COMMENT FONCTIONNENT LES SWITCHS ET ROUTEURS ?

Mewtow

17 février 2016

Zeste de savoir

Table des matières

1 Introduction

Souvent, les cours de réseau montrent comment fonctionnent les réseaux d'un point de vue conceptuel, sans vraiment indiquer comment cela se passe au niveau du matériel. Mais ce tutoriel va renverser la tendance : vous saurez tout sur votre matériel réseau.

2 Micro-architecture des switchs

Un switch est un matériel qui relie plusieurs composants électroniques ou informatiques entre eux, les interconnexions étant configurables. Il contient plusieurs ports d'entrées, sur lesquels on peut envoyer recevoir des paquets de données, et plusieurs ports de sorties, sur lesquels on peut envoyer des données. Dans certains cas, les ports d'entrée et de sortie sont confondus : un même port peut servir alternativement d'entrée et de sortie.

2.1 Switching

Le rôle du switch, c'est de faire en sorte que les informations en provenance d'un port soient envoyées sur un autre port, défini par la configuration du switch. Dans le cas le plus simple, on peut voir un Switch comme un ensemble de connections point à point configurables : le switch associe un port de sortie à chaque port d'entrée. Le composant qui relie ports d'entrée et ports de sortie est appelé la **switch fabric**.

Cette *switch fabric* peut s'implémenter de plusieurs manières, en utilisant :

— un réseau d'interconnexions configurable selon les besoins ;
— une mémoire ;
— un bus.

Les techniques de broadcast ou de multicast permettent d'envoyer une donnée présentée sur un port d'entrée sur plusieurs sorties : on peut envoyer un message identique à plusieurs ordinateurs en même temps, sans devoir envoyer plusieurs copies. Certains switchs permettent de gérer cela directement dans le matériel, soit en dupliquant les paquets, soit en connectant une entrée sur plusieurs sorties.

2.2 Arbitrage

Il arrive d'envoyer plusieurs paquets en même temps sur le même port de sortie, les deux paquets provenant d'entrées différentes. Dans ce cas, le switch essaye d'envoyer les paquets les uns après les autres, certains étant mis en attente (il arrive cependant que certains paquets soient perdus). De même, il arrive que la *switch fabric* soit relativement lente comparé au temps d'envoi et de réception d'un paquet : ceux-ci doivent être mis en attente.

Dans les deux cas, qui dit mise en attente dit : utilisation de mémoires tampons pour stocker les paquets mis en attente. Dans le cas d'un switch/routeur, les mémoires tampons sont des mémoires FIFO, histoire de conserver l'ordre d'arrivée et d'envoi des paquets. Si les FIFO sont remplies, les paquets en trop sont perdus et n'arrivent pas à destination : on laisse la situation entre les mains du logiciel.

2.2.1 Output Buffering

Si on met plus de temps à envoyer des paquets sur le réseau qu'à les router via la *switch fabric*, on risque de perdre des paquets : certains paquets arriveront sur les ports de sortie alors que l'envoi du paquet précédent n'est pas terminé. On peut éviter cela en mettant les FIFO entre la *switch fabric* et les ports de sortie : on parle d'***output buffering***.

