Sommaire

Liste des figures

Liste des Tableaux

Introduction générale

1-Motivations

Les prévisions stratégiques d'ITRS (***I**nternational **T**echnology **R**oadmap for **S**emiconductors*) annoncent que 70% des systèmes mono-puce (*SoC* pour "*System On Chip*" en anglais) comporteront au moins un processeur embarqué à partir de l'année 2005 [1]. Cette tendance semble non seulement se confirmer mais se renforcer : les SOCs contiendront plusieurs processeurs dans le cas d'applications telles que les terminaux mobiles, vidéos, réseaux de communication, traitement de signal, capteurs, etc. De plus, ces puces contiendront des éléments non digitaux (par ex. analogique ou RF) et des mécanismes de communication très sophistiqués. Le développement de produits électroniques de haute performance, de faible coût, destinés au grand public, et dotés de fonctions intelligentes a obligé l'industrie des semi-conducteurs à intégrer toutes les ressources sur une seule puce afin de réduire les coûts de production. Il est donc crucial de maîtriser la conception de tels systèmes tout en respectant les contraintes de mise sur le marché et les objectifs de qualité.

En plus, le modèle de synchronisation des futurs systèmes sur puce sera probablement de type « Globalement Asynchrone Localement Synchrone » (GALS) avec la possibilité d'utiliser plusieurs domaines isochrones (phases et domaines d'horloge différents)[2].

Cette augmentation de la capacité et de la complexité des systèmes mono-puce a stimulé les chercheurs pour concevoir de nouvelles plates-formes d'interconnexion fiable, à énergie réduite et à rendement élevé, baptisés réseaux sur puce (NOC pour "*Network On Chip*" en anglais**)**, afin de remédier aux problèmes de communication générés par les anciennes architectures d'interconnexion (les bus).

L'objectif essentiel de la conception des NOCs est de limiter l'espace de conception tout en respectant les contraintes de mise sur le marché et les objectifs de qualité, et d'assurer l'interfaçage entre l'espace de conception de l'application et l'implémentation[3].

Les réseaux sur puces semblent être une solution appropriée pour gérer la communication entre les ressources(Processeur, DSP, IP, ASIP, etc.…). La difficulté de la conception d'un NOC réside dans un compromis entre une Qualité de Service (QoS) optimale, une bande passante élevée, une latence faible, une flexibilité, une extensibilité d'utilisation importantes, et une possibilité de réutilisation de la conception, tout en limitant la consommation d'énergie et de surface dans la puce.

2- Contributions

Dans ce mémoire nous proposons un modèle de réseau sur puce basé sur une topologie en papillon à arbre élargi. Ce réseau interconnecte seize ressources. Une charge aléatoire uniformément répartie a été appliquée dans le réseau. Nous avons opté pour le simulateur NS-2 afin d'évaluer notre modèle. Nous avons pu implémenter toutes les composantes dans le simulateur choisi. Des études comparatives ont été réalisées entre deux techniques de routage pour le modèle proposé d'une part, et entre le réseaux BFT-NOC et Grille2D d'autre part. Plusieurs paramètres ont été explorés dans les simulations, à savoir la bande passante maximale, la technique de routage, la stratégie de mémorisation et le contrôle des performances.

3- Organisation du mémoire

Le chapitre 1 étudie les paramètres technologiques indispensables à la conception d'un réseau sur puce, et explore les réalisations faites dans cet axe de recherche, avec une analyse détaillée de la topologie, de la stratégie de mémorisation adoptée, des techniques de routage et de commutation utilisées, ainsi que du paquetage de données pour chaque réseau présenté.

Le chapitre 2 présente le simulateur NS-2 choisi pour évaluer notre modèle baptisé BFT-NOC. Nous détaillons les tâches à entreprendre afin d'implémenter et de simuler un réseau sur puce.

Les chapitres 3 et 4 développent en détail l'architecture du modèle BFT-NOC. L'implémentation de notre réseau dans le simulateur NS-2 est faite en trois phases. On commence par l'implémentation de la topologie. Dans la deuxième phase nous détaillons la gestion des communications inter-ressources, composée du générateur de trafic du réseau, ainsi que l'algorithme adopté pour le choix des ressources. La dernière phase renferme des modules implémentés en vue d'évaluer notre modèle, entre autres les stratégies de routage, le moniteur de la file d'attente, et l'analyseur de la bande passante.

Le chapitre 5 analyse les performances réelles du modèle BFT-NOC, que l'on a pu prédire grâce à une modélisation complète précise. Ces performances concernent des critères quantitatifs et qualitatifs minimaux que doit atteindre un réseau sur puce telle ques : la bande passante maximale, la latence, la fiabilité, la flexibilité, la consommation et la surface du réseau dans la puce.

La dernière partie évoque les extensions possibles de BFT-NOC, plus précisément au niveau du coût élevé de l'utilisation des protocoles IP (*Internet Protocol*) et UDP (*User Datagram Protocol*) pour la transmission des données à travers le réseau sur puce, ainsi que les problèmes liés aux techniques de routage et de commutation utilisées.

Chapitre 1.

Les réseaux sur puce

1.1 Introduction

Conduis par l'évolution phénoménale de la technologie submicronique[4], les futurs systèmes sur puce (SOC) utilisent des billions de transistors et intègrent des centaines de ressources sur une simple puce.

La conception d'une interconnexion inter-ressources fiable, à énergie réduite et à rendement élevé, s'avère un goulot d'étranglement dans le flot de conception des SOCs. Actuellement, ces communications sont assurées par les bus dans la conception des SOCs, ces moyens de communication sont limités en terme de débit et consommation d'énergie[5]. Les nouvelles générations des SOCs nécessitent une nouvelle architecture des communications inter-ressources offrant plusieurs avantages tels que : la flexibilité, la scalabilité, la bande passante, le débit garanti, et permettant de mieux contrôler les propriétés physiques (surface, bande passante, délais d'interconnexion, interférences et bruits …) de ces plates-formes.

Ce nouveau paradigme de conception des SoCs, baptisé réseau sur puce (***N**etwork **O**n **C**hip* appelé aussi ***NOC***), offre beoucoups solutions pour plusieurs problèmes d'interconnexion[6]. Une architecture de communication à haute performance doit en effet :

- être polyvalente, c'est à dire supporter des flux de données multiples générés par les mêmes ressources.

- Garantir l'intégrité des données, le contrôle de flux, de séquencement, et de la correction d'erreurs de transmission .

-Offrir des performances quantitatives et qualitatives très élevées (haut débit, faible latence et une consommation minimale d'énergie).

- Être extensible, flexible et reprogrammable.

- Permettre la réutilisation des ressources, des fonctionnalités de systèmes et de la conception.

1.2 Architecture générale des réseaux sur puce

1.2.1 Directives de conception

La conception d'un réseau sur puce est basée sur des paramètres technologiques indispensables pour réaliser une architecture d'interconnexion fiable, flexible et à énergie réduite. Cette plate-forme favorise la conception de systèmes sur puces à haute performance.

- **L'architecture:** elle devrait utiliser convenablement les ressources (Processeurs, DSPs, Ips, Mémoires, routeurs, interfaces-réseaux, fils d'interconnexion) disponibles sur la puce. Les signaux de commande n'ont pas besoin d'être acheminés en série avec les données, puisqu'ils peuvent être exécutés dans des fils dédiés aux commandes. L'implantation des mémoires tampons doit être limitée en terme de taille, afin de réduire la consommation de surface et d'énergie.

- **Le paquetage de données :** le système sur puce renferme plusieurs ressources (Processeur, DSP, IP, Mémoires, etc.…) appelées aussi nœuds qui sont interconnectés entre eux par un réseau sur puce. Le trafic sur le réseau est géré par des transactions entre ces nœuds qui génèrent des données découpées en séquences de bits de taille fixe ou variable appelées paquets. Par conséquent, la performance de la communication dans la puce n'est pas déterminée seulement par les aspects physiques du réseau (délais d'interconnexion, interférences), mais dépendent également du paquetage de données sur le réseau.

- **Les techniques de routage et de commutation:** L'acheminement des données sur la puce est assuré par des techniques de routage et de commutation. Ces techniques sont conçues de façon optimisée afin de minimiser l'utilisation substantielle de mémoire tampon sur la puce. L'état du réseau (contrôle de flux, congestion, intégrité de données) est contrôlé par des signaux dédiés aux commandes.

1.2.2 Catégories des réseaux sur puce :

Cette nouvelle architecture de communication est classée en deux catégories :

- **Les réseaux directs :** tous les nœuds sont interconnectés entre eux via des interfaces-réseaux. La ressource et le routeur sont co-implantés, par conséquent l'acheminement des données (routage et arbitrage) est assuré au niveau du nœud.

- Les réseaux indirects : les nœuds sont interconnectés par un ou plusieurs routeurs intermédiaires assurant l'acheminement des données à travers le réseau. En plus, si on adopte une ressource par routeur, ce dernier peut être soit implanté dans le nœud, soit conçu en dehors.

1.2.3 La gestion des Communications

Les communications entre les ressources dans la puce sont assurées par des messages passants ou mémoires partagées.

- Communication via messages passants : l'échange de données entre les nœuds est assuré par des commandes (telles qu'*envoyer()* ou *recevoir()*) exécutées explicitement par des APIs (*Application Program Interface*). Ce mécanisme requière des protocoles spéciaux de transmission, ce qui induit l'augmentation du temps de communication.

- Communication via mémoires partagées : l'échange des données est assuré implicitement par l'accès partagé et concurrentiel aux mémoires. Par conséquent, cette approche a été largement utilisée dans la conception des systèmes sur puce à haute performance[7].

1.2.4 Caractéristiques des NOCs :

Cette partie présente une description détaillée des caractéristiques des NOCs, nous définissons quelques paramètres technologiques indispensables pour la conception d'un réseau sur puce, tels que la topologie, les fils d'interconnexion, les files d'attentes, le paquetage des données et techniques de routage et de commutation.

1.2.4.1 La topologie

C'est un graphe permettant l'interconnexion des différents nœuds, il est régulier si sa topologie correspond à une structure mathématique, sinon il est irrégulier. Les paramètres caractérisant une topologie sont :

- le diamètre : le nombre maximal d'arêtes entre deux ressources.
- Le débit de bissection : le nombre minimal de liens reliant deux moitiés du graphe.
- Le degré : le degré maximal des routeurs
- Le coût marginal : le nombre de routeurs par ressource.

Les différentes topologies adoptées pour la conception des NOCs sont :

- Les topologies régulières : graphe linéaire(grille 1D), en anneau, grille 2D, anneau 2D, cube 2D, arbre binaire, arbre élargi, papillon, papillon en arbre élargi (figure 1)
- Les topologies irrégulières : (figure 1)

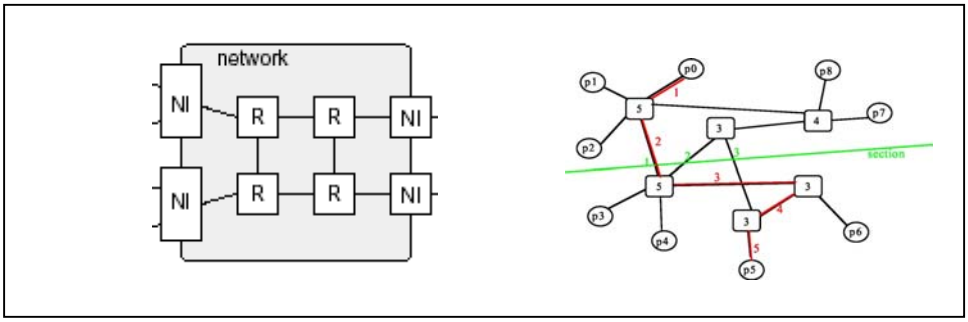


Figure 1 : les topologies irrégulières

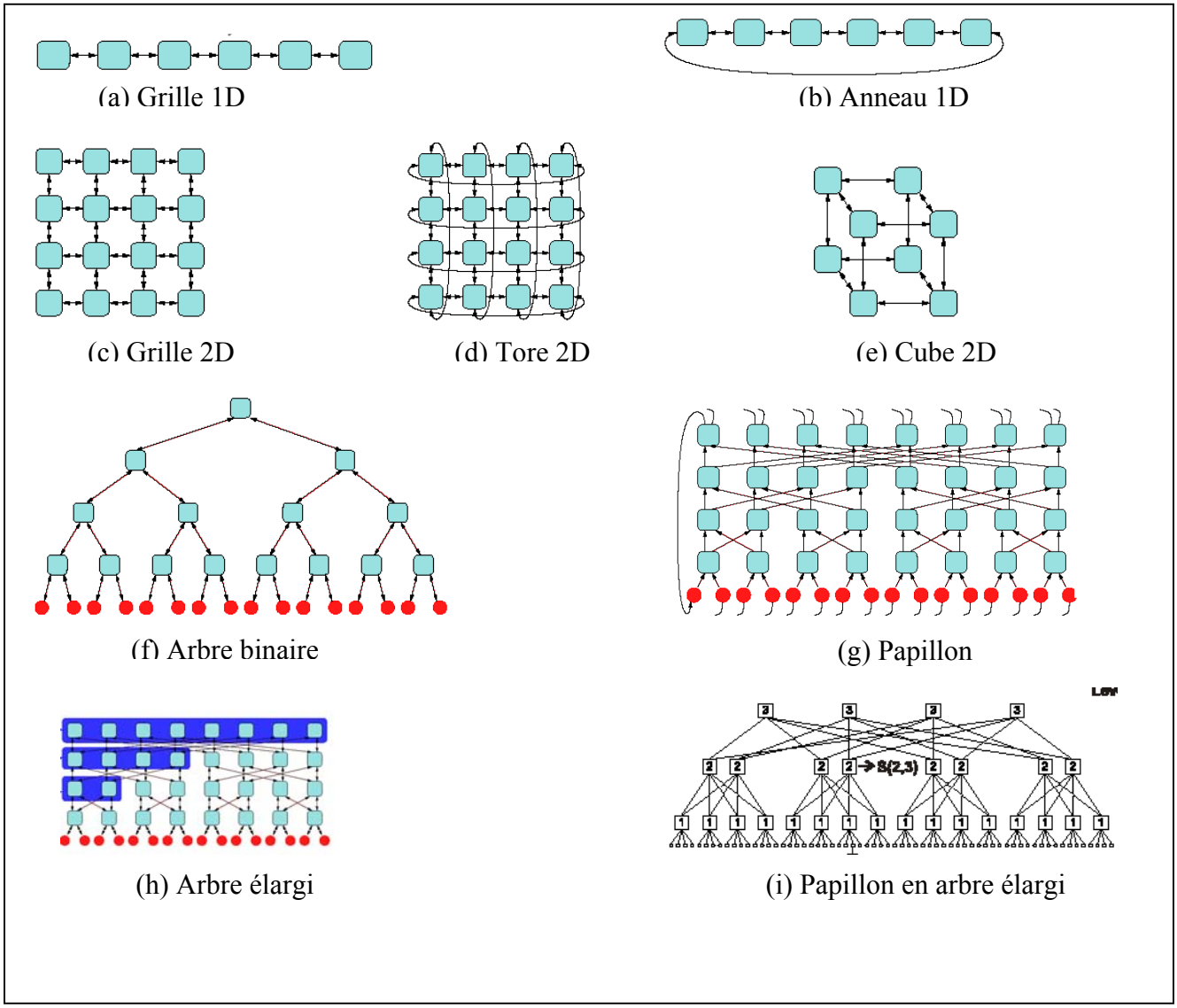


Figure 2 : les topologies régulières

1.2.4.2 Fils d'interconnexion (*wire*)

L'interconnexion des ressources est assurée par des fils. Le nombre de fils est déterminé en fonction des techniques de commutation et de routage utilisées par le réseau sur puce. Pour une technologie d'intégration de 0.13µm avec 8 couches de métal, la taille du fil varie entre 0.30µm et 0.50µm. En outre, un commutateur de surface 100µm x 100µm peut accommoder plusieurs centaines de fils dans plusieurs directions. Par conséquent, le coût d'ajouter plusieurs fonctionnalités de routage diminue quand le processus d'intégration évolue[8].

1.2.4.3 File d'attente

La limitation des fils ; en terme de gestion de conflits et de débit, est compensée par l'utilisation de files d'attentes appelées aussi mémoires tampons. Elles fournissent des espaces de stockage temporaire pour gérer les conflits inter-nœuds d'une part, et le dépassement de la bande passante maximale d'autre part. L'implantation de ces mémoires tampons influe sur la consommation d'énergie et sur la surface de la puce, ce qui induit un coût d'implémentation relativement important. Par conséquent, la taille de ces mémoires tampons est un facteur limitant dans la conception des réseaux sur puces.

1.2.4.4 Paquetage des données

Les données échangées entre les nœuds sont fragmentées en paquets, ces derniers dépendent des protocoles adoptés dans la conception des NOCs. Généralement, un paquet est composé de trois parties :

- **Un en-tête** : adresse source, adresse destination, codification de l'opération(lecture, écriture, commande, …), longueur de données, etc.….
- **Un message** : la donnée à échanger.
- **Une queue** : des données de vérification et de correction d'erreurs de transmission.

a) Les sources de données

Les paquets circulant à travers le réseau proviennent de plusieurs sources, ces paquets peuvent être classés en quatre catégories :

- **Paquet_requête_mémoire** : demande d'accès à la mémoire pour chercher des données (opération de lecture, adresse destination=id_mémoire, adresse source=id_cache, généralement pas de données).

- **Paquet_réponse_mémoire** : les données fournies par la mémoire (adresse destination =id_cache, adresse source= id_mémoire, message= données lues).
- **Paquet_M.A.J_mémoire** : réécriture des données dans la mémoire (opération d'écriture, adresse destination=id_mémoire, adresse source=id_cache, message= données à mettre à jour).
- **Paquet_entrée/sortie** : ce paquet est utilisé pour les opérations d'entrée /sortie (opération E/S, adresse destination=id_noeud, message= données à entrer/sortir).

b) Catégories de paquets

La section précédente montre que la plupart des paquets parcourant le réseau sont acheminés entre les mémoires et les caches, à l'exception des paquets d'entrées/sorties. Ces paquets sont générés par des sources différentes. On distingue deux catégories de paquets :

- **Paquets courts** : ce sont des paquets sans données, constitués d'un en-tête et d'une queue, l'information est décodée à partir de l'en-tête (ex. demande d'accès à une mémoire).
- **Paquets longs** : ce sont des paquets avec données, telle que la réécriture dans la mémoire. Généralement la taille des données est inférieure ou égale à la capacité d'un bloc de cache.

Techniques de routage et de commutation

a) Routage

Il s'agit de l'acheminement des paquets dans le réseau via des routeurs, qui sont chargés de véhiculer des paquets en vue de rejoindre la destination dans un délai minimum. Plusieurs techniques de routage sont utilisées dans la conception des réseaux sur puce :

- **le routage déterministe** : le chemin est déterminé seulement par les emplacements de la source et de la destination.
- **Le routage arithmétique** : l'adresse destination du paquet entrant est comparée à l'adresse du routeur, en fonction de laquelle le paquet sera acheminé(routage relatif).
- **Routage de source (***source routing***)** : le chemin est déterminé par la source, l'entête renferme les adresses des routeurs utilisés. Au passage du paquet, chaque routeur décolle son adresse.
- **Routage adaptatif** : le chemin est géré par les routeurs, pour faire face à l'indisponibilité des ressources (liens, routeurs).

