce modèle ne correspond pas tout à fait à la réalité. Dans un terminal réel, le buffer d'émission se comporte comme une file d'attente FIFO dans lesquels les différents processus qui veulent émettre des données écrivent leur paquet. Toutefois, sans hypothèses sur la périodicité du trafic, nous ne pouvons pas modéliser ce comportement. Il nous semble qu'un accès de type Round-Robin est une approximation raisonnable lorsque le trafic n'est pas saturé.

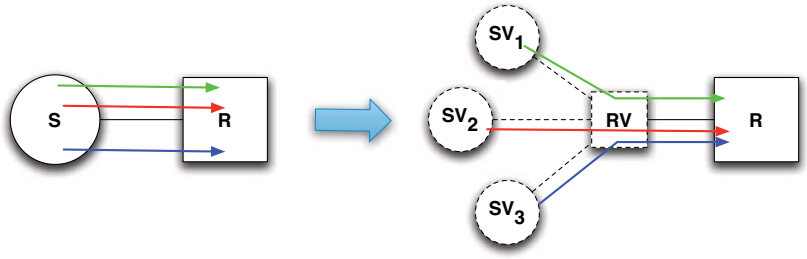


FIGURE 4.2 – Modèle d'un terminal source

Au total, le calcul de $d(f,l)$ peut être résumé par l'équation suivante :

$$d(f,l) = \begin{cases} \frac{T_f}{C} & \text{si } l = null \\ \max\limits_{((A^k_{f,l})_k)} D((A^k_{f,l})) + d(f, suiv(f,l)) + d_C & \text{sinon} \end{cases} \quad (4.8)$$

La preuve de terminaison de la méthode est donnée dans la Sous-Section suivante. L'équation (4.8) fournit ainsi une méthode de calcul d'une borne supérieure sur le délai pire-cas de bout en bout d'un flux quelconque f. Cette méthode peut ensuite être utilisée comme outil d'aide au dimensionnement de réseau SpaceWire en utilisant la métrique des délais pire-cas des différents flux comme critère de comparaison des réseaux obtenus en faisant varier la topologie du réseau.

4.1.3 Preuve de terminaison du calcul

Une définition préalable est nécessaire à la démonstration. On appelle *graphe de dépendance des liens* d'un réseau représenté par le graphe $\mathcal{G}(N,L)$, le graphe orienté $D = \mathcal{G}(L,E)$ dont les nœuds L sont les liens de $\mathcal{G}(N,L)$ et les arcs E sont les couples de liens utilisés successivement par au moins un flux :
$E = \{(l_i, l_j) \in L \; pour \; lesquels \; \exists f \in F | suiv(f, l_i) = l_j\}$

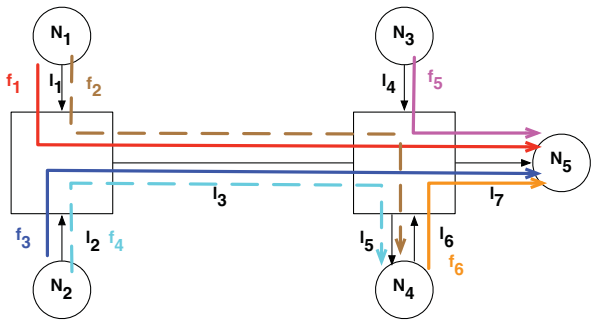


FIGURE 4.3 – Exemple de réseau SpaceWire (seuls les arcs utilisés par au moins un flux sont affichés)

A titre d'exemple, la Figure 4.4 présente le graphe de dépendance pour le réseau de la Figure 4.3. Il s'agit d'un réseau comportant deux routeurs et six terminaux. Six flux se partagent le réseau.

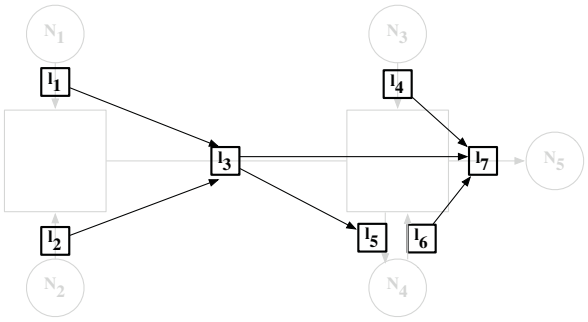


FIGURE 4.4 – Graphe de dépendance pour le réseau de la Figure 4.3

Théorème 1. *Pour tout flux f et tout lien l de liens(f), le calcul de $d(f,l)$ termine si et seulement si D est acyclique.*

Preuve : $\Rightarrow$ Supposons qu'il existe un cycle dans D. Etant donné que $suiv(f,l)$ ne peut renvoyer l pour aucun l (un paquet n'emprunte pas deux fois de suite le même lien), la longueur est supérieure ou égale à 2. Etant donné que d est définie de façon récursive, cela veut dire qu'il est possible de trouver un flux f et un lien l tels que calculer $d(f,l)$ nécessite un appel de $d(f,l)$ pendant la récursion. Par suite, le calcul de $d(f,l)$ ne termine pas.

$\Leftarrow$ Supposons que D est acyclique. Il est alors possible d'utiliser le tri topologique pour assigner un ordre total aux sommets de D tel que si $(l_i,l_j) \in E$ alors $l_i < l_j$ (si D n'est pas connexe, il est possible d'ordonner chaque composant connexe et d'ordonner ceux-ci arbitrairement ensuite). D'après la définition de E et de $suiv(f,l)$, il vient que calculer $d(f,l)$ ne nécessite que des appels à d avec des liens strictement supérieurs à l comme paramètres. Etant donné que F est fini et que $d(f,l)$ n'est appelé qu'une fois pour un f et un l donnés, il s'ensuit que le calcul de $d(f,l)$ termine toujours. ■

Notons qu'il est démontré dans [24] que pour un réseau Wormhole sous des hypothèses semblables aux nôtres, il est équivalent de dire que le graphe de dépendance est acyclique et que le réseau ne contient pas d'interblocages. Un réseau sujet aux interblocages n'étant pas utilisable en pratique de toutes façons, le Théorème 1 montre que le calcul se termine dans tous les cas de figure pertinents. Ceci est également cohérent avec le fait que, lorsque le calcul ne finit pas, $d(f,l)$ tend vers l'infini ce qui est caractéristique d'un interblocage.

Il est facile de construire un réseau wormhole de faible taille sujet à un problème d'interblocage. Le réseau de la Figure 4.5 en est un bon exemple. Son graphe de dépendance (voir Figure 4.6) est bien cyclique et il est facile d'imaginer que si chacun des flux émet un paquet au même instant, les trois paquets vont rester bloqués. Bien sûr, sur un réseau plus complexe, les interblocages sont beaucoup plus difficiles à repérer manuellement. Le concepteur réseau doit donc disposer d'un outil pour l'aider à garantir que le réseau n'en contient pas.

Une façon simple de tester si le calcul va terminer avant de le lancer, est d'utiliser la matrice d'adjacence du graphe de dépendance et de l'itérer. Si elle converge vers la

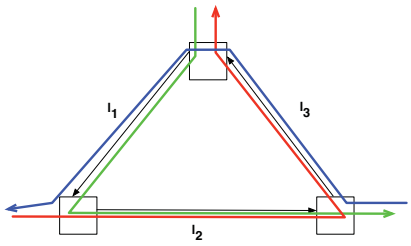


FIGURE 4.5 – Exemple de réseau soumis à un problème d'interblocage

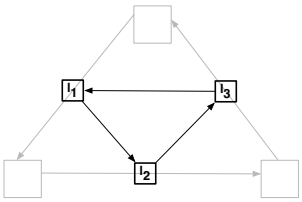


FIGURE 4.6 – Graphe de dépendance du réseau de la Figure 4.5

matrice nulle, alors le graphe n'est pas cyclique et le calcul terminera. Au contraire, si une puissance de la matrice n'a que des 1 sur sa diagonale, cela signifie que le graphe de dépendance est cyclique et que le calcul ne terminera pas.

4.1.4 Exemple de calcul de délais

4.1.4.1 Application de la méthode de calcul

Nous allons maintenant présenter le fonctionnement de la méthode sur l'exemple de réseau SpaceWire de la Figure 4.3. Il n'est pas nécessaire pour le calcul de connaître le débit de chaque flux, seulement la taille maximale des paquets que nous noterons $T_i,\ i \in \{1, \ldots, 6\}$ respectivement.

Nous devons d'abord vérifier que le graphe de dépendance du réseau (voir l'Appendice pour la définition) n'est pas cyclique. Comme on peut le voir sur la figure 4.4, ce graphe ne contient clairement aucun cycle et le calcul peut donc être effectué. En premier lieu, calculons une borne sur le délai pire-cas du flux f_1 qui a pour source N_1 et pour destination N_5.

Nous avons d'abord :

$$\begin{aligned} d(f_1, first(f_1)) &= d(f_1, l_1) \\ &= d(f_2, suiv(f_2, l_1)) + d(f_1, suiv(f_1, l_1)) \\ &= d(f_2, l_3) + d(f_1, l_3) \end{aligned} \tag{4.9}$$

La deuxième moitié devient :

$$\begin{aligned}
d(f_1, l_3) &= \max(d(f_3, l_7), d(f_4, l_5)) + d(f_1, l_7) + 2.d_C \quad \text{où} \\
d(f_1, l_7) &= d(f_5, null) + d(f_6, null) + d(f_1, null) + 3.d_C \\
&= T_1 + T_5 + T_6 + 3.d_C \\
d(f_3, l_7) &= d(f_5, null) + d(f_6, null) + d(f_3, null) + 3.d_C \\
&= T_3 + T_5 + T_6 + 3.d_C \\
d(f_4, l_5) &= d(f_4, null) + d_C = T_4 + d_C
\end{aligned}$$

La première partie de [4.9] se développe en :

$$\begin{aligned}
d(f_2, l_3) &= \max(d(f_3, l_7), d(f_4, l_5)) + d(f_2, l_5) + 2.d_C \\
d(f_2, l_5) &= d(f_2, null) + d_C = T_2 + d_C
\end{aligned}$$

avec $d(f_3, l_7)$ et $d(f_4, l_5)$ qui ont déjà été calculé.

En substituant les développements dans [4.9] nous obtenons :

$$d(f_1, l_1) = \frac{T_1 + T_2 + T_5 + T_6}{C} + 2.\max(\frac{T_3 + T_5 + T_6}{C} + 2.d_C, \frac{T_4}{C}) + 9.d_C \quad (4.10)$$

De la même façon, pour f_4 on obtient :

$$d(f_4, l_2) = \frac{T_3 + T_4 + T_5 + T_6}{C} + 2.\max(\frac{T_1 + T_5 + T_6}{C} + 2.d_C, \frac{T_2}{C}) + 9.d_C \quad (4.11)$$

Pour f_5, on a d'abord :

$$\begin{aligned}
d(f_5, l_4) &= d(f_5, suiv(f_5, l_4)) = d(f_5, l_7) \\
&= \max(d(f_1, null), d(f_3, null)) + d(f_6, null) + d(f_5, null) + 3.d_C
\end{aligned}$$

ce qui nous donne :

$$d(f_5, l_4) = \frac{T_5 + T_6 + \max(T_1, T_3)}{C} + 3.d_C \quad (4.12)$$

4.1.4.2 Quelques remarques sur ces résultats

Nous pouvons faire plusieurs remarques sur ces résultats. Notons premièrement que (4.10) et (4.11) sont similaires. (4.11) peut être obtenu à partir de (4.10) en échangeant T_1 et T_3 d'une part, T_2 et T_4 d'autre part. Ceci est cohérent avec le fait que N_1 et N_2 ont des schémas de communication identiques vers N_4 et N_5.

On peut également voir dans (4.10) que f_5 et f_6 ont un impact plus important que f_2 et f_3 dans le délai subi par f_1. Par suite, à moins que T_4 ne soit plus grand que $T_3 + T_5 + T_6$, au pire, un paquet de f_1 sera retardé trois fois par un paquet de f_5 et un paquet de f_6, deux fois par un paquet de f_3 et une fois par un paquet de f_2.

Notons que ceci est également vrai pour f_4 bien que ce flux n'ait pas la même destination que f_5 et f_6. Ceci est dû au fait qu'un paquet de f_4 peut être bloqué

derrière un paquet de f_3 ou f_1 qui peut, à son tour, être bloqué derrière un paquet de f_5 ou f_6. Cet exemple illustre bien le fait que les délais ne sont pas causés par des conflits d'accès à une destination commune mais par des conflits d'accès au lien partagé l_3. Par ailleurs, (4.12) montre qu'un paquet de f_5 ne peut être retardé qu'une fois par un paquet de f_4 et une fois par un paquet de f_1 ou f_3.

Ces observations suggèrent que lorsque plusieurs nœuds envoient des données à une même destination (une mémoire de masse par exemple) en partageant certains liens, les flux dont la source est proche de la destination ont un impact important sur les flux arrivant d'un nœud plus éloigné dans le réseau. Au contraire, les flux pour lesquels la source est éloignée de la destination ont un faible impact sur ceux partant d'un nœud proche de la destination. Par conséquent, il semble préférable de placer les nœuds émettant les plus longs paquets loin de la destination. Cela permet de réduire de façon importante le délai pire-cas pour les autres nœuds tout en n'augmentant que faiblement ce délai pour les paquets longs.

Toutefois, nous n'avons pris en compte ici que les délais pire-cas, pas les délais ou débits moyens. En effet, pour calculer un délai pire-cas, il nous faut supposer qu'à chaque lien, des paquets provenant des nœuds proches de la destination vont être entrelacés avec des paquets provenant de nœuds plus éloignés. Bien sûr, cela ne se produit pas forcément à chaque fois pour chaque paquet et chaque lien. Il se peut donc que placer les nœuds émettant des paquets longs loin de leur destination n'aboutisse qu'à bloquer tous les liens suivis par ces paquets pendant une durée plus importante. Cela peut également retarder d'autres flux adressés à d'autres destinations mais utilisant certains liens communs avec ces paquets de grande taille.

4.1.4.3 Application numérique

Afin de mieux comprendre la signification de ces formules, il peut être utile d'en considérer une application numérique. Nous utiliserons les valeurs suivantes pour les tailles maximales des différents paquets :

$$T_1, T_3 : 5120 \text{ octets} \ ; \ T_2, T_4 : 200 \text{ octets} \ ; \ T_5, T_6 : 1000 \text{ octets}$$

Ces valeurs correspondent à des tailles de paquets typiques si, par exemple, N_1 et N_2 sont des instruments d'observation, N_4 un processeur, N_5 une mémoire de masse et N_3 un équipement de monitoring. N_1 et N_2 envoient des données à haut débit et avec des tailles de paquet importantes vers N_5. Tous deux envoient également des paquets de monitoring vers N_4 qui en fait une synthèse puis transmet le résultat à N_5. N_3 envoie lui aussi ses données à N_5. Les flux f_2 et f_4 sont considérés comme critiques. Tous les liens ont une capacité $C = 200\ Mbps$ et le délai de commutation est $d_C = 0,5\ \mu s$ pour tous les routeurs. Nous pouvons maintenant calculer les délais pour f_1, f_4 et f_5 dans le meilleur et le pire des cas. Les résultats sont donnés dans la partie gauche du tableau 4.1.

Le délai dans le meilleur cas est le délai de livraison d'un paquet qui ne subirait aucun blocage lors de sa traversée du réseau. Ce délai est donc pour le flux f

$$d_f^{best} = \frac{T_F}{C} + d_C \times h \tag{4.13}$$

Flux	sans segmentation		avec segmentation	
	Meilleur cas	Pire cas	Meilleur cas	Pire cas
f_1 (segment)			$0,01\ ms$	$0,35\ ms$
f_1 (paquet)	$0,257\ ms$	$1,08\ ms$	$0,2\ ms$	$7\ ms$
f_4	$0,01\ ms$	$1,08\ ms$	$0,003\ ms$	$0,35\ ms$
f_5	$0,050\ ms$	$0,36\ ms$	$0,05\ ms$	$0,11\ ms$

TABLE 4.1 – Délais meilleur et pire-cas pour l'exemple de réseau

où h est le nombre de routeurs traversés. Si le trafic sur le réseau est important, il est possible que d_f^{best} soit une borne inférieure, impossible à atteindre en pratique.

Comme prévu d'après leurs formules, f_1 et f_4 ont le même délai pire-cas. En revanche, leurs délais dans le meilleur des cas sont très différents. Le rapport entre leurs délais dans le meilleur et le pire des cas est donc beaucoup plus faible pour f_1 (ratio de 4,2) que pour f_4 (ratio de 108). Le blocage récursif mène ainsi à de longs délais même pour les paquets de faible taille. Pour f_5, le rapport est de 7. Comme prévu, un flux dont la source est proche de la destination est peu influencé par les autres flux.

Nous pouvons ensuite essayer de segmenter les grands paquets de f_1 et f_3 en petits paquets de 256 octets pour réduire le délai pire-cas de f_4. Comme on peut le voir dans la partie droite du tableau 4.1, cela permet de réduire le délai pire-cas de f_4 à $0,35\ ms$ soit une réduction d'un facteur 3 environ. Le délai pire-cas de f_5 est réduit dans les mêmes proportions.

Cependant, la segmentation a un coût pour f_1. En effet, étant donné qu'il est maintenant nécessaire d'envoyer 20 segments pour transmettre la même quantité d'information qu'avant, le délai pire-cas pour un paquet complet est maintenant de $7\ ms$ soit presque 6,5 fois plus que sans segmentation.

La méthode de calcul peut ainsi aider à déterminer la taille optimale des paquets des différents flux en fonction des délais de bout en bout que l'on veut respecter pour chacun des flux.

4.1.5 Prise en compte des liens de capacités différentes

Il peut être nécessaire d'étudier un réseau dont tous les liens n'ont pas la même capacité. De même, les terminaux connectés au réseau ne sont pas forcément capables de lire les données reçues à la vitesse à laquelle le réseau les livre.

Une adaptation très simple du modèle permet de prendre en compte ces deux cas de figure. Lorsqu'un paquet traverse deux liens de capacités différentes C_1 et C_2, le contrôle de flux fait que le paquet est globalement transmis à la vitesse la plus faible des deux. Ainsi un paquet de taille T_f traversant ces deux liens sera livré avec un délai $\frac{T_f}{min(C_1,C_2)}$.

Plus généralement, la méthode de calcul présentée ci-dessus (4.1.2) s'adapte ainsi. Notons C_f^{min} la capacité minimale parmi celles des liens de $links(f)$. L'équa-

tion (4.1) s'écrit maintenant :

$$d(f, null) = \frac{T_f}{C_f^{min}}. \tag{4.14}$$

Remarquons que lorsqu'un paquet p_1 d'un flux f_1 est doublé par un paquet p_2 d'un flux f_2, p_2 bloque toujours p_1 pendant une durée $\frac{T_{f_2}}{C_{f_2}^{min}}$, y compris lorsque le lien de capacité $C_{f_2}^{min}$ est situé avant le lien pour lequel p_1 et p_2 sont en conflits. Ainsi, sur l'exemple de la Figure 4.7, les paquets du flux f_2 sont livrés à la vitesse $C_{f_2}^{min} = \min(20, 100) = 20$ Mbps et les paquets de f_1 à la vitesse $C_{f_1}^{min} = 50$Mbps. La borne pire-cas pour f_1 est donc $d_1 = \frac{T_1}{C_{f_1}^{min}} + \frac{T_2}{C_{f_2}^{min}}$.

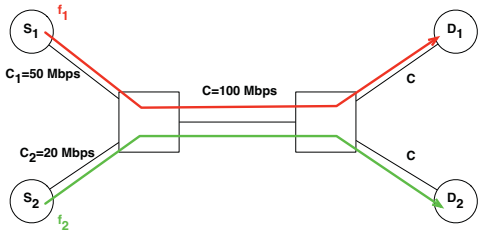


FIGURE 4.7 – Exemple de réseau avec des liens de vitesses différentes

Examinons maintenant ce qui se passe lorsqu'un terminal lit les données à une vitesse inférieure à celle du réseau. On appelle *taux de service*, noté C_T la vitesse à laquelle un terminal T consomme les données livrées dans son buffer de réception.

Si un paquet est livré par le réseau à une vitesse $C > C_T$, le buffer de réception du terminal va se remplir et le contrôle de flux va stopper la transmission du paquet. Lorsque le terminal aura lu 8 octets du buffer, un caractère FCT sera émis par l'interface réseau et la transmission pourra reprendre. Globalement, le paquet sera donc livré à la vitesse C_T exactement comme s'il traversait un lien de capacité C_T. Pour prendre en compte le taux de service du terminal, il suffit donc d'ajouter un lien de capacité C_T à la fin de $links(f)$ et d'intégrer ce lien au calcul de C_f^{min}.

4.1.6 Limites de cette méthode de calcul

Bien qu'intéressante, cette première méthode présente plusieurs limites non négligeables.

Pessimisme sur certaines configurations de réseau La première de ces limites est qu'en ne prenant pas en compte la quantité de trafic ou simplement la périodicité des flux, cette méthode est assez générale mais au prix d'un pessimisme parfois important. En effet, certaines configurations de réseau peuvent nous conduire à compter de nombreux paquets d'un même flux dans le délai pire cas d'un autre flux. Cela est notamment le cas pour les flux qui croisent le flux étudié juste avant sa destination.

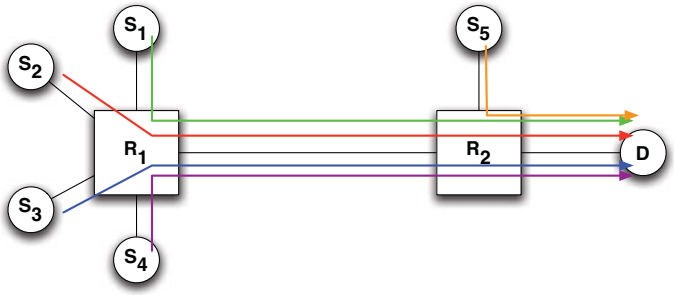


FIGURE 4.8 – Réseau sur lequel le délai calculé peut être pessimiste

Voyons cela sur un exemple. Sur la Figure 4.8, 5 flux $f_1, \ldots, f_5$ ont pour destination le terminal D. Si on applique la méthode de calcul récursive pour le flux f_1, on trouve

$$d(f_1) = \frac{L_1 + L_2 + L_3 + L_4 + 4.L_5}{C}. \tag{4.15}$$

Si L_5 est grand, ce délai risque d'être très élevé. Si l'intervalle minimal entre deux paquet de f_5 est faible, ce délai peut être atteint en pratique et, bien qu'élevé, il est réaliste. En revanche, si cet intervalle est suffisamment élevé, il n'est pas possible qu'autant de paquets de f_5 soient transmis avant un paquet de f_1. Malheureusement, sans hypothèse précise sur cet intervalle, nous sommes obligés de compter tous les paquets possibles de f_5 ce qui, sur un réseau de grande taille, peut rendre le délai pire cas calculé très pessimiste.

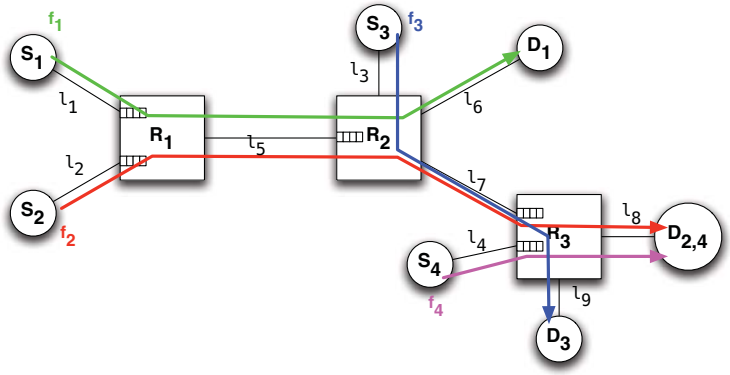


FIGURE 4.9 – Illustration du problème des paquets de faible taille

Les paquets de taille trop faible ne sont pas gérés Si un flux émet des paquets de taille inférieure à 64 × Nb_routeurs_traversés, notre hypothèse de livraison en deux phases ne tient plus. Il est alors possible que des fragments de plusieurs paquets d'un même flux soient présents dans le réseau au même instant. Dans ce cas, le délai pire-cas peut être plus élevé que la borne calculée par notre méthode.

Calculons par exemple, pour le réseau de la Figure 4.9, la borne pire-cas pour f_1. Sur ce réseau, tous les liens ont une capacité C. Le terminal $D_{2,4}$ a un taux de service $C_{2,4}$, c'est-à-dire qu'il lit les données reçues dans son buffer de réception à la vitesse $C_{2,4}$, plutôt qu'à la vitesse C. Ceci est pris en compte dans notre modèle par un lien supplémentaire l_{10} avec une capacité $C_{2,4}$ après le lien l_8.

Pour f_1, le calcul donne alors

$$\begin{aligned} d(f_1) &= d(f_1, l_1) = d(f_1, l_5) \\ &= d(f_1, l_6) + d(f_2, l_7) \\ &= d(f_1, null) + d(f_2, l_8) + d(f_3, l_9) \\ &= \frac{L_1}{C} + d(f_2, l_{10}) + d(f_4, l_{10}) + d(f_3, null) \\ &= \frac{L_1}{C} + d(f_2, null) + d(f_4, null) + \frac{L_3}{C} \\ &= \frac{L_1 + L_3}{C} + \frac{L_2 + L_4}{C_{2,4}} \end{aligned}$$

Cependant, si on se place dans le cas où $L_2 = 20$ caractères, il devient possible que 3 paquets de f_2 soient en attente dans le buffer d'entrée de R_3 au moment où, dans R_1 a lieu l'arbitrage entre un paquet de f_1 et un quatrième paquet de f_2, p_2^4. Dans le pire cas, un paquet de f_3 double p_2^4 et se retrouve ainsi bloqué derrière les trois paquets de f_2 déjà accumulés. De plus, dans le pire cas, chacun de ces paquets est entrelacé avec des paquets de f_4 ce qui augmente encore le délai. Enfin, le taux de service de $D_{2,4}$ peut être très faible, ce qui contribue a augmenter encore le délai pire cas. Dans ce cas, le délai pire cas serait $\frac{L_1+L_3}{C} + \frac{4.L_2+4L_4}{d_{2,4}}$ qui est bien supérieur à la borne calculée.

Comme on le voit, l'hypothèse sur la taille minimale des paquets est donc bien nécessaire pour que la borne calculée soit correcte. Ceci est dû au fait que, en supposant que la livraison d'un paquet a lieu en deux phases distinctes, on néglige l'impact des buffers d'entrée. Or, cette approximation n'est valable que lorsque les paquets sont suffisamment longs pour ne pas être stockés en entier dans les buffers intermédiaires.

Les paquets successifs d'un flux ne doivent pas être en conflit De même, il est nécessaire de supposer que les paquets successifs d'un flux n'interfèrent pas entre eux. En effet, si ça n'était pas le cas, un paquet pourrait arriver dans le buffer d'émission de son terminal source alors que le paquet précédent ne l'a pas encore quitté. Son délai de transmission pourrait alors être supérieur à la borne calculée car, en plus de son délai de transmission propre, il devrait attendre que l'autre paquet soit livré pour être transmis. Il est ainsi nécessaire de respecter un délai minimal entre les paquets d'un même flux.