- **Routage distribué** : le routeur dispose d'une table de routage reconfigurable dynamiquement .

b) Commutation

Il s'agit d'un traitement effectué sur le paquet dans le nœud avant de l'acheminer au nœud suivant. Plusieurs techniques de commutation sont envisageables :

- **commutation par mémorisation puis transfert** (*store-and-forward switching*) :le paquet doit être complètement mémorisé dans le nœud, afin de l'acheminer au nœud suivant.
- **Commutation par découpage** (*cut-through switching*) :le paquet est découpé en fragments acheminés à travers le réseau, le paquet peut être expédié à la prochaine étape avant qu'il ne soit complètement reçu par le nœud actuel.
- **Commutation virtuelle par découpage** (*virtual cut-through switching*) : idem que la dernière technique citée, à l'exception du fait que le paquet est totalement mémorisé dans le nœud en cas de blocage du fragment renfermant l'entête dû à la congestion du réseau.
- **Commutation de trou de ver** (*wormhole switching*) : les fragments du paquet suivent le canal réservé par le fragment entête, la queue résilie cette réservation. Cependant tous les fragments sont mémorisés dans les nœuds intermédiaires en cas de blocage du fragment renfermant l'entête dû à la congestion du réseau.

c) Problèmes liés aux techniques de commutation et de routage:

Nous présentons dans cette section les problèmes liés à l'utilisation des techniques de commutation et de routage qui peuvent se manifester au cours de la conception d'un réseau sur puce.

- **Blocage (***deadlock***)** : un ou plusieurs paquets peuvent se bloquer mutuellement, en outre ils attendent des ressources qui ne seront jamais disponibles.
- **Diffusion éternelle (***livelock***)** :le paquet circule éternellement à travers le réseau sans rejoindre sa destination, ce qui induit une saturation du réseau. Pour remédier à ce problème, il faut instaurer un compteur au niveau de l'entête du paquet renfermant le nombre maximal de routeurs traversés par le paquet, ce compteur est décrémenté au passage par chaque routeur. Le paquet est détruit lorsqu'il ne rejoint pas sa destination.

1.3 Etude exhaustive des recherches relatives au thème réseau sur puce

Cette partie présente les recherches relatives au réseau sur puce. Nous avons choisi des exemples provenant d'horizons très variés pour montrer la polyvalence et l'évolution de cette technologie. On s'est intéressé essentiellement à l'architecture générale des réseaux existants, ainsi qu'au paquetage des données et aux techniques de routage et de commutation adoptées.

1.3.1 SPIN (*Scalable Programmable Integrated Network*)

C'est un réseau d'interconnexion à commutation de paquets pour systèmes intégrés sur puce développé au LIP6[9]. Ce réseau est basé sur une topologie dite d'arbre quaternaire élargi(*Fat Tree*), cette topologie a les avantages suivants : le nombre maximal de liens entre les ressources est raisonnable (il est égal à $2*\log_4(n)$ avec n le nombre de niveaux), en plus elle est extensible, hiérarchique et utilise un petit nombre de routeurs pour un nombre donné de ressources. Les informations qui circulent sur le réseau SPIN sont des paquets. Un paquet SPIN est une séquence de fragments de 36bits, le premier fragment possède un marqueur de début de paquet et le dernier fragment renferme le marqueur de fin de paquet. La technique de routage utilisée est distribuée. Il en résulte que chaque routeur achemine les paquets sans l'intervention d'une synchronisation centrale. La technique de commutation utilisée est de type *Wormhole* pour limiter la latence. Le réseau SPIN (figure 3) fournit un mécanisme de communication dynamique entre les différents composants connectés dans le système. De plus, la bande passante croît linéairement avec le nombre de processeurs intégrés. L'implantation de ce réseau a soulevé des problèmes tels que la présence d'interblocages et la difficulté de la gestion des ressources autour du réseau. Pour remédier à ces problèmes, des *Wrappers* ont été conçus pour fournir aux ressources des interfaces de communication respectant le standard VCI (*Virtual Component Interface*) [10].

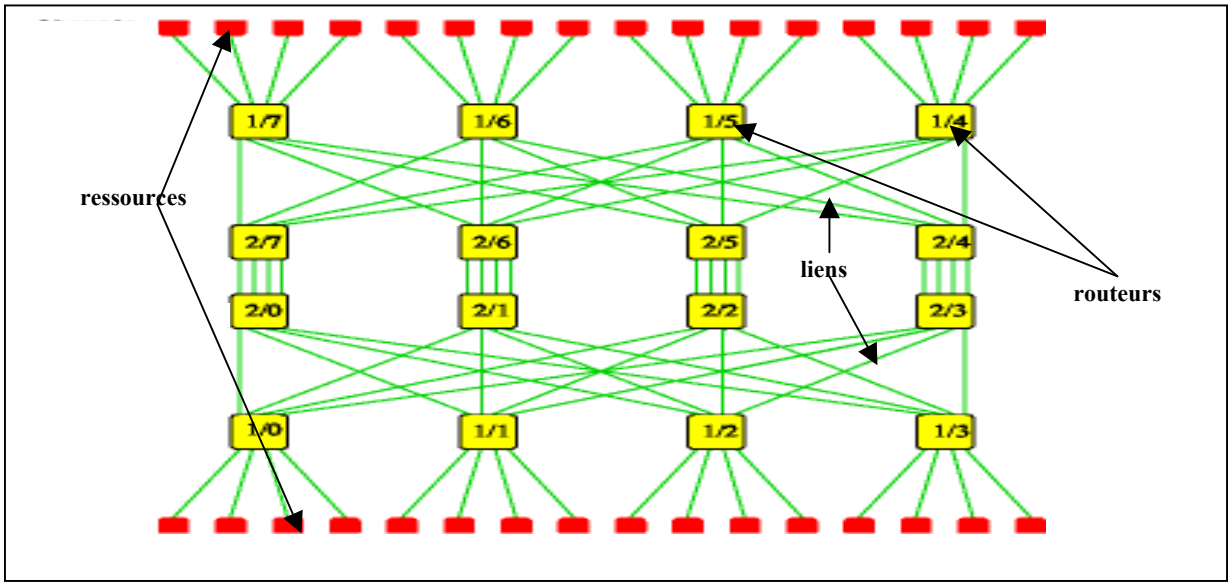


Figure 3: le réseau SPIN

1.3.2 Les réseaux orthogonaux

Ces réseaux sont interconnectés en grille ou en tore multidimensionnel. Un réseau direct en grille 2D a été proposé par Dally [11]. Ce modèle est basé sur une topologie qui interconnecte tous les processeurs entre eux directement (figure 4). Les fonctionnalités de routage, d'arbitrage et de paquetage de données sont réalisées indépendamment au niveau de chaque nœud. Les données échangées sont découpées en fragments de 294bits dont 38bits sont des données de contrôle (le type du paquet, la taille de la donnée, le numéro du canal virtuel, le chemin, l'indicateur de service). Le routeur utilise l'algorithme de source déterministe. La commutation des données est faite via des canaux virtuels pour limiter la latence, en plus ces données sont mémorisées dans des files d'attentes en entrée. Le processeur et le routeur sont co-implantés dans le même nœud. Les interfaces réseaux sont localisées aux quatre périphéries de chaque nœud . Ce réseau a des avantages de structure, d'exécution et de modularité, tels que :

- la prédiction des paramètres électriques (puissance, interférence et bruit) permettant l'obtention de circuits à haute performance en terme de latence , de bande passante et de puissance minimale.
- La réutilisation de la conception du réseau, c'est à dire le routeur est un composant réutilisable.
- La simplicité de l'interface-réseau, ce qui facilite l'interopérabilité avec une grande variété de protocoles.

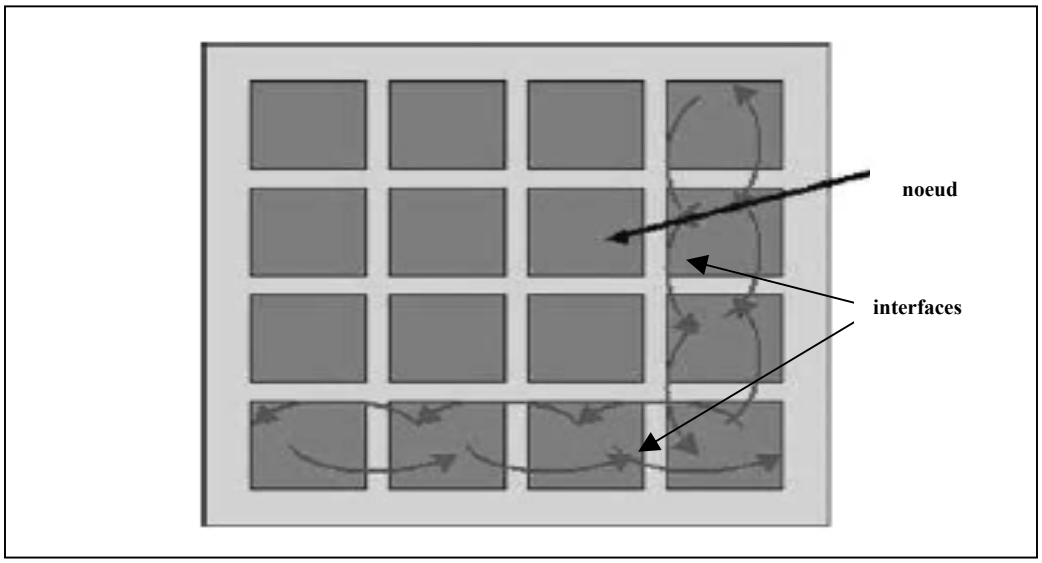


Figure 4: Dally NOC

1.3.3 Le réseau NOSTRUM

Ce réseau adopte la topologie grille 2D (figure 5). Chaque ressource est connectée à un routeur via une interface réseau. Chaque routeur est connecté aux quatre voisins. Ce modèle utilise la technique de commutation de paquets avec des circuits virtuels de communication pour garantir la bande passante et la latence. Chaque paquet est découpé en Flit (*Flow Unit*) de 300bits dont 10 sont réservés au contrôle. Le routeur utilise l'algorithme *Hot-potato routing* pour acheminer les données, ce qui minimise l'utilisation des mémoires tampons. Les données arrivant à un routeur sont mémorisées dans des files d'attente en entrée. Le contrôle de flux est assuré par des mécanismes d'acquittement, de fenêtre d'anticipation et de retransmission [12]. Un simulateur a été développé en *SystemC* pour évaluer cette architecture par une étude comparative entre le réseau NOSTRUM et une architecture à base de bus[13].

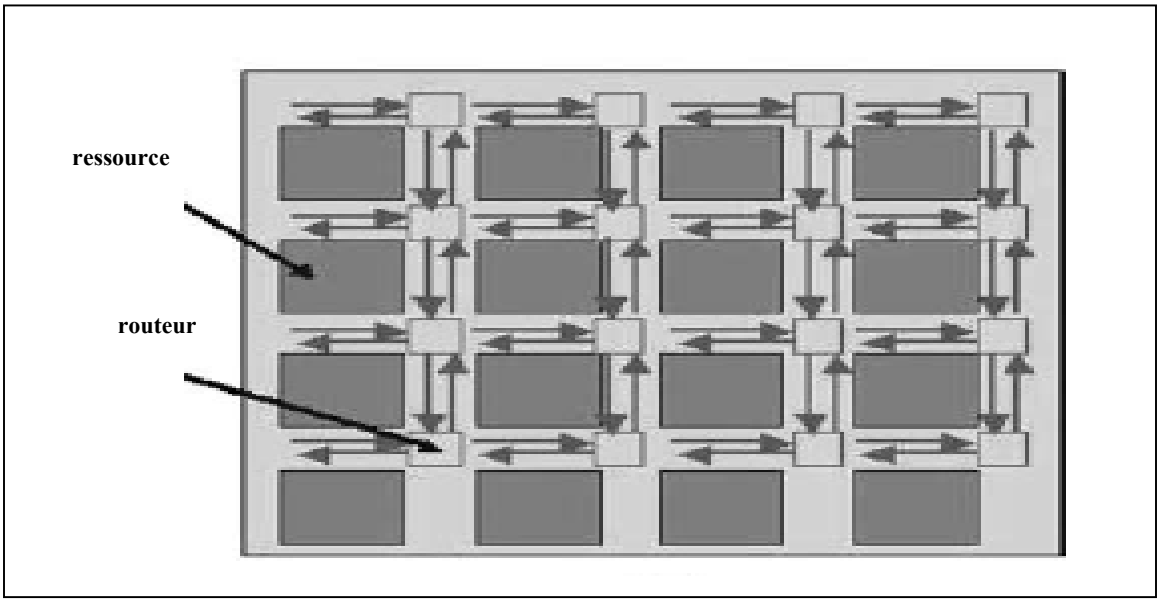


Figure 5: le réseau sur puce NOSTRUM

1.3.4 Le réseau Octagon

C'est un réseau direct, proposé par Karim [14]. Ce modéle est basé sur une topologie en anneaux raccordés (figure 6). Chaque anneau renferme huit nœuds. Les fonctionnalités de routage et de commutation sont co-implantées avec le processeur. Le paquet circulant à travers le réseau est de taille variable, l'entête du paquet renferme trois bits dédiés pour le contrôle(bits d'adresses). Ce réseau utilise la commutation de paquets et de circuits. La technique de routage adoptée est de type distribué et adaptative. La communication entre deux nœuds quelconques d'un anneau exige au plus deux liens intermédiaires. La bande passante de ce réseau peut atteindre 40Gbits/s, ce qui permet d'obtenir des circuits à rendement élevé .

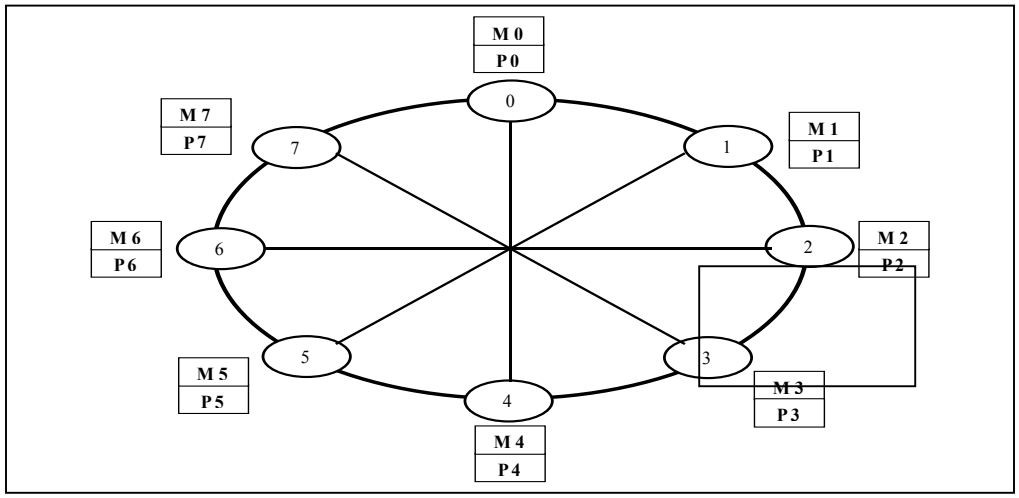


Figure 6: le réseau sur puce Octagon

1.3.5 Le réseau Æthereal

Ce réseau a été développé au laboratoire de recherches de Philips aux Pays Bas[15]. Il est basé sur une topologie irrégulière (figure 7). Les ressources(processeur, mémoire, Ip, etc.…) sont connectées au routeurs par des interfaces-réseaux. Le routeur Æthereal utilise un routage de source déterministe (*source routing*), une commutation de type *Wormhole* et une mémorisation de paquets en entrée. Chaque paquet est découpé en flits de 32bits, le premier flit renferme l'entête(identification de paquet, taille, chemin, fenêtre d'anticipation, indicateur de fin de paquet). Æthereal fournit un transfert fiable de données via des routeurs opérant en deux catégories de trafic (établissement de connexion de bout en bout puis échange de données). Les interfaces-réseaux assurent plusieurs fonctions telles que, le contrôle de flux, le paquetage de données, la connexion avec les protocoles standards d'interface, ainsi que l'ordonnancement des transactions générées par les ressources connectées au réseau.

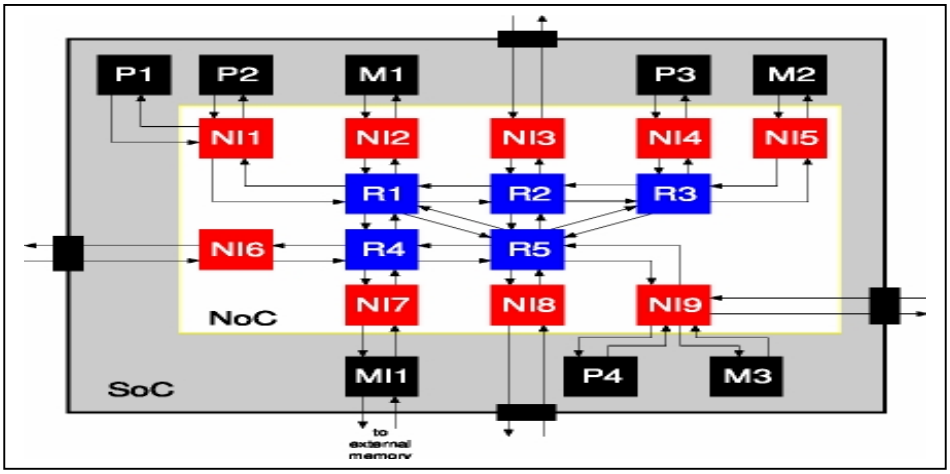


Figure 7: le réseau sur puce Æthereal

1.3.6 Le réseau Proteo

Cette architecture a été développée à l'Université de Tampere en Finlande[16]. Elle adopte une topologie en anneau bi-directionnel. Ce modèle est composé de plusieurs sous-réseaux interconnectés entre eux par des ponts (figure 8). Ce réseau offre une bibliothèque de communication flexible supportant plusieurs topologies et stratégies de routage. Le routeur dispose d'une table de routage veillant sur l'acheminement des données et utilise la technique de

commutation de paquets. Des files d'attente sont utilisées en entrées et en sorties. Un environnement de simulation à base de VHDL a été créé pour la validation de cette approche[17].

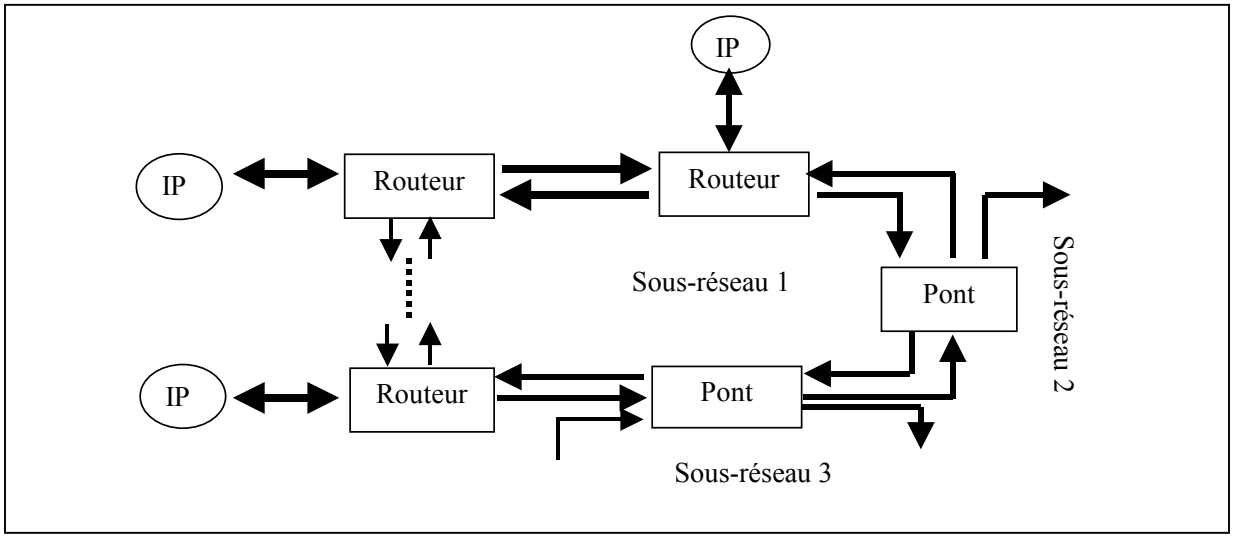


Figure 8: Proteo NOC

1.3.7 Le réseau Hermes

Cette architecture adopte la topologie en grille2D [18]. Chaque ressource (processeur, Ip) est connectée à un routeur (figure 9). Ce dernier est composé de cinq ports (Est, Ouest, Nord, Sud et Local). Le port local est relié à une ressource alors que les autres ports sont reliés aux routeurs voisins. Chaque port possède une file d'attente en entrée pour stocker provisoirement les données. La technique de commutation utilisée est de type *Wormhole* afin de diminuer la latence et l'utilisation de mémoires tampons. Les paquets circulant dans le réseau contiennent des données, un en-tête qui renferme l'adresse destination et un compteur indiquant le nombre de mots dans le paquet. L'acheminement des paquets dans le réseau est fait suivant une stratégie de routage arithmétique basée sur l'adresse du routeur exprimé en XY, où X représente sa position horizontale et Y sa position verticale. Les avantages primordiaux de cette plate-forme est sa performance, notamment en terme de latence et débit, ainsi que sa flexibilité du fait que les files d'attente et la taille des paquets sont paramétrables.

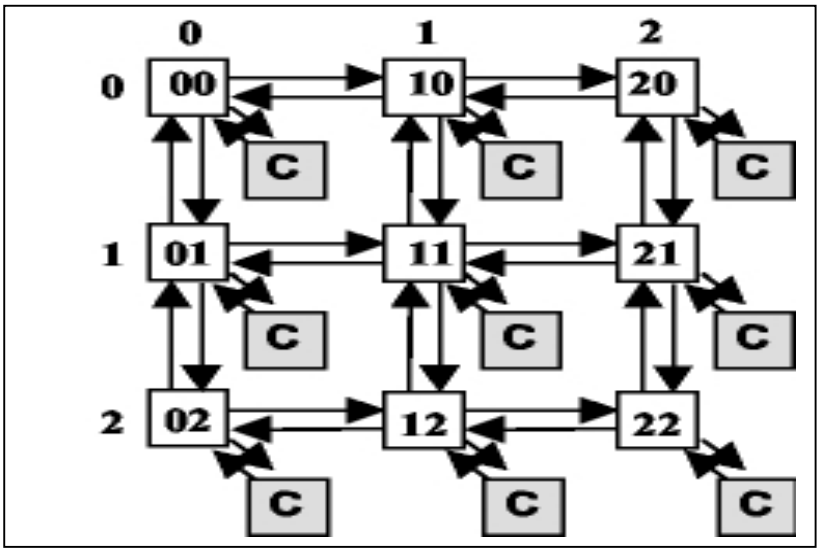


Figure 9: Hermes NOC

1.3.8 ECLIPSE(*Embedded Chip Level Integrated Parallel Super computEr*)

Cette approche adopte une topologie en maille 2D hiérarchisée(figure 10). Le nombre de routeurs est au moins le carré du nombre de ressource divisé par quatre[19]. Elle utilise des mémoires partagées afin d'éviter les problèmes de cohérence de cache. L'avantage primordial de cette nouvelle architecture réside dans le fait que les communications ne bloquent jamais le réseau dans le cas d'un trafic lourd.

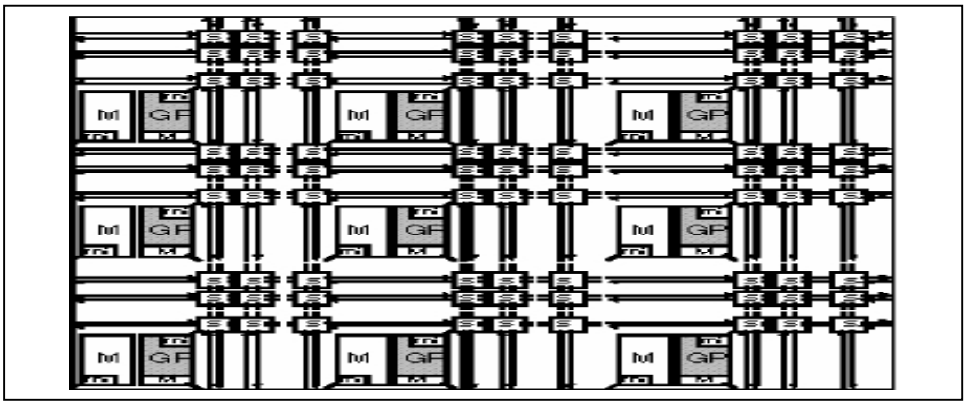


Figure 10: ECLIPSE NOC

1.4 Performances des réseaux sur puces

Nous avons passé en revue les différentes conceptions des réseaux sur puce. Bien que ces travaux restent théoriques jusqu'à nos jours, ils nous ont permis de déceler plusieurs besoins auxquels doit répondre une architecture d'interconnexion d'une part et des orientations générales indispensables pour la conception d'un réseau sur puce fiable et flexible d'autre part.

1.4.1 Critères d'évaluation d'un réseau sur puce

Nous présentons les objectifs quantitatifs et qualitatifs minimaux que doit atteindre un réseau sur puce comme suit :

- **la bande passante** : le réseau sur puce doit offrir une bande passante suffisante pour établir des communications multi-Gbit/s entre les ressources. En plus, cette bande passante devra croître linéairement avec le nombre de ressources connectées.
- **La latence** : plusieurs systèmes « temps réels » et applications spécifiques (vidéo) demandent une très faible latence. Pour cela, le réseau sur puce doit utiliser des mécanismes de routage acheminant les données dans un temps minimum.
- **La fiabilité** : c'est le contrôle du taux de disponibilité du réseau avec des hypothèses pertinentes de charge. Cette disponibilité doit être compatible avec plusieurs heures de bon fonctionnement continu.
- **La flexibilité** : les systèmes sur puce sont toujours composés de sous-systèmes hétérogènes. L'architecture de communication doit veiller à la coopération entre ces sous-systèmes par la mise en place d'une plate-forme supportant des trafics variés et ayant un interfaçage souple avec les protocoles standards. En plus, cette architecture doit offrir la possibilité de la programmation, de la reconfiguration et même de la réutilisation de la conception.
- **La consommation** : les propriétés physiques d'un réseau sur puce (telles que la surface, les interférences et les bruits) sont prévisibles et soumises à un contrôle afin de réaliser une architecture de connexion à consommation réduite.
- **La surface** : l'architecture de communication doit occuper une surface minimale de la puce.

1.4.2 Directives pour la conception d'un réseau sur puce

Nous présentons dans cette partie des recommandations indispensables pour la conception d'un réseau sur puce à énergie réduite et à rendement élevé.

- **L'architecture** : la topologie doit être extensible et utilise un nombre limité de liens et de routeurs.
- **La segmentation et la mémorisation de données** : les informations échangées doivent être découpées en séquences de bits de petite taille fixe, puis mémorisées dans des files d'attente de capacité limitée.
- **Les stratégies de routage et de la commutation** : le routeur doit utiliser des techniques de routage et de commutation permettant d'acheminer les paquets dans un temps minimum.

Les deux premiers paramètres influent sur la consommation de l'énergie et sur la surface du réseau sur la puce, le troisième paramètre définit la latence du système.

1.5 Conclusion

Dans cette partie, nous avons passé en revue les différentes conceptions des réseaux sur puces, les critères pertinents d'évaluation de la plate-forme d'interconnexion, ainsi que des mesures technologiques que le concepteur doit prendre en considération pour implémenter un réseau sur puce.

Ces critères serviront à orienter les choix de conception de notre modèle de réseau sur puce baptisé BFT-NOC. Les performances obtenues par les réalisations faites dans cet axe de recherche définissent les objectifs qu'on s'est fixé dans ce travail.

Chapitre 2.

Le simulateur NS-2

Après avoir exploré l'architecture générale du réseau sur puce et avant de modéliser notre architecture baptisée BFT-NOC, il a fallu choisir un simulateur pour évaluer notre modèle. Pour cela, nous décelons en première partie les similitudes entre les domaines des NOCs et des réseaux publics, puis nous étudions les simulateurs de réseaux existants et enfin nous présentons le simulateur NS-2 choisi en vue d'évaluer notre architecture.

2.1 NOC et les réseaux publics

L'étude approfondie faite dans le chapitre précédent nous a permis de déceler plusieurs similitudes entre les NOCs et les réseaux publics. Le réseau sur puce interconnecte plusieurs ressources tels que processeurs, DSPs, FPGAs, Ips, mémoires qui ressemblent aux terminaux connectés à un réseau public. Les routeurs ou commutateurs implémentés dans les NOCs ont presque les mêmes fonctionnalités que ceux des réseaux publics. L'acheminement des données est assuré respectivement par des fils en métal dans un NOC et des supports de transmission pour les réseaux publics. En plus, l'architecture d'un NOC est plus simple que celle d'un réseau public, étant donné tous les composants sont intégrés sur la même puce. Tous ces indicateurs nous ont laissé réfléchir à choisir un simulateur de réseau public à adopter en vue d'évaluer notre modèle.

2.2 Les simulateurs de réseaux publics

Nous étudions dans cette section quelques simulateurs utilisés dans le domaine des réseaux publics. Ce n'est pas une étude exhaustive puisque plusieurs centres de recherche ont conçu des simulateurs pour des objectifs précis tels que la simulation d'un protocole ou d'un problème particulier. Par conséquent, la documentation relative à un simulateur est souvent pauvre ou confidentielle, les bogues ne sont pas fixes et les simulateurs n'ont pas évolué. Dans la partie qui suit nous présentons des simulateurs de réseaux dont les informations les concernant sont disponibles[20].

- Netsim : conçu par MIT LCS, c'est un simulateur à événement discret, pour les réseaux à commutation de paquets, avec une stratégie de routage statique et mémorisation en file d'attente événementielle.
- Insane : conçu à l'université de Californie, c'est un simulateur à événement discret, orienté objet, utilise une table de routage statique, et une mémorisation FIFO. Il est utilisé pour simuler les réseaux ATM (Asynchronus transfer mode).
- Nest 2.5 : développé à l'université de Columbia, en vue de simuler les systèmes et les algorithmes distribués.
- Real 5.0 : c'est une version évoluée de NEST2.5, il évalue le comportement dynamique du flux d'information échangé, ainsi que la congestion des réseaux à commutation de paquets (notamment TCP/IP).
- Network simulator 2 (NS-2) : il a été développé par le groupe de recherche de réseau au laboratoire national de Laurent Berkeley (LBNL). C'est un simulateur orienté objet et à événement discret, il évalue les réseaux à commutation de paquets. De plus amples détails pour cet outil sont présentés dans la section suivante.
- Autres simulateurs : Opnet, Cpism, Cnet, Mars, Simunet, Gpss, Ipv6, Netsim++, . ..

Parmi les simulateurs cités ci-dessus, nous avons opté pour NS-2 en vue d'évaluer notre modèle baptisé BFT-NOC. Ce choix a été déterminé en fonction des avantages de ce simulateur, entre autres :

- un domaine large d'applications,
- des fonctionnalités multiples d'implémentation,
- des interfaces utilisateurs simples,
- l'efficacité, la scalabilité et la synergie,
- plusieurs niveaux d'abstraction .

2.3 Présentation du simulateur NS-2

2.3.1 Les fonctionnalités de base de NS-2 :

Le simulateur NS-2 propose aux utilisateurs un support de recherche pour la gestion des réseaux , telles que l'implémentation de la topologie, la modélisation des routeurs (file d'attente,

stratégies de routage…), la génération du trafic, la présentation de résultats sous forme de graphiques. Aussi, il dispose d'une source ouverte et distribuée (code partagé), ce qui facilite la comparaison des modèles, des protocoles et des résultats. En plus, cet outil fournit des niveaux multiples de détails, des interfaces utilisateurs simples, et plusieurs niveaux d'abstraction.

2.3.2 Architecture logicielle de NS-2 :

Le simulateur NS-2 est l'objet de perfectionnements périodiques effectués par un groupe de recherches de réseau au laboratoire national de Laurent Berkeley (LBNL). C'est un ensemble de ligne de code en C++ et OTCL (environ 200K), cet outil présente aussi plus de cent exemples testés, et un manuel d'utilisation de 371 pages qui illustre touts les fonctionnalités du simulateur[21]. Ce composant est supporté par toutes les plates-formes (Linux, Solaris, Windows, Mac).

2.3.3 Les composants de NS-2 :

Le simulateur NS-2 renferme plusieurs composants ayant des fonctionnalités précises, nous passons en revue ci-dessous les modules les plus utilisés au cours d'une simulation :

- NS, le noyau du simulateur, composé d'un ensemble de modules et de bibliothèques .
- Nam, l'animateur de réseau: c'est l'éditeur pour générer des scripts NS et pour visualiser les modèles implémentés.
- Les générateurs de topologie et d'applications .
- Le moniteur des files d'attente.
- L'analyseur de la bande passante.

En plus, le simulateur NS-2 offre la possibilité d'incorporer des protocoles de transmission, des mécanismes de routage et des générateurs d'applications, ces modules sont conçus par les utilisateurs.

2.3.4 Le modèle d'interconnexion de NS-2 :

Le simulateur NS-2 s'est inspiré du modèle OSI (Open System Interconnexion) de l'ISO (International Standard Organisation) le découpage en couches, mais se limite à cinq couches seulement à savoir :

- La couche application : Ftp, Web, Telnet, Cbr, Vbr
- La couche transport : TCP, UDP, LossMonitor…
- La couche réseau : protocole de routage (statique, dynamique ou manuel), file d'attente (Red, Drop-tail,…)
- La couche liaison : acheminement des données, contrôle d'erreurs. ..
- La couche physique : acheminement à travers des fils, ou sans-fils, ou par satellite.

2.3.5 Utilisation de NS-2

L'organigramme cité ci-dessous décrit les étapes à suivre pour l'utilisation de NS-2, nous signalons que cet outil offre aux utilisateurs l'opportunité de simuler plusieurs processus en même temps.

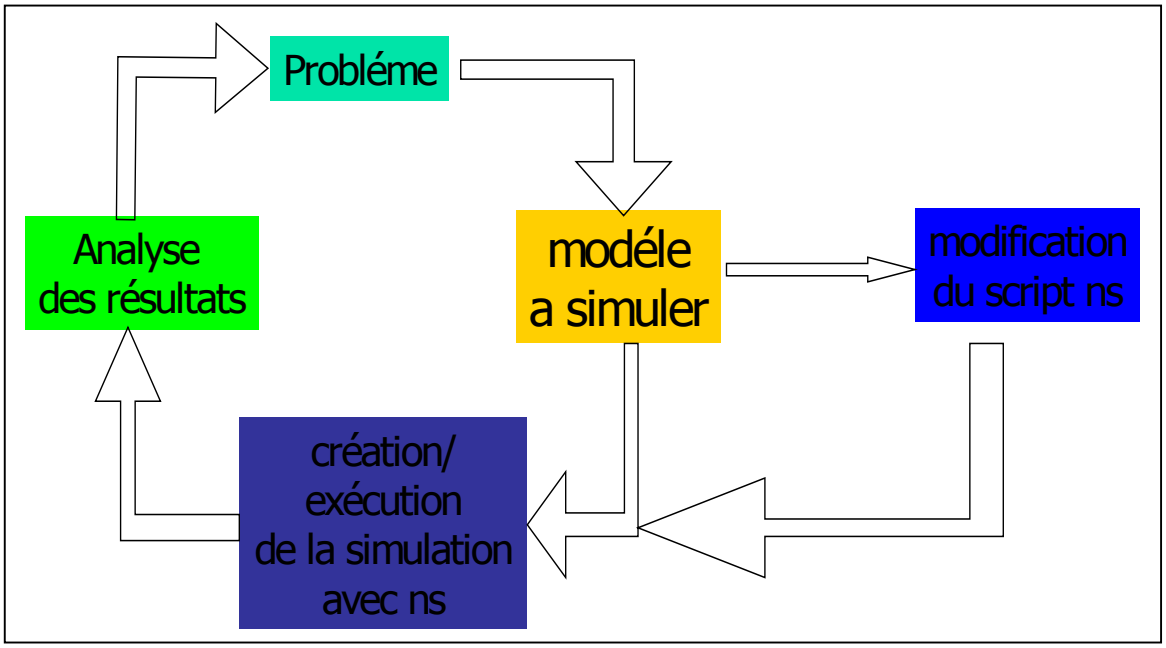


Figure 11: organigramme d'utilisation de NS-2

2.4 Tâches à entreprendre pour modeler et simuler un réseau avec NS-2

La conception, l'implémentation et la simulation d'un réseau avec NS-2 nécessitent des tâches réparties en trois phases :

2.4.1 Pré-traitement

- la création du modèle à simuler,
- la définition du déclencheur de début et de fin des événements,
- l'activation des moniteurs de trafic, de l'animateur de trafic et des traceurs de graphes (figure 12),
- l'implémentation de la topologie (la capacité du réseau en terme de bits, la taille de la file d'attente, la nature de la liaison (*simplex* ou *duplex*), le mécanisme de mémorisation (*DropTail, Red*, etc..), le moniteur de la file d'attente).

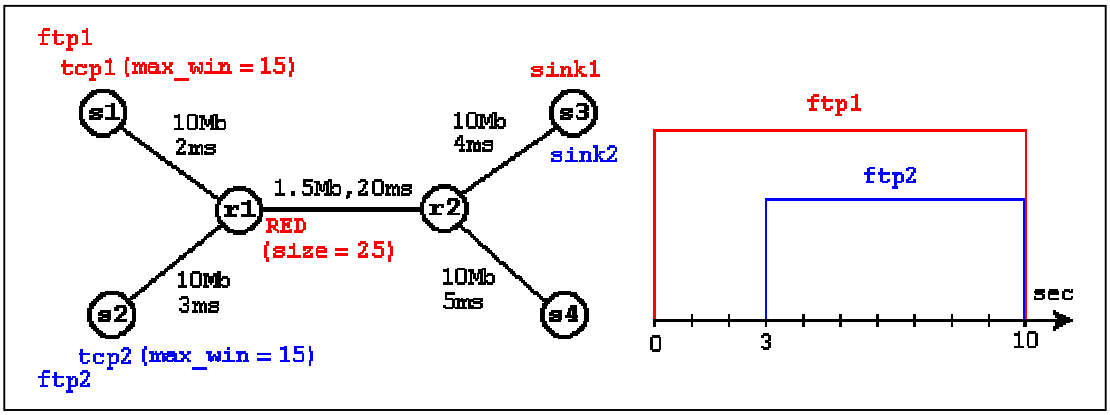


Figure 12: une topologie de réseau et un scénario de simulation

2.4.2 Traitement

- la mise en place de la stratégie et du protocole de routage (statique, dynamique ou manuelle),
- le choix du mécanisme de la gestion des files d'attente,
- la création de la connexion au niveau transport par des protocoles destinés à la transmission,
- la génération du trafic dans le réseau par des applications (figure 13).