4.2 Deuxième méthode récursive

Cette seconde méthode de calcul doit permettre de dépasser les limitations de la première méthode récursive tout en ne nécessitant pas d'hypothèses supplémentaires sur le trafic en entrée du réseau. Cette méthode est dérivée d'une méthode de calcul de délais pire-cas conçues pour les Networks-on-Chip best-effort (voir chapitre 3 et initialement présentée par Rahmati et al dans [21].

Le principe est le suivant. On décompose chaque routeur SpaceWire en un ensemble équivalent de routeurs élémentaires dont les buffers d'entrée sont de taille un caractère. On calcule ensuite le délai maximum que peut subir un paquet pour avancer d'un routeur élémentaire et on somme ces délais sur tout le chemin du flux. De plus, pour être certain d'obtenir un délai pire-cas, on suppose que tous les buffers du réseau sont saturés au début du calcul et que chaque flux tente d'émettre à la vitesse maximale.

Nous présentons d'abord en détail le modèle de routeur (4.2.1) puis la méthode de calcul elle-même (4.2.2 et plusieurs exemples d'application (4.2.3).

4.2.1 Modèle d'un routeur SpaceWire

Avec cette approche, nous cherchons à élaborer un modèle de routeur SpaceWire qui permette de dépasser les limitations de notre première approche. Pour mémoire, la principale limitation de cette méthode est qu'elle ne gère pas les paquets de faible taille car elle peut alors fournir des résultats optimistes par rapport au pire cas réel. Ceci est principalement lié au fait qu'avec cette méthode, on ne tient pas compte de l'impact des buffers d'entrée.

Prendre en compte ces buffers va nous permettre d'analyser plus finement les différents types de blocage possibles dans un routeur et d'en déduire un modèle de routeur SpaceWire qui simplifie l'analyse de ces blocages.

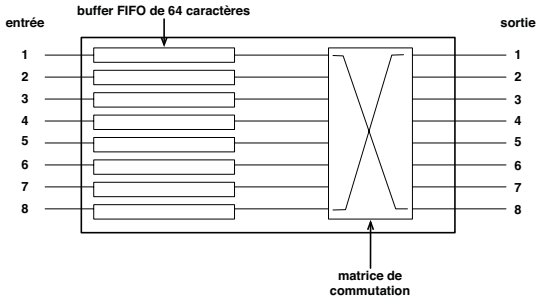


FIGURE 4.10 – Représentation schématique d'un routeur SpaceWire 8 × 8

La Figure 4.10 représente un routeur SpaceWire 8 × 8. Etant donné que chaque port fonctionne en *full-duplex*, nous pouvons considérer le port d'entrée et le port de sortie de chacun d'entre eux comme deux éléments indépendants dans notre modèle.

Chaque port d'entrée est connecté à une file d'attente qui tient lieu de buffer d'entrée. En général, cette file est de 64 octets pour un routeur SpaceWire. Les files

d'attente sont ensuite interconnectées avec les ports de sortie par une matrice de commutation.

4.2.1.1 Analyse des types de blocage

Dans un routeur wormhole, deux types de délais peuvent affecter un paquet. Le premier se produit lorsque le paquet demande l'accès à un port de sortie. Si un autre paquet utilise déjà ce port, le premier paquet ne peut obtenir l'accès au port et doit attendre la fin de la communication en cours. De plus, d'autres paquets peuvent également obtenir l'accès au port avant le paquet étudié.

Le second type de délai se produit lorsqu'un paquet p reste bloqué derrière un second paquet q dans la file d'attente d'un port d'entrée. Selon la longueur de q, son en-tête peut se trouver dans la même file, dans celle du routeur suivant, voire plus loin dans le réseau. Ainsi, même si q n'utilise pas le même port de sortie que p, p ne pourra pas accéder à son port de sortie tant que q n'aura pas complètement quitté la file d'attente.

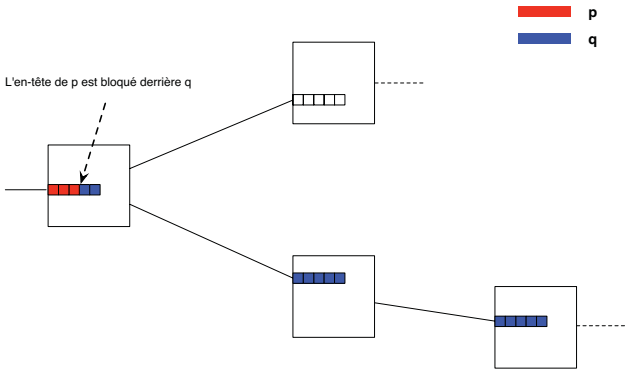


FIGURE 4.11 – Situation de blocage d'un paquet p dans un buffer d'entrée

La combinaison de ces deux types de délais engendre d'importantes variations sur les délais de bout en bout. Cependant, en décomposant chaque routeur en "routeurs élémentaires" plus aisé à analyser individuellement, il est possible d'obtenir un modèle simple permettant de déterminer une borne supérieure sur le délai de bout en bout.

4.2.1.2 Description du modèle

L'idée de base du modèle est que, en termes de délais, une file d'attente de 64 caractères est identique à un ensemble de 64 routeurs wormhole dotés d'un buffer d'un caractère et reliés par des liens de capacité infinie. Ce modèle est illustré sur la Figure 4.12.

On note $ER_{k,l}$ le l^{eme} routeur élémentaire ($l \in \{1, \ldots, 64\}$) associé au port d'entrée k, $k \in \{1, \ldots, 8\}$. Pour tout k, les routeurs $ER_{k,1}$ à $ER_{k,62}$ ont un seul port d'entrée et un seul port de sortie. Leur délai de commutation est de 0 seconde.

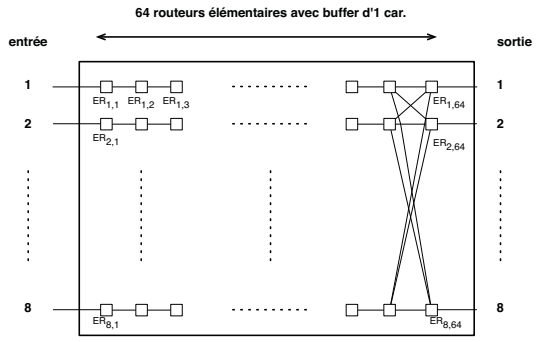


FIGURE 4.12 – Décomposition d'un routeur SpaceWire à l'aide de routeur élémentaires

Ils représentent simplement l'espace de stockage d'un caractère du buffer d'entrée du routeur.

Les routeurs $ER_{k,63}$ et $ER_{k,64}$ modélisent la matrice de commutation du routeur SpaceWire. Chaque routeur élémentaire $ER_{k,63}$ a un port d'entrée et 8 ports de sortie. Leur délai de commutation est égal au délai de commutation du routeur SpaceWire modélisé. Leur table de routage est également équivalente à celle du routeur SpaceWire. Pour tout k, $ER_{k,63}$ est connecté à tous les $ER_{k',64}$ afin de représenter les connexions effectuées par la matrice de commutation.

Chaque routeur $ER_{k,64}$ dispose de 8 ports d'entrée et d'un port de sortie relié à un routeur élémentaire d'un autre routeur SpaceWire. Une file d'attente d'un caractère est associée à chacun de leur port d'entrée. Les routeurs élémentaires de la dernière colonne modélisent les conflits d'accès aux ports de sortie et réalisent l'arbitrage entre leurs différents ports d'entrée grâce au même mécanisme de Round-Robin que le routeur SpaceWire modélisé.

Dans ce modèle, l'en-tête d'un paquet ne peut avancer que lorsque le routeur élémentaire immédiatement en aval est libre. Dans ce cas, le corps du paquet avance tout entier d'un cran le long du chemin du paquet. Le routeur élémentaire utilisé par le dernier caractère du paquet est ainsi libéré.

Toutefois, ce modèle ne reproduit pas exactement le fonctionnement d'un routeur SpaceWire. En effet, comme chaque routeur ER_{k64} contient huit files d'attente d'un caractère, le modèle semble disposer de plus de mémoire que le routeur réel. Pour corriger ce problème, il faut faire en sorte qu'à chaque instant, un seul routeur $ER_{k',64}, k' \in \{1, \ldots, 8\}$ puisse contenir un caractère provenant d'un routeur $ER_{k,63}$ donné.

Ainsi, la condition nécessaire pour qu'un caractère avance d'un routeur $ER_{k,63}$ à un routeur $ER_{k',64}$ n'est plus seulement que ce routeur soit libre comme c'est le cas pour les autres routeurs élémentaires. La condition est maintenant que tous les routeurs $ER_{k',64}, k' \in \{1, \ldots, 8\}$ soient libres. Avec cette condition , il est facile de voir que le flux de caractères provenant d'un port d'entrée n'utilise jamais plus de 64 caractère dans un routeur SpaceWire donné.

La Figure 4.13 illustre cette situation. Sur cette Figure, si le paquet vert pouvait

avancer vers le routeur $ER_{8,64}$ pendant que le paquet rouge utilise le routeur élémentaire $ER_{1,64}$, le flux de caractères en provenance du port d'entrée 1 occuperait 65 caractères, ce qui est impossible en réalité. En revanche, avec la nouvelle condition nécessaire, le paquet vert n'avancera que lorsque le paquet rouge aura encore avancé d'un cran, ce qui correspond à la réalité.

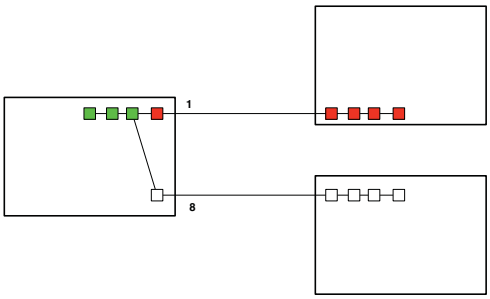


FIGURE 4.13 – Illustration de la condition nécessaire pour faire avancer un paquet vers le dernier routeur élémentaire

Notons que, comme tous les autres routeurs ne sont reliés qu'à un seul routeur aval, la nouvelle condition est également valable pour eux. Ceci permet de n'utiliser qu'un seul type de routeur élémentaire pour modéliser tout le réseau dont la condition est la suivante : "un routeur $ER_{k,l}$ peut faire avancer un caractère si et seulement si tous les routeurs élémentaires reliés à ses ports de sortie sont libres".

Nous pouvons maintenant utiliser ce modèle de routeur pour calculer une borne supérieure sur le délai pire-cas de bout en bout de chaque flux.

4.2.2 Calcul d'une borne sur le délai pire-cas

Le calcul a lieu en deux phases. Tout d'abord, le réseau SpaceWire étudié doit être transformé en un réseau équivalent constitué de routeurs élémentaires et de terminaux reliés par des liens unidirectionnels. Ensuite, ce réseau peut être utilisé pour calculer une borne sur le délai pire-cas de chaque flux.

4.2.2.1 Hypothèses

Comme pour la première méthode, on considère un réseau SpaceWire constitué de terminaux et de routeurs qui les interconnectent grâce à des liens SpaceWire. Chaque terminal peut émettre et/ou recevoir des données et ne dispose que d'une seule interface réseau.

Nous ne faisons pas d'hypothèses sur la périodicité des flux ni sur l'intervalle minimum séparant deux paquets. Ceci nous permet de prendre en compte les terminaux sans régulation de trafic qui constituent la majorité des terminaux disponibles aujourd'hui. Chaque terminal peut donc tenter d'émettre un paquet à n'importe quel moment. De plus, pour être certain de couvrir le pire scénario possible, nous faisons l'hypothèse que les buffers de tous les routeurs sont remplis au début du calcul et que tous les flux génèrent des paquets à la plus grande vitesse possible.

Nous ne faisons pas non plus d'hypothèse sur la taille des paquets.
Nous faisons également les hypothèses suivantes :
- le routage est statique
- tous les liens ont la même capacité C
- un terminal qui reçoit un paquet le lit aussi vite que le réseau le transmet
- chaque terminal a besoin d'un délai constant d_{inj} pour injecter un paquet dans le réseau
- chaque terminal a besoin d'un délai constant d_{ej} pour éjecter un paquet du réseau

4.2.2.2 Génération du réseau élémentaire

La première étape est de remplacer chaque routeur SpaceWire par un ensemble de routeurs élémentaires équivalents comme expliqué en Section 4.2.1.2. Ainsi, chaque flux de données va suivre un chemin constitué de routeurs élémentaires reliés par des liens unidirectionnels.

Notons h_i le nombre de routeurs élémentaires suivis par le flux i et R_i^j, $j \in \{1, \ldots, h_i\}$ le j^{eme} routeur traversé par f_i. Remarquons que h_i ne compte que les routeurs élémentaires qui représentent les routeurs SpaceWire. Ceci aura son importance par la suite.

Il nous faut ensuite modéliser les terminaux. Chacun d'entre eux peut avoir deux rôles à jouer : source et destination. Ces rôles doivent être modélisés séparément puisque notre modèle utilise des liens unidirectionnels.

Un terminal source injecte dans le réseau des paquets appartenant à un ou plusieurs flux à la vitesse maximum. Tous ces flux sont en compétition pour l'accès au port de sortie du terminal. Dans ce modèle, on considère que cet accès est arbitré par un mécanisme de type Round-Robin similaire à celui des routeurs. Par conséquent, il est possible de considérer un terminal source comme un ensemble de sources élémentaires (une par flux) connectée à un routeur élémentaire qui réalise l'arbitrage entre elles. Nous noterons ce routeur élémentaire R_i^0 pour chaque flux f_i (voir Figure 4.14).

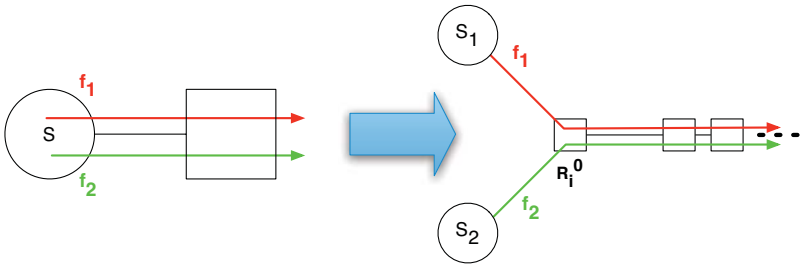


FIGURE 4.14 – Modèle d'un terminal source

Symétriquement, un terminal destination reçoit des paquets appartenant à un ou plusieurs flux et les lit au fur et à mesure qu'ils arrivent. Le modèle de destination doit rendre compte de deux phénomènes : la livraison complète du paquet et le

délai fixe associé à la lecture de chaque paquet. Cependant, modéliser ainsi chaque destination comme un seul bloc élémentaire ne permet pas d'obtenir le délai de bout en bout correct pour un paquet.

En effet, en additionnant les délais subis par un paquet dans chaque routeur élémentaire traversé, on obtient uniquement le délai permettant à l'en-tête du paquet d'atteindre la destination. Pour obtenir le délai de transmission du paquet complet, nous avons choisi d'ajouter $L_i - 1$ routeurs élémentaires au chemin suivi par chaque flux f_i après $R_i^{h_i}$. Ainsi, chaque destination est modélisée par un routeur élémentaire qui reçoit tous les flux à destination de ce terminal et les répartit sur des chemins formés de $L_i - 2$ routeurs élémentaires supplémentaires (voir Figure 4.15).

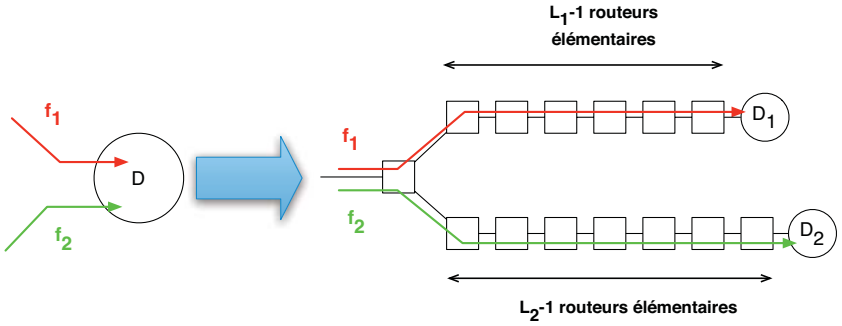


FIGURE 4.15 – Modèle d'un terminal destination

Tous ces routeurs additionnels sont reliés par des liens de capacités infinies. Lorsque le paquet avance le long de ses routeurs élémentaires additionnels, il traverse encore au moins un lien de vitesse finie, ce qui nous permet de compter le délai de livraison du paquet correctement. Enfin, chacun des chemins se termine par une destination élémentaire qui modélise le délai fixe d_{ej} et marque la fin du transfert du paquet.

Au final, un paquet du flux f_i doit traverser $h_i + L_i$ routeurs élémentaires numérotés de 0 à $h_i + L_i - 1$ pour atteindre sa destination élémentaire. Nous obtenons ainsi un modèle complet du réseau à basé de routeurs élémentaires que nous pouvons maintenant utiliser pour calculer une borne sur le délai pire-cas de chaque flux.

4.2.2.3 Calcul de la borne pire-cas

Notre méthode de calcul s'inspire de la méthode RTB-HB présentée dans [21] et nous utiliserons des notations similaires.

Considérons un paquet p_i du flux f_i. Soit u_i^j le délai maximum nécessaire pour que l'en-tête de p_i (i.e. le premier caractère) avance du routeur R_i^j au routeur R_i^{j+1}. Le corps du paquet avance simultanément d'un routeur élémentaire à la suite de l'en-tête.

La borne supérieure (*Upper-Bound*) sur le délai de bout en bout du flux f_i peut

alors s'exprimer ainsi :

$$UB_i = d_{inj} + d_{ej} + \sum_{j \in \{0,\ldots,h_i+L_i-2\}} u_i^j \tag{4.16}$$

UB_i représente la durée maximale nécessaire pour que p_i traverse tous les routeurs élémentaires le long de son chemin.

De même, l'Intervalle Minimal entre l'émission de deux paquets d'un même flux est donné par

$$MI_i = d_{inj} + \sum_{j \in \{0,\ldots,L_i-2\}} u_i^j \tag{4.17}$$

MI_i représente le délai minimum nécessaire pour qu'un paquet sorte complètement du buffer source. A partir de cette métrique, on peut déduire la bande passante maximum disponible pour chaque flux et, par suite, déterminer si un réseau est capable de transmettre tous le trafic requis.

Calcul de u_i^j Il y a deux causes de délai possibles dans un routeur élémentaire.

D'une part, puisque nous supposons que le réseau est totalement rempli de paquets, le routeur élémentaire aval R_i^{j+1} contient le dernier caractère d'un autre paquet $p_{i'}$. Il faut donc que $p_{i'}$ avance d'un cran pour que p_i puisse avancer lui aussi. L'en-tête de $p_{i'}$ se situe dans le routeur $R_{i'}^{j+L_{i'}}$ et lorsqu'il avance, tout le paquet $p_{i'}$ le suit. Par conséquent, le délai pour que $p_{i'}$ libère R_i^{j+1} est $u_{i'}^{j+L_{i'}}$. Cette situation est illustrée sur la Figure 4.16.

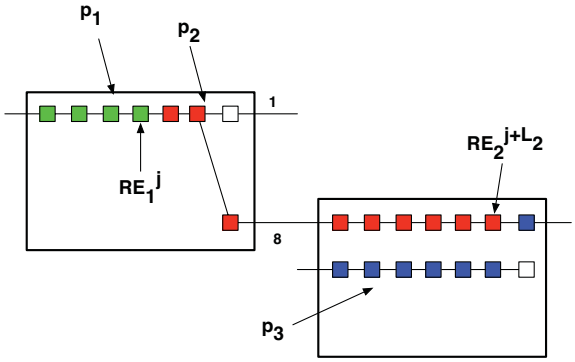


FIGURE 4.16 – Premier cas pour u_i^j : un paquet est bloqué derrière un paquet d'un autre flux

Soit $z_c(i,j)$ le nombre de flux en conflits avec f_i dans R_i^j. Dans le cas des routeurs élémentaires qui ne comportent qu'un port de sortie, $z_c(i,j)$ est égal au nombre de flux passant par le routeur, en excluant f_i. Pour les routeurs élémentaires comportant plusieurs ports de sortie, il nous faut nous assurer que tous ces ports sont libres avant que p_i ne puisse avancer. Par conséquent, nous pouvons également considérer que tous les flux qui passent par R_i^j sont en conflit avec f_i pour l'accès au port de sortie.

Par ailleurs, $p_{i'}$ peut également appartenir au flux f_i lui-même. Cette situation est illustrée sur la Figure 4.17.

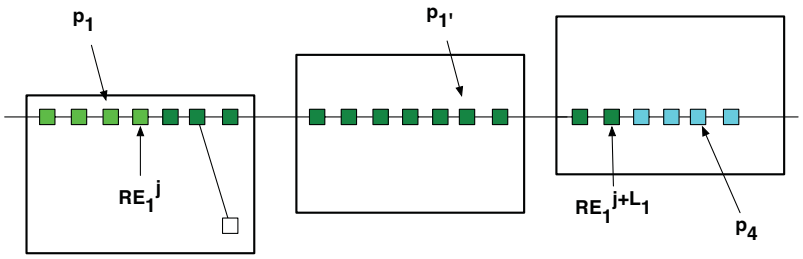


FIGURE 4.17 – Second cas pour u_i^j : un paquet est bloqué derrière un paquet du même flux

Au final, pour chaque routeur élémentaire, le délai causé par $p_{i'}$ dans le pire des cas est $\max[u_i^{j+L_i}, \max_{i' \in \{1,\ldots,z_c(i,j)\}}(u_{i'}^{j+L_{i'}})]$.

D'autre part, des paquets provenant d'autres ports d'entrée et utilisant le même port de sortie peuvent également retarder p_i. Dans le cadre d'un réseau wormhole, lorsqu'un autre paquet utilise déjà le port de sortie, p_i doit attendre que ce paquet ait été complètement transféré avant de pouvoir accéder au port de sortie.

Cependant, comme nous l'avons montré en Section 4.1, grâce au Round-Robin, un seul paquet par autre port d'entrée peut passer avant p_i. Pour obtenir le délai pire-cas causé par l'ensemble des flux entrant par un port donné, il faut donc calculer le délai que chaque flux pourrait causé puis prendre le maximum de ces délais. Les délais associés à chaque port d'entrée peuvent ensuite être additionnés pour obtenir le délai maximum subi par p_i.

En outre, un paquet $p_{i'}$ entrant dans R_i^j par un port d'entrée différent de celui de p_i doit avancer de $L_{i'}$ crans dans le réseau avant de libérer le port de sortie. Le délai causé par $p_{i'}$ est donc $\sum_{k=1}^{L_{i'}} u_{i'}^{j+k}$.

Soit $I_i^j = \{\text{ports d'entrée de } R_i^j \text{ sauf celui de } f_i\}$ et $F_l^j = \{\text{flux entrant dans } R_i^j \text{ par le lien } l\}$. Le délai pire-cas causé par l'ensemble des flux provenant d'autres ports d'entrée que p_i est donné par

$$\sum_{l \in I_i^j} \left(\max_{i' \in F_l^j} \{ \sum_{k=1}^{L_{i'}} u_{i'}^{j+k} \} \right).$$

Au final,

$$u_i^j = \max[u_i^{j+L_i}, \max_{i' \in \{1,\ldots,z_c(i,j)\}} (u_{i'}^{j+L_{i'}})] + \sum_{l \in I_i^j} \left(\max_{i' \in F_l^j} \{ \sum_{k=1}^{L_{i'}} u_{i'}^{j+k} \} \right) \tag{4.18}$$

(4.18) peut maintenant être utilisée pour calculer récursivement (4.16) et (4.17) pour chaque flux f_i. Le calcul prend fin lorsque chaque paquet atteint sa destination élémentaire. La condition d'arrêt de la récursion peut s'écrire ainsi :

$$\forall j > h_i, u_i^j = \frac{10}{C} \tag{4.19}$$

puisque les caractères SpaceWire sont de taille 10 bits.

Nous avons réalisé une implémentation en Java de cette méthode de calcul afin de valider son bon fonctionnement.

4.2.2.4 Prise en compte des liens de capacités différentes

Cette méthode de calcul peut être adaptée pour prendre en compte des liens de capacités différentes ou un taux de service dans les terminaux destinations.

Le principe est le suivant. Un paquet p_i ne peut avancer qu'à condition que le paquet situé dans le routeur immédiatement en aval avance lui aussi. Comme ce paquet est lui-même bloqué par un autre paquet, au final, p_i ne peut avancer qu'à la vitesse du paquet le plus lent en aval. Ce paquet peut ne pas interférer avec p_i directement mais seulement par l'intermédiaire d'autres paquets qui bloquent p_i.

De plus, si l'un des liens actuellement utilisé par p_i est plus lent que ceux utilisés par les paquets en aval, p_i avancera seulement à la vitesse de ce lien.

La capacité des liens traversés n'étant prise en compte que lors de la phase finale du calcul, nous avons choisi d'ajouter un paramètre c à u_i^j durant le calcul. c représente la capacité la plus faible rencontrée par un paquet lors de son transfert. La condition d'arrêt devient alors

$$\forall j > h_i, u_i^j = \frac{10}{C_{min}} \tag{4.20}$$

où C_{min} est la capacité la plus faible rencontrée par un paquet durant toute la récursion.

A chaque calcul de $u_i^j(C)$, on compare le paramètre C et la capacité la plus faible parmi celles des liens actuellement utilisés par p_i, c'est-à-dire entre ER_i^j et $ER_i^{j-L_i}$. La capacité la plus faible est ensuite passée en paramètre lors des calculs de $u_{i'}^j(C')$ nécessaires.

4.2.3 Application de la méthode de calcul

4.2.3.1 Exemple simple de calcul

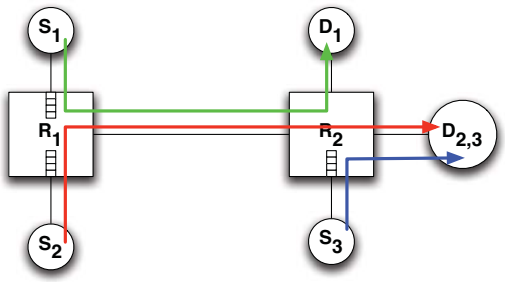


FIGURE 4.18 – Exemple de réseau

Nous illustrerons l'application de notre seconde méthode de calcul sur l'exemple de la Figure 4.18. Pour simplifier, nous considérerons que les routeurs ont des buffers

de taille $B = 4$ caractères. Nous utiliserons les valeurs suivantes pour les tailles des paquets : $L_1 = 4$, $L_2 = 8$ et $L_3 = 6$ caractères.

Le flux f_1 traverse 8 routeurs élémentaires associées aux routeurs SpaceWire R_1 et R_2 : les routeurs élémentaires R_1^1 à R_1^4 sont associés à R_1 ; les routeurs élémentaires R_1^5 à R_1^8 à R_2. Comme S_1 n'est source que du seul flux f_1, le routeur virtuel R_1^0 n'est partagé par aucun autre flux. Enfin, comme D_1 n'est pas partagé non plus, f_1 traverse 8 routeurs élémentaires R_1^9 à R_1^{16} qui représentent la lecture du paquet complet par la destination.

On procède de façon identique pour f_2 et f_3. Notons que f_1 et f_2 partagent le buffer d'entrée de R_2. Par conséquent, on a $R_1^i = R_2^i$ pour $i = 4, \ldots, 7$. De plus, comme f_2 et f_3 ont la même destination $D_{2,3}$, ils se partagent le premier routeur élémentaire associé à ce terminal, d'où $R_2^8 = R_3^4$ et $R_2^9 = R_3^5$.

Par définition, on a

$$UB_1 = \sum_{j=0}^{15} u_1^j \tag{4.21}$$

En appliquant la définition de u_i^j, on trouve $u_1^0 = u_1^8$ car R_1^0 n'est pas partagé et $L_1 = 8$. Puis, $u_1^8 = u_1^{16} = \frac{10}{C}$. De même, $u_1^j = \frac{10}{C}$ pour $j = 1, 2, 3$.

Dans R_1^4, f_1 est en conflit avec f_2. Par conséquent,

$$u_1^4 = \max(u_1^{12}, u_2^8) + \sum_{k=1}^{4} u_2^{4+k}$$

On a ensuite $u_1^{12} = \frac{10}{C}$ et $u_2^8 = \max(u_2^{12}, u_3^{14}) + \sum_{k=1}^{6} u_3^{4+k}$ car f_2 et f_3 sont en conflit à partir de $R_2^8 = R_3^4$.

On voit facilement que $\forall k = 1, \ldots, 6, u_3^{4+k} = \frac{10}{C}$ et que $u_2^{12} = u_3^{14} \frac{10}{C}$. Par suite, $u_2^8 = 7.\frac{10}{C}$ et $u_1^4 = 11.\frac{10}{C}$.

Ensuite, $u_1^5 = \max(u_1^{13}, u_2^9) = \frac{10}{C}$. De même, $u_1^6 = \max(u_1^{14}, u_2^{10}) = \frac{10}{C}$ et $u_1^7 = \max(u_1^{15}, u_2^{11}) = \frac{10}{C}$. Enfin, on a clairement $\forall j \in \{8, \ldots, 16\}, u_1^j = \frac{10}{C}$ car il n'y a plus de conflits sur la fin du trajet.

Ainsi, au total $UB_1 = 26.\frac{10}{C}$ secondes.

4.2.3.2 Avantages par rapport à la première méthode

Cette seconde méthode de calcul avait pour buts de dépasser deux des limitations de la première méthode de calcul sans nécessiter d'hypothèses supplémentaires sur la périodicité des flux de données. D'une part, elle doit permettre de prendre en compte des paquets de très faible taille. D'autre part, elle doit prendre en compte les interactions entre plusieurs paquets d'un même flux qui se trouveraient simultanément dans le réseau.

Pour illustrer ces deux points, reprenons l'exemple de la Figure 4.9, reproduit ci-dessous sur la Figure 4.19.

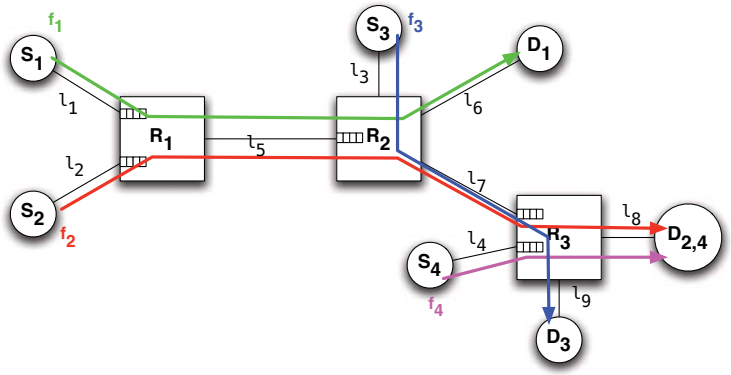


FIGURE 4.19 – La seconde méthode prend en compte les paquets de faible taille

Prise en compte des paquets de faible taille Avec la seconde méthode, le cas où $L_2 = 20$ octets est correctement pris en compte. Considérons l'application numérique suivante :

Flux	Taille des paquets (octets)
f_1	4000
f_2	20
f_3	1000
f_4	20

Les liens fonctionnent avec une capacité $C = 200$ Mbps et le terminal $D_{2,4}$ a un taux de service $C_{2,4} = 2$ Mbps.

Le délai calculé par la première méthode pour f_1 est $d(f_1) = \frac{L_1+L_3}{C} + \frac{L_2+L_4}{C_{2,4}}$ (voir Section 4.1.6). On a vu à la même Section que si $L_2 = 20$ octets, le délai peut en réalité être au moins $\frac{L_1+L_3}{C} + \frac{4.L_2+4.L_4}{C_{2,4}}$. Le délai pour la seconde méthode est calculée à l'aide de notre implémentation.

Les résultats sont les suivants :

Méthode	Délai pour f_1
1ère méthode	0,45 ms
Délai pire-cas possible	1,05 ms
2ème méthode	3,67 ms

Comme on peut le voir, la première méthode calcule une borne non valide, inférieure au délai subi par f_1 dans une situation atteignable. En revanche, la seconde méthode fournit une borne valide mais pessimiste. Nous reviendrons sur le pessimisme de cette méthode ultérieurement.

Prise en compte des interférences dues au paquet du même flux D'autre part, la seconde méthode permet de prendre en compte les interactions entre les paquets d'un même flux. Nous pouvons l'illustrer à l'aide du flux f_4 sur l'exemple précédent.

Avec la première méthode, la borne calculée pour f_4 est

$$d(f_4) = d(f_2, null) + d(f_4, null) = \frac{L_2 + L_4}{C_{2,4}}.$$

En réalité, puisque les paquets de f_4 sont de longueur $L_4 = 20$ octets, trois d'entre eux peuvent être en attente dans le buffer d'entrée de R_3 pendant que $D_{2,4}$ lit un paquet de f_2. Chacun de ces paquets peut être entrelacés avec un paquet de f_2 ce qui donne un délai d'au moins $4.\frac{L_2+L_4}{D_{2,4}}$.

L'application numérique est la suivante :

Méthode	Délai pour f_4
1ère méthode	0,2 ms
Délai pire-cas possible	0,8 ms
2ème méthode	0,86 ms

Sur cet exemple, la seconde méthode calcule une borne correcte et qui n'est pas pessimiste par rapport au cas réel exhibé ici.

4.2.3.3 Limite de cette méthode

La limite de cette méthode est le pessimisme engendré par la présence de liens de vitesse faible et par la prise en compte des messages de faible taille et de leurs interactions entre eux.

Par exemple, sur le réseau de la Figure 4.19, la borne calculée par la seconde méthode pour le flux f_2 est beaucoup plus pessimiste que celle du flux f_4.

Etudions d'abord la transmission d'un paquet de f_2 en supposant que tous les buffers de son chemin sont remplis de paquet de f_2. f_2 traversent les trois routeurs R_1, R_2 et R_3. Le buffer d'entrée de chacun de ces routeurs peut contenir 3 paquets de f_2 au début du calcul.

- Chacun des paquets bloqués dans R_3 peut être bloqué par un paquet de f_4. Le délai dû à ces paquets est donc $3.\frac{L_2+L_4}{C_{2,4}}$.
- Chaque paquet bloqué dans R_2 peut être bloqué par un paquet de f_3 et par un paquet de f_4. Le délai dû à ces paquets est donc $3.\frac{L_2+L_4}{C_{2,4}} + 3.\frac{L_3}{C}$.
- Chaque paquet bloqué dans R_1 peut être bloqué par un paquet de f_1, un paquet de f_3 et un paquet de f_4. Le délai dû à ces paquets est donc $3.\frac{L_2+L_4}{C_{2,4}} + 3.\frac{L_1+L_3}{C}$.
- Enfin, le paquet de f_2 étudié qui est émis au moment où les neufs paquets précédents sont bloqués dans les buffers intermédiaires peut aussi être bloqué par un paquet de f_1, de f_3 et de f_4.

Au total, un paquet de f_2 peut donc subir au moins un délai

$$d_2 = \frac{4.L_1 + 7.L_3}{C} + 10.\frac{L_2 + L_4}{C_{2,4}}.$$

Avec les valeurs choisies ici, on obtient un délai de $3,15$ ms. Or,la borne calculée par la seconde méthode est de $17,9$ ms. Ce résultat est fortement pessimiste pour

un petit réseau sur lequel ne circule que peu de flux. Le pessimisme de la méthode s'explique par le fait que le calcul du délai pire-cas se fait octet par octet et que le calcul n'a pas de "mémoire".

En effet, à chaque pas d'un octet, on cherche la pire situation possible sans vérifier que la séquence de situation pire-cas choisie est possible en réalité. Par exemple, dans le pire-cas déterminé manuellement ci-dessus, un paquet p_2 de f_2 bloqué dans le buffer d'entrée de R_2 peut être bloqué par un paquet de f_3. Celui-ci peut être bloqué derrière trois paquets de f_2 chacun bloqué par un paquet de f_4. Le délai de blocage subi par p_2 est donc $\frac{L_3}{C} + 3.\frac{L_2+L_4}{C_{2,4}}$.

En revanche, comme la méthode de calcul procède pas à pas, elle compte que p_2 subit le blocage précédent puis que, lorsque p_2 arrive dans le buffer d'entrée de R_3, celui-ci contient à nouveau trois paquets du flux f_2 qui vont être bloqués chacun par un paquet du flux f_4. Ce phénomène se reproduit pour chaque paquet de f_2 contenu dans les buffers de R_1 et R_2 ce qui conduit à une borne pessimiste.

Cependant, le pessimisme de la méthode est directement lié à la présence d'un lien de capacité réduite dans le réseau. Ainsi, si on choisit comme valeur $C_{2,4} = C = 200$ Mbps, on a $d_2 = 1,2$ ms et la borne calculée par la seconde méthode est $1,2$ ms également. Le pessimisme de la méthode n'apparaît donc bien dans ce cas que lorsqu'un lien lent existe dans le réseau.

4.3 Bilan global sur cette approche

Dans ce chapitre, nous avons présenté une approche du calcul de délai pire-cas de bout en bout sans hypothèses fortes sur le trafic en entrée du réseau. Cette approche est divisée en deux méthodes de calcul aux hypothèses différentes. Cependant, aucune de ces méthodes ne nécessitent d'hypothèses sur la périodicité des flux.

La première méthode de calcul suppose que les paquets sont suffisamment longs pour réserver un trajet complet de la source à la destination lors de leur transfert à travers le réseau. Elle suppose également que les paquets d'un même flux n'interfèrent pas entre eux. Elle fournit une fonction récursive qui permet de calculer assez simplement le délai de bout en bout d'un paquet traversant un réseau SpaceWire. Sur l'exemple présenté, elle nous a permis de mettre en valeur le fait que, parmi plusieurs flux ayant la même destination, le fonctionnement du SpaceWire favorise les flux dont la source est proche de la destination au détriment des flux dont la source est plus éloignée. Cette méthode n'est pas suffisante en pratique car les paquets des flux critiques sont souvent de faible taille. Il est donc nécessaire de pouvoir les inclure dans l'analyse.

Nous avons donc mis au point une seconde méthode de calcul qui lève les restrictions de la méthode précédente. Cette méthode peut prendre en compte des paquets de n'importe quelle taille et prend en compte les interactions entre les paquets successifs d'un même flux. Ceci est réalisé en décomposant les routeurs SpaceWire en routeurs élémentaires dotés d'un buffer d'un octet et en supposant au cours de calcul

que des paquets sont déjà présents dans le réseau. De plus, à chaque étape du calcul, on suppose que les paquets présents dans le réseau créent la pire situation possible, même si cela mène globalement à un ordonnancement des paquets irréalisable et plus pessimiste que le pire cas possible en réalité. Ce pessimisme ressort surtout dans un réseau contenant des liens de vitesse plus faible que le reste du réseau. Au final, le fait que cette méthode compte de nombreux paquets excédentaires au cours du calcul met en relief la nécessite d'une seconde approche qui prenne en compte la périodicité des paquets afin de restreindre le nombre de paquets comptés.

Chapitre 5

Approche basée sur le Calcul Réseau

Sommaire

Ce chapitre présente notre deuxième approche du calcul de borne supérieure sur le délai pire-cas d'un paquet. Cette approche se base sur la théorie du Calcul Réseau. Un bref rappel de cette théorie est présentée en Section 5.1. Nous présentons ensuite le nouvel élément du Calcul Réseau que nous proposons en 5.2 puis le calcul d'une courbe de bout en bout en 5.3. Puis, nous donnons plusieurs façons d'améliorer cette courbe et nous illustrons les limites de cette méthode. Enfin, nous faisons le bilan dans la dernière section.

5.1 Introduction au Calcul Réseau

Cette Section présente rapidement les notions de Calcul Réseau indispensables à la compréhension de la suite du chapitre.

Le Calcul Réseau est une théorie basée sur l'algèbre Min-Plus qui a été créée pour étudier les systèmes à files d'attente déterministes tels que ceux présents dans les réseaux de télécommunication.

Dans le cadre de cette théorie, un flux est représenté par une fonction cumulative $x(t)$, définie comme le nombre de bits passant par un point donné du réseau pendant l'intervalle $[0, t]$. Chaque élément réseau S est considéré comme un système dans lequel entre un flux $x(t)$ et duquel sort un flux $y(t)$ (voir Figure 5.1). On cherche alors à caractériser y en fonction de x et des propriétés de l'élément réseau.

FIGURE 5.1 – Un élément réseau S étudié par le Calcul Réseau

Courbe d'arrivée Pour qu'il soit possible de fournir des garanties aux flux, il est nécessaire de limiter la quantité de données émises par chaque flux. Autrement, les délais de transmission ne pourraient pas être bornés. Pour ce faire, le Calcul Réseau utilise la notion d'enveloppe de trafic ou courbe d'arrivée.

Définition 1. Courbe d'arrivée
Soit α une fonction non-décroissante définie pour tout $t \geq 0$. On dit qu' α est une courbe d'arrivée *pour x si et seulement si*

$$\forall s \leq t, x(t) - x(s) \leq \alpha(t - s)$$

α limite la quantité de données que peut transmettre le flux x durant un intervalle de temps, uniquement en fonction de la durée de cet intervalle.

Courbe de service Inversement, chaque élément réseau doit fournir une garantie sur la quantité minimale de trafic qu'il va transférer pendant une période de temps donnée. Ceci est modélisé par une courbe de service.

Définition 2. Courbe de service
Soit un système (élément réseau) S et un flux traversant S avec comme fonction d'entrée x et comme fonction de sortie y. On dit que S offre une courbe de service β au flux si et seulement si β est positive et non décroissante avec $\beta(0) = 0$ et, pour tout $t \geq 0$, il existe $t_0 \geq 0$, $t_0 \leq t$ tel que

$$y(t) - x(t_0) \geq \beta(t - t_0)$$

Lorsqu'on connaît à la fois la courbe d'arrivée d'un flux et la courbe de service offerte par le système qu'il traverse, on obtient une borne supérieure sur le délai de traversée en prenant la distance horizontale maximum entre ces deux courbes $h(\alpha, \beta)$. De même, on obtient une borne sur le backlog maximum du système en prenant la distance verticale entre les deux courbes $v(\alpha, \beta)$. La Figure 5.2 illustre ces notions avec une courbe d'arrivée α affine et une courbe de service β de type Rate-Latency.

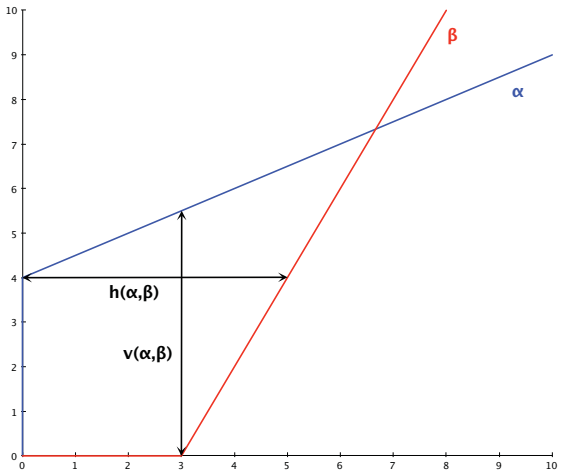


FIGURE 5.2 – Exemple de courbe d'arrivée (en bleu) et de courbe de service (en rouge)

On peut également définir la notion de courbe de service stricte :

Définition 3. Courbe de service stricte
Un système S offre une courbe de service stricte β si pendant toute période "occupée" de durée u, la quantité de donnée transmise est égale à $\beta(u)$. Une période occupée du système est un intervalle de temps pendant lequel le système a des données à transmettre.

Si S offre une courbe de service stricte β, alors β est aussi une courbe de service offerte par S. La réciproque est fausse.

Convolution Min-Plus L'opération la plus utilisée par le Calcul Réseau est la convolution Min-Plus.

Définition 4. Convolution Min-Plus
Soient α et β deux fonctions non-décroissantes telles que $\alpha(t) = \beta(t) = 0$ pour $t \leq 0$. La convolution min-plus de α et β est définie par

$$(\alpha \otimes \beta)(t) = \inf_{0 \leq s \leq t} \{\alpha(t-s) + \beta(s)\}$$

Grâce à cette opération, la définition d'une courbe d'arrivée peut être reformulée en $x \leq \alpha \otimes x$ et celle d'une courbe de service en $y \geq \beta \otimes x$.

Nous pouvons maintenant donné un théorème essentiel du Calcul Réseau :

Théorème 2. *Soit un flux traversant successivement deux systèmes S_1 et S_2. Supposons que S_1 et S_2 offrent respectivement des courbes de service β_1 et β_2. Alors la concaténation de ces deux systèmes offre une courbe de service $\beta_1 \otimes \beta_2$ au flux.*

Ce théorème permet de combiner des éléments réseau traversés successivement par un flux pour obtenir la courbe de service de bout en bout offerte au flux par le réseau dans son ensemble.

Par ailleurs, traverser un système affecte la courbe d'arrivée d'un flux. Lorsqu'un flux de courbe d'arrivée α traverse un système offrant une courbe de service β, sa courbe d'arrivée en sortie du système devient $\alpha^* = \alpha \oslash \beta$ où $\oslash$ est la déconvolution min-plus de α et β définie comme suit.

Définition 5. Déconvolution min-plus
Soient α et β deux fonctions non-décroissantes telles que $\alpha(t) = \beta(t) = 0$ pour $t \leq 0$. La déconvolution min-plus de α et β est définie par

$$(\alpha \oslash \beta)(t) = \sup_{s \geq 0} \{\alpha(t+s) - \beta(s)\}$$

Une autre opération utile est la clôture positive non-décroissante de β définie par

Définition 6. Clôture positive non-décroissante
Soit une fonction β. La clôture positive non-décroissante *de β est définie par*

$$(\beta)_{\uparrow}(t) = \max(\sup_{0 \leq s \leq t} \beta(s), 0)$$

Courbe de service maximale Une dernière notion qui nous sera utile par la suite est celle de courbe de service maximale.

Définition 7. Courbe de service maximale
Soit un système S et un flux traversant S avec comme fonction d'entrée x et comme fonction de sortie y. On dit que S offre une courbe de service maximale *γ au flux si γ est fonction non-décroissante telle que $\gamma(t) = 0$ pour tout $t \leq 0$ et $y \leq x \otimes \gamma$.*

Nous utiliserons le théorème suivant dans notre modèle :

Théorème 3. *Soit un flux contraint par une courbe d'arrivée α et traversant un système offrant une courbe de service β et une courbe de service maximale γ. Alors le flux en sortie du système est contraint par la courbe d'arrivée $\alpha^* = (\alpha \otimes \gamma) \oslash \beta$.*

En général, on a $(\alpha \otimes \gamma) \oslash \beta \leq \alpha \oslash \beta$ ce qui permet d'obtenir une meilleure caractérisation du flux en sortie.

Pour une introduction plus détaillée au Calcul Réseau, on pourra consulter [25] et [23].

5.2 Un nouvel élément réseau : la section wormhole

Avant de pouvoir utiliser le Calcul Réseau pour analyser un réseau SpaceWire, il nous faut modéliser les différents composants de ce réseau par des courbes de service. En général, on décompose un réseau en éléments classiques tels que multiplexeurs, démultiplexeurs et buffers dont les courbes de service sont connues. Cependant, le Calcul Réseau a d'abord été conçu pour étudier des réseaux de type *Store-and-Forward* plutôt que des réseaux de type wormhole et, comme nous le verrons en section 5.2.2 les éléments classiques ne conviennent pas pour modéliser un réseau wormhole de façon efficace. Dans cette Section, nous introduisons donc un nouvel élément réseau, la section wormhole, qui nous permettra de modéliser plus simplement un réseau wormhole.

5.2.1 Hypothèses

On considère un réseau composé de routeurs SpaceWire et de terminaux. Chaque terminal n'est doté que d'une seule interface SpaceWire mais peut être la source et/ou la destination de plusieurs flux. Chaque flux f est modélisé par une source, une destination, un chemin à travers le réseau entre ces deux terminaux et une courbe d'arrivée α_f. Nous utiliserons de préférence des courbes d'arrivée en escalier (voir Figure 5.3) qui fournissent un modèle de trafic plus précis que des courbes affines : si f est périodique de période T_f et que les paquets de f ont pour taille maximale L_f, alors f est contraint par une courbe d'arrivée

$$\alpha_f = L_f . \left\lceil \frac{t}{T_f} \right\rceil .$$

On ne considère que le cas du routage statique qui est le seul supporté par les routeurs SpaceWire.

On suppose également que le réseau est stable, c'est-à-dire qu'aucun lien ne doit transmettre plus de données que sa capacité ne le permet. Sur une période bornée, la quantité de données à transmettre peut être supérieure à la capacité du lien mais, à long terme, la quantité moyenne de données à transmettre sur un lien doit être inférieure à la capacité du lien. Dans un réseau classique de type *Store-and-Forward*, il suffit de vérifier que, sur chaque lien, la somme des débits moyens des flux qui empruntent ce lien est inférieure à la capacité du lien. Pour un réseau Wormhole,

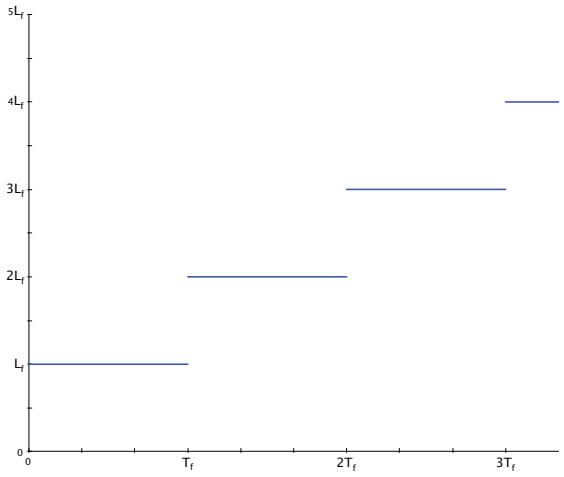


FIGURE 5.3 – Courbe d'arrivée en escalier

cette condition est insuffisante. En effet, comme un paquet peut utiliser plusieurs liens simultanément, les caractères du paquet avanceront à la vitesse du lien le plus lent plutôt qu'à celle du lien qu'ils empruntent à un moment donné.

Nous utiliserons donc la condition suffisante suivante pour vérifier que le trafic n'est pas trop important pour le réseau. Pour chaque flux f_i, notons C_i^{min} la capacité minimale rencontrée par f_i sur son trajet et r_i le débit moyen de f_i à l'entrée dans le réseau. r_i peut être facilement calculé à partir de la courbe d'arrivée du flux. Soit $\Phi_i = \frac{r_i}{C_i^{min}}$. Φ_i représente la fraction de temps pendant laquelle f_i utilise son lien de capacité minimale. Comme tous les liens empruntés par f_i sont reliés par le contrôle de flux, Φ_i est la fraction de temps nécessaire à f_i sur tous les liens qu'il traverse.

Notons $f_1, \ldots, f_{k_j}$ les flux utilisant le lien j. La condition de stabilité est donc

$$\forall j, \sum_{k=1}^{k_j} \Phi_k \leq 1 \tag{5.1}$$

Dans un premier temps, nous faisons l'hypothèse que tous les liens du réseau ont la même capacité C et offrent, au départ, la même courbe de service $\beta(t) = C.t$ aux flux qui les traversent. Seuls les terminaux peuvent avoir un taux de service plus faible.

Nous faisons également l'hypothèse que chaque terminal dispose d'un buffer de réception capable de stocker un paquet de taille maximale. Ainsi le taux de service du terminal est "isolé" et n'intervient pas dans le calcul des poids Φ_i.

Enfin, nous faisons l'hypothèse que deux flux ne peuvent se croiser qu'une seule fois, c'est-à-dire que quand deux flux ont été démultiplexés, ils ne peuvent plus se croiser à nouveau.

5.2.2 Pourquoi créer un nouvel élément ?

Pour déterminer la courbe de service offerte par un routeur wormhole, l'approche classique du Calcul Réseau aurait lieu en deux phases. La première consisterait à décomposer le routeur en éléments plus simples dont on saurait exprimer la courbe de service. Durant la seconde phase, on pourrait ensuite combiner ces éléments grâce au Théorème 2 pour obtenir la courbe de service du routeur lui-même.

Cependant, dans un réseau wormhole, le service offert par un routeur n'est pas indépendant du service offert par les routeurs en aval. Le mécanisme de contrôle de flux peut forcer un routeur à ne transmettre ses données qu'à une vitesse beaucoup plus faible que la capacité nominale du lien de sortie.

Par conséquent, nous ne proposons pas un modèle classique du routeur à base de multiplexeur et de démultiplexeur. A la place, nous essayons de construire une vue logique du réseau qui prenne en compte l'interdépendance des routeurs.

Lorsqu'un flux entre en conflit avec un autre flux, l'impact du conflit est double. D'une part, le flux en conflit cause un délai lorsqu'il est multiplexé avec le flux étudié. D'autre part, une fois les flux séparés, le contrôle de flux propage les délais subis par chaque flux vers l'arrière. Ces délais sont alors répercutés sur l'autre flux car les deux flux partagent au moins une file d'attente FIFO sur leur trajet commun.

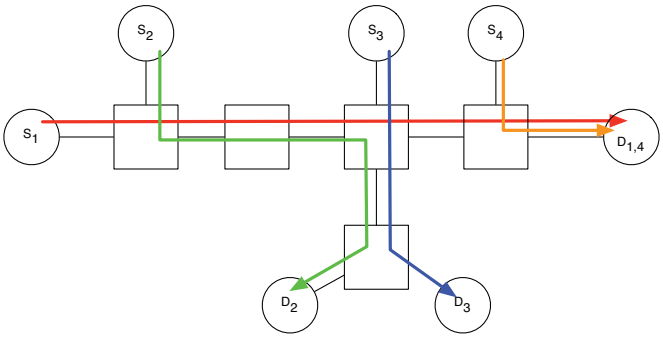
C'est pour prendre correctement en compte ce dernier phénomène que nous proposons l'élément réseau "section wormhole". L'idée de base est de diviser le chemin suivi par un flux en une série de sections. Chaque section est composé d'un ensemble de ports de sortie successifs partagés par les mêmes flux. Ainsi, chaque section correspond à l'arrivée ou au départ d'un flux en conflit avec le flux étudié.