2.4.3 Post-traitement

- l'exécution de la simulation
- l'analyse des résultats de simulation fournis par des moniteurs de files d'attente et de bande passante (figure 14) .

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)                    src_addr : node.port (3.0)
+ : enqueue (at queue)                      dst_addr : node.port (0.0)
- : dequeue (at queue)
d : drop    (at queue)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

Figure 13: détails du trafic inter-nœuds

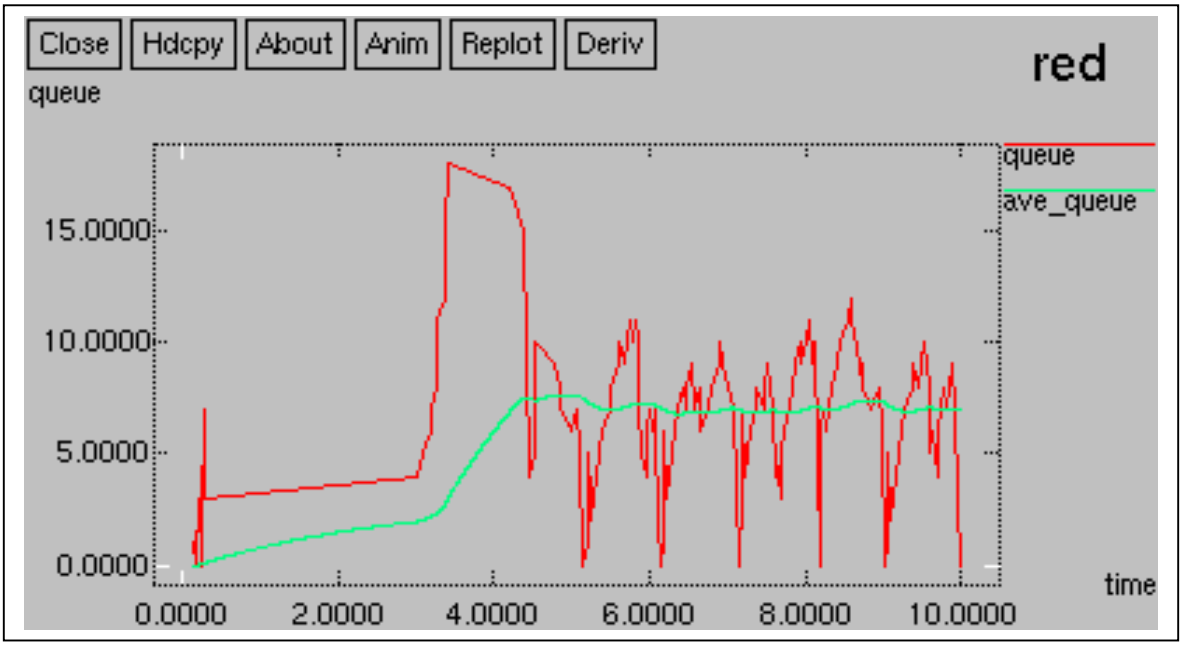


Figure 14: graphe évaluant la taille de la file d'attente durant la simulation

Chapitre 3.

Architecture générale du réseau sur puce (BFT_NOC)

En réinvestissant les connaissances acquises dans le domaine du réseau sur puce par l'étude exhaustive des travaux relatifs à ce thème, et en déterminant les motivations et les objectifs de ce travail, nous présentons dans cette partie le modèle de notre réseau sur puce baptisé BFT_NOC. La conception et la simulation du modèle sont assurées par le simulateur NS-2.

Cette étape consiste à concevoir le modèle en topologie papillon en arbre élargi, choisir les blocs fonctionnels (ressources et routeurs), les paramètres liés à la communication inter-ressources (bande passante, latence, débit…), les techniques de routage, les files d'attente, et la conception d'un analyseur de trafic du réseau permettant l'obtention des résultats de simulation.

3.1 Introduction

Notre modèle est composé essentiellement de trois parties (figure 15), la mise en place de la topologie, la génération des communications inter-ressources, l'implémentation d'un analyseur contrôlant des paramètres liés au trafic dans le réseau. Ces parties satisfont les besoins de communication de ce modèle, et permettent l'obtention des résultats de simulation analysés dans le chapitre 6.

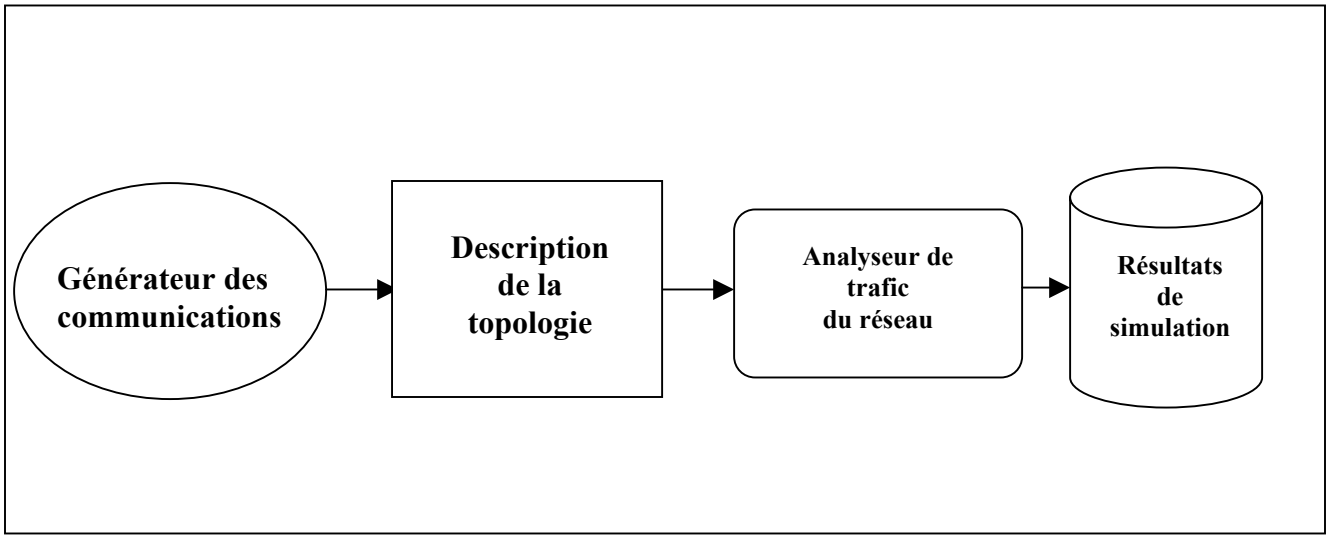


Figure 15: le modèle abstrait du BFT_NOC

3.2 Architecture générale du modèle BFT_NOC

Cette partie présente la mise en place de la topologie papillon en arbre élargi (*Butterfly Fat Tree*) sur notre modèle BFT_NOC. Dans la première phase, nous décrivons la topologie adoptée, puis l'architecture et le fonctionnement des composants (ressources et routeurs) intervenant dans ce modèle et en dernière phase les communications nécessaires pour cette plate-forme.

3.2.1 Implémentation de la topologie sur BFT_NOC

3.2.1.1 Architecture de la topologie papillon en arbre élargi

Nous avons opté pour la topologie papillon en arbre élargi (figure 16) pour interconnecter de nombreux blocs (processeurs, mémoires, DSP, IP ….). Dans ce modèle, les ressources sont placées aux feuilles et les routeurs aux sommets. Chaque nœud est marqué par une paire de coordonnées (l, a) où l dénote le niveau d'un nœud et le p dénote sa position dans ce niveau. En général, au niveau le plus bas (l = 0), il y a N ressources avec des adresses s'étendant de 0 à (N-1). La paire (0, N) dénote les endroits des ressources à ce niveau le plus bas. Le routeur marqué par R(l, p) possède six ports : $ascendant_0$, $ascendant_1$, $descendant_0$, $descendant_1$, $descendant_2$ et $descendant_3$. Les ressources sont reliées aux N/4 routeurs au premier niveau. Le nombre de niveaux dépend du nombre de ressources. En outre pour N ressources, le nombre de niveaux est $\log_4(N)$. Au l ^{éme} niveau (de l=1 à $\log_4(N)$) , on dispose de $N/2^{l+1}$ routeurs[22].

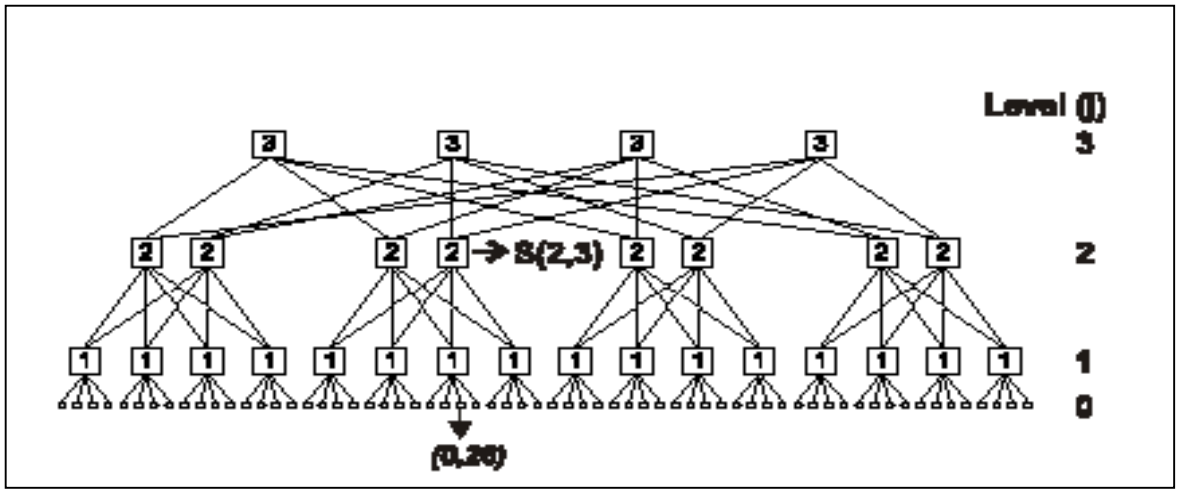


Figure 16: une architecture papillon en arbre élargi composée de 64 ressources et 28 routeurs

Les connections ressource-routeur et routeur-routeur sont déterminées comme suit :

- une ressource N(0, a) est connectée au descendant$_{(amod4)}$ dénoté S(1, (adiv4))
- l'ascendant$_0$ du routeur R(l, a) est connecté au descendant$_{(i)}$ dénoté S(l+1, (adiv2^{l+1})*2l + (a mod 2^l)), avec i= (a mod 2^{l+1})div 2^{l-1}
- l'ascendant$_1$ du routeur R(l, a) est connecté au descendant$_{(i)}$ dénoté S(l+1, (adiv2^{l+1})*2l + ((a+2^{l-1})mod 2^l)), avec i= (a mod 2^{l+1})div 2^{l-1}

L'adoption de cette architecture nous a permis plusieurs avantages :

- Le nombre de routeurs dans cette topologie converge vers une constante indépendante du nombre de niveaux, et ce nombre tend vers la moitié du nombre de ressources quand ce dernier croit arbitrairement.

$$R=\frac{N}{4}+\frac{1}{2}\frac{N}{4}+\frac{1}{4}\frac{N}{4}+\dots\left(\frac{1}{2}\right)^{l}\frac{N}{4}=\frac{N}{4}\left(\frac{1-\left(\frac{1}{2}\right)^{l}}{1-\frac{1}{2}}\right) \quad R \xrightarrow[l\ \to]{} \frac{N}{2}$$

- Il existe plus qu'un chemin court entre une paire de feuilles, en outre un message est acheminé à travers l'un des deux liens hauts du routeur.
- Les fils entre les ressources et les routeurs sont logiquement structurés, donc leurs longueurs peuvent être rendues prévisibles.
- Le trafic émergeant de/arrivant aux ressources est combiné dans un fil simple, ceci induit à la réduction de la congestion du fil.

3.2.1.2 Modélisation de la topologie

Notre objectif essentiel est de concevoir un réseau sur puce dont la topologie est papillon en arbre élargi. Une architecture composée de 16 ressources, 6 routeurs et 3 niveaux sera modélisée puis simulée (figure 17).

Les routeurs et les ressources sont présentés respectivement par des carrés et des cercles. Les ressources sont hétérogènes (mémoires, processeur, DSP, IP ..) et sont choisis aléatoirement soit source ou destination.

La connexion ressource-routeur est faite via un composant intermédiaire implémenté dans la ressource et baptisé RNI (*Ressource Interface Network*). La connexion routeur-routeur permet la communication entre les ressources. Chaque composant(ressource, routeur) possède une adresse unique indiquant le niveau du composant et sa position dans ce niveau.

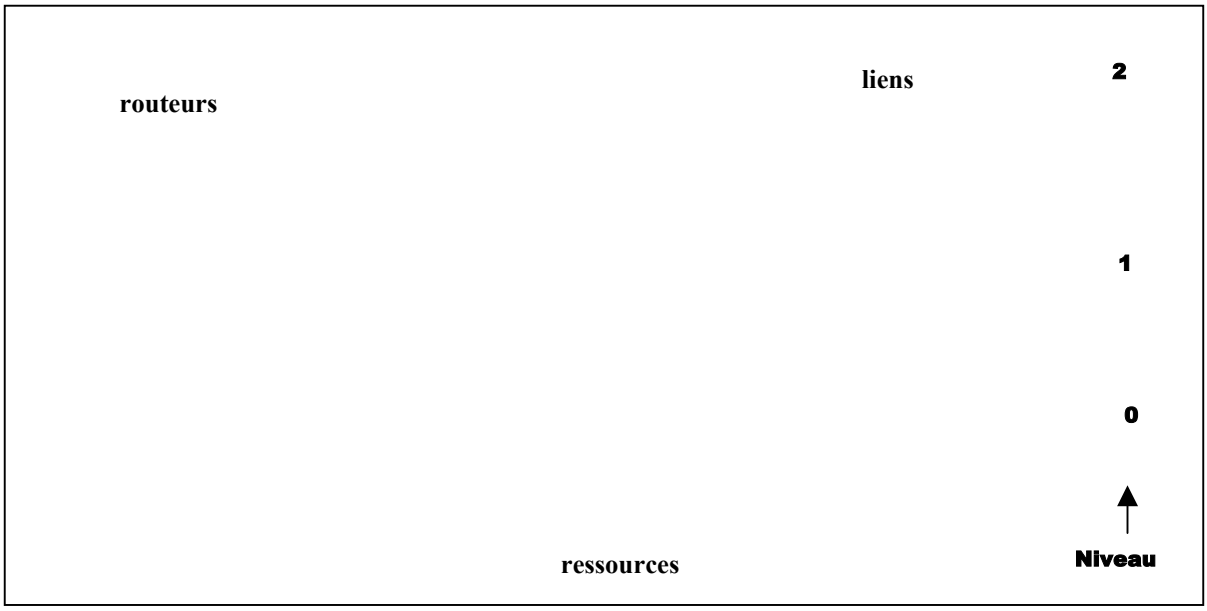


Figure 17: Synoptique du BFT-NOC

3.2.2 Architecture des blocs fonctionnels

Le modèle proposé comprend deux composants importants : le routeur et la ressource, ainsi que les communications établies entre eux.

3.2.2.1 Le routeur

Le routeur est un composant vital de tout réseau. Il doit donc être très compact. D'autre part, la performance globale du réseau dépend du comportement individuel des routeurs et de leurs interactions, les meilleurs résultats étant évidemment obtenus avec les routeurs les plus sophistiqués.

Nous proposons un routeur qui assure l'établissement de la connexion, l'acheminement des paquets et la gestion de la file d'attente. Ce routeur comprend 6 ports reliés par des fils en métal. Chaque port comprend un lien_départ et un lien_arrivée. Une file d'attente de taille fixe est assignée à chaque lien_arrivée. Nous avons adopté le mécanisme FIFO pour la gestion de la file d'attente, c'est à dire quand un paquet arrive à un routeur (figure 18), deux hypothèses se présentent : La première, s'il y a de l'espace dans la file d'attente, le paquet doit attendre dans la file jusqu'à devenir prioritaire afin de l'acheminer au prochain nœud. La seconde, si la file d'attente est saturée, le paquet est détruit définitivement. En ce qui concerne le mécanisme de routage, nous avons adopté deux stratégies de routage pour la détermination du prochain nœud.

La première stratégie est statique, en outre les calculs sont faits une seule fois pour une seule simulation. La deuxième stratégie de routage est dynamique, fondée sur un algorithme modifiant la table de routage dynamiquement durant la simulation .

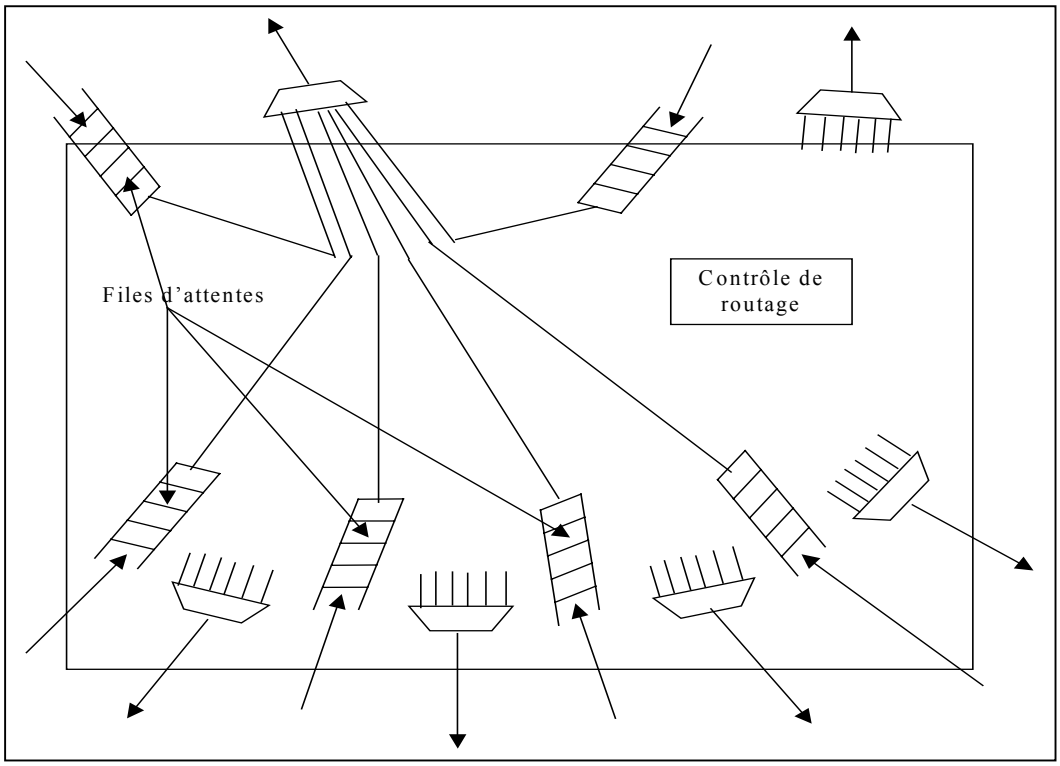


Figure 18: Synoptique du routeur BFT-NOC

3.2.2.2 Les ressources

Les ressources ont été modélisées comme processeur, mémoire, IP, DSP . .., aussi elles peuvent être à la fois source et destination (figure 19) dans lesquelles les paquets sont générés et consommés. On suppose que la taille de la file d'attente est infinie dans les ressources. Etant donné que chaque ressource (processeur, mémoire, IP, DSP …) est habilitée à traiter les données entrantes et sortantes à une grande vitesse, le risque de destruction de paquets dans une ressource est négligeable.