Chaque section wormhole offre un service qui dépend des courbes d'arrivée des flux en conflits. Comme des flux entrent et sortent du chemin du flux étudié entre les sections, il n'est pas possible de convoluer leurs courbes de service directement. Nous allons donc chercher, pour chaque section, à intégrer l'impact des flux en conflits directement dans la courbe de service de la section. La vue logique du réseau pour ce flux peut ainsi être réécrite en éliminant au fur et à mesure les flux en conflits. Une fois tous les flux en conflits intégrés dans les courbes de service des sections, il est possible de calculer la courbe de service de bout en bout offerte au flux grâce au Théorème 2.

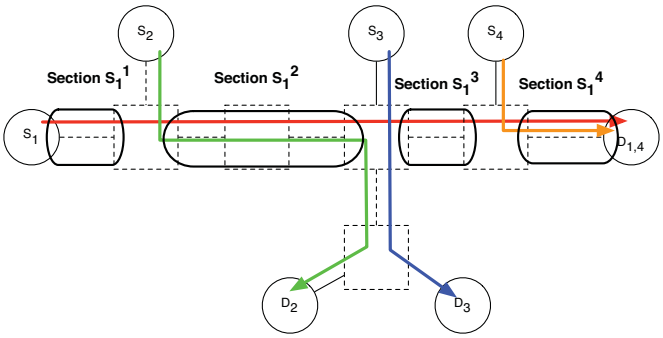
En outre, une fois qu'un flux en conflit avec le flux étudié a quitté le chemin de celui-ci, il doit traverser des sections wormhole relatives à son propre trajet avant d'atteindre sa destination. Les délais subis dans ces sections sont les délais répercutés sur le flux étudié par le biais du contrôle de flux.

La division du réseau en section wormhole est la même pour tous les flux. Chaque flux suit ensuite son propre trajet dans le réseau logique constitué de sections. Nous numéroterons donc les sections relativement à chaque flux. Une section partagée par plusieurs flux aura ainsi plusieurs noms.

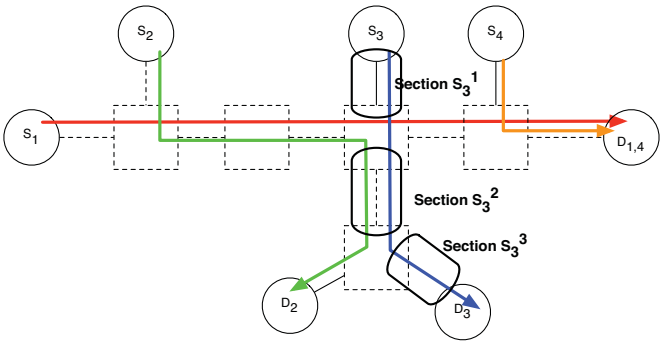
Considérons par exemple le réseau de la Figure 5.4a. Sur ce schéma, chaque flux f_i va de sa source S_i à sa destination D_i. Une destination partagée par deux flux f_i et f_j est notée $D_{i,j}$.



(a) Exemple de réseau



(b) Sections wormhole pour f_1



(c) Sections wormhole pour f_3

Examinons d'abord f_1 de plus près. Le chemin suivi par f_1 se décompose en quatre sections wormhole $(S_1^1, S_1^2, S_1^3, S_1^4)$ avant d'atteindre $D_{1,4}$ (voir Figure 5.4b). Les sections S_1^2 et S_1^4 sont partagés avec d'autres flux. Ces deux sections sont donc des sources de délais pour f_1. De plus, étant donné qu'après avoir quitté f_1, f_2 traverse d'autres sections, ces sections sont sources de délai pour f_1 mais seulement de façon indirecte.

En revanche, pour f_3, la décomposition en section sera différente. f_3 traverse trois sections wormhole (S_3^1, S_3^2, S_3^3) et est en conflit avec f_2 dans S_3^2 (voir Figure 5.4c). Seule cette portion du réseau est importante pour f_3.

Cet exemple illustre comment la notion de section wormhole rend les conflits qui prennent place au sein d'un réseau wormhole plus facile à analyser. Nous allons maintenant en présenter un modèle détaillé.

5.2.3 Modèle général d'une Section wormhole

Une Section wormhole présente deux aspects à modéliser sous forme de courbes de service. D'une part, la Section elle-même est partagée entre plusieurs flux et seule une partie du service total de la Section wormhole est allouée à chaque flux. Nous modéliserons cet aspect par une courbe de service $\beta_i^{S,partage}$ offerte au flux f_i.

D'autre part, à la sortie de la Section, certains délais subis par les flux qui quittent le flux étudié peuvent être reportés sur celui-ci. Ce second aspect sera modélisé par une courbe de service $\beta_i^{S,sortie}$.

Ainsi, au total le service offert par la Section wormhole S au flux f_i sera

$$\beta_i^S = \beta_i^{S,partage} \otimes \beta_i^{S,sortie}. \tag{5.2}$$

5.2.4 Partage d'une section

Plaçons-nous dans le cas où un seul flux partage avec le flux étudié une section S composée de M ports de sortie en séquence. Nous présenterons en Section 5.3.2 une méthode permettant de calculer une courbe de service de bout en bout uniquement à partir du modèle d'une section wormhole partagée par deux flux.

Notons f_1 le flux étudié et f_2 le flux qui interfère. Chaque flux f_i est contraint par une courbe d'arrivée $\alpha_i^{S_{in}}$ à l'entrée de S. Pour $j = 1, \ldots, M$, on note β^j la courbe de service offerte par le port de sortie R^j aux deux flux.

La quantité de données qui sort réellement du port de sortie R^j peut être inférieure à β^j du fait du contrôle de flux. β_j doit donc plutôt être vue comme un paramètre intermédiaire permettant de calculer le service offert à f_1 par la section complète. Une fois l'analyse terminée, il est possible d'utiliser le Théorème 2 pour déterminer la courbe de service offerte par la section puis par le chemin complet. Cette courbe de service de bout en bout représente le service réellement offert à f_1 car elle inclut non seulement les délais directs causés par les flux qui interfèrent avec f_1 mais également les délais indirects. Par conséquent, elle modélise ce que peut réellement transmettre le réseau pendant une période de temps donnée et est donc une courbe de service valide.

Etant donné que tous les routeurs de la section S sont partagés par les deux flux et qu'aucune autre interférence ne se produit, on peut considérer les M routeurs de la section comme un seul élément de courbe de service $\beta^S = \bigotimes_{j=1}^{M} \beta^j$.

Le mécanisme de Round-Robin simplifié utilisé par les routeurs SpaceWire garantit à chaque port d'entrée un accès équitable à chaque port de sortie. Cependant, une fois qu'il a accès à un port de sortie, un paquet peut l'utiliser sans limitation de durée. Le modèle classique du Round-Robin en Calcul Réseau est d'attribuer un poids à chaque flux puis de partager la bande passante entre les flux proportionnellement à ces poids. Comme le standard SpaceWire ne suit pas ce modèle à poids proportionnels, nous sommes forcés de nous rabattre sur le modèle "blind multiplexing" ([23], Théorème 6.2.1), plus pessimiste.

Ce théorème nécessite que la courbe de service soit une courbe de service stricte. Dans notre cas, on a $\beta^j(t) = C.t, \forall j$ qui est bien une courbe de service stricte.

Ainsi la courbe de service offerte par la section S au flux f_1 est

$$\beta_1^{S,partage} = \left(\bigotimes_{j=1}^{M} \beta^j - \alpha_2^{S_{in}} \right)_{\uparrow}. \tag{5.3}$$

Notons au passage que la convolution Min-Plus est prioritaire par rapport à la soustraction.

$\beta_1^{S,partage}$ est une courbe de service pire-cas qui implique que f_2 a une priorité plus haute que f_1 et peut préempter celui-ci instantanément. En réalité, f_2 ne peut pas interrompre la transmission d'un paquet de f_1. Nous avons voir avec l'illustration suivante pourquoi ce modèle peut être pessimiste. Il nous permet par contre d'être certain que nous sommes bien dans le pire scénario possible.

Illustration du service résiduel

Pour mieux comprendre l'impact du partage de section, nous pouvons montrer graphiquement une courbe de service résiduelle (voir Figure 5.4). Supposons que le flux f_2 ait une courbe d'arrivée $\alpha_2^{S_{in}}$ en escalier (en vert sur le graphique). Dans ce cas, le service résiduel offert à f_1 par la section est donné par la courbe $\beta_1^{S,partage}$ en bleu.

On peut observer sur ce graphique que chaque segment de pente nulle de $\beta_1^{S,partage}$ est de longueur $\frac{L_2}{C}$. Les intervalles de service nul correspondent donc bien au passage d'un paquet de f_2. Dans le cas particulier où la courbe d'arrivée de f_1 est telle que les paquets des deux flux s'entrelacent, ce service est réaliste. Autrement, il est pessimiste car à chaque fois qu'un paquet de f_2 arrive à l'entrée de S, une transmission en cours d'un paquet de f_1 est interrompue.

5.2.5 Sortie de la section

5.2.5.1 Limites des modèles existants

Dans un modèle de réseau *Store-and-Forward* classique, les paquets des différents flux sont démultiplexés instantanément. De plus, une fois qu'un paquet a quitté un

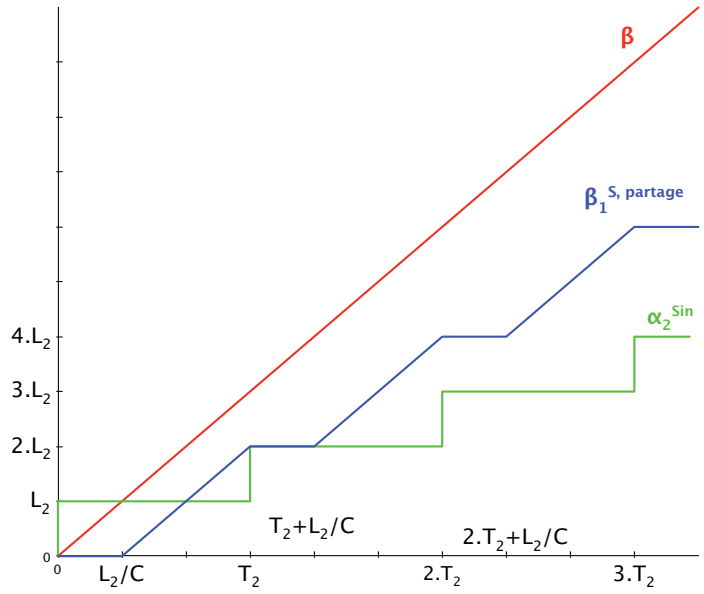


FIGURE 5.4 – Calcul du service résiduel dans le cas d'une courbe d'arrivée en escalier

routeur, les délais qu'il peut éventuellement subir ne sont pas répercutés en amont du réseau. Le démultiplexage n'a donc aucun impact sur le délai (voir [26]).

En revanche, ceci n'est pas vrai dans le cadre d'un réseau wormhole. Comme nous l'avons déjà vu, le contrôle de flux reporte sur un flux les ralentissements subis par les flux qui ont quitté son chemin.

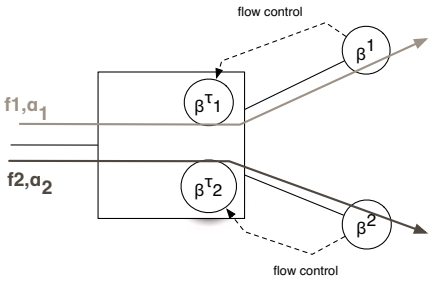


FIGURE 5.5 – Deux flux démultiplexés en sortie d'un routeur wormhole

Dans [22], les auteurs donnent un modèle de ce phénomène. Ils étudient le réseau présenté sur la Figure 5.5. Dans leur modèle, les flux f_1 et f_2 sont démultiplexés à l'intérieur d'un routeur puis doivent chacun traverser un contrôleur de flux qui régule l'accès à leur port de sortie respectif. Le contrôleur τ_i modélise l'impact du contrôle de flux sur le lien sortant utilisé par f_i grâce à une courbe de service β^{τ_i}. Cette courbe β^{τ_i} dépend de la courbe de service du routeur aval qui dépend elle-même de celle du routeur suivant et ainsi de suite jusqu'à la destination du flux.

Les auteurs considèrent que, du point de vue du flux agrégé $f_{\{1,2\}}$, τ_1 et τ_2 sont des régulateurs de trafic placés en parallèle. En effet, étant donné qu'une fraction

	f_1	f_2
r_i (Mbps)	40	5
b_i (bits)	1000	200
C_i (Mbps)	100	20

TABLE 5.1 – Paramètres pour l'étude du réseau Figure 5.5

de $f_{\{1,2\}}$ traverse τ_1 et une autre τ_2, on peut considérer, par une approximation pessimiste, que le flux $f_{\{1,2\}}$ complet traverse simultanément les deux régulateurs de trafic. Par suite, le service qu'ils offrent à $f_{\{1,2\}}$ est $\beta_{\{1,2\}} = \min(\beta^{\tau_1}, \beta^{\tau_2})$. Ce service agrégé est ensuite partagé entre les deux flux pour obtenir le service assuré à chacun d'entre eux.

Ce service est valide mais très pessimiste. Considérons de nouveau l'exemple de la Figure 5.5. Supposons que les flux f_1 et f_2 aient des courbes d'arrivées affine : $\alpha_i(t) = r_i.t + b_i$ et que les deux courbes de service soient linéaires : $\beta_i(t) = C_i.t,\ i = 1, 2$. Nous utiliserons les valeurs du Tableau 5.1.

D'après le modèle de [22], Le service offert à $f_{\{1,2\}}$ est $\beta_{\{1,2\}}(t) = \min(C_1, C_2).t = C_2.t$. Par ailleurs, la courbe d'arrivée de $f_{\{1,2\}}$ est $\alpha_{\{1,2\}}(t) = \alpha_1(t) + \alpha_2(t) = r_{1,2}.t + b_{1,2}$ avec $r_{1,2} = r_1 + r_2 = 50$ Mbps et $b_{1,2} = b_1 + b_2 = 1200$ bits.

La distance horizontale $h(\alpha_{\{1,2\}}, \beta_{\{1,2\}})$ entre les deux courbes est clairement infinie et, dans ce modèle, nous sommes forcés de conclure que le réseau ne peut pas transporter les deux flux. Ce résultat est très pessimiste car il est facile de voir qu'avec les débits choisis, il est parfaitement possible de transporter les deux flux simultanément sur ce réseau.

Ceci illustre bien la nécessité d'un nouveau modèle plus précis du démultiplexage.

5.2.5.2 Un nouveau modèle de démultiplexage

En réalité, étant donné que chacun des flux part dans une direction différente après le routeur, il n'est pas possible de déterminer précisément le service offert au flux agrégé. Chaque flux reçoit son propre service mais peut causer des délais à l'autre flux à travers le contrôle de flux.

Le délai causé à un paquet particulier de f_1 est très difficile à déterminer car il dépend des conflits exacts qui se produisent sur le chemin aval de f_2. Nous pouvons par contre rechercher une borne pire-cas sur ce délai.

Le délai maximum causé par f_2 à un bit de f_1, que nous noterons d_2, est au plus le délai maximum de livraison d'un paquet de f_2, du dernier routeur partagé à la destination de f_2. Ainsi,

$$d_2 = h(\alpha_2^{S_{out}}, \beta_2^{dest}) \tag{5.4}$$

où $\alpha_2^{S_{out}}$ est la courbe d'arrivée de f_2 à la fin de S et β_2^{dest} la courbe de service offerte à f_2 entre la fin de S et sa destination.

Etant donné que nous avons utilisé un modèle "blind multiplexing" avec f_2

comme flux de plus haute priorité, nous avons

$$\alpha_2^{S_{out}} = \alpha_2^{S_{in}} \oslash \bigotimes_{j=1}^{M} \beta^j \tag{5.5}$$

La courbe de service est ainsi

$$\beta_1^{S,sortie} = \delta_{d_2}$$

où δ est la fonction délai, définie par

$$\delta_T(t) = \begin{cases} 0 & \text{si } t < T \\ +\infty & \text{si } t \geq T \end{cases} \quad \text{pour tout } T \geq 0.$$

Revenons maintenant à l'exemple de la Figure 5.5. Avec ce modèle, le délai causé par f_2 à f_1 est seulement $h(\alpha_2, \beta_2) = \frac{200}{2.10^7} s = 10\mu s$. Ce modèle est donc beaucoup moins pessimiste que celui de [22].

5.2.6 Courbe de service complète offerte par la section

Nous pouvons maintenant exprimer la courbe de service offerte par la section S.

Lorsque deux flux f_1 et f_2 partagent une section wormhole S composée de M ports de sortie consécutifs, la courbe de service offerte à f_1 est donnée par :

$$\begin{aligned} \beta_1^S &= \beta_1^{S,partage} \otimes \beta_1^{S,sortie} \\ &= \left[\bigotimes_{j=1}^{M} \beta^j - \alpha_2^{S_{in}} \right]_{\uparrow} \otimes \delta_{h(\alpha_2^{S^{out}}, \beta_2^{dest})} \end{aligned} \tag{5.6}$$

5.3 Calcul de la courbe de service de bout en bout

Nous pouvons ensuite combiner les courbes de service des différentes sections pour obtenir la courbe de service de bout en bout offerte à chaque flux. Toutefois, avant cela, il nous faut d'abord modéliser les terminaux sous forme de courbes de service.

5.3.1 Modèle des terminaux

Etant donné que les liens SpaceWire fonctionnent en *full-duplex*, les ports d'entrée et de sortie du terminal se comportent de manière indépendante. Nous avons donc choisi de modéliser séparément les terminaux source et les terminaux destination.

Chaque terminal source contient une file d'attente FIFO de taille quelconque. Cette file est partagée par toutes les applications et est connectée au port de sortie. Celui-ci émet les données présentes dans la file dès que le lien est libre. Chaque application utilise un flux de données différent. Un terminal source est donc un

élément réseau partagé de façon FIFO par un nombre fixe de flux. Nous pouvons donc utiliser le service résiduel pour un élément FIFO défini dans ([23], Proposition 6.2.1) :

Pour un élément FIFO partagé entre deux flux de courbes d'arrivée α_1 et α_2, soit la famille de fonction β_1^θ définie par

$$\beta_1^\theta(t) = [\beta(t) - \alpha_2(t-\theta)]_\uparrow . \mathbf{1}_{\{t>0\}} \tag{5.7}$$

Alors pour tout θ tel que β_1^θ soit non-décroissante, β_1^θ est une courbe de service pour f_1. Nous utiliserons ce résultat pour $\theta = 0$ dans la suite. Notons que l'hypothèse d'arrivée instantanée des paquets est vérifiée car les applications écrivent instantanément leurs paquets dans le buffer d'émission.

Un terminal destination contient un buffer de réception FIFO capable de contenir au moins un paquet de taille maximale parmi les flux qui aboutissent à ce terminal. Il peut imposer un délai constant à chaque paquet avant de le traiter. Ce délai correspond au temps de traitement du paquet précédent. Le terminal lit ensuite le paquet à une vitesse constante, appelée le taux de service, qui est en général inférieure à la capacité du lien entrant. Une fois un paquet lu, il est retiré du buffer et détruit immédiatement.

Pour modéliser un terminal destination D, nous avons donc choisi une courbe de service $\beta^D(t) = r_D.(t - T_D)^+$ où r_D est le taux de service du terminal et T_D le délai imposé aux données.

5.3.2 Résolution d'interférences plus complexes

Nous avons vu en Section 5.2 comment calculer la courbe de service offerte par une section wormhole lorsqu'elle est partagée par deux flux. En général, un flux interfère avec un nombre plus importants de flux lorsqu'il traverse le réseau et une section wormhole peut être partagée par plus de deux flux. Dans cette sous-section, nous allons montrer comment calculer la courbe de service de bout en bout offerte à un flux en utilisant le modèle établi pour une section partagée par deux flux.

Par ailleurs, si le flux étudié interfère avec un grand nombre de flux qui entrent et sortent à tous les routeurs de son chemin, nous somme forcés de créer une section wormhole pour chaque routeur. Cependant, cette méthode mènerait à un résultat sous-optimal car elle nous forcerait à compter plusieurs fois (une par routeur) l'influence d'un flux partageant plusieurs routeurs successifs avec le flux étudié. Il est donc préférable d'essayer d'optimiser la courbe de service de bout en bout en groupant des flux qui se partagent plusieurs routeurs successifs en un flux agrégé. Nous pouvons ensuite déterminer des sections wormhole pour ce flux agrégé et en déduire la courbe de service qui lui est offerte. Cette courbe de service peut ensuite être partagée entre les composantes individuelles du flux agrégé.

L'automatisation de cette procédure est basée sur l'utilisation des schémas d'interférence définis dans [22]. Ces schémas d'interférence définissent l'ordre dans lequel les courbes de service résiduel doivent être calculées. Les auteurs de [22] définissent trois schémas qui décrivent les interactions entre un flux étudié et deux flux en conflit

avec lui (et entre eux). Ils montrent également que tous les conflits impliquant un nombre supérieur de flux peuvent être résolus une fois que ces schémas peuvent l'être.

L'application de ces schémas d'interférence peut être vue comme un système de réécriture qui permet de calculer une courbe de service résiduelle offerte à un flux agrégé qui inclut le flux étudié. Le but est de traiter successivement tous les flux en conflit pour obtenir la courbe de service individuelle offerte au flux étudié. Au fur et à mesure de l'application des règles de réécriture, les courbes de service des ports de sortie intermédiaires sont remplacées par des courbes résiduelles obtenues en soustrayant la courbe d'arrivée de chaque flux en conflit.

L'application répétée du calcul de courbes de service résiduelles ne respecte pas les hypothèses du théorème utilisée en 5.2.4. En effet, une courbe de service résiduelle de la forme de celle de la Figure 5.4 n'est pas une courbe de service stricte. Cependant, nous faisons l'hypothèse que la courbe de service obtenue est quand même valide. Cette hypothèse est corroborée empiriquement par l'interprétation "physique" des courbes de service obtenue (voir Figure 5.9) et nous paraît donc réaliste.

Nous donnons maintenant la définition de chacun de ces schémas et la courbe de service résiduelle calculée dans chaque cas.

On note β^j la courbe de service offerte par le routeur R_j et $d_i = h(\alpha_i^{out}, \beta_i^{dest})$, $i = 2, 3$. On note également $L_{i,j}$ l'ensemble des liens partagés par les flux f_i et f_j. Nous expliciterons les courbes α_i^{out} dans chaque cas car elles dépendent des interactions entre f_2 et f_3.

Schéma imbriqué Cette configuration (voir Figure 5.6a) peut être définie formellement par $L_{1,3} \subset L_{1,2}$.

Ici, on traite tout d'abord f_1 et f_2 comme un flux agrégé qui partage une section contenant le port de sortie du routeur R_3 avec f_3 (Figure 5.6b). Ainsi, en appliquant (5.6) le service offert par R_3 à $f_{\{1,2\}}$ est $\beta^3_{\{1,2\}} = \left[\beta^3 - \alpha_3^3\right]_\uparrow \otimes \delta_{d_3}$.

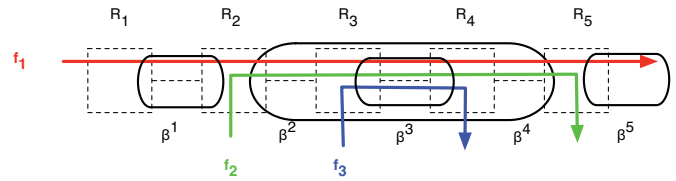
Ensuite, on considère que f_1 et f_2 se partage la section qui comprend les ports de sortie des routeurs R_2, R_3 et R_4 (Figure 5.6c). Le service offert par cette section à f_1 est donc $\beta_1^{2\to4} = \left[\beta_2 \otimes \beta^3_{\{1,2\}} \otimes \beta^4 - \alpha_2^2\right]_\uparrow \otimes \delta_{d_2}$.

Il reste enfin à f_1 à traverser une section $R_1 \to R_5$ sans aucun conflit (Figure 5.6d). Le service final offert à f_1 est donc

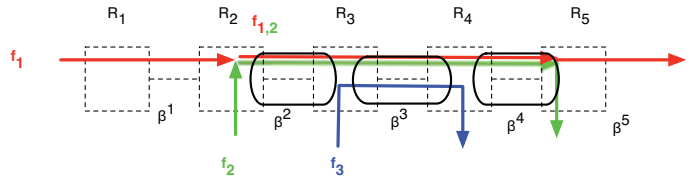
$$\begin{aligned}\beta_1^{1\to5} &= \beta^1 \otimes \beta_1^{2\to4} \otimes \beta^5 \\ &= \beta^1 \otimes \left[\beta_2 \otimes \beta^3_{\{1,2\}} \otimes \beta^4 - \alpha_2^2\right]_\uparrow \otimes \delta_{d_2} \otimes \beta^5 \\ &= \beta^1 \otimes \left[\beta_2 \otimes \left[\beta^3 - \alpha_3^3\right]_\uparrow \otimes \delta_{d_3} \otimes \beta^4 - \alpha_2^2\right]_\uparrow \otimes \delta_{d_2} \otimes \beta^5\end{aligned} \tag{5.8}$$

Par ailleurs, nous avons

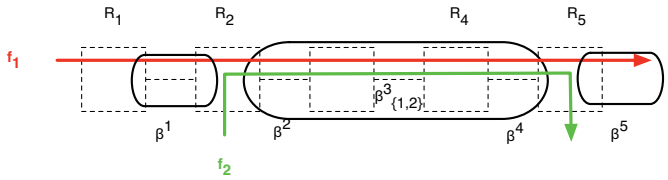
$$\begin{aligned}\alpha_2^{out} &= \alpha_2^2 \oslash \left[\beta_2 \otimes \left[\beta^3 - \alpha_3^3\right]_\uparrow \otimes \delta_{d_3} \otimes \beta^4\right] \\ \alpha_3^{out} &= \alpha_3^3 \oslash \left[\left[\beta^3 - \alpha_2\right]_\uparrow \otimes \delta_{d_2}\right]\end{aligned}$$



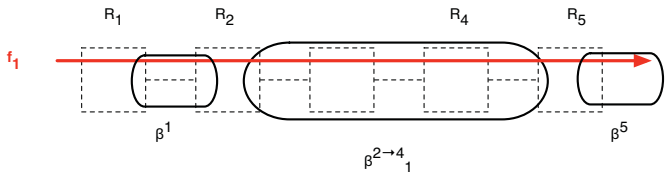
(a) Schéma d'interférence imbriqué



(b) On agrège d'abord f_1 et f_2 dans la section R_3



(c) f_1 et f_2 se partagent la section $R_2 \rightarrow R_4$

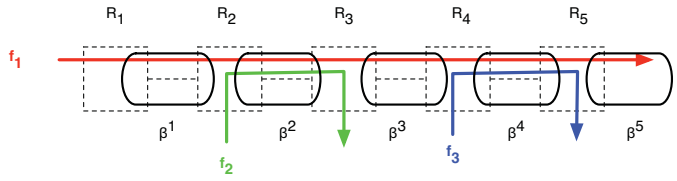


(d) f_1 est maintenant seul sur $R_1 \rightarrow R_5$

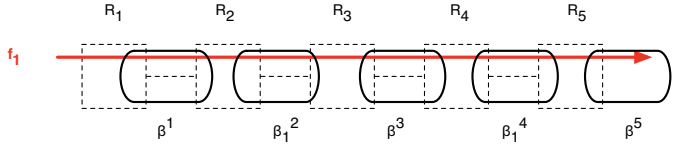
FIGURE 5.6 – Résolution du schéma d'interférence imbriqué

Schéma parallèle Dans cette configuration, f_1 partage la section contenant R_2 avec f_2 et la section contenant R_4 avec f_3.

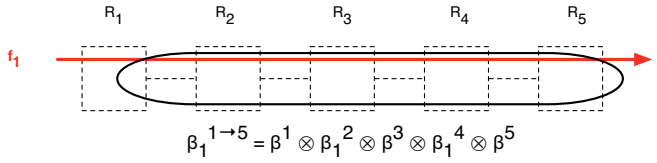
Ceci peut être défini formellement par $L_{1,2} \cap L_{1,3} = \emptyset$.



(a) Schéma d'interférence parallèle



(b) On substitue une courbe de service résiduelle aux deux sections avec conflit



(c) f_1 traverse une section $R_1 \to R_5$ sans conflits

FIGURE 5.7 – Résolution du schéma d'interférence parallèle

Remarque : on parle ici de schéma parallèle alors que du point de vue de f_1, les deux conflits ont lieu en série. Cependant, du point de vue de f_2 et f_3 les conflits sont en parallèle car les deux flux n'ont aucun lien en commun.

Ici, on peut associer à chaque section comprenant un conflit une courbe de service résiduelle (Figure 5.7b) :

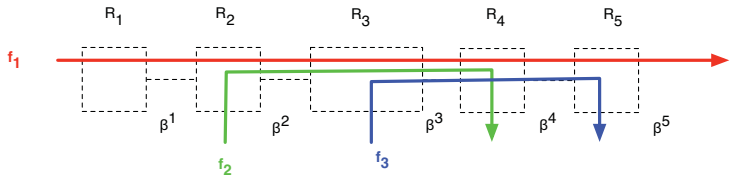
$$\beta_1^2 = \left[\beta^2 - \alpha_2^2\right]_\uparrow \otimes \delta_{d_2}$$
$$\beta_1^4 = \left[\beta^4 - \alpha_3^4\right]_\uparrow \otimes \delta_{d_3}$$

Une fois la substitution effectuée, f_1 traverse une section sans conflit allant de R_1 à R_5 (Figure 5.7c). Le service offert est donc

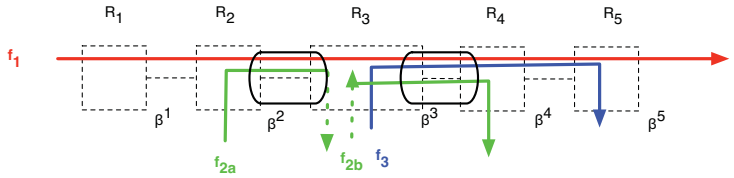
$$\begin{aligned}\beta_1^{1\to5} &= \beta^1 \otimes \beta_1^2 \otimes \beta^3 \otimes \beta_1^4 \otimes \beta^5 \\ &= \beta^1 \otimes \left[\beta^2 - \alpha_2^2\right]_\uparrow \otimes \delta_{d_2} \otimes \beta^3 \otimes \left[\beta^4 - \alpha_3^4\right]_\uparrow \otimes \delta_{d_3} \otimes \beta^5\end{aligned} \tag{5.9}$$

De plus,

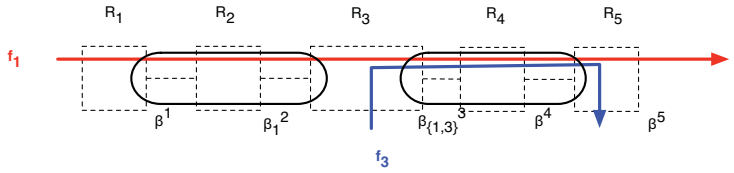
$$\alpha_2^{out} = \alpha_2^2 \oslash \beta^2$$
$$\alpha_3^{out} = \alpha_3^4 \oslash \beta^4$$



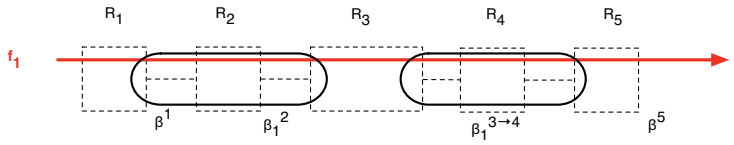
(a) Schéma d'interférence croisé



(b) On coupe f_2 en deux sous-flux f_{2a} et f_{2b}



(c) Ces deux sous-flux peuvent être retirés et remplacés par des courbes de service résiduelles



(d) f_3 peut ensuite être facilement substitué

FIGURE 5.8 – Résolution du schéma d'interférence croisé

Schéma croisé Cette configuration (voir Figure 5.8a) peut être définie formellement ainsi : $L_{1,2} \cap L_{1,3} \neq \emptyset$ *et* $L_{1,3} \not\subset L_{1,2}$ *et* $L_{1,2} \not\subset L_{1,3}$.

Ici, f_1 partage R_2 et R_3 avec f_2 et R_3 et R_4 avec f_3. Pour résoudre ce schéma, il faut couper f_2 en deux sous-flux f_{2a} et f_{2b} entre le port de sortie de R_2 et le port de sortie de R_3 (Figure 5.8a). Maintenant, f_1 partage la section comprenant R_2 avec f_{2a}. Cette interférence peut-être résolue indépendamment des autres.

Ensuite, pour les flux f_1, f_3 et f_{2b}, on est ramené à un schéma imbriqué. On peut donc considérer que le flux agrégé $f_{\{1,3\}}$ partage la section composée de R_3 avec f_{2b} (Figure 5.8c) puis, une fois f_{2b} traité, que f_1 partage la section R_3, R_4 avec f_3. Enfin, f_1 est seul sur la section $R_1 \to R_5$ (Figure 5.8d).

Le service offert à f_1 peut donc s'écrire

$$\beta_1^{1\to5} = \beta^1 \otimes \left[\beta^2 - \alpha_2^2\right]_\uparrow \otimes \left[\left[\beta^3 - \alpha_2^3\right]_\uparrow \otimes \delta_{d_2} \otimes \beta^4 - \alpha_3^3\right]_\uparrow \otimes \delta_{d_3} \tag{5.10}$$

avec

$$\begin{aligned}
\alpha_2^{out} &= \alpha_2^3 \oslash [\left[\beta^3 - \alpha_3^3\right]_\uparrow \otimes \delta_{d_3}] \\
\alpha_2^3 &= \alpha_2^2 \oslash \beta^2 \\
\alpha_3^{out} &= \alpha_3^3 \oslash [\left[\beta^3 - \alpha_2^3\right]_\uparrow \otimes \delta_{d_2} \otimes \beta^4]
\end{aligned}$$

Illustration du schéma d'interférence imbriqué

Plaçons nous dans la situation de la Figure 5.6a. Le premier calcul de service résiduel est similaire à celui de la Figure 5.4 et donne une courbe de service résiduelle $\beta_{\{1,2\}}^3$ du type de celle de la Figure 5.9. On soustrait ensuite la courbe d'arrivée α_2^2 du flux f_2 pour obtenir la courbe de service $\beta_1^{2\to4}$ offerte à f_1.

De même que sur la Figure 5.4, on constate sur la Figure 5.9 que le premier segment de pente nulle de $\beta_1^{2\to4}$ est de longueur $\frac{L_2+L_3}{C}$ ce qui correspond au passage d'un paquet de f_2 et d'un paquet de f_3. f_1 dispose ensuite du débit complet de la section jusqu'à l'instant T_2 où un second paquet de f_2 arrive et utilise à nouveau tout le débit disponible. On voit ainsi que calculer deux services résiduels l'un après l'autre modélise correctement, bien que de façon pessimiste, le partage d'une section entre trois flux.

5.3.3 Courbes d'arrivées des flux en conflits

Les calculs présentés en 5.3.2 nécessitent de connaître les courbes d'arrivées $\alpha_i^{S_{in}}$ des flux en conflits avec le flux étudié f_e au début de leur section commune S. Pour un flux f_i, $\alpha_i^{S_{in}}$ dépend de la courbe d'arrivée α_i du flux à l'entrée dans le réseau ainsi que du service offert au flux entre sa source et le point où il rencontre f_e.

Notons $\beta_i^{src\to S}$ cette courbe de service.

$\alpha_i^{S_{in}}$ est alors donnée par

$$\alpha_i^{S_{in}} = \alpha_i \oslash \beta_i^{src\to S}. \tag{5.11}$$

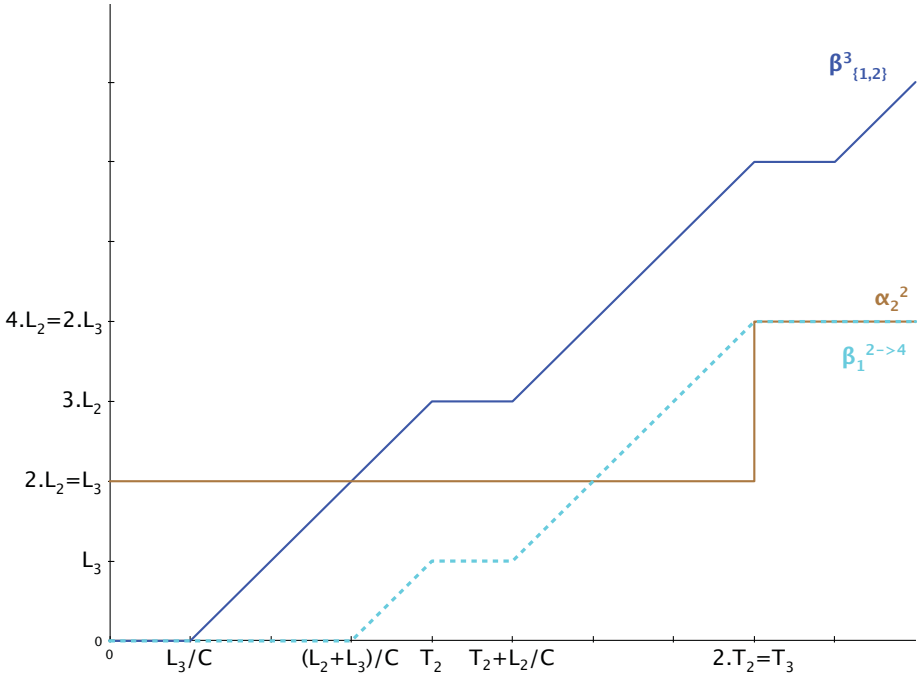


FIGURE 5.9 – Calcul du service résiduel pour le schéma imbriqué

$\beta_i^{src \to S}$ peut être calculée sur le même principe qu'une courbe de service de bout en bout, comme expliquée dans la Section suivante.

La courbe d'arrivée $\alpha_i^{S_{out}}$ peut quant à elle être déterminée à partir de $\alpha_i^{S_{in}}$. En revanche, il reste à déterminer la courbe de service β_i^{dest} offerte à f_i entre la fin de S et sa destination. Ce service, tout comme celui reçu entre la source et le début de S dépend des courbes d'arrivée d'autres flux qui interfèrent avec f_i.

Comme ces flux peuvent eux-mêmes dépendre de de f_e, il est possible de se retrouver dans une situation de dépendance circulaire.

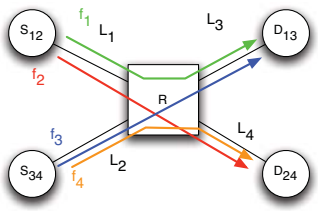


FIGURE 5.10 – Exemple de réseau posant un problème de dépendance circulaire

A titre d'exemple, considérons le réseau de la Figure 5.10. Ce réseau comprend quatre terminaux et un routeur SpaceWire. Chaque lien offre la même courbe de service β. Chacun des flux est contraint par une courbe d'arrivée α_i à l'entrée dans le réseau. Dans cet exemple, on négligera les courbes de service des terminaux destination qui n'ont pas d'influence sur le problème.

Les courbes de service de bout en bout peuvent s'exprimer ainsi :

$$\beta_1^{ETE} = [\beta - \alpha_2]_\uparrow \otimes \delta_{d_2} \otimes \left[\beta - \alpha_3^R\right]_\uparrow$$
$$\beta_2^{ETE} = [\beta - \alpha_1]_\uparrow \otimes \delta_{d_1} \otimes \left[\beta - \alpha_4^R\right]_\uparrow$$
$$\beta_3^{ETE} = [\beta - \alpha_4]_\uparrow \otimes \delta_{d_4} \otimes \left[\beta - \alpha_1^R\right]_\uparrow$$
$$\beta_4^{ETE} = [\beta - \alpha_3]_\uparrow \otimes \delta_{d_3} \otimes \left[\beta - \alpha_2^R\right]_\uparrow$$

avec

$$\forall\ i\ \in\ \{1, \ldots, 4\}, d_i = h(\alpha_i, \beta_i^d)$$

et

$$\beta_1^d = \left[\beta - \alpha_3^R\right]_\uparrow$$
$$\beta_2^d = \left[\beta - \alpha_4^R\right]_\uparrow$$
$$\beta_3^d = \left[\beta - \alpha_1^R\right]_\uparrow$$
$$\beta_4^d = \left[\beta - \alpha_2^R\right]_\uparrow$$

De plus,

$$\alpha_1^R = \alpha_1 \oslash [[\beta - \alpha_2]_\uparrow \otimes \delta_{d_2}]$$
$$\alpha_2^R = \alpha_2 \oslash [[\beta - \alpha_1]_\uparrow \otimes \delta_{d_1}]$$
$$\alpha_3^R = \alpha_3 \oslash [[\beta - \alpha_4]_\uparrow \otimes \delta_{d_4}]$$
$$\alpha_4^R = \alpha_4 \oslash [[\beta - \alpha_3]_\uparrow \otimes \delta_{d_3}]$$

Comme on peut le voir, pour calculer β_1^{ETE}, il faut connaître d_2 qui nécessite de connaître α_4^R qui nécessite d_3 qui nécessite α_1^R qui nécessite d_2.

5.3.4 Méthode du point fixe

Pour résoudre ce problème de dépendance circulaire, nous utilisons la méthode du point fixe. Les courbes d'arrivée de tous les flux à l'entrée du réseau sont connues ainsi que les courbes de service de tous les ports de sortie. De plus, toutes les courbes de service résiduelles peuvent être calculées immédiatement si les courbes d'arrivée de tous les flux au début de chaque section sont connues. Il suffit donc de calculer les courbes d'arrivée des différents flux à chaque point du réseau pour avoir résolu le problème.

On note α_i la courbe d'arrivée du flux i dans le réseau et α_i^j la courbe d'arrivée du flux i à l'entrée du port j.

On procède par itération.

Initialisation On pose $\alpha_i^{j,(0)} = \alpha_i$ pour tout i, j. Cela revient à supposer que les conflits rencontrés par un flux au fur et à mesure de sa progression dans le réseau n'affecte pas sa courbe d'arrivée. Cette influence va être prise en compte progressivement au fur et à mesure des itérations.

Itération En utilisant les schémas d'interférence définies en Sous-Section 5.3.2, on exprime $\alpha_i^{j,(n+1)}$ en fonction de α_i, d'un ensemble de courbes de service $\{\beta^{j'}\}$ et d'un ensemble de courbes d'arrivées $\{\alpha_k^{l,(n)}\}$ calculées à l'étape précédente.

A chaque itération, les courbes d'arrivées $\alpha_i^{j,(n)}$ augmentent, jusqu'à atteindre leur valeur véritable à chaque point du réseau.

Au bout d'un certain temps, il existe p tel que $\forall i, \forall j, \alpha_i^{j,(p+1)} = \alpha_i^{j,(p)}$ et le calcul a convergé vers une solution du système.

5.3.5 Exemple de calcul itératif

Reprenons l'exemple de la Figure 5.10. Le calcul est le suivant.

Initialisation

$$\forall\, i \;\in\; \{1,\ldots,4\}, \alpha_i^{R,(0)} \leftarrow \alpha_i$$

Itération Par exemple, pour le flux f_1 :

- $\alpha_1^{R,(n+1)} = \alpha_1 \oslash [[\beta - \alpha_2]_\uparrow \otimes \delta_{d_2^{(n)}}]$
- $d_2^{(n)} = h(\alpha_2, \beta_2^{d,(n)})$
- $\beta_2^{d,(n)} = \left[\beta - \alpha_4^{(n)}\right]_\uparrow$

Application numérique Pour l'application numérique, nous considérons que tous les liens ont une capacité $C = 50$ Mbps. On suppose également que les terminaux ont des taux de service de même valeur et qu'ils n'ont donc pas d'influence sur le calcul. Enfin, on suppose que les quatre flux sont contraints par des courbes d'arrivées affines $\alpha_i(t) = r_i.t + b_i$. Nous utiliserons les valeurs suivantes pour r_i et b_i :

Flux	r_i (Mbps)	b_i (octets)
f_1	15	4000
f_2	7	200
f_3	18	4000
f_4	5	200

Avec ces valeurs, le calcul converge en 17 itérations et fournit les bornes suivantes :

Flux	borne calculée (ms)
f_1	3,3
f_2	3,9
f_3	3,1
f_4	3,8

Nous reviendrons sur ce réseau au chapitre 6 lorsque nous comparerons les trois méthodes de calcul.

5.3.6 Etude des liens de vitesses différentes

Selon la topologie du réseau, la prise en compte des liens de vitesses différentes peut poser problème avec la méthode de calcul présentée plus haut. Ceci est dû au fait que le modèle "blind multiplexing" utilisé n'est pas une représentation exacte du comportement d'un réseau wormhole.

Cas général Considérons l'exemple de la Figure 5.11. Les deux flux f_1 et f_2 ont la même destination D et se partagent le lien l_3. Avant d'accéder à l_3, f_1 traverse l_1 de capacité C_1 et f_2 traverse l_2 de capacité C_2. On note $\beta^j(t) = C_j.t$ la courbe de service de chaque lien. Chaque flux a une courbe d'arrivée α_i en escalier et une taille de paquet L_i.

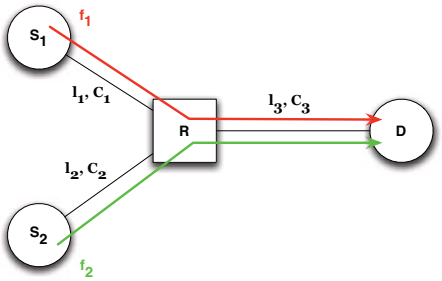


FIGURE 5.11 – Illustration du problème des liens de vitesses différentes

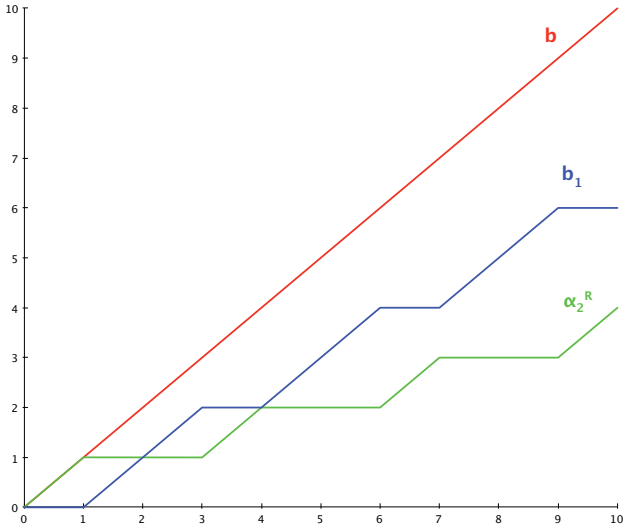
Supposons dans un premier temps que $C_1 = C_2 = C_3 = C$. Notons $\beta_C(t) = C.t$. Dans ce cas, le service résiduel offert à C_1 est $\beta_1 = \left[\beta - \alpha_2^3\right]_\uparrow$ avec $\alpha_2^3 = \alpha_2 \oslash \beta^2$. La courbe correspondante est donnée en Figure 5.12a. Chaque segment de pente nulle de β_1 est de longueur $\frac{L_2}{C}$ et correspond au passage d'un paquet de f_2. La courbe correspond donc bien au service résiduel d'un élément réseau partagé avec priorité préemptive.

Supposons maintenant que $C_1 = C_3$ et que $C_2 < C_1$. Comme $C_2 < C_3$, α_2^3 a une pente C_2 puisque le contrôle de flux contraint les paquets de f_2 à s'écouler à la vitesse la plus faible. Dans ce cas, β_1 a l'aspect de la courbe de la Figure 5.12b. Comme on le voit, cette courbe ne correspond pas à un service valide puisqu'elle autorise f_1 à transmettre un paquet à vitesse réduite $C_3 - C_2$ pendant qu'un paquet de f_2 est en cours de transfert.

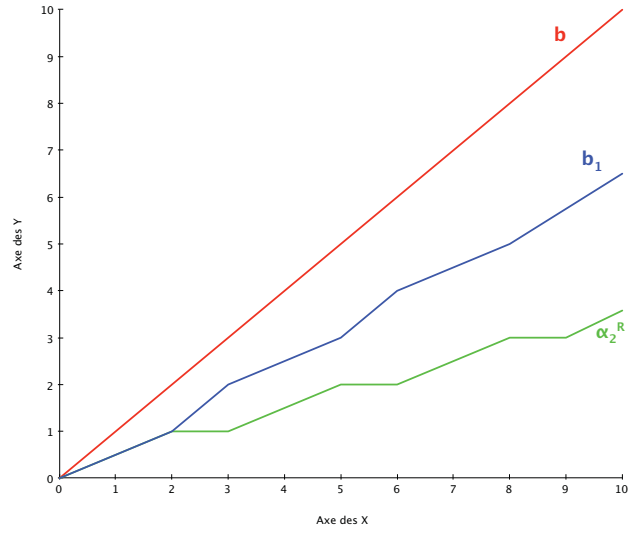
Ainsi, dans le cas général, l'utilisation du "blind multiplexing" n'est possible que si tous les liens ont la même capacité.

Taux de service des terminaux Il est cependant possible de prendre correctement en compte la présence de terminaux dont le taux de service est inférieur à la capacité des liens du réseau. Etant donné que ce terminal se situe forcément à la fin du chemin suivi par un (ou plusieurs flux), la situation présentée au paragraphe précédent ne peut pas se produire.

La prise en compte du taux de service d'un terminal se subdivise en deux cas.



(a) si tous les liens ont la même vitesse



(b) en présence d'un lien de vitesse faible

FIGURE 5.12 – Service résiduel offert au flux f_1

Nous présenterons ces deux cas dans le cas d'un réseau contenant deux flux mais la généralisation est aisée.

D'une part, le terminal de taux de service faible peut être partagé par les deux flux. Supposons que le terminal D du réseau de la Figure 5.11 ait un taux de service $C_D < C = C_1 = C_2 = C_3$. Dans ce cas, la courbe de service offerte à f_1 est $\beta_1 = \left[\beta \otimes \beta_D - \alpha_2^3\right]_\uparrow$. Cette courbe correspond bien au service offert par la Section wormhole qui comprend l_3 et D, moins la part du service utilisé par f_2.

D'autre part, si f_1 et f_2 ont des destinations différentes, chacune peut avoir un taux de service différent, inférieur à la capacité des liens du réseau. Bien sûr, si f_2 a pour destination un terminal D_2 de faible taux de service, cela aura un impact sur f_1. Cet impact est pleinement pris en compte dans le délai imposé par f_2 à f_1 à la fin de leur Section wormhole commune. En effet, ce délai dépend de la courbe de service β_2^{dest} offerte à f_2 et cette courbe inclut la courbe de service de D_2.

Notre modèle basé sur le Calcul Réseau prend donc correctement en compte le taux de service des terminaux. De même, le modèle peut facilement prendre en compte le fait que le dernier lien traversé par un flux ait une capacité plus faible que le reste du réseau. Dans ce cas, il est nécessaire que le terminal destination du flux ne soit pas la source d'un autre flux.

5.4 Amélioration de la courbe de service de bout en bout

5.4.1 Agrégation des flux en conflits

Si on applique la méthode de calcul présentée en Section 5.3 de façon immédiate, on risque de surestimer fortement les délais causés au flux étudié par des flux en conflits qui quitteraient le flux par un même port de sortie. En effet, dans ce cas, on va compter que chacun de ces flux doublent l'autre à partir de la fin de la section wormhole commune avec le flux étudié, ce qui va énormément augmenter le délai. Nous allons d'abord illustrer ce problème sur un exemple puis nous présenterons une solution pour calculer un délai plus faible pour ces flux.

5.4.1.1 Exemple de surestimation du délai de sortie

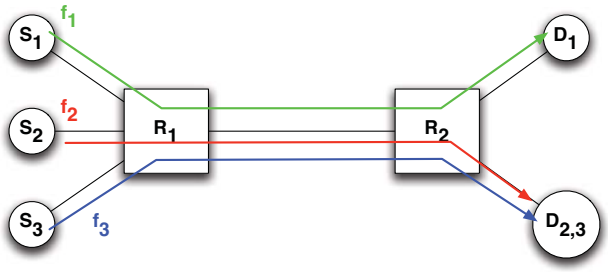


FIGURE 5.13 – Exemple de surestimation du délai de sortie

On suppose que chaque lien offre une courbe de service $\beta(t) = C.t$ et que chaque flux est contraint par une courbe d'arrivée α_i à l'entrée du réseau.

En appliquant le modèle présenté en Section 5.3 à ce réseau, la courbe de service de bout en bout offerte à f_1 est

$$\beta_1^{ETE} = \beta \otimes \left[\beta - \alpha_2^{R_1} - \alpha_3^{R_1}\right]_\uparrow \otimes \delta_{d_2+d_3}$$

avec $d_i = h(\alpha_i^{R_2}, \beta_i^{dest}), i = 2, 3$.

Ensuite, on a

$$\beta_2^{dest} = \left[\beta - \alpha_3^{R_2}\right]_\uparrow$$

$$\beta_3^{dest} = \left[\beta - \alpha_2^{R_2}\right]_\uparrow$$

et

$$\alpha_2^{R_2} = \alpha_3^{R_1} \oslash \left[\beta - \alpha_2^{R_1}\right]_\uparrow$$

$$\alpha_3^{R_2} = \alpha_2^{R_1} \oslash \left[\beta - \alpha_3^{R_1}\right]_\uparrow$$

Comme on peut le constater, avec ce calcul, on suppose que f_3 bloque f_2 en sortie de R_2 et, simultanément, que f_2 bloque f_3. En réalité, seul un des deux blocages peut se produire et ce calcul est inutilement pessimiste.

On peut donc améliorer la courbe de service offerte à f_1 en traitant f_2 et f_3 comme un seul flux agrégé à partir de R_1. Ce flux agrégé $f_{2,3}$ aura pour courbes d'arrivée

$$\alpha_{2,3}^{R_1} = \alpha_2^{R_1} + \alpha_3^{R_1}$$

$$\alpha_{2,3}^{R_2} = \alpha_{2,3}^{R_1} \oslash \beta$$

De plus, aucun flux n'interfère avec le flux $f_{2,3}$ entre R_2 et $D_{2,3}$, le délai de sortie $d_{2,3}$ est donc

$$d_{2,3} = h(\alpha_{2,3}^{R_2}, \beta)$$

et la courbe de service de bout en bout offerte à f_1 :

$$\beta_1^{ETE} = \beta \otimes \left[\beta - \alpha_{2,3}^{R_1}\right]_\uparrow \otimes \delta_{d_{2,3}}$$

Il est aisé de constater que cette courbe de service est bien meilleure que la précédente mais nous pouvons utiliser une application numérique pour illustrer l'amélioration de la borne calculée.