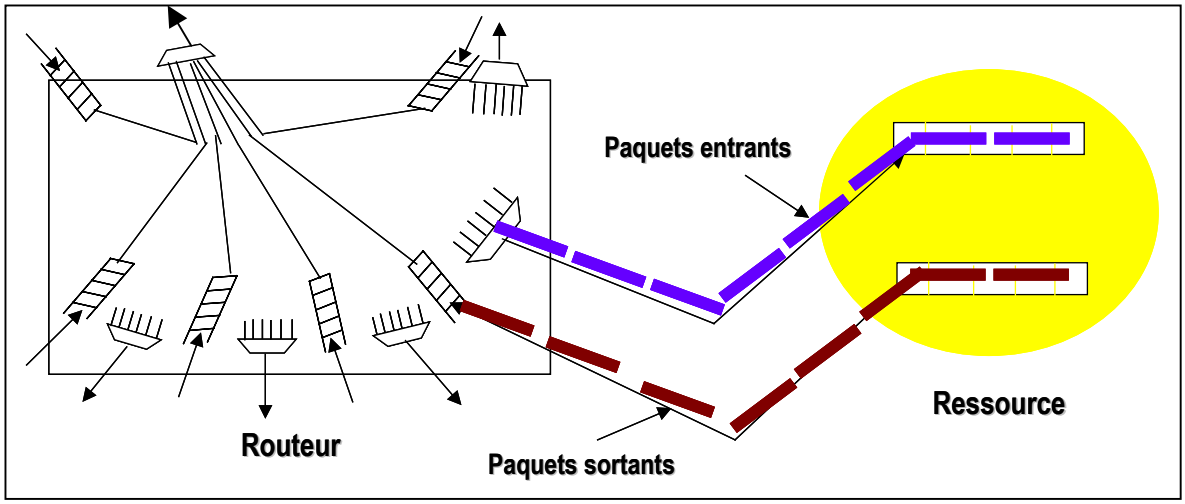


Figure 19: Liaison routeur-ressource

3.2.3 Communications inter-blocs

Dans cette section on présente les conditions requises pour gérer les communications inter-blocs.

- Les informations entre deux ressources sont acheminées en trois phases, de la ressource-source au routeur auquel elle est connectée, puis du routeur au routeur et en dernière phase du routeur à la ressource-cible.
- Les informations échangées sont découpées en paquets dont la taille est fixée à 8 octets, et pour une bande passante maximale de 8Gbits/s .
- Notre routeur (*dropTail router*) adopte le mécanisme FIFO pour la gestion de la file d'attente, en outre, le routeur détruit automatiquement les paquets arrivant et dépassant la capacité de la file d'attente.
- Deux stratégies de routage ont été adoptées pour notre modèle. La première est statique, basée sur un algorithme qui détermine le chemin le plus court entre la source et la destination, la table de routage est mise à jour une seule fois au début de la simulation. La deuxième stratégie de routage est dynamique, basée sur un algorithme modifiant la table de routage dynamiquement durant la simulation. Les avantages primordiaux de ces stratégies sont la simplicité, la tolérance aux fautes, et le contrôle des propriétés physiques (surface, bande passante, débit, latence,…) par l'utilisation optimale des mémoires et des délais d'interconnection.

- Etant donné que NS_2 est un simulateur à événement discret, notre modèle est piloté par événement en utilisant des déclencheurs de début et de fin de simulation.
- Au niveau du contrôle de la transmission entre équipements terminaux (ressources), nous avons opté pour le protocole UDP (*User Datagram Protocol*) opérant en mode non connecté, c'est à dire les paquets sont acheminés à travers le réseau sans garantie d'arrivée (pas d'acquittements), sans contrôle de flux, de duplication et pas de récupération d'erreurs. Ce choix est déterminé par la simplicité du protocole par rapport à TCP (transmission control protocol), et par l'optimisation de la taille de la file d'attente afin de minimiser la destruction des paquets.
- La génération du trafic est pilotée par une application produisant un trafic selon une distribution exponentielle (figure 20). Des paquets de taille fixe sont émis à travers le réseau durant une période opérationnelle (*burst_time*) suivant un débit binaire fixé à l'avance. Le trafic est bloqué durant une période de repos (*idle_time*).
- Le choix des sources est effectué suivant une distribution uniforme, pour notre modèle, toutes les ressources émettent des paquets selon une distribution exponentielle, au contraire les cibles sont choisies aléatoirement, en outre une ressource peut être une cible pour plusieurs sources à la fois.

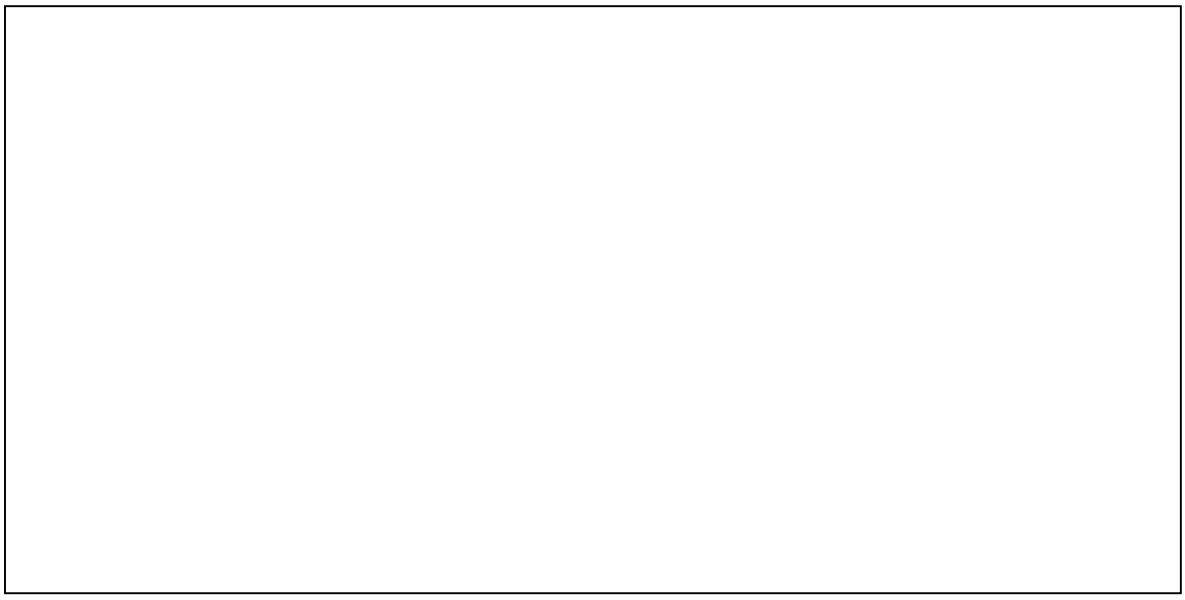


Figure 20: génération du trafic dans le réseau

Chapitre 4.

Conception et modélisation du réseau BFT_NOC dans NS-2

Cette étape consiste à implémenter le modèle BFT-NOC dans le simulateur NS-2, nous avons découpé le modèle en trois composants. Nous commençons par la conception de la topologie adoptée, dans la deuxième phase nous détaillons la gestion des communications inter-ressources composée du générateur de trafic du réseau, ainsi que l'algorithme adopté pour le choix des ressources. La dernière étape renferme des modules implémentés en vue d'évaluer notre modèle, entre autres les stratégies de routage, le moniteur de la file d'attente et l'analyseur de la bande passante.

4.1 Implémentation de la topologie

Nous introduisons dans cette étape le nombre de ressources utilisées dans notre modèle, ce nombre est fixé à **16 ressources**. Une ressource peut être à la fois source et destination dans laquelle les paquets sont générés et consommés. Le nombre de niveaux de la topologie adoptée est déterminé implicitement en fonction du nombre de ressources, en outre pour N=**16 ressources**, le nombre de niveaux **L est égal à $\log_4(16) = 2$**. Le nombre de routeurs est déduit implicitement en fonction du nombre de ressources et de niveaux, et ce par la fonction suivante : au **l** éme niveau (**de l=1 à $\log_4$ (N)**), on dispose de $\mathbf{N/2^{l+1}}$ routeurs. Pour notre modèle, nous implémentons au **premier** niveau **quatre** routeurs($\mathbf{16/2^{1+1}}$), et au **deuxième** niveau **deux** routeurs ($\mathbf{16/2^{2+1}}$), en outre le nombre de routeurs utilisés s'élève à **six**. Nous présentons dans l'exemple qui suit la syntaxe de création des ressources :

```
For {set i 0} {$i < $Nr} {incr i} {
Set n($i)   [$bftnoc node]
Set $n($i) shape circle
Set $n($i) color red
}
```

Nous définissons aussi, trois paramètres fondamentaux utilisés dans la conception de notre réseau sur puce, à savoir la bande passante maximale, le délai et la taille de FIFO.

- **La bande passante maximale** : détermine la capacité maximale du réseau en terme de bits ou paquets, cette valeur est exprimée en **MBits/s** La limitation du CPU a réduit la bande passante à **500MBits/s**
- **Le délai d'une liaison** : c'est le temps de transfert d'un paquet d'un nœud à un autre via une liaison en fil métallique. Ce **délai** est fixé à **0.1ms**.
- **La taille de FIFO** : c'est un paramètre critique pour les NOCs, il influe directement sur les performances des routeurs implémentés dans le modèle, le délai du message dépend aussi de la taille de FIFO. Plusieurs variantes de FIFOs sont implémentées et simulées en vu d'évaluer les performances de notre réseau sur puce ($2^{\mathbf{tf}}$ avec $\mathbf{tf} \in [0,4]$).

Nous illustrons un exemple qui définit ces trois paramètres, ainsi que leur utilisation pour la conception de la topologie :

```
Set bandepassante 500Mb
Set delailfil 0.1ms
Set capacitefifo 4
For {set i 0} {$i < $(Nr/2l+1 )} {incr i} {
For {set j 0} {$j < 4} {incr j} {
$nocbft duplex-link $n($j) $r($i) $bandepassante $delaifil Droptail
```

 # commentaires :

- ✓ nocbft : l'objet a simulé
- ✓ duplex-link : liaison en full_duplex(échanges de paquets dans les deux sens)
- ✓ $n($j) $r($i) : désigne respectivement une ressource et un routeur
- ✓ Droptail : le mécanisme de la gestion de la FIFO, tout paquet entrant, et n'ayant pas une place dans la FIFO est détruit automatiquement.

```
$nocbft queue-limit $n($j) $r($i) $capacitefifo
set monfifo(n($j)r($i)) [$nocbft monitor-queue $n($j) $r($i)  ]
```

 # commentaires :

- ✓ queue-limit : définit la capacité de FIFO de la liaison entre la ressource et le routeur
- ✓ monitor-queue : une pile renferme des données relatives à la FIFO(paquets entrants, paquets sortants, paquets détruits…)

```
}
```

4.2 La gestion des communications inter-ressources

Cette partie comporte deux modules implémentés, le premier décrit l'application qui génère le trafic au niveau des sources, le deuxième présente l'algorithme permettant le choix des sources et des destinations.

4.2.1 Générateur du trafic dans le réseau sur puce(BFT-NOC)

Nous choisissons une application qui génère le trafic au niveau des sources suivant une distribution exponentielle. Des paquets de taille fixe sont émis à travers le réseau durant une période opérationnelle (*burst_time*) et suivant un débit binaire fixé à l'avance, et le trafic est bloqué durant une période de repos (*idle_time*). Ce générateur du trafic qui est implémenté au niveau de la couche application est attaché au protocole UDP (*User Datagram Protocol*) utilisé par notre modèle, et ce protocole est attaché à un nœud source. De l'autre coté un agent LossMonitor est crée, cet agent est attaché à un nœud destination. Les deux protocoles sont connectés entre eux au niveau de la couche transport, en outre la connexion entre une source et une destination est établie implicitement.

Le comportement de ce générateur du trafic est défini par quatre paramètres :

- **débit** : la quantité d'informations envoyée pour une période donnée, une variante de débits a été simulée(100,120,150,250,350 et 450Mbits/s)
- **taille du paquet** : des données de taille fixe générées par l'application
- **burst_time** : la période opérationnelle durant laquelle les paquets sont générés
- **idle_time** : la période durant laquelle le trafic est bloqué

l'exemple cité ci-dessous présente les étapes d'implémentation d'un générateur du trafic :

```
Set taillepaquet 8
Set debit 450000k        # déclaration  des paramètres
Set burst  0.5s
Set idle 1ms
Proc application {noeud_source cible  taillepaquet   burst  idle   debit } {
Set bftnoc [Simulator instance]
Set UDP[new Agent/UDP]
$bftnoc attach-agent $noeud_source $udp
set expooTraffic [new Application/TrafficExponential]
$expooTraffic  set  packetSize_ $ taillepaquet
$expooTraffic  set  burst_time_ $burst
$expooTraffic  set  idle_time_ $idle
$expooTraffic  set  rate_ $debit
$expooTraffic  attach-agent UDP
$bftnoc connect $UDP $cible
return expooTraffic
}
```

```
#appel de la procédure application par une source
set source($i) [application $n($noeud_src) $cible($i) $taillepaquet  $burst  $idle  $debit ]
```

```
# creation d'un agent LossMonitor au niveau du nœud cible
for {set j 0} {j < $na} {incr j} {
set cible($i) [new Agent/LossMonitor]
}

# connexion du nœud destination à l'agent cible
$bftnoc attach-agent $n(noeud_dest) $cible($I)
```

4.2.2 Algorithme d'affectation des nœuds sources et cibles

Afin de mieux évaluer les performances de notre réseau sur puce baptisé BFT-NOC, nous avons appliqué à notre réseau une charge aléatoire uniformément répartie. Les sources sont connectées à une application qui génère des paquets suivant une distribution exponentielle, les cibles peuvent être destination pour plusieurs sources à la fois. En outre, l'algorithme proposé repose sur les hypothèses suivantes :

- toutes les applications connectées aux16 ressources génèrent des paquets.
- une cible est destination à un ou plusieurs sources
- 80% des cibles sont situées à quatre liaisons de la source.
- 20% des cibles sont situées à deux liaisons de la source.

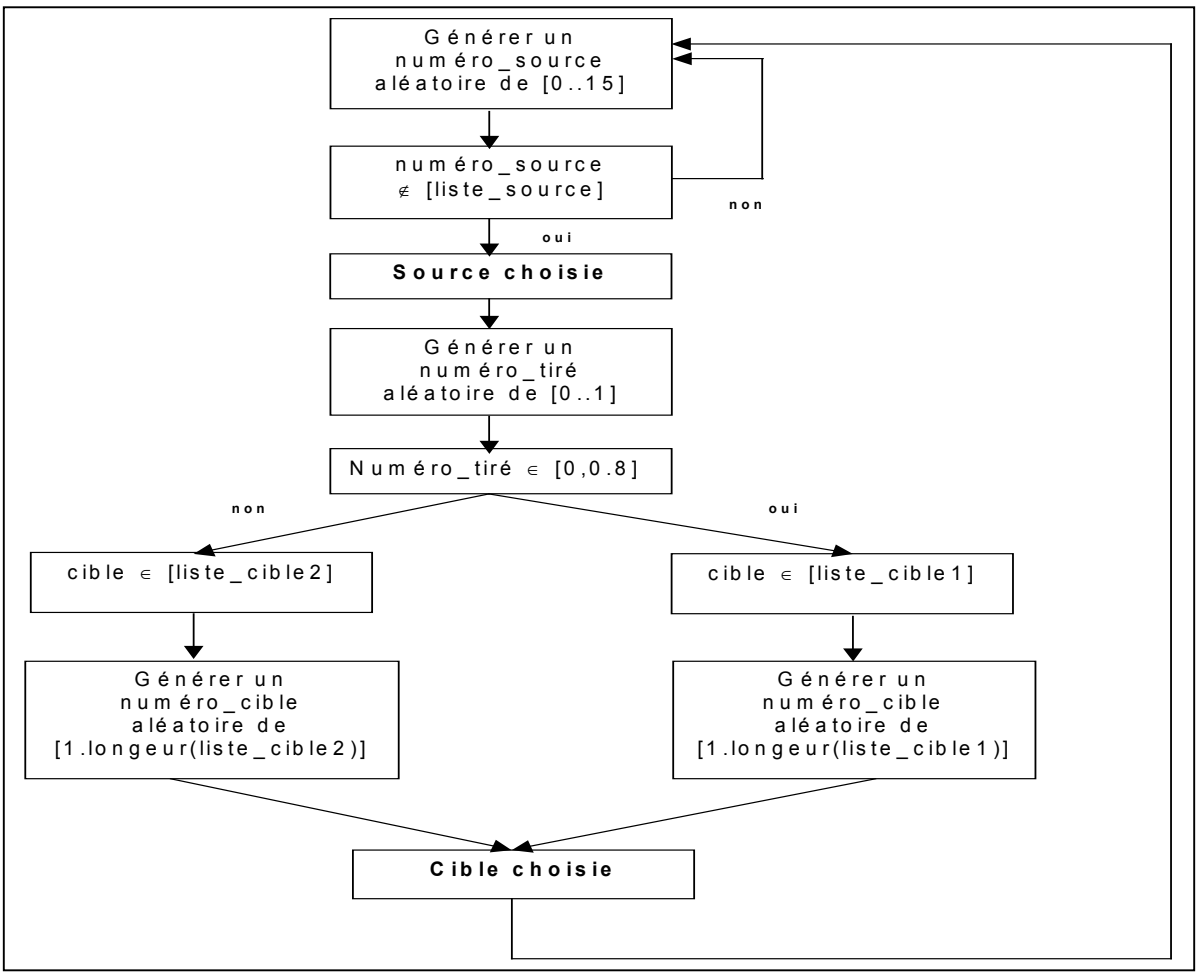
Quelques résultats d'affectation des sources et des cibles sont explicités dans le tableau suivant :

Scénario 1 :

```
noeud 0 est la  source   noeud 4 est la destination
noeud 1 est la  source   noeud 13 est la destination
noeud 2 est la  source   noeud 14 est la destination
noeud 3 est la  source   noeud 1 est la destination
noeud 4 est la  source   noeud 13 est la destination
noeud 5 est la  source   noeud 10 est la destination
noeud 6 est la  source   noeud 13 est la destination
noeud 7 est la  source   noeud 12 est la destination
noeud 8 est la  source   noeud 9 est la destination
noeud 9 est la  source   noeud 1 est la destination
noeud 10 est la  source   noeud 13 est la destination
noeud 11 est la  source   noeud 8 est la destination
noeud 12 est la  source   noeud 10 est la destination
noeud 13 est la  source   noeud 0 est la destination
noeud 14 est la  source   noeud 4 est la destination
```

noeud 15 est la source noeud 12 est la destination
Scénario 2 :
noeud 0 est la source noeud 4 est la destination noeud 1 est la source noeud 11 est la destination noeud 2 est la source noeud 10 est la destination noeud 3 est la source noeud 1 est la destination noeud 4 est la source noeud 13 est la destination noeud 5 est la source noeud 2 est la destination noeud 6 est la source noeud 15 est la destination noeud 7 est la source noeud 8 est la destination noeud 8 est la source noeud 14 est la destination noeud 9 est la source noeud 6 est la destination noeud 10 est la source noeud 11 est la destination noeud 11 est la source noeud 14 est la destination noeud 12 est la source noeud 9 est la destination noeud 13 est la source noeud 5 est la destination noeud 14 est la source noeud 2 est la destination noeud 15 est la source noeud 0 est la destination

L'organigramme suivant décrit les étapes de sélection des sources et cibles :



4.2.3 Stratégies et protocoles de routage

La performance globale du réseau dépend du comportement des routeurs, et de leurs interactions, entre autre le routage des paquets. Le simulateur NS-2 choisi offre plusieurs variantes de stratégies de routage, nous adoptons deux variantes pour notre modèle de réseau sur puce, un routage statique et un routage dynamique.