Prenons les paramètres suivants. Les liens ont une capacité $C = 200$ Mbps. Les flux ont une période de 2 ms et leurs paquets sont de taille 4000 caractères.

Dans le premier cas, on a $d_2 = d_3 = 0,4$ ms et le délai de bout en bout pour f_1 est $1,4$ ms. En agrégeant f_2 et f_3, on a $d_{2,3} = 0,2$ ms et le délai de bout en bout de f_1 est $0,8$ ms. On voit donc que même sur un réseau de faible taille, cette optimisation apporte une amélioration substantielle de la borne de bout en bout.

L'algorithme utilisé pour agréger les flux en conflit est donné en Annexe A.2

5.4.2 Utilisation de la courbe de service maximale

Comme expliqué en Section 5.1, l'utilisation de la courbe de service maximale permet d'optimiser le calcul de la courbe d'arrivée d'un flux après la traversée d'un élément système. Ici, nous pouvons l'utiliser pour avoir une contrainte plus stricte sur les flux en conflit avec le flux étudié f_e.

Comme nous avons supposé que tous les liens du réseau ont la même capacité C, la courbe de de service maximale γ offerte à chacun des flux est $\gamma(t) = C.t$.

Concrètement, l'utilisation d'une courbe de service maximale revient à dire que la quantité de données d'un flux entrant par un port d'entrée est bornée par la vitesse maximale à laquelle le flux a pu s'écouler avant d'atteindre l'entrée de S.

Si on note $\beta_i^{S_{in}}$ la courbe de service reçue par f_i avant S, la courbe d'arrivée $\alpha_i^{S_{in}}$ sera

$$\alpha_i^{S_{in}} = (\alpha_i \otimes \gamma) \oslash \beta_i^{S_{in}}. \tag{5.12}$$

Pour les applications, nous utiliserons (5.12) dans le calcul itératif à la place de (5.11).

5.5 Limites de cette méthode

La méthode de calcul basée sur le Calcul Réseau présente deux limites notables. La première est que nous n'avons pas de preuve formelle de la convergence de la méthode du point fixe. Cette méthode converge et fournit des résultats en un nombre réduits d'itération (quelques dizaines au plus) sur tous les réseaux étudiés au chapitre 6.

Pour aider la convergence du système d'équations, il est également possible de spécifier à l'avance une borne maximale pour chaque flux. Ainsi, au fur et à mesure des itérations, les délais de bout en bout des différents flux augmentent et il est possible d'arrêter le calcul lorsque la borne d'un flux dépasse la borne maximale.

La seconde limite de cette méthode est lié au modèle de partage de section choisi. Le modèle "blind multiplexing" suppose que lorsque deux flux partagent une section wormhole, l'un des deux est plus prioritaire et obtient tout le service qu'il demande. Le deuxième flux n'a à sa disposition qu'un service résiduel et le service garanti peut être très faible.

Ce modèle reste réaliste lorsque les flux ont des débits du même ordre de grandeur. En revanche, lorsque l'un des flux a un débit très élevé et l'autre un débit très faible, le modèle préemptif défavorise largement ce dernier.

Considérons par exemple, le réseau de la Figure 5.11. Nous supposons ici que tous les liens ont la même capacité $C = 50$ Mbps. Les paramètres des flux sont les suivants :

	Période (ms)	Taille des paquets (octets)
f_1	0,025	100
f_2	40	4000

Avec ces paramètres, on obtient une borne de 0,82 ms pour f_1 et 4 ms pour f_2. En réalité, un paquet de f_2 peut être bloqué par un seul paquet de f_1, soit un délai de $\frac{40000+1000}{5.10^7} = 0,82$ ms. Sur ce petit exemple, le modèle préemptif donne donc une borne cinq fois plus élevée que le pire cas réel. Notons que le pire-cas exact calculé ici correspond à la borne donnée par notre première méthode de calcul (voir Section 4.1).

Nous comparerons plus en détail les performances des différentes méthodes au chapitre 6.

5.6 Bilan sur cette méthode

Dans cette Section, nous avons présenté un modèle de réseau SpaceWire basé sur la théorie du Calcul Réseau. Ce modèle est construit à partir d'un nouvel élément réseau, la section wormhole. Cet élément permet de modéliser les interférences entre les flux ainsi que l'effet du mécanisme de contrôle de flux de façon plus précise que les éléments réseau traditionnels comme les multiplexeurs et les démultiplexeurs. En particulier, notre modèle permet de répercuter les blocages en aval subis par un flux démultiplexé de façon beaucoup moins pessimiste que le modèle de [22].

Nous avons également présenté une méthode d'analyse d'un réseau complet qui s'inspire des schémas d'interférence définis par [22] et qui permet de calculer la courbe de service de bout en bout offerte à chacun des flux utilisant le réseau.

Enfin, nous avons présenté une méthode itérative permettant de résoudre les problèmes de dépendance circulaire qui peuvent survenir en calculant ces courbes de service.

Nous avons implémenté la méthode de calcul complète en utilisant Matlab [27] et la Toolbox RTC [28]. Cette boîte à outil fournit les fonctions basiques du Calcul Réseau, telles que convolution et déconvolution. L'implémentation inclut l'application des schémas d'interférence, le calcul itératif des courbes d'arrivée et l'agrégation des flux sortants pour optimiser la courbe de service de bout en bout. Le algorithmes détaillés sont données en Annexe A.

Comparée aux méthodes du chapitre 4, le Calcul Réseau impose de caractériser de façon plus précise le trafic injecté dans le réseau à l'aide de courbes d'arrivée. Une courbe d'arrivée impose de connaître la taille maximale des paquets qu'un flux peut émettre ainsi que la période du flux ou un intervalle minimal entre deux paquets. On peut ainsi lever les restrictions de la première méthode de calcul récursive tout en n'ayant pas besoin de supposer que tous les flux émettent à leur vitesse maximale en permanence.

Chapitre 6

Comparaison des différentes méthodes de calcul

Ce chapitre présente une comparaison entre les trois méthodes de calcul présentées aux chapitres 4 et 5. La première comparaison se base sur une configuration industrielle fournie par Thales Alenia Space. La première méthode de calcul récursive n'étant pas applicable à ce cas d'étude, nous ne comparerons que les résultats de la seconde méthode récursive avec ceux fournis par le Calcul Réseau ainsi qu'avec les résultats de simulation fournis par Thales Alenia Space.

Dans un second temps, nous comparerons les trois méthodes de calcul sur plusieurs exemples plus simples qui permettront de déterminer quelle méthode est la plus appropriée à chaque situation.

Dans la suite du chapitre, la première méthode de calcul récursive sera dénommée RC1 (Recursive Calculus) et la seconde RC2. La méthode basée sur le Calcul Réseau sera notée NC (Network Calculus).

6.1 Application au cas d'étude Thales Alenia Space

6.1.1 Description du cas d'étude

Ce cas d'étude est basé sur une architecture réseau destinée à un satellite d'observation. Il s'agit d'une architecture dans laquelle les paquets de commande/contrôle et scientifique se partagent un seul et même réseau.

L'architecture réseau est présentée sur la Figure 6.1. Elle comprend

- 4 routeurs SpaceWire qui utilisent l'adressage logique ;
- 9 applications correspondant à des instruments de charge utile scientifique ;
- 5 équipements liés à la plate-forme.

L'ensemble du trafic échangé dans le réseau peut être classé selon 4 types de paquets :

Paquets SC (SCience data) : transportent les mesures acquises par les applications et sont destinés à la mémoire de masse

Paquets HK (HouseKeeping) : indiquent l'état des différents instruments et du processeur et sont stockés dans la mémoire de masse

Paquets CMD (CoMmanD data) : correspondent aux commandes envoyées par le processeur au contrôleur de la mémoire de masse ou aux instruments

Paquets TM (TeleMetry) : transportent les mesures stockées dans la mémoire de masse et qui seront émises vers la station sol

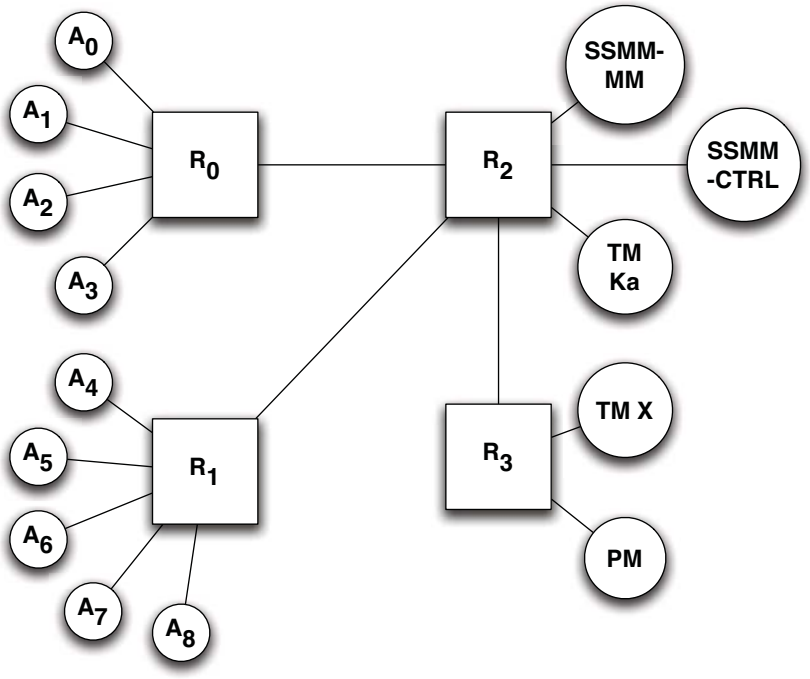


FIGURE 6.1 – Topologie réseau du cas d'étude

Chaque type de paquet suit un chemin bien défini :
- Paquet SC
  - Application → SSMM-CTRL → SSMM-MM
- Paquet HK
  - Application → PM → SSMM-CTRL → SSMM-MM
  - PM → SSMM-CTRL → SSMM-MM
- Paquet CMD
  - PM → SSMM-CTRL
  - PM → Application
- Paquet TM
  - SSMM-MM → TM-Ka
  - SSMM-MM → TM-X

Comportement des nœuds

Les nœuds applicatifs (A_0 à A_8) génèrent et reçoivent des paquets de façon décorrélée. Les paquets reçus sont lus instantanément et détruits. Les paquets sont générés selon deux processus indépendants : un pour les paquets SC, un pour les paquets HK. Pour chacun des processus, les paquets sont générés de façon parfaitement périodique et ont une taille fixe.

L'élément SSMM-MM (pour Mass Memory) lit et détruit instantanément tous les paquets reçus. Il émet des paquets à destination des deux TM Encoders. Les paquets sont générés alternativement pour chaque destinataire à un débit légèrement

inférieur à la capacité de réception de chaque TM Encoder. La mémoire est supposée suffisamment remplie pour que l'émission vers les TM encodeurs soit continue.

L'élément SSMM-CTRL (contrôleur de mémoire de masse) mémorise chaque paquet reçu de type HK ou SC. L'élement ne peut pas stocker plus d'un paquet et bloque la réception d'autres paquets tant que son buffer est utilisé. Le contrôleur détruit les paquets de commande reçus. Il ne génère aucun paquet mais retransmet les paquets HK et SC vers le SSMM-MM après un délai de traitement fixe.

Les deux éléments TM Encoders (TransMitters) lisent les paquets reçus à un débit constant et n'émettent aucun paquet.

L'élement PM (Processing Module) reçoit les paquets HK de tous les instruments. Il lit les paquets à un débit constant et enregistre chaque paquet reçu complet. Il ne peut stocker qu'un seul paquet HK a un instant donné et bloque la réception des paquets suivants jusqu'à ce que sa mémoire soit à nouveau libre. Le PM retransmet les paquets HK reçus au contrôleur SSMM-CTRL après un délai de traitement. Il génère également ses propres paquets HK à destination de l'élément SSMM-CTRL. De plus, le PM génère des paquets de type CMD à destination de toutes les applications et du contrôleur SSMM-CTRL. La taille des commandes et l'intervalle de génération sont constants.

Les délais de traitement des équipements PM et SSMM-CTRL sont paramétrables dans les simulations et dans les outils de calcul mais ne le serait pas dans un équipement réel.

Valeur des paramètres

On considère deux scénarios. Dans le premier, les applications émettent des paquets scientifiques de grande taille. Dans le second, les paquets scientifiques sont segmentés au niveau de l'application et sont de taille beaucoup plus faible.

Les paramètres pour les nœuds A_0 à A_8 sont donnés dans les Tableaux 6.1 et 6.2 pour ces deux scénarios. Dans le second scénario seules les tailles et les périodes des paquets scientifiques sont modifiées.

Tous les autres nœuds fonctionnent de la même façon dans les deux scénarios.

Scénario 1	A0	A1	A2	A3	A4	A5	A6	A7	A8
HK Période (s)	0,16	0,0533	0,16	5,33	0,32	1,33	0,2	8	16
Taille HK (bytes)	20	20	20	20	20	20	20	20	20
Science Période (s)	0,00221	0,0264	0,16	3,2	0,0075	0,0985	0,4	9,697	20
Taille Science (bytes)	4000	4000	4000	4000	4000	4000	4000	4000	4000
Taille buffer réception	1 kB	1 kB	1 kB	1 kB	1 kB	1 kB	1 kB	1 kB	1 kB
Débit entrant (Mbps)	50	50	50	50	50	50	50	50	50

Table 6.1 – Paramètres pour les nœuds applicatifs : scénario 1

Scénario 2	A0	A1	A2	A3	A4	A5	A6	A7	A8
Science Période (s)	$5{,}5 \times 10^{-5}$	0,00066	0,004	0,08	0,00019	0,00246	0,01	0,242	0,5
Taille Science (bytes)	100	100	100	100	100	100	100	100	100

Table 6.2 – Paramètres pour les nœuds applicatifs : scénario 2

Période (s)	Taille des paquets (bytes)	Taille buffer (bytes)	Taux de service (bytes/s)	Débit entrant (Mbps)
0,04	4000	64	6 250 000	50

Table 6.3 – Paramètres pour l'élement SSMM-MM

Taille buffer réception (bytes)	Taille buffer applicatif (bytes)	Délai (μs)	Débit entrant (Mbps)
64	4000	10	50

Table 6.4 – Paramètres pour l'élément SSMM-CTRL

Taille buffer réception (bytes)	Taux de service (kbytes/s)
4000	87,5

Table 6.5 – Paramètres pour les éléments TM Encoders

Taille buffer réception (bytes)	64
Taux de service (bytes/s)	1250
Période HK (s)	0,08
Taille HK (bytes)	100
Période CMD (s)	0,08
Taille CMD (bytes)	1000
Délai (μs)	10
Débit entrant (Mbps)	20

TABLE 6.6 – Paramètres pour l'élément PM

6.1.2 Remarques préliminaires sur les résultats

Dans les Sous-Sections suivantes, nous présentons deux valeurs de types différents. D'un côté, le résultat fourni par les méthodes de calcul des chapitres 4 et 5 est une borne supérieure sur le délai pire-cas d'un paquet. Cette borne est une valeur théorique qui n'est pas nécessairement atteinte dans la réalité. De l'autre, la valeur fournie par une simulation est le délai maximum observé pour un paquet au cours de cette simulation particulière. Il ne s'agit donc pas forcément du plus grand délai possible mais seulement du plus grand délai survenu au cours d'une simulation nécessairement incomplète et limitée dans le temps.

D'une manière générale, on peut ordonner ces trois valeurs par ordre croissant : délai maximum observé $\leq$ pire cas réel, atteignable $\leq$ borne supérieure analytique. La Figure 6.2 illustre la disposition de ces trois valeurs. Le graphique présente plusieurs distributions possibles des délais de bout en bout, la distribution réelle étant inconnue.

6.1.3 Résultats des différentes méthodes de calcul

Nous comparerons les résultats fournis par les différentes méthodes que nous avons présentées avec les valeurs observées grâce au simulateur MOST (Modeling Of SpaceWire Traffic) [29] de Thales Alenia Space.

MOST est un simulateur au niveau caractère basé sur OPNET [30] qui se présente comme un ensemble de briques de base telles que les routeurs SpaceWire et différents types de terminaux. Une interface graphique permet d'assembler les différentes briques et de créer la topologie du réseau. Le trafic injecté dans le réseau peut obéir à différentes lois statistiques ou être simplement périodique. MOST permet d'observer en détail le trafic échangé sur le réseau, notamment la distribution statistique des délais de bout, dont le pire cas observé pour chaque flux de données. Il permet également d'évaluer la taille maximale nécessaire des différents buffers du réseau. MOST est considéré comme abouti par Thales Alenia Space ; nous considérerons donc les résultats fournis par le simulateur comme représentatif d'un réseau réel.

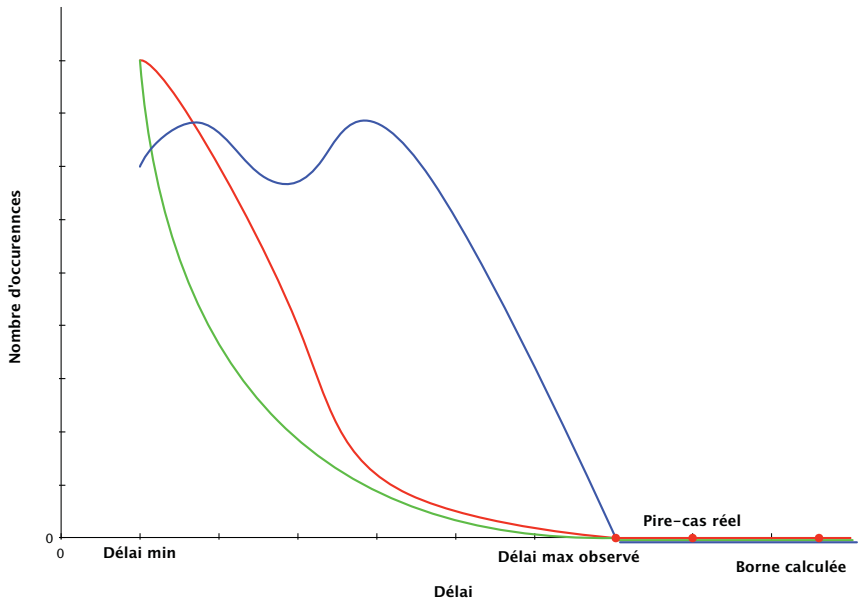


FIGURE 6.2 – Illustration des différentes valeurs comparées dans ce chapitre

Par ailleurs, chaque paquet d'un flux SC ou HK suit un trajet correspondant à plusieurs flux au sens défini pour nos différentes méthodes de calcul. Le délai calculé pour un paquet sera donc la somme des délais pour chacun des flux constituant son trajet. De plus, nous présentons les résultats sous forme synthétique, par type de trafic. Ainsi, dans ce qui suit la borne HK correspond à la borne maximale parmi celles des neufs flux HK.

6.1.3.1 Application de RC1

La première méthode de calcul que nous avons présentée (voir Section 4.1) n'est pas applicable au cas d'étude TAS. En effet, les paquets HK de taille 20 octets et les paquets SC du scénario 2 (100 octets) sont de faible taille et peuvent être stockés entièrement dans les buffers des routeurs intermédiaires.

On peut d'ailleurs vérifier que les résultats fournis par cette méthode sont incorrects. Dans le scénario 2, la borne calculée pour les flux de type CMD est inférieure au délai le plus élevé observé lors des simulations (voir Tableau 6.7).

Flux	CMD	HK	SC	TM
MOST	2,15	145	16,6	58,2
RC1	**0,63**	875,58	876,21	154,31

TABLE 6.7 – Scénario 2 : Comparaison entre RC1 et MOST. Délais en ms

Ceci justifie l'insuffisance de la méthode RC1 et la nécessité de chercher d'autres moyens d'étude.

6.1.3.2 Application de RC2

Les résultats fournis par la seconde méthode de calcul sont donnés dans les Tableaux 6.8 et 6.9.

Flux	CMD	HK	SC	TM
MOST	2,15	145	16,6	58,2
RC2	111,3	14 214	8 632	175,4

TABLE 6.8 – Scénario 1 : Comparaison entre RC2 et MOST. Résultats en ms

Flux	CMD	HK	SC	TM
MOST	0,58	143	2,9	54
RC2	6,8	13 915	10 835	175

TABLE 6.9 – Scénario 2 : Comparaison entre RC2 et MOST. Résultats en ms

Sur le cas d'étude TAS, cette méthode de calcul fournit des bornes très élevées. Par exemple, les bornes calculées pour les flux de type HK et SC peuvent être supérieures à 10 s. Les bornes fournies par cette méthode ne nous paraissent donc pas exploitables sur le cas d'étude.

L'explication du pessimisme de cette méthode est la même que pour l'exemple présenté en 4.2.3.3. Comme notre méthode fonctionne octet par octet, elle crée un scénario pire-cas impossible à réaliser en pratique car elle compte une succession de paquets en conflits qui ne peut se produire.

Sur ce réseau, ce phénomène est particulièrement marqué et renforcé par plusieurs facteurs. Premièrement, les paquets des flux HK sont de taille très faible (20 octets), ce qui oblige à en compter trois dans chaque buffer d'entrée traversé par un paquet. Deuxièmement, le terminal PM qui traite les paquets HK a un taux de service très faible ce qui fait que chacun des paquets HK comptés en trop cause un délai important. Enfin, les paquets HK traversent successivement trois routeurs (R_0 ou R_1, R_2, R_3) et sont entrelacés avec de nombreux flux différents (plusieurs flux de type SC et TM) ce qui augmente les possibilités de création de scénarios impossibles.

Au total, notre méthode de calcul finit par compter plusieurs centaines de paquets HK livrés dans le délai de livraison du paquet HK ou SC étudié. Etant donné que le terminal PM a un taux de service de 10 kbps et que les paquets HK sont de taille 20 octets, la livraison de 500 paquets HK nécessite un délai $D_{500} = \frac{20*8*500}{10000} = 8\ s$, du même ordre de grandeur que les bornes observées.

Le fait que cette méthode soit pessimiste parce qu'elle compte de nombreux paquets supplémentaires a motivé la conception d'une autre méthode de calcul, qui permette de prendre en compte le nombre exact de paquets de chaque flux en mesure d'interférer avec le paquet étudié.

6.1.3.3 Application de la méthode NC

Vérification de la condition de stabilité Avant d'appliquer la méthode NC à ce réseau, il nous faut vérifier la condition de stabilité définie en 5.2.1. Nous ne présenterons pas le calcul pour tous les liens et tous les flux mais seulement pour quelques exemples représentatifs. Pour rappel, la condition de stabilité du réseau est la suivante : le poids d'un flux est égal au rapport du débit moyen du flux sur la capacité minimale traversée. Sur chaque lien, la somme des poids du flux doit être inférieure à 1.

Le débit de chaque flux peut être déduit facilement de la période et de la taille des paquets du flux. La plupart des flux traversent uniquement des liens à 50 Mbps. Seuls les flux $TM\ Ka$ et $TM\ X$ traversent des liens à 20 Mbps. De plus, les deux flux ont le même débit de 0,5 Mbps. Leur poids est donc de $0,025$.

Les débits et poids des flux HK et SC sont donnés dans le Tableau 6.10.

	SC		HK	
N° du flux	Débit moyen (Mbps)	Poids	Débit moyen (Mbps)	Poids
1	18,2	0,36	0,0013	25.10^{-6}
2	1,6	0,03	0,0038	75.10^{-6}
3	0,25	0,005	0,0013	25.10^{-6}
4	0,0125	0,00025	0,0004	$0,76.10^{-6}$
5	0,54	0,11	0,0006	$12,5.10^{-6}$
6	0,41	0,008	0,0002	3.10^{-6}
7	0,1	0,002	0,001	20.10^{-6}
8	0,004	0,00008	0,00003	$0,5.10^{-6}$
9	0,002	0,00004	0,00002	$0,26.10^{-6}$

TABLE 6.10 – Débits et poids pour les flux SC et HK

Tous les flux CMD ont le même débit de $1,12$ Mbps et un poids de $0,02$. Le flux de paquets HK générés par le PM à destination du terminal $SSMM-CTRL$ a un débit de $0,013$ Mbps et un poids de $0,25.10^{-3}$. Enfin, le flux constitué par tous les paquets HK retransmis par le PM au $SSMM-CTRL$ a un débit de $0,008$ Mbps et un poids de $0,16^{-3}$.

Nous pouvons maintenant calculer la somme des poids des différents flux sur chaque lien. Le lien $R_0 \rightarrow R_2$ est utilisé par les flux SC_0 à SC_3 et HK_0 à HK_3. La somme de leurs poids est $0,40$. De même, le lien $R_1 \rightarrow R_2$ est utilisé par les flux SC_4 à SC_8 et HK_4 à HK_8. La somme de leurs poids est de $0,12$.

Le lien $R_2 \rightarrow SSMM-CTRL$ est utilisé par tous les flux SC, le flux HK généré par le PM ainsi que le flux CMD à destination de $CTRL$ et les flux HK retransmis par le PM. Leur poids est de $0,53$. Il s'agit du lien le plus chargé du réseau.

Le lien $R_2 \rightarrow R_3$ est utilisé par les neuf flux HK et le flux $TM\ X$. Leur poids est

de $25,2.10^{-3}$. Enfin, le lien $R_3 \rightarrow R_2$ est utilisé par les dix flux CMD (à destination des neufs terminaux A_i et du $CTRL$) et par les flux HK du PM vers le terminal $CTRL$. Leur poids est de $0,2$.

La condition de stabilité est ainsi vérifiée sur ces liens et, par extension, par tout le réseau puisque les liens sont clairement moins chargés que les liens pour lesquels nous avons calculé la somme des poids des flux.

Nous pouvons maintenant nous intéresser aux résultats pour la méthode NC qui sont donnés dans les Tableaux 6.11 et 6.12.

Remarque : les algorithmes présentés dans l'Annexe A n'ont été implémenté que tardivement et le code n'est pas finalisé. Les résultats présentés dans ce chapitre pour la méthode NC ont donc été obtenus à l'aide d'une implémentation "manuelle" : l'application des schémas d'interférence et l'agrégation des flux ont été réalisées manuellement. Nous avons ensuite utilisé Matlab pour réaliser le calcul itératif sur le système d'équations obtenues à la main.

On peut d'abord remarquer que les bornes fournies par cette méthode sont correctes (i.e. supérieures aux résultats de MOST) et beaucoup moins élevées que celles fournies par RC2.

Flux	CMD	HK	SC	TM
MOST	2,2	145	16,6	58
NetCal	91	635	212	161

Table 6.11 – Scénario 1 : Comparaison entre NC et MOST. Résultats en ms

Premier scénario Sur le premier scénario, les bornes des flux HK et TM sont peu pessimistes par rapport aux pires cas observés dans les simulations. Les bornes pour ces deux flux sont entre 3 et 5 fois supérieures aux délais observés. Ce résultat est très satisfaisant car, en général, le rapport entre Calcul Réseau et simulations est beaucoup plus important.

La borne calculée pour les flux SC est plus élevée en comparaison du pire cas observé avec MOST : la borne est environ 13 fois supérieure au pire cas observé. Une première explication est que le pire cas observé est une valeur ponctuelle observée sur une seule simulation et il est probable que des délais plus élevés soient réalisables en pratique avec d'autres paramètres ou une simulation plus longue.