- Le routage **statique** : utilise l'algorithme **Dijkstra's all-pairs SPF**(*Shortest Path Forward)*, le chemin entre la source et la cible est calculé une seule fois et ce au début de la simulation, donc la table de routage est inéchangeable durant la période de simulation.
- Le routage **dynamique** : utilise un algorithme de routage distribué(***Distributed Bellman-Ford algorithm***), la table de routage est reconfigurable dynamiquement durant la simulation, cependant le chemin est géré par les routeurs, pour faire face à l'indisponibilité des ressources(liens, routeurs), et la possibilité d'acheminer les paquets à travers plusieurs chemins redondants.

Nous présentons dans l'exemple ci-dessous l'implémentation des stratégies de routage dans notre modèle :

```
Set bftnoc [Simulator instance]
# stratégie de routage dynamique avec utilisation de plusieurs chemins redondants
$bftnoc rtproto DV
For {set i 0} {$i < $(Nr/2^(l+1) )} {incr i} {
$r($i) set multipath_ 1
}

# stratégie de routage statique
$bftnoc rtproto Static
```

Une étude comparative des deux stratégies de routage adoptées est largement explicitée dans la partie analyse des performances du modèle BFT-NOC.

4.3 Implémentation des modules de contrôle de performances

Cette partie présente les modules implémentés en vue d'évaluer les performances de notre modèle BFT-NOC et plus précisément la pertinence de la modélisation du routeur. Nous détaillons les paramètres nécessaires pour le contrôle de simulation, le moniteur de la file d'attente et en dernière partie l'analyseur de la bande passante.

4.3.1 Contrôle d'événements

Nous avons choisi NS-2 pour simuler notre réseau sur puce, cet outil opère en événement discret, il a fallut définir des déclencheurs d'événement pour piloter notre modèle. Pour cela, trois paramètres ont été définis respectivement le **début**, la **fin** et la **période maximale** de la simulation. La taille des paquets et les débits adoptés nous ont imposé de limiter la durée totale de la simulation à **4ms**, par exemple pour un débit de **450Mbits/s**, et un paquet de **8octets**(64 bits), l'application génère environ **300000** paquets pour la période maximale adoptée. Cette charge importante appliquée est suffisante pour évaluer les performances de notre modèle.

4.3.2 Le moniteur de la file d'attente

Afin de déterminer la performance brute de notre modèle, nous avons implémenté un moniteur au niveau de chaque file d'attente. Ce moniteur renferme des données nécessaires pour l'évaluation de notre réseau, à savoir les **données entrantes**(paquets, octets), les **données sortantes**(paquets, octets), **les données détruites** (paquets, octets), **la taille maximale de la file d'attente**, **la moyenne d'occupation de la file d'attente**…. Ces données sont mises à jour périodiquement durant la période de simulation(**4ms**) suivant un intervalle fixé à **0.1ms**. Une fois la simulation est achevée, toutes les données concernant les files d'attente seront enregistrées dans un fichier en vue de les exploiter dans la partie analyse des performances du réseau sur puce BFT-NOC.

L'exemple cité ci-dessous illustre l'extraction des données a partir du moniteur de la file d'attente :

```
Set bftqad [open bftnoc.qad w]
Set bftnoc [Simulator instance]
Foreach {index valeur} {array get qmon} {
Set paquets [eval $value get-pkts-integrator]
Set moy_fifo_pkts [expr [$paquets set sum_]/$timemax]
Puts $bftqad " [eval $value set parrivals_] [eval $value set pdepartures_] [eval $value set pdrops_] $moy_fifo_pkts $max_fifo "
}
```

4.3.3 Analyseur de la bande passante

Le paramètre le plus souvent mesuré dans les simulations des réseaux sur puce est la **bande passante**. On la définit ainsi: c'est le nombre de paquets (en unités de Mbit/s) arrivant à une cible par seconde. Pour cela, nous avons implémenté un moniteur dans les **16 ressources** permettant d'enregistrer l'évolution de la bande passante de chaque destination durant la période de simulation, en captant des valeurs intégrées (paquets) à des intervalles de temps fixé à **0.1ms**.

Les étapes d'implémentation de l'analyseur de la bande passante sont décrites ci-dessous :

- avoir une instance du simulateur ;
- fixer l'intervalle du captage des paquets ;
- enregistrer les paquets reçus par chaque cible durant l'intervalle du temps ;
- calculer la bande passante ;
- remette à zéro le nombre de paquets reçus par chaque cible ;
- ré-exécuter la procédure.

Chapitre 5.

Analyse des performances du réseau BFT_NOC

5.1 Introduction

Ce chapitre présente des mesures de performances du réseau BFT_NOC. Le simulateur NS-2 a été utilisé pour implémenter et évaluer notre plate-forme d'interconnexion. Cet outil permet de simuler des modules contenant des descriptions en langage TCL (*Tool Command Langage*) du comportement du réseau. Grâce aux excellentes performances de NS-2, nous avons simuler un modèle composé de 16 ressources et de 6 routeurs. La durée moyenne d'une simulation est de 3 minutes environ, sur une station PC à 3 Ghz sous Linux.

Dans la première partie de cette section, nous analysons les performances du modèle BFT_NOC en termes de **bande passante**, **latence moyenne**, **taille de file d'attente** et **fiabilité**. Pour cela, deux **stratégies** de **routage** ont été **adoptées** et **comparées**. La première stratégie est statique, en outre les calculs sont faits une seule fois pour une seule simulation. La deuxième stratégie de routage est dynamique, basée sur un algorithme modifiant la table de routage dynamiquement durant la simulation.

Dans la deuxième partie, une étude comparative est faite entre notre modèle BFT_NOC et un réseau sur puce basé sur une topologie en Grille2D, tout en adoptant les paramètres technologiques qui ont été retenus dans la première partie.

5.2 Performance du réseau sur puce BFT_NOC

Afin de déterminer la performance de l'interconnexion et la pertinence de la modélisation du routeur, nous avons appliqué à notre modèle une charge aléatoire uniformément répartie : Un générateur de trafic produit des paquets suivant un débit moyen (paramétrable) dans un temps opérationnel (*burst_time*). Le trafic est bloqué dans une période de repos (*idle_t*ime). Les paramètres les plus souvent mesurés dans ces simulations sont : la bande passante maximale, la latence moyenne, la taille de la file d'attente et la charge moyenne du réseau. Tous ces paramètres ont été mesurés et commentés dans cette section.

5.2.1 Fiabilité du réseau

C'est le contrôle du taux de disponibilité du réseau sous des hypothèses pertinentes de charge. Deux paramètres ont été mesurés en vue d'évaluer cette disponibilité à savoir **la charge moyenne du réseau** et **le taux de paquets détruits**.

5.2.1.1 La charge moyenne du réseau

On la définit ainsi: C'est le rapport entre la charge réelle et la charge maximale du réseau. La charge réelle (**Cr**) est définie par le nombre des paquets reçus par les cibles durant la simulation. La charge maximale (**Cmax**) est définie par le nombre maximal de paquets transmis à travers les liens du réseau.

$$\mathbf{Cmoy}=\frac{Cr}{C\max}\ (1)$$

$$\mathbf{Cr}=\sum_{i=1}^{nl}(Paquets\_reçus)i\ \ (2)$$ et $(Paquets\_reçus)i$: le nombre de paquets reçus par une cible

$$\mathbf{Cmax}=\mathbf{T*\ 2*nl*\ (}\frac{Bw}{taille}\mathbf{)}\ (3)$$

T :durée de simulation (4ms) ; Bw :la bande passante (500Mb/s)

Taille :la taille du paquet (64bits) ; nl : le nombre de liens dans le réseau (24 liens).

$nl = \sum_{i=0}^{l}\frac{Nr}{x}$ avec $x=2^1$ et l :le nombre de niveaux dans le réseau, Nr :nombre de ressources.

Les figures 21 et 22 présentent les résultats de mesures de la charge moyenne de réseau pour les deux stratégies de routage adoptées. Nous constatons que la charge moyenne du réseau croît linéairement avec le débit pour l'intervalle [100 à 300Mb/s], et elle n'est pas sensitive à la taille de la file d'attente (les tableaux 1 et 2) . En plus, l'adoption du routage dynamique permet au réseau d'atteindre une charge moyenne plus importante : pour un débit de 250Mb/s et une FIFO de taille 8 paquets, cette charge s'élevé à 85% contre 58% pour la technique de routage statique.

	débit en Mb/s					
taille FIFO	**100**	**120**	**150**	**250**	**300**	**450**
2paquets	**0.313**	**0.343**	**0.417**	**0.576**	**0.544**	**0.515**
4paquets	**0.338**	**0.406**	**0.477**	**0.576**	**0.557**	**0.54**
8paquets	**0.298**	**0.406**	**0.477**	**0.576**	**0.595**	**0.487**

Tableau 1: : la charge moyenne du BFT_NOC pour un routage statique

	débit en Mb/s					
taille FIFO	100	120	150	250	300	450
2paquets	0.222	0.27	0.313	0.539	0.475	0.606
4paquets	0.339	0.407	0.508	0.847	0.87	0.925
8paquets	0.301	0.407	0.508	0.847	0.947	0.993

Tableau 2 : la charge moyenne du BFT_NOC pour un routage dynamique

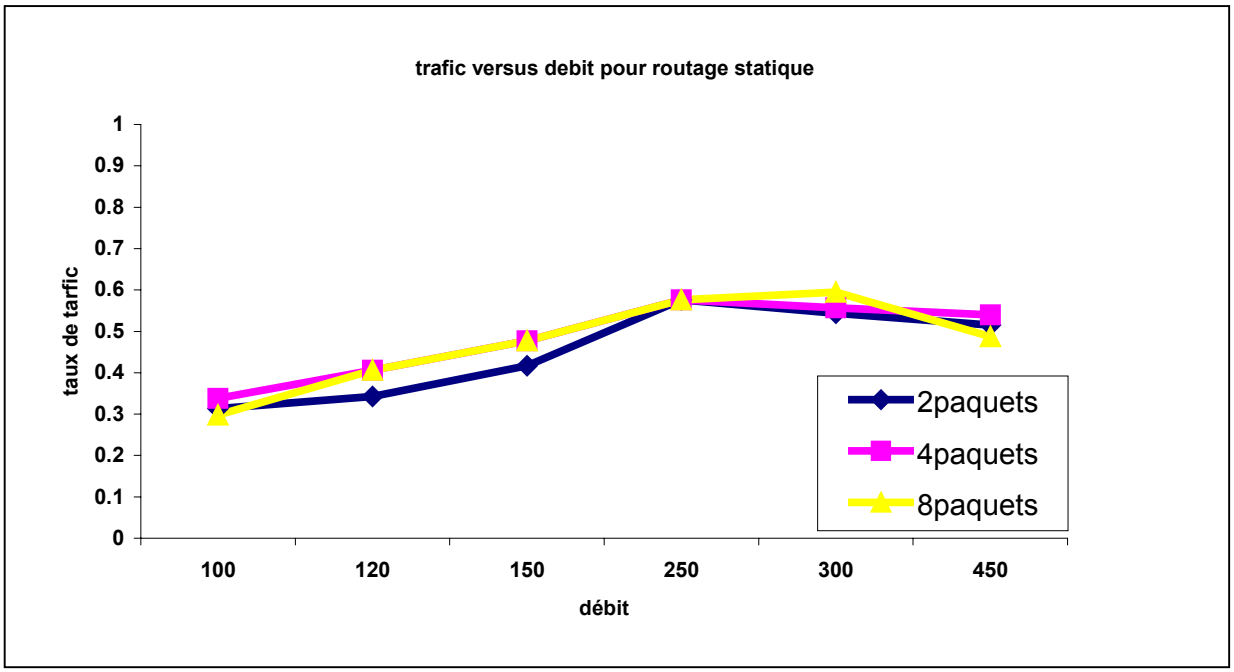


Figure 21: la charge moyenne du BFT_NOC pour un routage statique

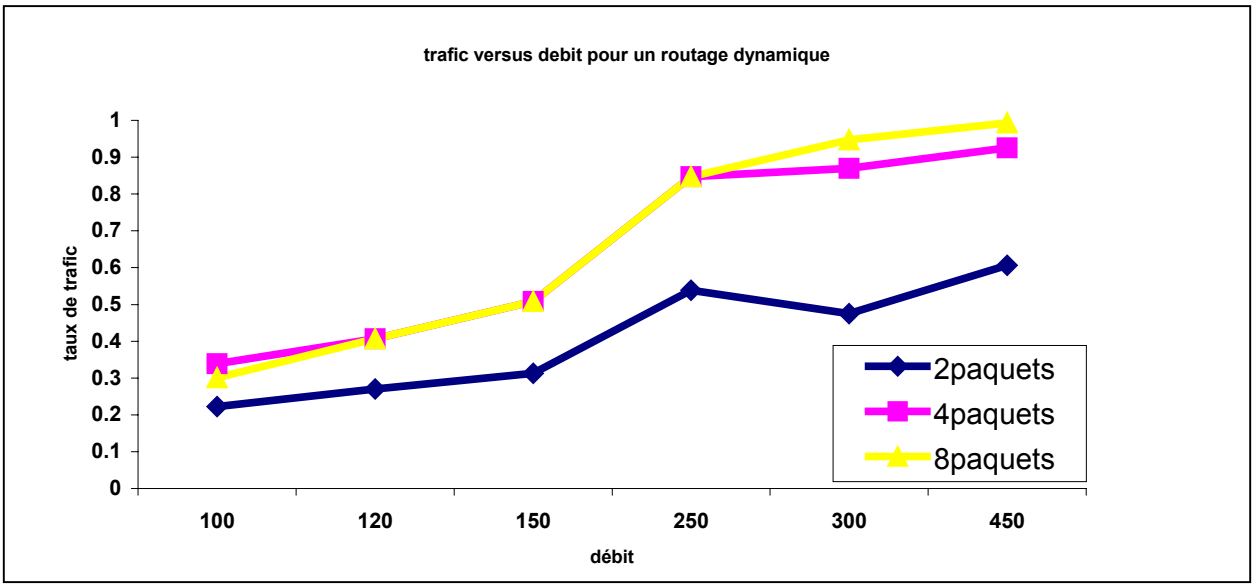


Figure 22: la charge moyenne du BFT_NOC pour un routage dynamique

5.2.1.2 Le taux de paquets détruits

On la définit ainsi: C'est le rapport entre le nombre des paquets détruits par les routeurs et le nombre total des paquets échangés à travers le réseau.

D'après les résultats figurant dans les tableaux 3 et 4, nous avons pu observer les constations suivantes :

- le taux de paquets détruits décroît quand la taille de file d'attente croît pour une variation de la FIFO de 2 à 8 paquets et un réseau peu chargé (figures 23 et 24).
- le taux de paquets détruits croît avec le débit pour des valeurs dépassant 250Mb/s et 120 Mb/s respectivement pour le routage dynamique et statique (figures 25 et 26).
- l'obtention d'une **bande passante maximale** de **250Mb/s** pour une taille de la file d'attente entre **4** et **8 paquets** pour une stratégie de routage **dynamique**.
- l'obtention d'une **bande passante maximale** de **120Mb/s** pour une taille de la file d'attente entre **4** et **8 paquets** pour une stratégie de routage **statique**.

	taille de la file d'attente		
débit en Mb/s	2paquets	4paquets	8paquets
100	**0.0328**	**0.0000**	**0.0000**
120	**0.0576**	**0.0000**	**0.0000**
150	**0.0344**	**0.0224**	**0.0223**
250	**0.1440**	**0.1440**	**0.1440**
300	**0.1770**	**0.1770**	**0.1770**
450	**0.2610**	**0.2610**	**0.2610**

Tableau 3 : le taux de paquets détruits pour un routage statique

	taille de la file d'attente		
débit en Mb/s	2paquets	4paquets	8paquets
100	**0.0053**	**0.0000**	**0.0000**
120	**0.0002**	**0.0000**	**0.0000**
150	**0.0437**	**0.0000**	**0.0000**
250	**0.0202**	**0.0000**	**0.0000**
300	**0.0320**	**0.0290**	**0.0290**
450	**0.1240**	**0.1120**	**0.1110**

Tableau 4 : le taux de paquets détruits pour un routage dynamique

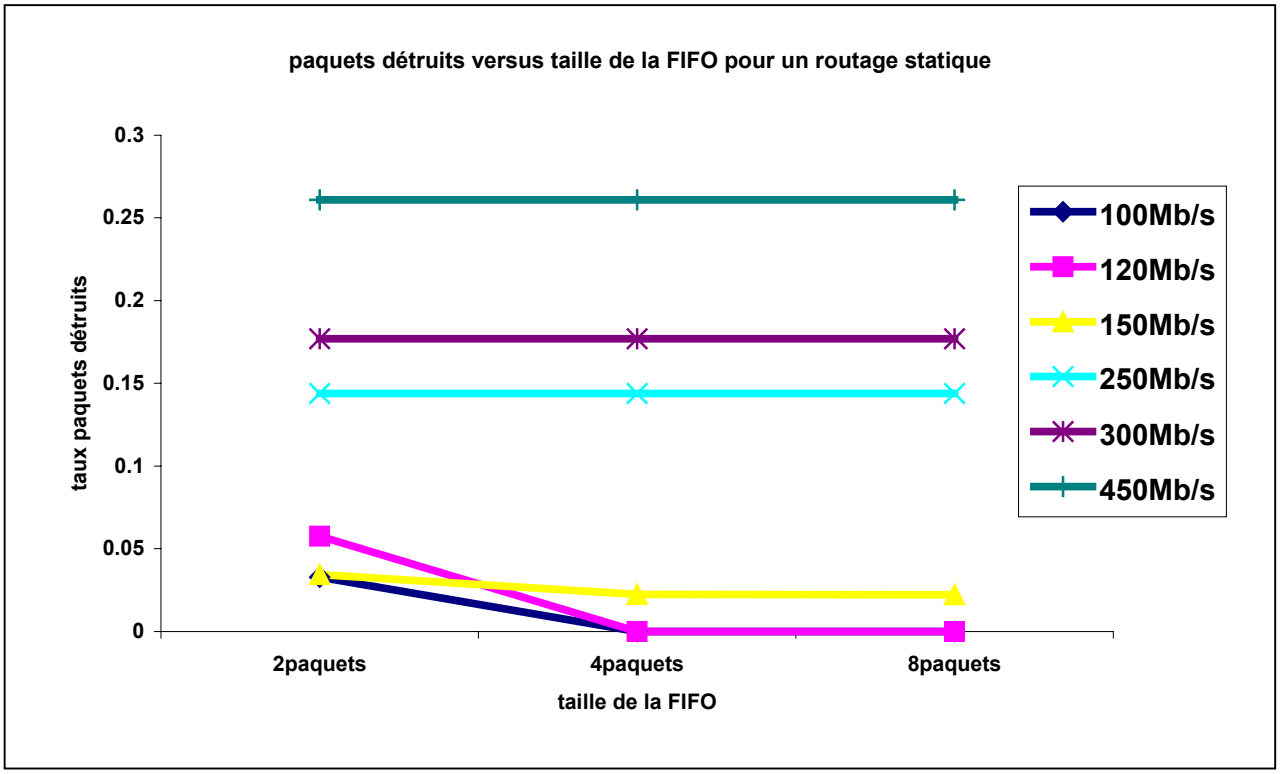


Figure 23: évaluation de la taille de la FIFO pour un routage statique

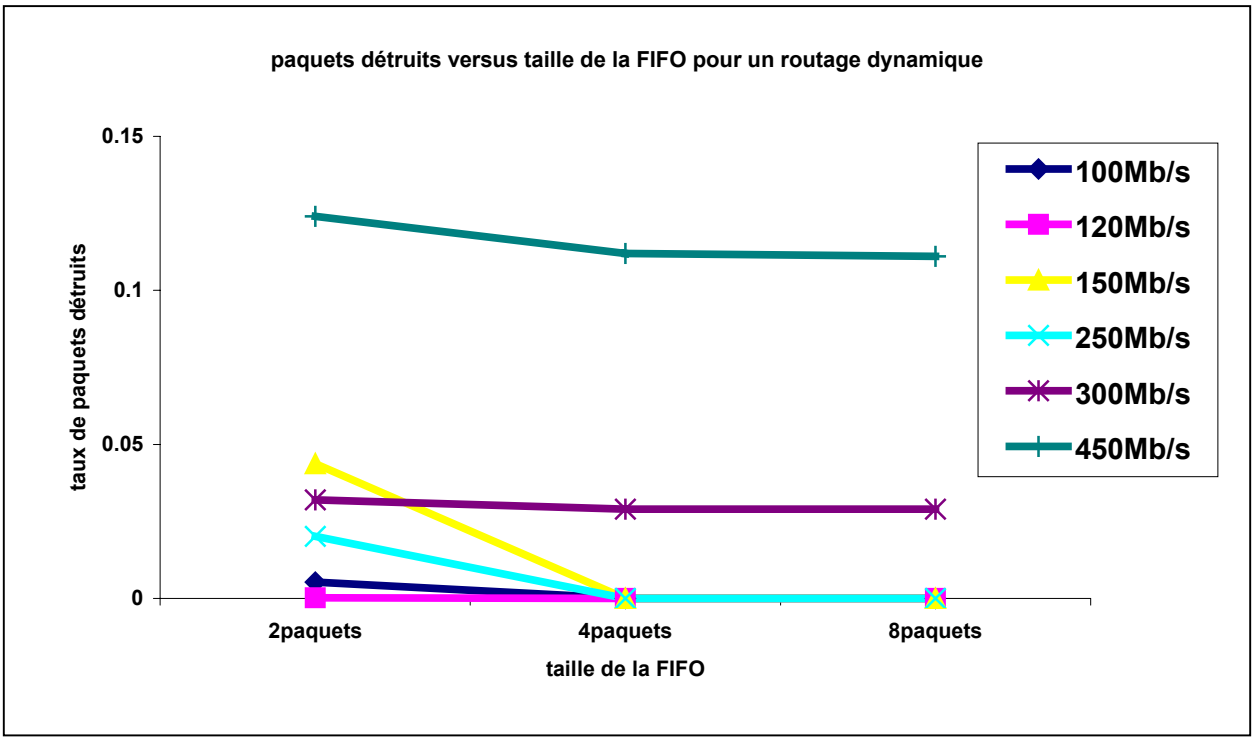


Figure 24 : évaluation de la taille de la FIFO pour un routage dynamique

Figure 25 :le taux de paquets détruits pour un routage statique

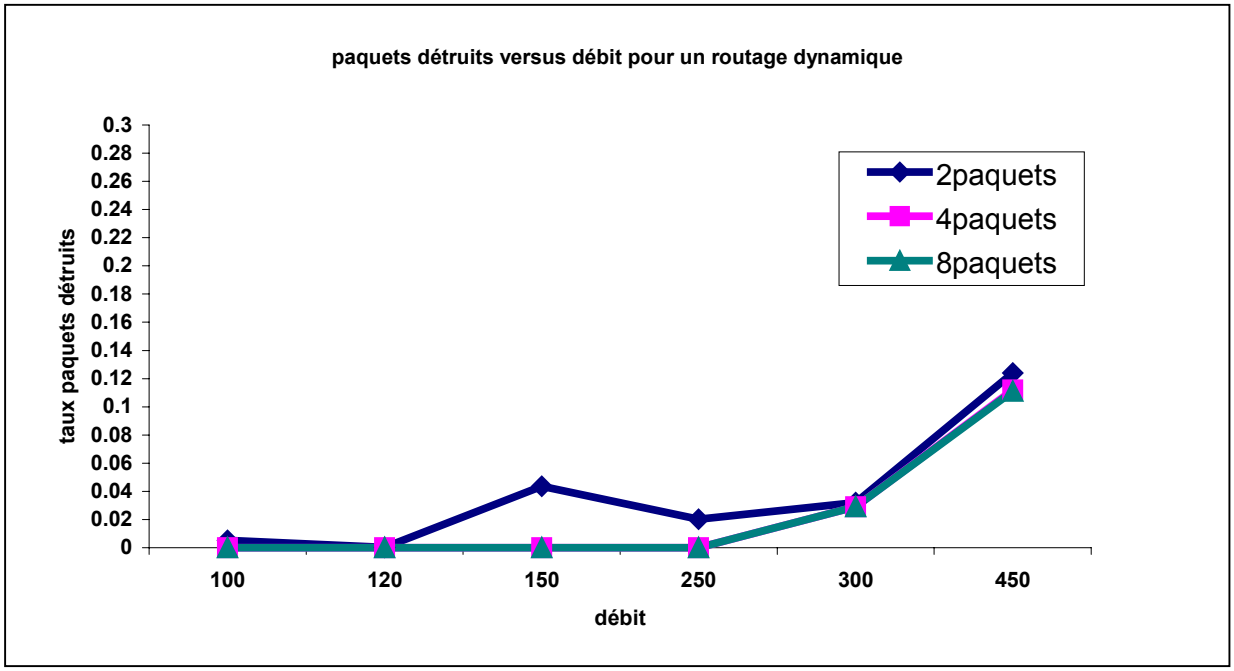


Figure 26: le taux de paquets détruits pour un routage dynamique

5.2.2 La bande passante maximale

Afin de déterminer la bande passante maximale du réseau, nous avons mesuré l'impact de débit et de la taille de la file d'attente sur le taux de paquets rejetés. Les résultats obtenus (figures 5.5 et 5.6) nous ont permis de dégager les constations suivantes :

- La bande passante maximale ne peut pas dépasser 250Mb/s pour une stratégie de routage dynamique et une file d'attente de taille 4 ou 8 paquets.
- La bande passante maximale ne peut pas dépasser 120Mb/s pour une stratégie de routage statique et une file d'attente de taille 4 ou 8 paquets.

5.2.3 La stratégie de mémorisation

La taille de la file d'attente est un paramètre fondamental dans la conception d'un routeur. Elle ne doit pas être de grande taille pour qu'elle n'occupe pas trop de surface sur la puce. Pour cela, nous avons adopté des FIFOs de taille variable. Les résultats obtenus (tableaux 1,2,3 et 4 ; figures 5.3 et 5.4) montrent l'impact de la taille de la file d'attente sur la fiabilité du réseau. En outre, une file d'attente de taille 8 paquets est la plus performante.

5.2.4 La latence moyenne

On la définit ainsi : C'est le temps de la transmission du paquet dans le routeur. Lorsque le réseau est peu chargé, la file d'attente est vide et la mesure de la latence moyenne (**Lm**) correspond au temps du transit du paquet dans le routeur (par exemple pour un paquet de taille 64bits et une bande passante de 500Mb/s, la latence moyenne(**Lm**)= 128ns). Si le réseau est trop saturé, les paquets s'accumulent dans la file d'attente et la latence moyenne dépend du délai de mémorisation du paquet dans la FIFO.

	taille de la file d'attente	
débit en Mb/s	4paquets	8paquets
100Mbs	**128**	**128**
120Mbs	**128**	**128**
150Mbs	**128**	**128**
250Mbs	**128**	**128**
300Mbs	**144.512**	**345.472**
450Mbs	**286.976**	**723.584**

Tableau 5 : latence de routage dynamique

	taille de la file d'attente	
débit en Mb/s	4paquets	8paquets
100Mbs	**128**	**128**
120Mbs	**128**	**128**
150Mbs	**144.512**	**342.912**
250Mbs	**271.104**	**707.84**
300Mbs	**322.56**	**805.632**
450Mbs	**334.848**	**818.176**

Tableau 6 : la latence de routage statique

Les tableaux 5 et 6 présentent les mesures de la latence moyenne du paquet dans le routeur en fonction de la charge de réseau et la taille de la file d'attente pour les différentes stratégies de routage adoptées.

Les figures 27 et 28 montrent l'évolution de la latence moyenne pour les deux tailles de la file d'attente. Pour un réseau trop chargé, la latence moyenne est très sensitive à la taille de la file d'attente, ceci est du au délai de mémorisation des paquets dans les FIFOs.

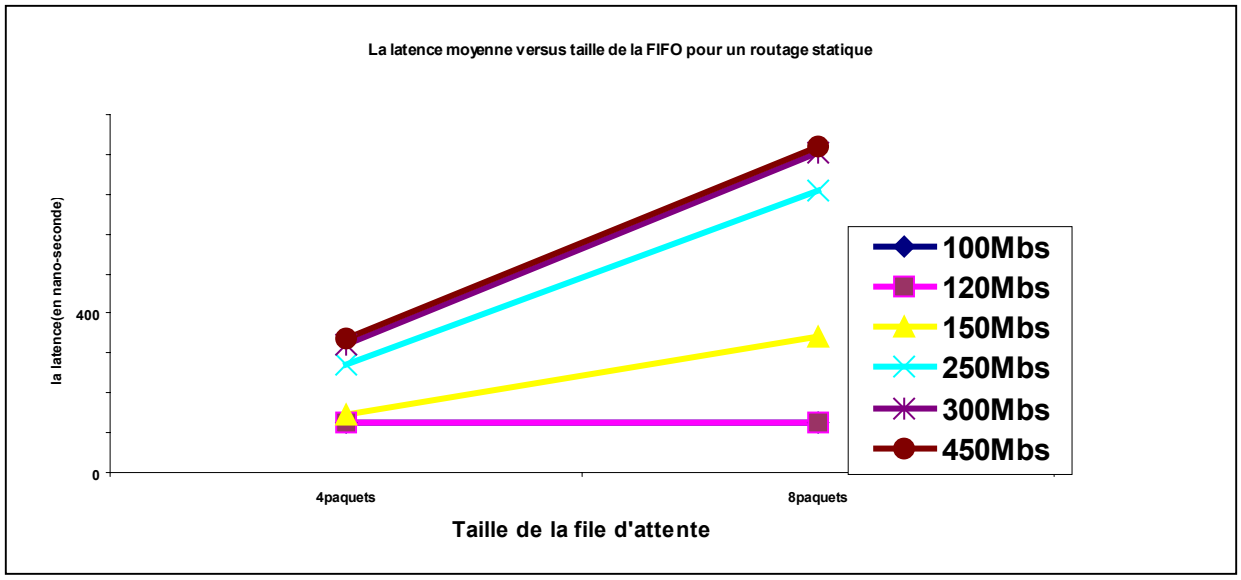


Figure 27: évaluation de la latence par rapport à la taille de la FIFO pour un routage statique

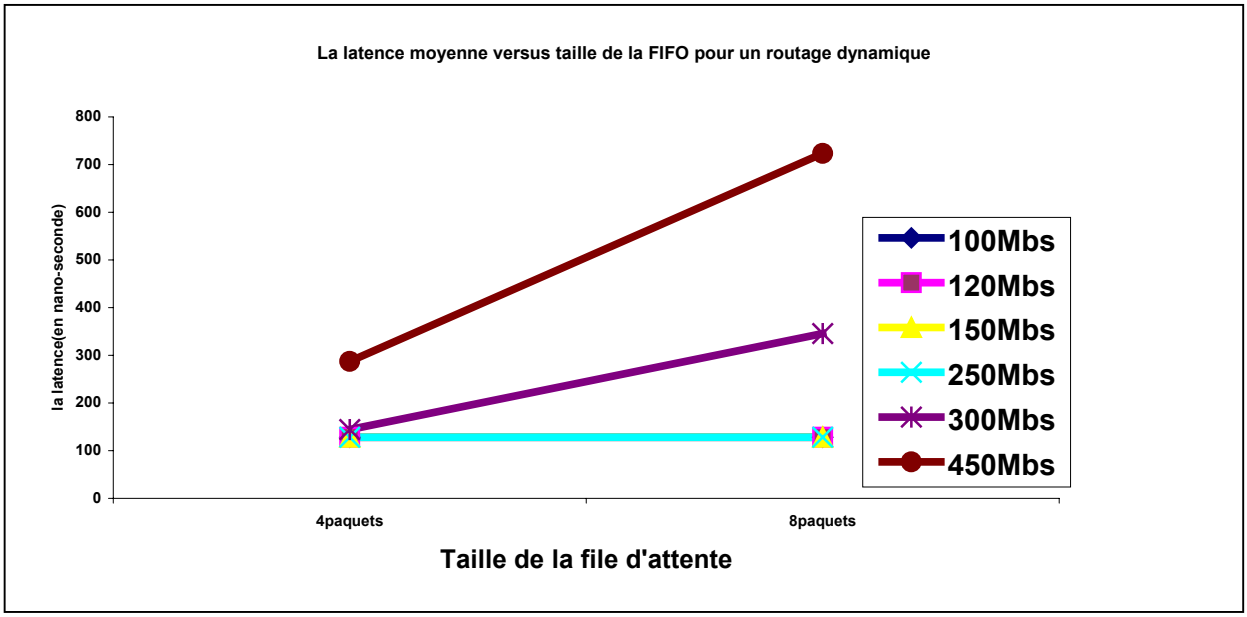


Figure 28 : évaluation de la latence par rapport la taille de la FIFO pour un routage dynamique

Les figures 29 et 30 présentent la latence moyenne en fonction de la charge du réseau. Nous constatons qu'un réseau légèrement chargé offre une latence moyenne constante pour les deux stratégies de routage adoptées. Toutefois, si le débit dépasse la capacité maximale du réseau, on constate une augmentation phénoménale de la latence due au délai de mémorisation des paquets dans les files d'attente.

Figure 29: évaluation de la latence en fonction du débit pour un routage statique

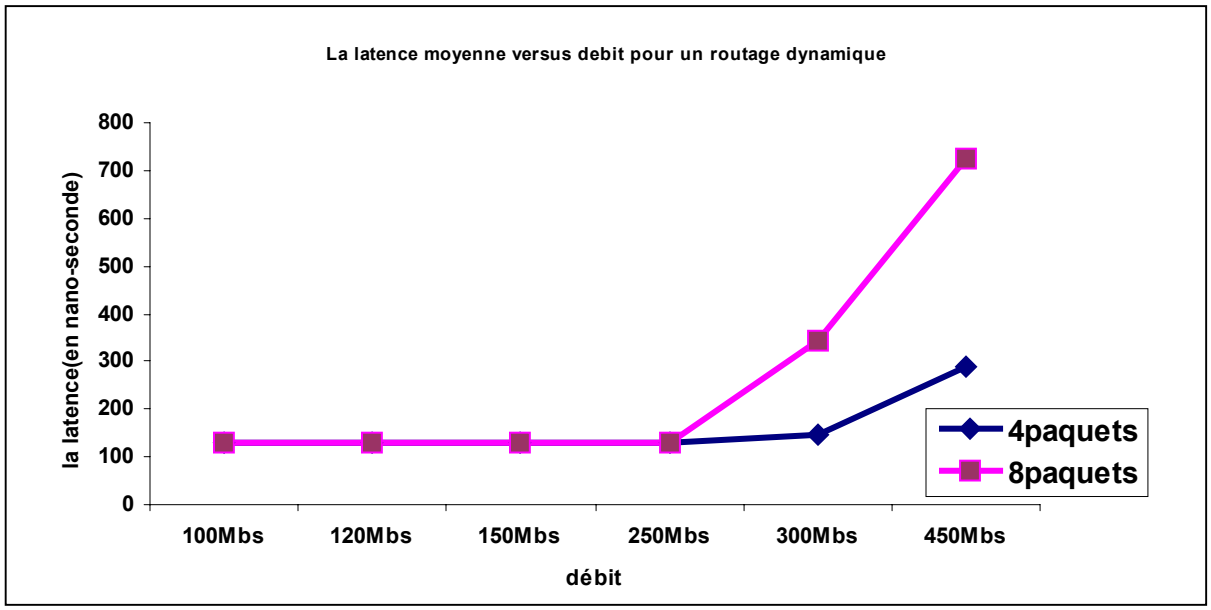


Figure 30 :évaluation de la latence en fonction du débit pour un routage dynamique

5.3 Analyse critique

Ces résultats appellent plusieurs critiques, notamment pour les deux stratégies de routage adoptées dans la simulation de notre modèle. Premièrement, le taux de disponibilité du réseau et de la bande passante maximale sont plus élevés pour le routage dynamique que pour le statique. Ce taux dépend de la charge du réseau. Deuxièmement, les meilleures latences moyennes sont obtenues pour un réseau peu chargé, mais si la débit dépasse la capacité du réseau, la latence moyenne du paquet est ralentie par la mémorisation des paquets dans les FIFOs. Enfin, la performance de l'interconnexion reste tributaire d'une bonne optimisation de la taille de la file d'attente.

Cependant, les résultats obtenus ont montré les performances de l'interconnexion du BFT_NOC, avec des réserves quant à la nécessité d'une bonne optimisation de la taille de la file d'attente et à l'utilité de l'adoption du routage dynamique pour l'acheminement des paquets à travers le réseau.