Une deuxième explication est que, dans notre modèle, les flux HK ont des délais élevés car leur destination (PM) a une courbe de service très faible. Comme l'analyse pire-cas suppose que les flux HK émettent en même temps, le délai dû au démultiplexage de ces flux est important et augmente fortement la borne des flux SC. Cette hypothèse peut facilement être vérifiée expérimentalement. En effet, lors du calcul on observe que le délai $d_{HK_{0,3}}$ que les quatre flux HK_0 à HK_3 causent à chacun des flux SC_0 à SC_3 est de $59,6$ ms. Le délai $d_{HK_{4,8}}$ causé par les flux HK_4 et HK_8 aux flux SC_4 à SC_8 est quant à lui de 114 ms. Ceci confirme bien que la

valeur élevée de la borne des flux SC est due essentiellement aux interférences avec les flux HK. Ce résultat est remarquable dans la mesure où, dans cette configuration, les paquets urgents HK de taille 20 caractères causent des délais importants aux paquets non-urgents SC de taille 4000 caractères.

Enfin, la borne calculée pour les flux CMD est très importante. NC fournit une borne 46 fois supérieure au délai maximum observé durant la simulation. Ceci illustre la faiblesse principale de notre méthode de calcul, à savoir l'utilisation d'un modèle préemptif du partage d'une section wormhole (voir les sections 5.2.4 et 5.5). En effet, l'un des flux CMD (qui ont tous pour source le terminal PM) a pour destination le terminal CTRL. Ce flux interfère donc avec tous les flux SC qui ont la même destination.

D'une part, le modèle préemptif implique que les neufs flux SC vont avoir accès au lien $R_2 \rightarrow CTRL$ avant le paquet du flux CMD ce qui est impossible en réalité. En effet, les flux SC n'arrivent que sur deux ports d'entrée du routeur R_2 et le mécanisme d'arbitrage Round-Robin garantit qu'au plus un paquet de chacun de ces ports peut passer avant le paquet CMD. D'autre part, étant donné que les flux SC subissent des interférences avec les flux HK avant d'atteindre le routeur R_2 leurs courbes d'arrivée dans le routeur R_2 sont supérieures à leur courbes d'arrivée à leur entrée dans le réseau. Cela revient à dire que plus d'un paquet de chaque flux SC passe avant le paquet du flux CMD. Nous avons déjà vu que les flux HK ont un impact très important sur les flux SC. Cet impact est indirectement reporté sur le flux CMD à destination du terminal $CTRL$. Comme tous les flux CMD interfèrent entre eux au début de leur trajet, ils ont tous la même borne très élevée.

Flux	CMD	HK	SC	TM
MOST	0,58	143	2,9	54
NetCal	70	562	169	151

Table 6.12 – Scénario 2 : Comparaison entre NC et MOST. Résultats en ms

Second scénario Dans le second scénario, les paquets des flux SC sont segmentés au niveau applicatif en petits paquets de 100 octets.

La première observation est que cela n'a presque aucune influence sur les flux TM que ce soit avec MOST ou NC. Ceci est normal étant donné que ces flux n'ont pas d'interaction avec les flux SC.

De même, avec MOST, les flux HK ont le même délai maximum observé dans les deux scénarios. En revanche, la borne calculée par NC est réduite de plus de 10 %. Dans la mesure où les flux HK et SC sont en conflits directs, il n'est pas étonnant que la borne des flux HK soit plus faible dans le scénario 2.

On peut par contre se demander pourquoi cette amélioration n'est pas plus forte. La réponse logique est que la majorité du délai subi par les paquets HK est due au goulot d'étranglement formé par le terminal PM. La taille des paquets SC n'a

donc qu'un impact modéré sur la borne pire-cas.

Pour les flux SC et CMD, les délais observés avec MOST sont nettement meilleurs dans le second scénario que dans le premier. Il paraît logique que livrer un paquet de 100 octets soit plus rapide que livrer un paquet de 4000 octets. Comme le délai des flux CMD dépend directement du conflit entre le flux CMD à destination du terminal $CTRL$ et les flux SC, la réduction du délai de livraison se retrouve naturellement reportée sur ces flux.

Par contre, avec la méthode NC, les bornes des flux CMD et SC sont réduites dans le second scénario mais de façon relativement faible. Tout comme dans le premier scénario, l'essentiel du pessimisme de notre méthode provient des délais importants causés par les flux HK aux flux SC et reportés indirectement sur les flux CMD.

Bilan En conclusion, la méthode NC permet de couvrir complètement le cas d'étude de Thales Alenia Space et fournit des bornes exploitables dans un outil d'aide à la conception réseau. Sur cette configuration, elle nous a permis d'identifier le principal goulot d'étranglement du réseau, à savoir le terminal PM.

En revanche, le modèle préemptif utilisé dans cette méthode conduit à surestimer l'impact du goulot d'étranglement et à propager son effet à des flux qui ne devraient pas être affectés de façon aussi forte.

6.1.3.4 Comparaison graphique

Le graphique de la Figure 6.3 permet d'illustrer les différences d'ordre de grandeur entre les différentes méthodes pour chaque type de délai sur le scénario 1. Nous n'avons pas inclus le même graphique pour le scénario 2 car il est similaire à celui du scénario 1.

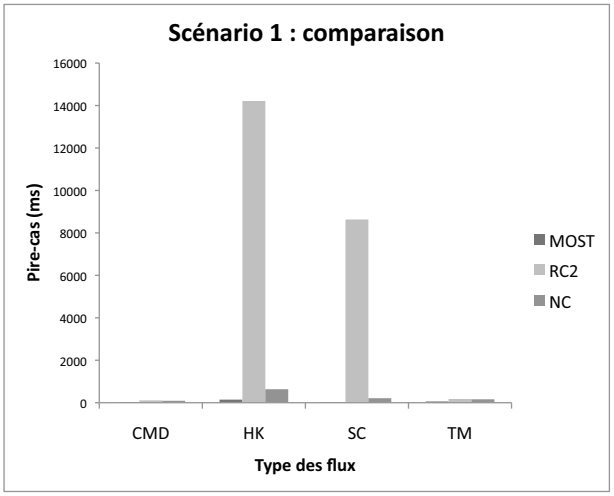


Figure 6.3 – Comparaison entre MOST, RC2 et NC

6.2 Comparaison sur différents réseaux

Cette Section présente une comparaison des trois méthodes de calcul sur plusieurs réseaux de topologie plus simples que le cas d'étude TAS. Le but ici est de mettre en valeur les situations dans lesquelles une méthode est plus efficace qu'une autre.

Dans cette Section, sauf mention contraire, on suppose que tous les liens fonctionnent à la capacité $C = 50$ Mbps.

6.2.1 Etude de l'impact de la taille des paquets

Ce réseau comprend cinq terminaux et un routeur (voir Figure 6.4). Quatre flux f_1 à f_4 traversent ce réseau depuis les nœuds S_1 à S_4 respectivement. Les quatre flux ont pour destination le terminal D. Les paramètres des trois scénarios considérés sont fournis dans le Tableau 6.13 et les résultats dans le Tableau 6.14.

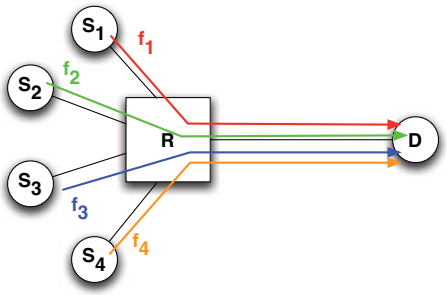


FIGURE 6.4 – Premier réseau

Flux	Scénario 1		Scénario 2		Scénario 3	
	Taille	Période	Taille	Période	Taille	Période
f_1	4000	7	4000	2,2	4000	1,65
f_2	4000	7	4000	6,4	20	0,32
f_3	4000	7	4000	2,2	4000	1,65
f_4	4000	7	4000	6,4	20	0,32

TABLE 6.13 – Paramètres pour les différents scénarios sur le premier réseau

Dans le premier scénario, la méthode RC1 fournit une borne optimale. RC1 fournit la même borne pour tous les flux : $d = \frac{L_1+L_2+L_3+L_4}{C}$. RC2 fournit un résultat quasi identique.

La méthode NC fournit des résultats différents suivant les périodes des flux. Pour f_1, la courbe de service de bout en bout utilisée par NC est

$$\beta_1^{ETE} = \beta \otimes \left[\left[[\beta - \alpha_2^R]_\uparrow - \alpha_3^R \right]_\uparrow - \alpha_4^R \right]_\uparrow$$

	Scénario 1	Scénario 2	Scénario 3
RC1	3,2	3,2	_
RC2	3,21	3,21	f_1, f_3 : 1,6 f_2, f_4 : 6,4
NC	3,2	f_1, f_3 : 5,47 f_2, f_4 : 10,6	1,6

TABLE 6.14 – Résultats sur le premier réseau (ms)

où $\beta(t) = C.t$ et $\alpha_i^R = (\alpha_i \otimes \beta) \oslash \beta$ est la courbe d'arrivé du flux f_i dans le routeur R. β_1^{ETE} est obtenue par application successive du schéma d'interférence imbriqué. Les autres flux ont des courbes de service similaires.

Si les périodes des flux sont élevées (Scénario 1), le service résiduel conduit à prendre en compte qu'un paquet de chaque autre flux double chaque paquet du flux étudié, comme pour RC1 et RC2. La borne calculée est donc identique.

En revanche, si la période d'un flux est faible (Scénario 2), le modèle préemptif utilisé fait que la méthode NC compte plus de paquets que RC1 et RC2. Elle donne ainsi une borne plus pessimiste. Dans le scénario 2, où les flux ont des périodes différentes, la borne calculée par NC pour les flux de plus grande période (f_2 et f_4) est 3 fois plus élevée que celle donnée par $RC1$ et $RC2$.

La situation est un peu différente lorsque l'on considère des paquets de faible taille (Scénario 3). Supposons par exemple que les flux f_2 et f_4 émettent des paquets de taille 20 octets. Dans ce cas, RC1 n'est pas applicable. De plus, RC2 donne des bornes élevées pour f_2 et f_4. Ceci s'explique par le fait qu'avec des paquets de faible taille, cette méthode considère que plusieurs paquets de ces deux flux sont présents simultanément dans le réseau. Comme chacun de ces paquets peut être doublé par un paquet de f_1 et un paquet de f_3, cela augmente la borne calculée de façon importante. Par contre, la méthode NC tient compte des périodes réelles des flux et ne compte que le nombre de paquets de f_2 et f_4 pouvant être réellement dans le réseau au même instant. La borne fournit n'augmente donc pas comme pour RC_2. Cependant, si l'un des flux avait une période très faible, le service résiduel pour les autres flux serait très faible, quelle que soit la taille des paquets de ce flux.

6.2.2 Influence de la périodicité des flux

Le second réseau comprend 4 terminaux et un routeur (voir Figure 6.5). 7 flux se partagent ce réseau. f_1 a pour source S_1. f_2, f_3 et f_4 ont pour source $S_{2,3,4}$. Enfin, f_5, f_6 et f_7 ont pour source $S_{5,6,7}$.

Les paramètres sont présentés dans les Tableaux 6.15 et 6.16 et les résultats dans le Tableau 6.17. La taille des paquets est la même dans les deux scénarios. De plus, dans un scénario donné, tous les flux ont la même période. Nous étudierons une période de 32 ms avec le scénario 1 et de 5 ms avec le scénario 2.

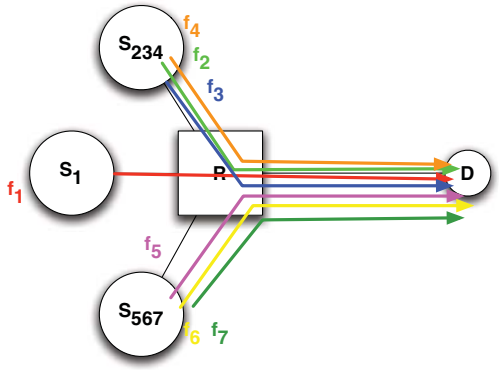


FIGURE 6.5 – Second réseau

Flux	Taille (octets)
f_1	4000
f_2	4000
f_3	300
f_4	1000
f_5	5000
f_6	500
f_7	2000

TABLE 6.15 – Taille des paquets pour les différents flux sur le second réseau

Pour la méthode RC1, la borne calculée pour f_1 est

$$d_1 = \frac{L_1 + \max(L_2, L_3, L_4) + \max(L_5, L_6, L_7)}{C}.$$

f_2, f_3 et f_4 ont tous les trois la même borne :

$$d_2 = d_3 = d_4 = \frac{3.L_1 + L_2 + L_3 + L_4 + 3.\max(L_5, L_6, L_7)}{C}.$$

De même, f_5, f_6 et f_7 ont tous la même borne :

$$d_5 = d_6 = d_7 = \frac{3.L_1 + 3\max(L_2, L_3, L_4) + L_5 + L_6 + L_7}{C}.$$

RC1 et RC2 donnent des bornes identiques toutes les deux ce qui est normal en l'absence de paquets de faible taille ou de liens de vitesses différentes. NC est plus

Scénario 1	32 ms
Scénario 2	5 ms

TABLE 6.16 – Période des flux sur le second réseau

Flux	RC1	RC2
f_1	2,6	2,61
f_2, f_3, f_4	6,46	6,47
f_5, f_6, f_7	6,3	6,31

Résultats pour NC		
Flux	Scénario 1	Scénario 2
f_1	3,65	6,89
f_2	3,65	6,89
f_3	3,75	9,88
f_4	3,73	9,13
f_5	3,63	6,36
f_6	3,74	9,66
f_7	3,70	8,24

TABLE 6.17 – Résultats sur le second réseau (ms)

pessimiste pour f_1 car notre modèle préemptif conduit à considérer qu'un paquet de chacun des flux f_2 à f_7 double chaque paquet de f_1. En revanche, dans le scénario 1, NC donne de meilleurs résultats que RC1 et RC2 pour les flux f_2 à f_7.

L'explication est la suivante. Considérons par exemple le flux f_2. Comme l'illustre la formule ci-dessus pour le délai de f_2, RC1 considère qu'un paquet de f_2 est doublé par un paquet de f_3 et un paquet de f_4. RC1 suppose ensuite que chacun de ces 3 paquets est bloqué par un paquet de f_1 et un paquet du flux causant le plus de délai parmi f_5, f_6 et f_7. Or, suivant la période de ce dernier flux, il n'est pas forcément possible que trois paquets soient émis successivement et bloquent les paquets de f_2, f_3 et f_4. La méthode NC quant à elle compte un paquet de f_5, f_6 et f_7 ce qui permet de réduire la borne obtenue. La différence peut être conséquente si un terminal émet des paquets de tailles très variables.

En revanche, dans le scénario 2, les périodes des flux sont plus faibles ce qui amène NC à compter plusieurs paquets de flux en conflit pour chaque paquet du flux étudié. Le calcul réseau ne présente donc un avantage qu'en présence de flux de période importante.

6.2.3 Impact du calcul de point fixe

Ce troisième exemple reprend le réseau de la Figure 5.10. La topologie du réseau est reproduite sur la Figure 6.6. Quatre flux f_1 à f_4 se partagent le réseau. Nous considérerons trois scénarios dont les paramètres sont donnés dans le Tableau 6.18.

La méthode RC1 donne la même borne pour les 4 flux :

$$\forall i, d_i = \frac{L_1 + L_3}{C} + \frac{L_2 + L_4}{C_4}.$$

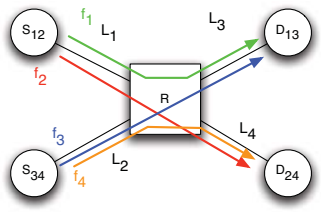


FIGURE 6.6 – Exemple de réseau posant un problème de récursion infinie

Flux	Scénario 1		Scénario 2		Scénario 3		Scénario 4	
C_4 (Mbps)	50		6,25		0,2		0,2	
	Taille	Période	Taille	Période	Taille	Période	Taille	Période
f_1	4000	20	4000	2	4000	20	4000	2
f_2	500	8	500	8	20	32	20	4
f_3	5000	20	5000	2	5000	20	5000	2
f_4	400	8	400	8	20	32	20	4,5

TABLE 6.18 – Paramètres pour les différents scénarios sur le troisième réseau

Les courbes de service de bout en bout calculées par la méthode NC sont données dans la Section 5.3.3. Pour rappel, pour f_1 cette courbe est

$$\beta_1^{ETE} = [\beta - \alpha_2]_\uparrow \otimes \delta_{d_2} \otimes \left[\beta - \alpha_3^R\right]_\uparrow .$$

Les autres flux ont des courbes similaires.

		Scénario 1	Scénario 2	Scénario 3	Scénario 4
RC1		1,98	3,24	_	
RC2	f_1	1,99	3,3	10	
	f_2	1,99	3,4	16,2	
	f_3	1,99	3,3	10	
	f_4	1,99	3,4	16,2	
NC	f_1	2,08	3,34	3,80	47,0
	f_2	2,26	4,04	4,6	50,8
	f_3	2,06	3,32	3,80	39,2
	f_4	2,06	3,32	3,80	33,7

TABLE 6.19 – Résultats sur le troisième réseau (ms)

Dans le premier scénario, RC1 représente le pire cas atteignable et fournit donc une borne optimale. RC2 fournit un résultat équivalent. Le calcul réseau donne un résultat légèrement plus pessimiste. Ceci est dû au fait que sur le deuxième lien traversé par un flux (l_3 pour f_1 et f_3, l_4 pour f_2 et f_4), la courbe utilisée est supérieure à la courbe d'arrivée du flux à la source. Dans la courbe de service de f_1

ci-dessus, on retranche α_3^R (courbe d'arrivée de f_3 en sortie de R) qui est supérieure à α_3 (courbe d'arrivée de f_3 à sa source) du fait de l'interférence avec f_4 sur l_2. Le calcul réseau considère donc qu'un peu plus qu'un paquet de f_3 double un paquet de f_1 sur l_3. Ceci est lié au fait que nous utilisons un modèle fluide qui est un peu plus pessimiste qu'un modèle par paquet. La situation est bien sûr similaire sur les autres flux.

Dans le deuxième scénario, on diminue la capacité du lien l_4. On a maintenant $C_4 = 6,25$ Mbps. La principale observation ici est que rien ne change par rapport au premier scénario. Les bornes calculées par les différentes méthodes sont bien sur plus élevées que dans le premier scénario mais le rapport entre elles reste le même : RC2 est légèrement plus pessimiste que RC1 (qui est là aussi optimale) et NC est plus pessimiste que ces deux méthodes mais dans le même ordre de grandeur.

Dans les deux scénarios suivants, RC1 n'est pas applicable car f_2 et f_4 émettent des paquets de taille trop faible. De plus, $C_4 = 0,2$ Mbps et f_2 et f_4 émettent des paquets de 20 octets.

Dans le scénario 3, le trafic est faible. Dans ce cas, NC donne une meilleure borne que RC2 car RC2 suppose la présence de plusieurs paquets de f_2 et f_4 dans les buffers réseaux. NC ne compte que le nombre réellement présent de paquets.

En revanche, dans le scénario 4, avec une charge de trafic élevée, NC devient très pessimiste. La borne pour f_2 est ainsi de $50,8$ ms alors que la borne fournie par RC2 est de $16,2$ ms. Cet exemple illustre le fait que dans un réseau où les flux se croisent et qui nécessitent l'utilisation du calcul itératif décrit dans 5.3.4, les bornes fournies par le calcul réseau peuvent très rapidement augmenter avec la quantité de trafic.

6.2.4 Impact des paquets de faible taille, lus par un terminal de faible taux de service

Dans ce quatrième exemple, nous comparerons uniquement les méthodes RC2 et NC sur le réseau déjà étudié en Section 4.2.3.2. Ce réseau est reproduit sur la Figure 6.7. Les paramètres des 3 scénarios considérés sont donnés dans le Tableau 6.20. Les résultats sont présentés dans le Tableau 6.21. Dans cet exemple, tous les liens ont une capacité $C = 200$ Mbps sauf le lien l_8 dont la capacité varie selon les scénarios.

On observe dans le Tableau 6.21 que sur cette configuration, NC fournit de bien meilleures bornes que RC2. Nous avons déjà expliqué en Section 4.2.3.3 que RC2 compte un nombre très importants et irréalistes de paquets du flux f_2 sur ce type de configuration. Ceci est visible dans le fait que la borne calculée pour f_2 à l'aide de RC2 est beaucoup plus élevée que celle calculée par NC. De même, la borne de f_1 est élevé car il croise f_2 dès le premier routeur ce qui oblige la méthode RC2 à supposer la présence de nombreux paquets de f_2 dans les routeurs suivants du chemin de f_2.

Le scénario 2 (dans lequel $C_8 = 0,2$ Mbps au lieu de 2 Mbps dans le scénario 1) permet de constater que les bornes calculées par RC2 et NC dépendent directement de la capacité C_8. Lorsque C_8 est divisée par 10, les bornes des 4 flux sont multipliées

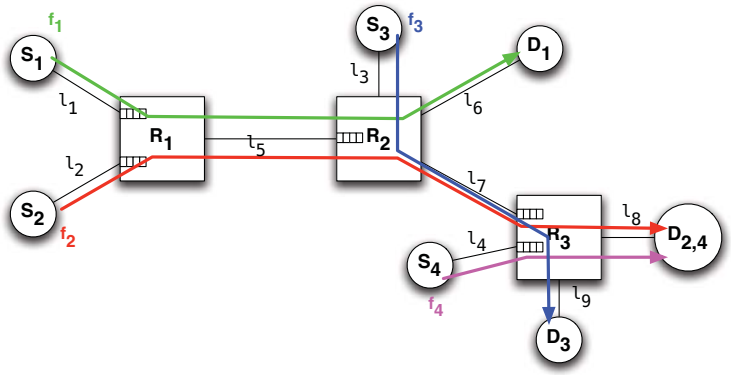


FIGURE 6.7 – Quatrième réseau étudié

	Scénario 1	Scénario 2
C_8 (Mbps)	2	0,2

	Taille (octets)	Période ms
f_1	4000	2
f_2	20	2,5
f_3	1000	0,3
f_4	20	2

TABLE 6.20 – Paramètres pour les deux scénarios sur le quatrième réseau

par 10 ou plus. Dans ce scénario, l'avantage de la méthode NC sur la méthode RC2 est donc d'autant plus marqué.

Cet exemple illustre bien le fait qu'il est d'autant plus important de ne pas compter un nombre trop élevé de paquets lorsque ceux-ci utilisent un lien beaucoup plus lent que le reste du réseau.

	Scénario 1		Scénario 2	
	RC2	NC	RC2	NC
f_1	3,6	0,45	33,2	4,25
f_2	17,9	0,25	164	4,05
f_3	0,87	0,3	8,25	4,1
f_4	0,83	0,25	8,25	4,05

TABLE 6.21 – Résultats pour le quatrième réseau (ms)

Chapitre 7

Conclusion et perspectives

7.1 Conclusion

L'objectif de cette thèse était de développer des outils analytiques destinés à l'étude des délais de livraison pire-cas dans un réseau SpaceWire. Ces outils permettent aux concepteurs réseau de vérifier qu'une architecture réseau est capable d'assurer la livraison d'un ensemble de flux critiques en respectant leurs contraintes temporelles.

Nous avons choisi d'utiliser comme métrique une borne supérieure du délai de livraison pire-cas d'un paquet de données. Cette métrique fournit une garantie certaine du respect des contraintes temporelles tout en restant analytiquement calculable.

Plusieurs méthodes analytiques existaient déjà pour d'autres types de réseaux wormhole mais aucunes ne s'appliquaient au SpaceWire. Ainsi plusieurs méthodes permettent d'étudier les réseaux internes des ordinateurs massivement parallèles d'un point de vue statistique. D'autres études sont consacrées à des réseaux wormhole dotés de mécanismes temps-réel dédiés que SpaceWire ne fournit pas. Elles ne sont donc pas applicables dans notre contexte. Enfin, plusieurs méthodes analytiques ont été mises au point pour étudier des réseaux wormhole de type best-effort assez similaires au SpaceWire, comme les Networks-on-Chip. Certaines de ces méthodes fournissent un point de départ intéressant pour l'analyse d'un réseau SpaceWire mais aucune ne peut s'appliquer directement au SpaceWire car elles ont été conçues pour des cas d'utilisation différents.

Nous avons donc choisi de concevoir notre propre méthode analytique de calcul des délais pire-cas. Nous avons utilisé deux approches différentes lors de notre recherche d'un tel outil. La première approche ne nécessite pas d'hypothèses fortes sur le trafic injecté dans le réseau. En particulier, il n'est pas nécessaire que les flux soient périodiques pour que la méthode soit applicable. Cette approche a donné lieu à la mise au point de deux méthodes complémentaires de calcul des délais pire-cas.

La première méthode que nous avons proposée calcule le délai de transmission d'un paquet en supposant que les paquets successifs d'un même flux n'ont pas d'influence les uns sur les autres. La seconde méthode suppose quant à elle que le réseau est saturé lorsque le paquet étudié est émis et que tous les terminaux continuent à émettre à leur vitesse maximale. Le réseau peut éventuellement contenir d'autres paquets appartenant au même flux que le paquet étudié. Elle fournit ainsi par construction une borne supérieure absolue sur le délai pire-cas. Nous avons présenté quelques exemples d'application de chaque méthode qui illustrent leur utilité

dans une analyse pire-cas. Nous avons aussi mis en valeur les limites de chacune de ces méthodes et les architectures réseau qu'elle ne permettent pas de traiter.

Au contraire, la deuxième approche est une analyse des délais pire-cas basée sur des hypothèses plus fortes sur le trafic en entrée du réseau. Cette analyse utilise la théorie du Calcul Réseau pour modéliser un réseau SpaceWire. Le Calcul Réseau permet de modéliser précisément le trafic injecté dans le réseau grâce au concept de courbes d'arrivée. Réciproquement, il nécessite de modéliser les routeurs par des courbes de service qui décrivent la quantité minimale de données que transmet un élément réseau pendant une certaine durée.

Pour déterminer ces courbes de service, nous avons introduit un nouvel élément de Calcul Réseau, la section wormhole. Celle-ci permet de modéliser les conflits entre les flux de façon plus précise que les éléments classiques comme les multiplexeurs et les démultiplexeurs. En particulier, nous avons proposé un modèle du démultiplexage de deux flux en sortie d'une section wormhole beaucoup moins pessimiste que les modèles existants. Enfin, nous avons décrit une méthode d'analyse d'un réseau complet qui utilise le concept de schéma d'interférence introduit dans [22]. Notre analyse emploie également la méthode du point fixe pour résoudre les problèmes de dépendances circulaires entre les courbes d'arrivée des différents flux en conflit. Notre système d'équations converge rapidement sur tous les réseaux présentés dans cette thèse. Il nous reste cependant à démontrer formellement sa convergence dans un cas général.

Le chapitre final présente une comparaison des trois méthodes de calcul introduites dans les chapitres 4 et 5. Dans un premier temps, nous avons analysé une architecture de référence fournie par Thales Alenia Space. Nous avons montré que les deux premières méthodes ne suffisent pas à analyser complètement cette architecture. La première méthode n'est pas applicable à cette architecture car elle requiert une hypothèse sur la taille minimale des paquets qui n'est pas respectée dans ce cas d'étude. La seconde méthode est applicable à l'architecture mais aboutit à des bornes trop élevées pour être utiles en pratique. En revanche, la méthode basée sur le Calcul Réseau fonctionne sur cette architecture et fournit des bornes exploitables. Cet exemple permet de montrer que notre méthode est capable d'analyser un réseau complexe et hétérogène.

Dans un second temps, nous avons comparé les trois méthodes de calcul sur différents réseaux plus simples. Ces exemples illustrent le fait que chaque méthode est plus adaptée à certaines situations que les autres. Ainsi, suivant la configuration du réseau, il peut être plus important soit de modéliser précisément le mécanisme Round-Robin des routeurs comme le font les deux premières méthodes, soit de tenir compte des débits réels des flux comme le fait la troisième méthode à l'aide des courbes d'arrivée.

Au final, le travail réalisé au cours de cette thèse a permis de mettre au point plusieurs outils d'analyse des délais pire-cas pour un réseau SpaceWire. Ces outils permettent l'étude complète du cas d'étude Thales Alenia Space qui a servi de support à cette thèse. L'étude réalisée durant cette thèse ouvre également plusieurs perspectives de recherche que nous allons maintenant détailler.

7.2 Perspectives de recherche

Trois perspectives de recherche s'offrent à nous pour poursuivre le travail engagé dans cette thèse.

La première perspective est d'élargir le champ d'application des méthodes de calcul proposées. En particulier, il serait utile de disposer d'un modèle du Group Adaptative Routing pour les méthodes RC2 et NC. L'outil analytique permettrait alors de vérifier si le Group Adaptative Routing permet de supprimer les goulots d'étranglement ou s'il n'a pas d'utilité pratique en ce qui concerne les délais pire-cas. Il serait également intéressant de disposer d'un modèle de l'extension déterministe SpaceWire-D lorsqu'elle sera finalisée. Nous pourrons ainsi en mesurer les avantages et les inconvénients avec précision.

Le second développement futur serait d'intégrer les trois méthodes de calcul en un unique outil qui déterminerait automatiquement quelles méthodes sont applicables parmi les trois. L'outil calculerait ensuite la borne pire-cas par chaque méthode applicable et renverrait la meilleure. De plus, cet outil pourrait être intégré dans un outil de design interactif d'architectures réseau. Le but serait d'étudier une architecture, d'identifier les contraintes temporelles non respectées et les goulots d'étranglement. L'architecture pourrait ainsi être améliorée progressivement par le concepteur réseau. Une application intéressante de cet outil serait de déterminer quels types de topologie de réseau wormhole sont les plus appropriés pour les réseaux bord satellites.

La troisième perspective est d'améliorer les méthodes de calcul proposées pour en réduire le pessimisme. La principale évolution serait de modéliser de façon plus réaliste le partage non-préemptif d'une section wormhole. Nous utilisons actuellement un modèle générique qui ne prend pas en compte le fonctionnement spécifique des routeurs SpaceWire. Il s'agit d'une évolution importante qui nécessiterait notamment d'utiliser un modèle par paquets plutôt qu'un modèle fluide comme nous le faisons actuellement.

Une autre piste serait d'utiliser des résultats provenant de la théorie de l'ordonnancement. Nous avons vu au chapitre 3 que certaines méthodes d'étude des réseaux wormhole comme [18] s'inspiraient de cette théorie. A l'heure actuelle, aucun modèle inspiré de la théorie de l'ordonnancement n'a été proposé pour un réseau wormhole best-effort comme SpaceWire. Cependant, il nous semble envisageable de rechercher un modèle de réseau wormhole qui se baserait sur la méthodes des trajectoires [31]. Cette approche a notamment été utilisé avec succès dans [32] pour réaliser l'analyse pire-cas d'un réseau AFDX. Ce développement nous semble complexe dans la mesure où l'approche trajectorielle n'a été employée que pour modéliser des réseaux de type *Store-and-Forward.* Toutefois, un couplage avec l'une des méthodes récursives proposées dans cette thèse pourrait éventuellement simplifier la modélisation.

Annexe A

Algorithme de calcul des courbes de service de bout en bout

A.1 Algorithme sans agrégation

Le principe de l'algorithme est le suivant. On suppose initialement que, pour chaque flux, la courbe d'arrivée du flux en n'importe quel point de son chemin est identique à la courbe d'arrivée à la source.

On calcule ensuite les courbes de service résiduelles offertes à chaque flux en fonction des courbes d'arrivées des autres flux. On en déduit de nouvelles courbes d'arrivée (supérieures ou égales à l'approximation précédente) pour chacun des flux aux différents points de leur chemin.

On utilise ensuite ces nouvelles courbes d'arrivée pour calculer de nouvelles courbes de service résiduelles et ainsi de suite jusqu'à ce que plus aucune courbe d'arrivée ne soit modifiée lors d'une itération.

On obtient ainsi une solution du système et on en déduit la courbe de service de bout en bout pour chacun des flux.

A.2 Algorithme d'agrégation

L'agrégation des flux est implémenté à travers la procédure detectGroupableFlows et les fonctions createAggFlows et new AggregatedFlow. Avant de lancer le calcul itératif, la procédure detectGroupableFlows est appelée sur chaque flux.

Algorithm 1: Itération du calcul

Input :
- $F = \{f_1, \ldots, f_n\}$, the set of Flows
- $L = \{l^1, \ldots, l^m\}$, the set of Links
- each Link l^j has a service curve $l^j.SC$
- each Flow f_i has an arrival curve $f_i.AC$ and a path $(l_i^1, \ldots, l_i^{m_i})$
- each Flow f_i has two tables oldAC and newAC storing the arrival curves at each Link in $(l_i^1, \ldots, l_i^{m_i})$ (i.e before traversing the Link). oldAC stores the arrival curves computed during the previous iteration and newAC the arrival curves already computed during the current iteration.
- each Flow f_i has two tables oldSC and newSC storing the service curves offered to f_i from each Link in $(l_i^1, \ldots, l_i^{m_i})$. oldSC stores the service curves computed during the previous iteration and newSC the service curves already computed during the current iteration.
- Precision

Output:
- for all i, $f_i.eteSC$ the end to end service curve offered to f_i
- for all i, $h(f_i.AC, f_i.eteSC)$, an upper-bound on the worst-case end-to-end delay of f_i

Initial values

`/* Initially, we assume that the arrival curve of each Flow is the same at every point of the path */`

For all i, j, f_i.oldAC[j] $\leftarrow f_i.AC$ and f_i.oldSC[j] $\leftarrow 0$

For all i, f_i.oldETE_SC $\leftarrow 0$

Optional step : Flows aggregation

`/* Calling detectGroupableFlows is an optional step that optimizes the end-to-end service curve. Aggregation must be done before starting the iteration. */`

For all Flow f_i, f_i.`detectGroupableFlows`

Iteration

`/* At each step, compute the residual service curves offered to each Flow and the resulting arrival curves. These new arrival curves will be used to compute the residual service curves at the next iteration. */`

for *all Flow* f_i **do**
  for *all Link* l_i^j **do**
    f_i.newAC[j] $\leftarrow f_i$.AC $\oslash$ `computeSC`(f_i, l_i^1, l_i^{j-1})
    f_i.newSC[j] $\leftarrow$ `computeSC`$(f_i, l_i^j, l_i^{m_i})$
    f_i.newETE_SC $\leftarrow$ `computeSC`$(f_i, l_i^1, l_i^{lm_i})$
  end for
end for
if *end condition not verified* **then**
  f_i.oldAC[j] $\leftarrow f_i$.newAC[j]
  f_i.oldSC[j] $\leftarrow f_i$.newSC[j]
  f_i.oldETE_SC $\leftarrow f_i$.newETE_SC
end if

End condition

`/* We stop the iterative computation once all the arrival and service curves are constant. */`

For all i, j, $v(f_i$.oldAC[j], f_i.newAC[j]$) \leq$ Precision
AND $v(f_i$.oldSC[j], f_i.newSC[j]$) \leq$ Precision
AND $v(f_i$.oldETE_SC, f_i.newETE_SC$) \leq$ Precision

`/*` $v(f, g)$ `is the vertical distance between` f `and` g`, where` f `and` g `are positive, non-decreasing functions */`

Algorithm 2: class Flow

The class Flow contains the following important attributes.

- `itfFlows` list of all the Flows hat interfere with the Flow
- `links` list of Links used by the Flow
- `matrix` Describes the conflicts with the interfering flows. There is 1 row for each interfering Flow and 1 column for each Link used by the Flow itself. `matrix`[i,j]==true if and only if `itfFlows`[i] interferes with the Flow on Link `links`[j].

Function computeSC(f_i, l^{in}, l^{out}, exf)

Input :
- f_i Flow for which we compute the service curve
- l^{in} first Link in the section
- l^{out} last Link in the section
- exf lowest priority flow, excluded from the computation because it does not interfere at all with f_i

Output: the service curve offered to f_i between Links l^{in} and l^{out} excluding Flow exf

```
begin
    // use local copies of f_i attributes

    mat ← f_i.matrix[ :, l^in :l^out]
    auxlinks ← f_i.links[ :, l^in :l^out]
    iflows ← f_i.itfFlows
    offsets ← [0, ..., 0]
    scurves ← [l^in.SC, ..., l^out.SC]
    delay ← 0
    /* computeSC calls the three auxiliary functions until there is
       only one Link left in auxlinks. Then the service curve of this
       Link is the service curve offered by the whole section        */
    goCut ← true
    while goCut do
        go ← true
        while go do
            // Look for consecutive Links with the same interfering
               flows and merge them
            go1 ← f_i.mergeLinks
            /* Look for Flows interfering with f_i on only one Link,
               subtract their arrival curve from the service curve of
               the Link and remove them                              */
            go2 ← f_i.handleFlows (exf)
            go ← go1 OR go2
        end while
        // If the first two functions are not enough to solve the
           conflicts, cut one Flow into two parts to simplify the
           resolution
        goCut ← f_i.cutFlow
    end while
    return (scurves [1], delay)
end
```

Function mergeLinks

Input :
- f_i

Output:
changed : true if at least one merge happened during execution, false otherwise

Result: merge all the consecutive Links used by exactly the same Flows into one

```
begin
    changed ← false
    over ← false
    while NOT over do
        over ← true
        for j=1 to length( auxlinks) -1 do
            if mat [ :,j]==mat [ :,j+1] then
                // two successive columns are identical
                // remove the second one, then start the loop again
                delete (mat [ :,j+1])
                delete (auxlinks [j+1])
                scurves [j] ← scurves [j] ⊗ scurves [j+1]
                delete (scurves [j+1])
                offsets [j] ← offsets [j]+offsets [j+1]+1
                delete (offsets [j+1])
                over ← false
                changed ← true
                break
            end if
        end for
    end while
end
```

Function handleFlows(exf)

Input :
- f_i
- exf lowest priority flow, excluded from the computation because it does not interfere at all with f_i
- nnz function that counts the number of non-zero elements in a row

Output: changed true if at least one merge happened during execution, false otherwise

Result: remove the Flows that interfere with f_i on only one Link and subtract their arrival curve from the service curve of that Link

```
begin
    changed ← false
    over ← false
    while NOT over do
        over ← true
        for i=1 to length(iflows) do
            if nnz (mat [i, :])==0 OR iflows [i]==exf then
                // iflows [i] does not interfere: it can be removed safely
                delete (mat [i, :])
                delete (iflows [i])
                over ← false
                break
            else if nnz (mat [i, :])==1 then
                j ← {index of mat such that mat [i,j]==1}
                scurves [j] ← (scurves[j] − iflows[i].oldAC)↑
                // if iflows [i] is demultiplexed between l^in and l^out,
                // add its demultiplexing delay to delay
                if (auxlinks [j+offsets [j]]==f_i.endItf(iflows [i])
                    AND f_i.lastLink != iflows [i].lastLink) then
                    delay ← delay +
                    h(iflows[i].oldAC[j+offsets [j]], iflows[i].oldSC[j+offsets [j]])
                end if
                delete (mat [i, :])
                delete (iflows [i])
                over ← false
                changed ← true
                break
            end if
        end for
    end while
end
```

Function cutFlow

Input : f_i

Output: changed true if at least one merge happened during execution, false otherwise

Result: The first Flow of iflows is cut into two subFlows. The first one uses only the first Link of iflows [1]. The second one uses all the Links of iflows [1], except the first one.

```
begin
    if length(iflows) ≤ 1 OR length(auxlinks) ≤ 2 then
        changed ← false
        return
    end if
    changed ← true
    // create two subFlows of iflows [1]
    j ← {first index such that iflows [1,j] ≠ 0 }
    line1 ← [0, . . . , 0]
    line1[j] ← 2
    line2 ← mat [1, :]
    line2[j] ← 0
    // add the two subFlows to the interfering flows of f_i
    addLastLine (mat, line1)
    addLast (iflows, iflows [1])
    addLastLine (mat, line2)
    addLast (iflows, iflows [1])
    delete (mat [1, :])
    delete (iflows [1])
end
```

Procedure detectGroupableFlows

Input : f_i

Result: The Flows that interfere with f_i and have the same destination are grouped in one AggregatedFlow. This allows us to count the interferences between those Flows only once.

```
destFlows ← new Map()
begin
    // sort the flows by destination
    for f ∈ f_i.itfFlows do
        destFlows.add(f.lastLink().name, f)
    end for
    // sort each list by first Link of interference with f_i
    for dflows ∈ destFlows.values() do
        // dflows is a list of Flows with the same destination
        auxmap ← new Map()
        for fd ∈ dflows do
            auxmap.add(f_i.startItf(fd).name, fd)
        end for
        for (cflows ∈ auxmap.values(), ckey ∈ auxmap.keys()) do
            if length(cflows) ≥ 2 then
                // Aggregate the Flows in cflows
                // newAggFlows is a list of new AggregatedFlows
                // the first element of newAggFlows aggregates all
                   the others
                // and is the one that interferes with f_i instead of
                   the Flows in cflows
                newAggFlows = createAggregateFlows (cflows, ckey, f_i)
                f_i.addItfFlow(newAggFlows[1])
                f_i.removeItfFlows(cflows)
            end if
        end for
    end for
end
```

Function createAggFlows

Input :
- flows the set of Flows to aggregate
- cli Link such that, starting from cli, all the Flows follow the same path
- mflow master Flow. The Flows are all interfering with mflow and are aggregated relatively to this Flow

Output: newAggFlows a list of new AggregatedFlow. newAggFlows [1] aggregates all the other subAggregatedFlows.

```
begin
    firstCommonLink ← (first Link common to all the Flows in flows )
        // firstCommonLink is always upstream of cli
    // sort the Flows by Link before firstCommonLink
    sortedFlows ← new Map()
    for f ∈ flows do
        sortedFlows.add(f.prevLink(firstCommonLink).name, f)
    end for
    flowsToAgg ← Nil
    newAggFlows ← Nil
    for kflows ∈ sortedFlows.values() do
        if length(kflows)==1 then
            flowsToAgg.addFirst (kflows[1])
        else
            newAggFlows.addFirst (createAggregateFlows (kflows,
            firstCommonLink, mflow))
        end if
    end for
    newAggFlows.addFirst (new AggregatedFlow(flowsToAgg,
    newAggFlows, firstCommonLink, mflow))
end
```

Function new AggregatedFlow

Input :
- flows list of Flows to aggregate in this new AggregatedFlow
- agflows a list of AggregatedFlows to aggregate in this new AggregatedFlow
- firstCommonLink first Link common to all the Flows and AggregatedFlows
- mflow master Flow. The Flows are all interfering with mflow and are aggregated relatively to this Flow

Output: newAggFlow a new AggregatedFlow, aggregating all the Flows in flows and the AggregatedFlows in agflows

begin

newAggFlow.name = (*concatenation of* mflow.*name and the names of all the Flows and AggregatedFlows*)

if *length(*flows*)* ≥ 1 **then**

newAggFlow.links $\leftarrow$ flows [1].links[firstCommonLink :lastLink]

else if *length(*agflows*)* ≥ 1 **then**

newAggFlow.links $\leftarrow$ agflows [1].links[firstCommonLink :lastLink]

else

ERROR

end if

newAggFlow.flows $\leftarrow$ flows

newAggFlow.agflows $\leftarrow$ agflows

newAggFlow.mflow $\leftarrow$ mflow

```
/* newAggFlow.matrix includes all the Flows interfering on any
   Link of newAggFlow.links except those in flows or already
   aggregated in one AggregatedFlows of agflows                */
```

newAggFlow.matrix $\leftarrow$ *create* newAggFlow.*matrix*

return newAggFlow

end

Annexe B

Résultats détaillés pour le cas d'étude Thales Alenia Space

Cet annexe présente les résultats détaillés fournis par méthodes de calcul RC2 et NC sur le cas d'étude TAS étudié en section 6.1.

B.1 Résultas fournis par la méthode RC2

Flux	Scénario 1	Scénario 2
SC0 à SC3	6 699	6 674
SC4 à SC8	8 632	8 599
HK0 à HK3	11 122	10 835
HK4 à HK8	14 214	13 915
inclut HK (re)		
CMD0 à CMD9	111,3	6,75
TM X	175	175
TM Ka	175	175

L'élément le plus notable sur les résultats de la méthode RC2 est que plusieurs groupes de flux ont exactement la même borne pire-cas. Pour les flux SC et HK, les groupes (SC_0 à SC_3, SC_4 à SC_8, HK_0 à HK_3 et HK_4 à HK_8) correspondent à la division entre les terminaux applicatifs reliés au routeur R_0 et ceux reliés aux routeurs R_1.

Dans la mesure où le lien $R_1 \rightarrow R_2$ est partagé par 10 flux alors que le lien $R_0 \rightarrow R_2$ n'est partagé que par 8 flux, il n'est pas étonnant que la borne des flux passant par R_1 soit plus élevée que celle des flux passant par R_0. Rappelons cependant que ceci n'est valable que dans la mesure où la méthode RC2 ne prend pas en compte les périodes des différents flux.

Enfin, les flux CMD ont tous le même délai car ils sont sérialisés sur le lien $PM \rightarrow R_3$ avant de suivre des trajets similaires jusqu'à leurs destinations respectives.

B.2 Résultats fournis par la méthode NC

Flux	Scénario 1	Scénario 2
SC0	89,6	72,0
SC1	148,3	118,8
SC2	156,1	124,9
SC3	157,7	126,1
SC4	173,4	138,4
SC5	208,4	166,2
SC6	211,1	168,3
SC7	211,9	169,0
SC8	211,9	169,0
HK0	392,6	357,5
HK1	271,9	247,7
HK2	392,4	357,5
HK3	500,0	455,1
HK4	466,2	423,1
HK5	514,9	467,3
HK6	433,8	393,7
HK7	529,4	480,5
HK8	530,9	481,8
HK (re)	104,1	80,3
CMD0	91,5	70,4
TM X	161,0	150,6
TM Ka	161,0	150,6

A l'inverse de RC2, la méthode NC prend en compte les périodes des flux et donne donc une borne différente pour chacun des flux SC et HK. Il est intéressant de noter que les bornes varient assez faiblement (au maximum, de $89,6$ ms à 211 ms pour les flux SC) alors que les périodes des flux varient beaucoup plus (d'un facteur 10000 pour les flux SC).

Bibliographie

[1] Steve Parkes and Philippe Armbruster. Spacewire : A spacecraft onboard network for real-time communications. *IEEE-NPSS Real Time Conference*, (14), Feb 2005. 6

[2] ECSS. ECSS-E-ST-50-12C : SpaceWire – Links, nodes, routers and networks. Aug 2008. 6, 12

[3] IEEE Computer Society. IEEE standard 1355-1995. *IEEE Standard for Heterogeneous Interconnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)*, 1996. 6

[4] ECSS. ECSS-E-ST-50-52 : Remote Memory Access Protocol (RMAP). 10

[5] Doug Roberts and Steve Parkes. Spacewire missions and applications. *2010 SpaceWire International Conference*, 2010. 13

[6] William J. Dally. Virtual-channel flow control. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2), Mar 1992. 13

[7] Lionel Ni and Philip Mckinley. A survey of wormhole routing techniques in direct networks. *Computer*, Jan 1993. 26

[8] P. Kermani and L Kleinrock. Virtual cut-through : A new computer communication switching technique. *Computer Networks*, 3(4), Jan 1979. 26

[9] William J. Dally and Charles L. Seitz. The torus routing chip. *Distributed computing*, Jan 1986. 27

[10] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *Computers, IEEE Transactions on*, 39(6), Jun 1990. 29

[11] Wei Jing Guan, Wei K. Tsai, and Douglas Blough. An analytical model for wormhole routing in multicomputer interconnection networks. *Proceedings of the Seventh International Parallel Processing Symposium*, 1993. analyse statistique des délais de transmission. 29

[12] Jeffrey T. Draper and Joydeep Ghosh. A comprehensive analytical model for wormhole routing in multicomputer systems. *Journal of Parallel and Distributed Computing*, Jan 1994. 29

[13] Jong-Pyng Li and Matt W.Mutka. Real-time virtual channel flow control. *Computers and Communications, IEEE 13th Annual International Conference on*, 1994. 29

[14] S.L. Hary and F. Özgüner. Feasibility test for real-time communication using wormhole routing. *Computers and Digital Techniques, IEE Proceedings*, 144(5), 1997. 30, 31, 32

[15] Shobana Balakrishnan and Füsun Özgüner. A priority-driven flow control mechanism for real-time traffic in multiprocessor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 9(7), 1998. 31, 32

[16] Byungjae Kim, Jong Kim, Sungje Hong, and Sunggu Lee. A real-time communication method for wormhole switching networks. *Parallel Processing, Proceedings of the 1998 International Conference on*, 1998. 31

[17] Zhongai Lu, Axel Jantsch, and Ingo Sander. Feasibility analysis of messages for on-chip networks using wormhole routing. *Asia and South Pacific Design Automation Conference. Proceedings of the*, 2, 2005. 32

[18] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. *NoCS 2008, Second ACM/IEEE International Symposium on Networks-on-Chip*, Apr 2008. 32, 115

[19] Zheng Shi and Alan Burns. Real-time communication analysis with a priority share policy in on-chip networks. *ECRTS '09, 21st Euromicro Conference on Real-Time Systems*, Jul 2009. 32

[20] Sunggu Lee. Real-time wormhole channels. *Journal of Parallel and Distributed Computing*, Jan 2003. 33

[21] Dara Rahmati, Srinivasan Murali, Luca Benini, Federico Angiolini, Giovanni De Micheli, and Hamid Sarbazi-Azad. A method for calculating hard qos guarantees for networks-on-chip. *ICCAD 2009, IEEE/ACM International Conference on Computer-Aided Design*, 2009. 33, 34, 52, 57

[22] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. *International Symposium on Networks-on-Chip, Proceedings of the 2009 3rd ACM/IEEE*, 2009. 34, 77, 78, 79, 80, 94, 114

[23] Jean-Yves Le Boudec and Patrick Thiran. Network calculus : A theory of deterministic queuing systems for the internet. *Springer Verlag*, (LNCS 2050), Apr 2004. 34, 71, 76, 80

[24] William J. Dally and Charles M. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *Computers, IEEE Transactions on*, C-36(5), May 1987. 44

[25] Jean-Yves Le Boudec and Patrick Thiran. A short tutorial on Network Calculus. I. fundamental bounds in communication networks. *ISCAS 2000, Proceedings of The 2000 IEEE International Symposium on Circuits and Systems*, 4, 2000. 71

[26] Rene L. Cruz. A calculus for network delay, part i : Network elements in isolation. *Information Theory, IEEE Transactions on*, 37(1), Jan 1991. 77

[27] MATLAB. *version 7.9 (R2009b)*. The MathWorks Inc., Natick, Massachusetts, 2009. 94

[28] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox, 2006. 94

[29] Philippe Fourtier, Alain Girard, Antoine Provost-Grellier, and François Sauvage. Simulation of a spacewire network. *International SpaceWire Conference 2010, Proceedings of the*, Oct 2010. :2010p1685

[30] Inc. OPNET Technologies. Opnet network simulator. 99

[31] Steven Martin and Pascale Minet. Schedulability analysis of flows scheduled with fifo : application to the expedited forwarding class. *IPDPS 2006, 20th International Parallel and Distributed Processing Symposium*, Apr 2006. 115

[32] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Improving the worst-case delay analysis of an afdx network using an optimized trajectory approach. *Industrial Informatics, IEEE Transactions on*, 6(4) :521 –533, nov. 2010. 115

Maîtrise des latences de communication dans les réseaux bord SpaceWire

SpaceWire est un standard de réseau embarqué promu par l'Agence Spatiale Européenne qui envisage de l'utiliser comme réseau bord unique dans ses futures satellites. SpaceWire utilise un mécanisme de routage Wormhole pour réduire la consommation mémoire des routeurs et les coûts associés. Cependant, le routage Wormhole peut engendrer des blocages en cascade dans les routeurs et, par conséquent, d'importantes variations des délais de livraison des paquets.Comme le réseau doit être partagé par des flux critiques et non-critiques, les concepteurs réseau ont besoin d'un outil leur permettant de vérifier le respect des contraintes temporelles des messages critiques.

Pour réaliser cet outil, nous avons choisi comme métrique une borne supérieure sur le délai pire-cas de bout en bout d'un paquet traversant un réseau SpaceWire. Au cours de la thèse, nous avons proposé trois méthodes permettant de calculer cette borne. Les trois méthodes utilisent des hypothèses différentes et ont chacune des avantages et des inconvénients. D'une part, les deux premières méthodes sont très générales et ne nécessitent pas d'hypothèses restrictives sur le trafic en entrée du réseau. D'autre part, la troisième méthode nécessite des hypothèses plus précises sur le trafic en entrée. Elle est donc moins générale mais donne la plupart du temps des bornes plus serrées que les deux autres méthodes.

Dans cette thèse, nous avons appliqué ces différentes méthodes à une architecture de référence fournie par Thales Alenia Space afin d'en comparer les résultats. Nous avons également appliqué ces méthodes à des exemples plus simples afin de déterminer l'influence de différents paramètres sur les bornes fournies par nos méthodes.

Mots-clés : réseaux embarqués, temps-réel, SpaceWire, wormhole, évaluation de performance, délais pire-cas

Controlling communication latencies in on-board SpaceWire networks

The SpaceWire network standard is promoted by the ESA and is scheduled to be used as the sole on-board network for future satellites. SpaceWire uses a wormhole routing mechanism to reduce memory consumption and the associated costs. However, wormhole routing can lead to packet blocking in routers which creates large variations in end-to-end delays. As the network will be shared by real-time and non real-time traffic, network designers require a tool to check that temporal constraints are verified for all the critical messages. The metric we chose for this tool is an upper-bound on the worst-case end-to-end delay of a packet traversing a SpaceWire network. This metric is simpler to compute than the exact delay of each packet and provide enough guarantee to the network designers.

During the thesis, we designed three methods to compute this upper-bound. The three methods use different assumptions and have different advantages and drawbacks. On the one hand, the first two methods are very general and do not require strong assumptions on the input traffic. On the other hand, the third method requires more specific assumptions on the input traffic. Thus, it is less general but usually gives tighter bounds than the two other methods.

In the thesis, we apply those methods to a case study provided by Thales Alenia Space and compare the results. We also compare the three methods on several smaller networks to study the impact of various parameters on their results.

Keywords: embedded networks, real-time, SpaceWire, wormhole, performance evaluation, worst-case delay