5.4 Etude comparative entre BFT_NOC et un NOC en Grille2D

Afin de mieux évaluer notre modèle à base de papillon en arbre élargi (figure31), il ne suffit pas de montrer les performances d'interconnexion du BFT_NOC, mais il faut également le comparer à une architecture concurrente. Pour cela, nous avons choisi d'implémenter un réseau sur puce à base d'une topologie en Grille2D (figure 32). Cette architecture interconnecte les 16 ressources à 16 routeurs. Ces derniers sont interconnectés entre eux en Grille2D. Nous avons adopté les paramètres technologiques qui ont été retenus dans l'analyse de la performance du BFT_NOC, notamment la taille de la file d'attente, les stratégies de routage et la charge appliquée dans le réseau.

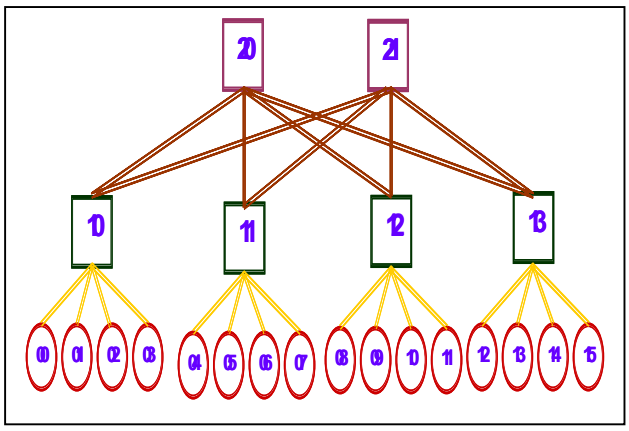


Figure 31: synoptique du BFT-NOC

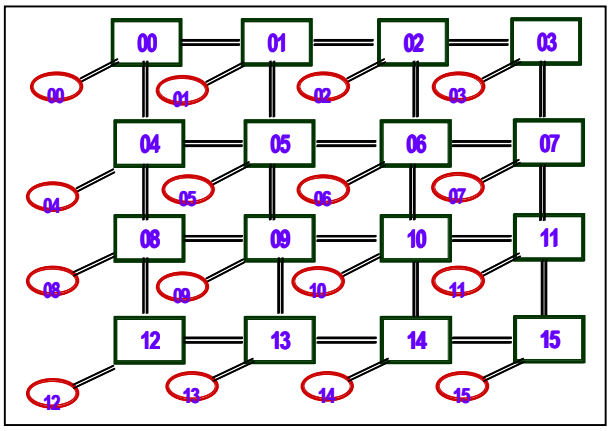


Figure 32 : synoptique du Grille2D-NOC

Les tableaux 7, 8 et 9 indiquent les paramètres mesurés, respectivement la charge moyenne du réseau, le taux de paquets détruits et la latence moyenne d'un paquet dans le réseau.

	architecture	
débit	BFT-NOC	MESH-NOC
150	0.301	0.372
200	0.508	0.468
250	0.847	0.58
300	0.947	0.659
450	0.993	0.948

Tableau 7 : la charge moyenne du réseau

	architecture	
débit	BFT-NOC	MESH-NOC
150	0	0
200	0	0.01
250	0	0.02
300	0.029	0.069
450	0.111	0.113

Tableau 8 : le taux de paquets détruits

	architecture	
débit	BFT-NOC	MECH-NOC
150	128	128
200	128	613.12
250	128	651.776
300	345.472	774.784
450	723.584	824.832

Tableau 9 :la latence moyenne du paquet

Le figure 33 présente l'impact du débit sur la charge moyenne du réseau pour les deux architectures. En outre, nous avons obtenu des valeurs de charge qui sont presque égales pour certains débits. En plus, nous constatons une augmentation brusque de la charge du modèle BFT_NOC à partir d'un débit de 250Mb/s.

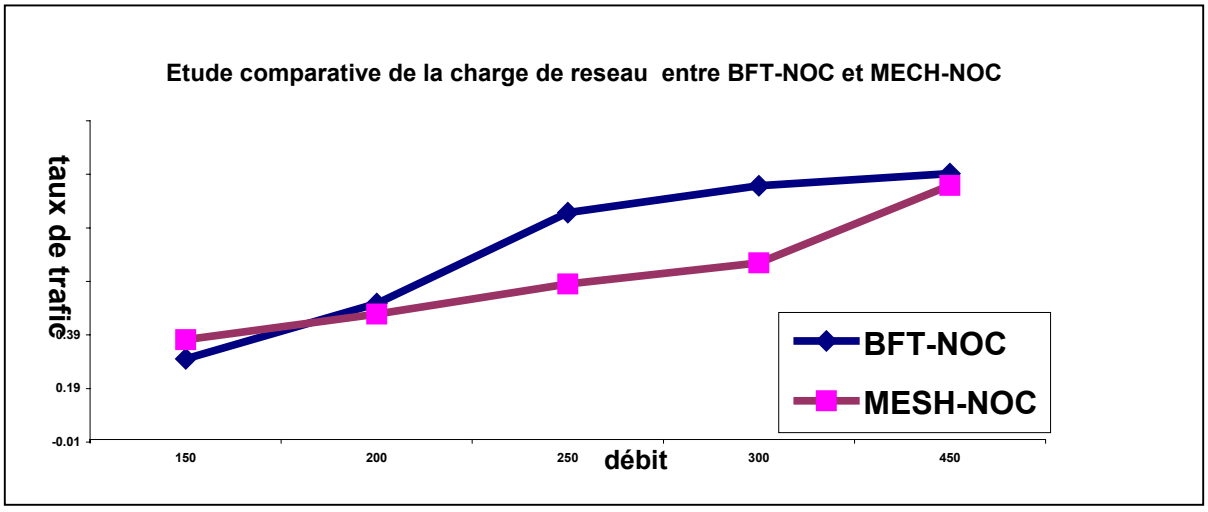


Figure 33 : évaluation de la charge moyenne du réseau pour les deux NOCs

La figure 34 montre que le taux de paquets détruits dépend de la charge du réseau pour les deux architectures. Le modèle BFT_NOC prétend à une bande passante maximale (autrement dit, le taux de paquets détruits reste nul) de 250Mb/s contre 150Mb/s pour le MESH_NOC.

Les valeurs de la latence moyenne d'un paquet dans le réseau (figure 35) nous ont permis de formuler les constations suivantes :

- La latence moyenne est sensitive à la taille de la file d'attente pour un débit dépassant la capacité maximale du réseau pour les deux architectures.
- La latence moyenne dépend du nombre de routeurs traversés.

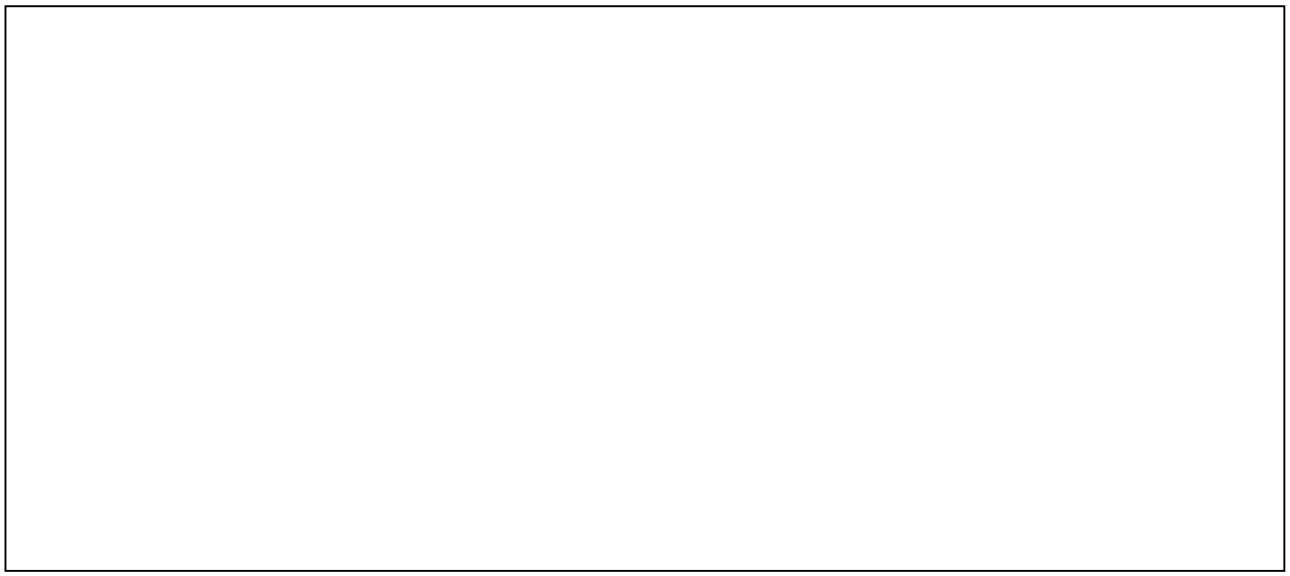


Figure 34 : évaluation du taux de paquets détruits pour les deux NOCs

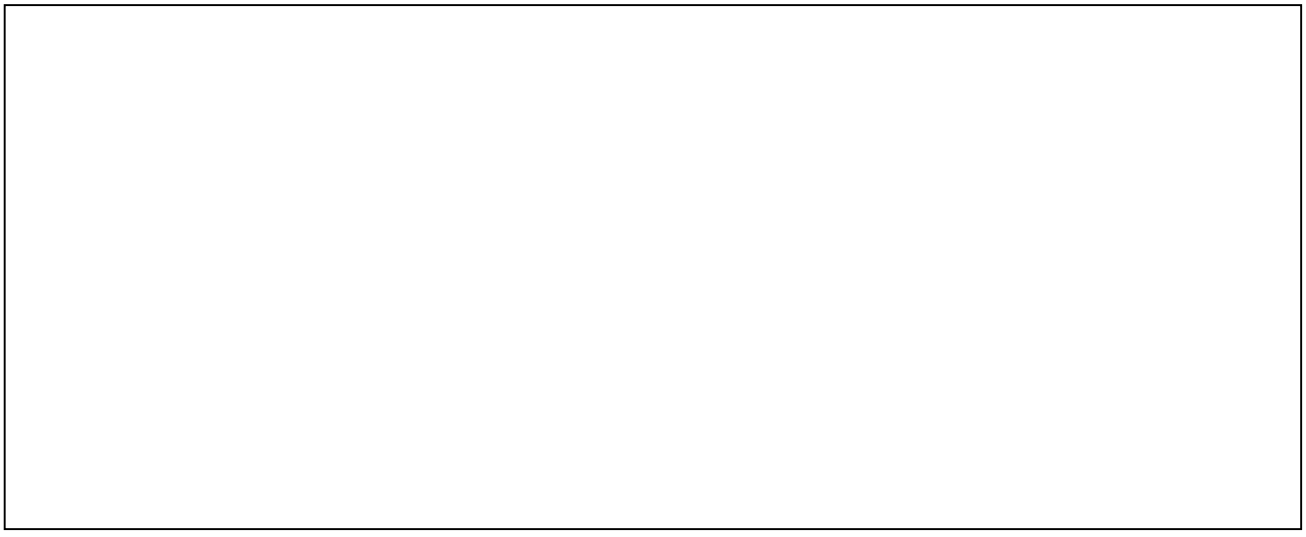


Figure 35 : évaluation de la latence moyenne pour les deux NOCs

Cette étude comparative a montré que BFT_NOC semble offrir une solution avantageuse par rapport à MESH_NOC. Le tableau 10 résume cette comparaison en termes de composants, liens, bande passante, fiabilité et latence moyenne.

	BFT-NOC	Grille 2D
Ressources	16	16
Routeurs	6	16
liens	24	40
Charge moyenne maximale	0.993	0.948
Fiabilité	☺	😐
Bande passante maximale	250Mb/s	150Mb/s
Latence moyenne maximale	723.584	824.832

Tableau 10 : Comparaison entre BFT_NOC et MESH_NOC

5.5 Conclusion

Ce chapitre analyse les performances du modèle BFT-NOC, que l'on a pu prédire grâce à une modélisation complète précise. Ces performances concernent des critères quantitatifs et qualitatifs minimaux que doit atteindre un réseau sur puce : la bande passante maximale, la latence, la fiabilité, la flexibilité, la consommation et la surface du réseau dans la puce. Les performances de l'interconnexion ont été concrétisées par une étude comparative entre BFT_NOC et une architecture concurrente en Grille2D.

Conclusion et Perspectives

Nous récapitulons ici les travaux de mastère et ses apports. Nous avons passé en revue les conceptions des réseaux sur puce. Bien que ces travaux restent théoriques jusqu'à nos jours, ils nous ont permis de déceler plusieurs besoins auxquels doit répondre une architecture d'interconnexion à haute performance d'une part, et des directives générales indispensables pour la conception d'un réseau sur puce fiable et flexible.

Notre architecture d'interconnexion baptisée BFT_NOC a été modélisée en modules contenant des descriptions en langage TCL du comportement du réseau, puis elle a été évaluée par simulation avec NS-2. La conception du modèle à été réalisée en trois phases. Nous avons commencé par l'implémentation de la topologie. Dans la deuxième phase nous avons détaillé la gestion des communications inter-ressources composée du générateur de trafic du réseau, ainsi que de l'algorithme adopté pour le choix des ressources. La dernière phase renferme des modules conçus en vue d'évaluer notre modèle, entre autres les stratégies de routage, le moniteur de la file d'attente et l'analyseur de la bande passante.

Grâce aux performances de NS-2, nous avons pu simuler un modèle composé de 16 ressources et 6 routeurs. les résultats obtenus ont montré les performances de l'interconnexion du BFT_NOC, avec des réserves sur la nécessité d'une bonne optimisation de la taille de la file d'attente, de la bande passante maximale et de l'utilité de l'adoption du routage dynamique pour l'acheminement des paquets à travers le réseau. Les performances de l'interconnexion ont été concrétisées par une étude comparative entre BFT_NOC et une architecture concurrente basée sur une topologie en Grille2D.

Nous énumérons ci-dessous les limitations et les extensions possibles du BFT_NOC qui méritent un complément d'investigation, et qui feront l'objet de futures recherches :

- Le coût élevé d'utilisation des protocoles, notamment IP (*Internet Protocol*) et UDP (*User Datagram Protocol*) pour la transmission des données à travers le réseau. En effet,

les en-têtes des protocoles cités (28 octets) ont été ajoutées aux données transmises (paquets de 8 octets), ce qui entraîne la diminution du rendement du réseau (22%).

- La limitation de stratégies de routage implantées dans le simulateur NS-2. Par conséquent le concepteur est tenu à ajouter des modules en vue d'implémenter de nouvelles techniques de routage telles que : *Wormhole routing* et *Hot Potato routing* .
- Le problème de synchronisation : toutes les communications dans le simulateur NS-2 sont contrôlées par des événements (Event Driven), en outre on ne peut pas prédire les aspects physiques (blocage, accès à la mémoire, etc..).
- L'évaluation des performances de l'architecture d'interconnexion est faite suite à une charge appliquée au réseau, elle peut estimer le comportement du réseau, mais elle n'est pas trop représentative des applications réelles.

Nous proposons deux scénarios pour palier à ces limitations. Le premier consiste à étendre le simulateur NS-2 par l'ajout de modules spécifiques aux architectures des réseaux sur puces. Le deuxième consiste à concevoir un simulateur de réseau sur puce, tout en tenant compte des performances du simulateur NS-2.

Références

[1] International Technology Roadmap for Semiconductors 2004 http://public.itrs.net/

[2] R. Lauwereins; "Creating a world of Smart Re-configurable Devices", Field Programmable Logic FPL'2002, pp790-794.

[3] A. Jantsch, J. Oberg, H. Tenhunen; "Special issue on Networks on chip", Journal of Systems Architecture (50) (2004), pp61-63

[4] W.O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, L. Gauthier, M. Diaz-Nava, A.A. Jerraya; "Multi-processor SoC platforms: a component-based design approach", IEEE Design and Test of Computers 19 (6) (2002),pp52–63

[5] R. Ho, K. Mai, M. Horowitz; "The future of wires", Proceedings of the IEEE, April (2001),pp490–504.

[6] L. Benini, G. De Micheli; "Networks on chips: a new SoC paradigm", IEEE Computer 35 (1) (2002),pp70–78.

[7] D.E. Culler, J.P. Singh, A. Gupta; "Parallel computer architecture: a hardware/software approach", Morgan Kaufmann Publishers, 1998.

[8] Ye, T.; Benini, L.; De Micheli: "Packetized On-Chip Interconnection Communication Analysis for MPSoC". In: Design Automation and Test in Europe (DATE'03), Mars 2003, pp344-349.

[9] P. Guerrier, A. Greiner; "A generic architecture for on-chip packet-switched interconnections" ;Proceedings of Design Automation and Test in Europe, 2000.

[10] H. Charlery, A. Greiner ; "SPIN,un micro-réseau d'interconnexion à commutation de paquets respectant la norme VCI ", Proceedings of SympAAA 2003.

[11] W. Dally, B. Toles; "Route packets, not wires: on-chip interconnection networks", Proceedings of the 38th Design Automation Conference, Juin 2001, pp684– 689.

[12] S. Kumar, A. Jantsch, J. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, A. Hemani; "A network on chip architecture and design methodology", Proceedings of IEEE Computer Society Annual Symposium on VLSI, Avril 2002, pp105–112.

[13] R. Thid, M. Millberg, A. Jantsch; "Evaluation NOC communication backbones with simulation", Proceedings IEEE NorChip conference, Novembre 2003, pp27-30.

[14] F. Karim, A. Nguyen, S. Dey; "On-chip communication architecture for OC-768 network processors", Proceedings of 38th Design Automation Conference, Juin 2001, pp678–683.

[15] Rijpkema, E.Goossens, K.Radulescu: "A. Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip". In: Design, Automation and Test in Europe (DATE'03), Mars 2003, pp350-355.

[16] I.Saastamoinen, M.Alho, J.Pirttimaki, J.Nurmi; "Proteo Interconnect IPs for Networks-on-Chip". In: IP Based SoC Design, Octobre 2002.

[17] D. Siguenza-Tortosa, J. Nurmi; "VHDL-Based Simulation Environment for Proteo NoC", Proceedings of the Seventh Annual IEEE International Workshop on High Level Design Validation and Test, Cannes,France,Octobre 2002, pp1-6

[18] F. Moraes, A. Mello, L. Meller, L. Ost, N. Calazans; "A Low Area Overhead Packet-switched Network on Chip: Architecture and Prototyping". In: IFIP Very Large Scale Integration (VLSI-SOC) 2003.

[19] M. Forsell; "A Scalable High-Performance Computing Solution for Networks on Chips" . IEEE Micro, Septembre. 2002, v 22(5), pp46-55.

[20] T. Ernst ; "Notes about network simulators" – INRIA – Sophia-Antipolis

[21] K. Fall, K. Varadhan; "The ns Manual", the VINT Project, Juin 2001, http://www.isi.edu/nsnam.

[22] R.I Greenberg., L. Guan. ; "An improved analytical model for wormhole routed networks with application to butterfly fat-trees", Proceedings of the 1997 International Conference on Parallel Processing, pp44 -48