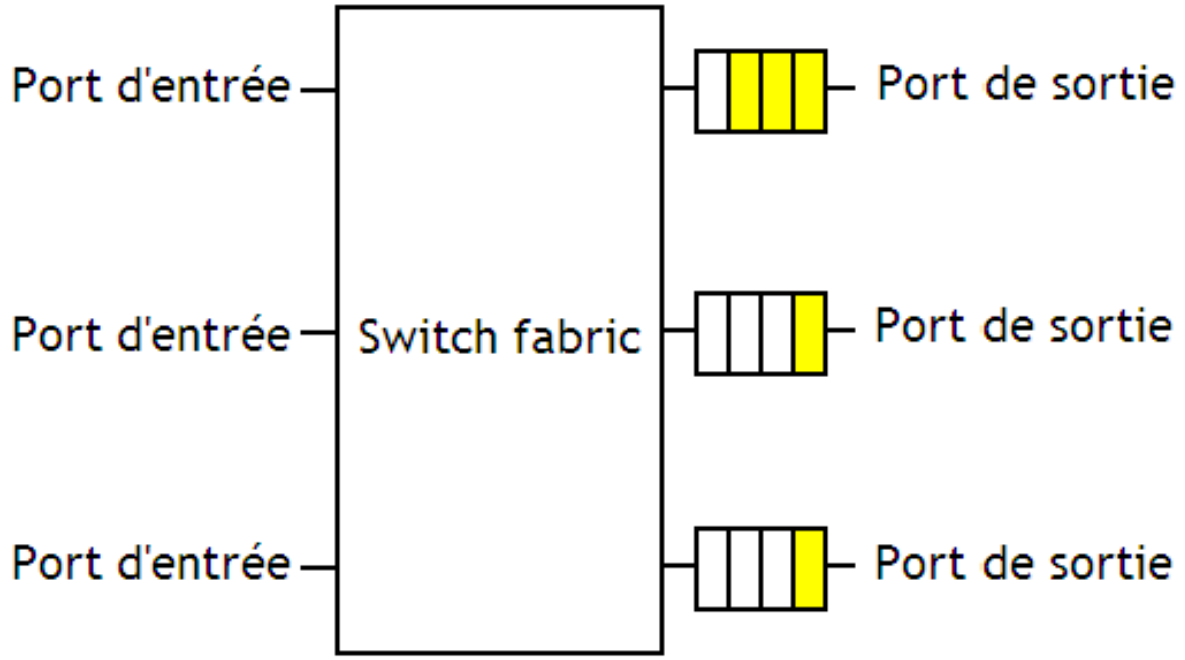

Figure 2.1 – Output Buffering

2.2.2 Input Buffering

Il arrive que l'on se trouve face à la situation inverse : le temps de router un paquet, plusieurs paquets sont arrivés sur les ports d'entrée. Pour éviter cela, il faut mettre les FIFO sur les entrées : on parle d'**input buffering**. Avec cette organisation, on est obligé d'utiliser un algorithme d'ordonnancement pour décider quelle FIFO traiter en priorité. La taille optimale des FIFO dépend de la qualité de l'algorithme d'ordonnancement.

Mais si une requête est en attente, elle va bloquer les requêtes suivantes sur le port d'entrée, même si celles-ci ont un port de sortie différent : on parle d'**head of line blocking**.

Pour éliminer l'*head of line blocking*, certains switchs utilisent une variante de l'*input buffering* : ***Virtual output queing***. Avec cette méthode, chaque FIFO est découpée en sous-FIFO, chacune prenant en charge les paquets destinés à un port de sortie précis. Avec cette technique, on peut traiter les requêtes dans le désordre, afin de profiter au maximum des ports libres.

2.2.3 Input-output Buffering

Et enfin, dernière solution : mettre des FIFO aussi bien sur les ports d'entrée que sur les ports de sortie. Il est même possible d'utiliser le *virtual output queuing*. Dans tous les cas, les algorithmes d'ordonnancement de ces switchs sont particulièrement complexes.

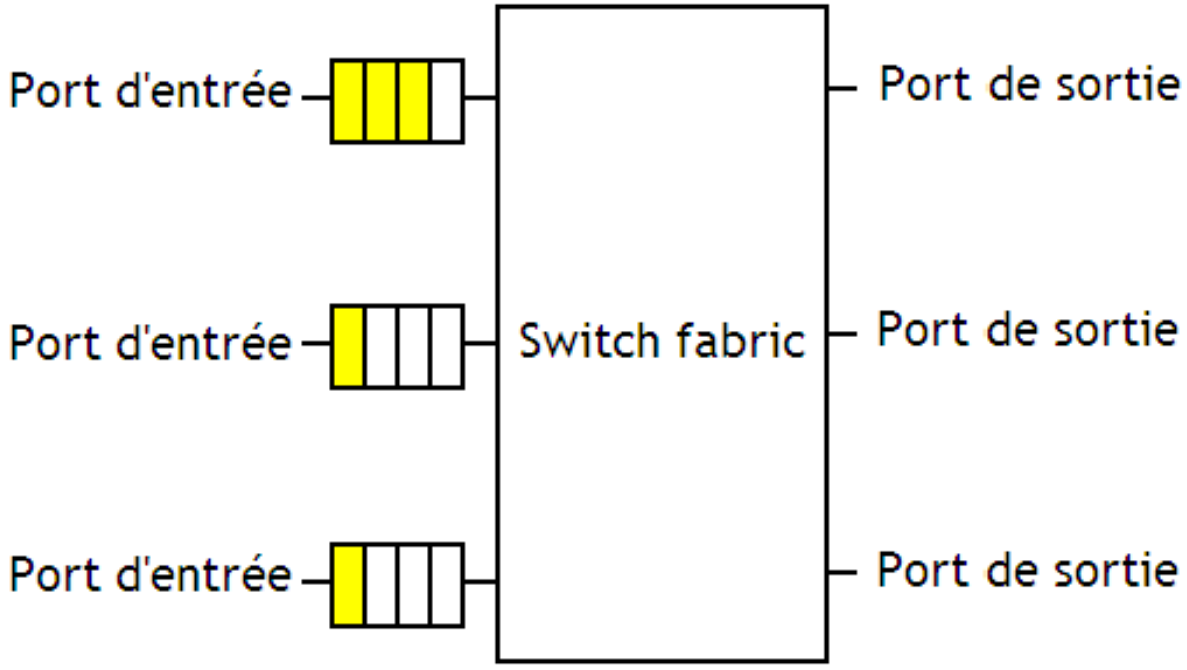

Figure 2.2 – Input Buffering

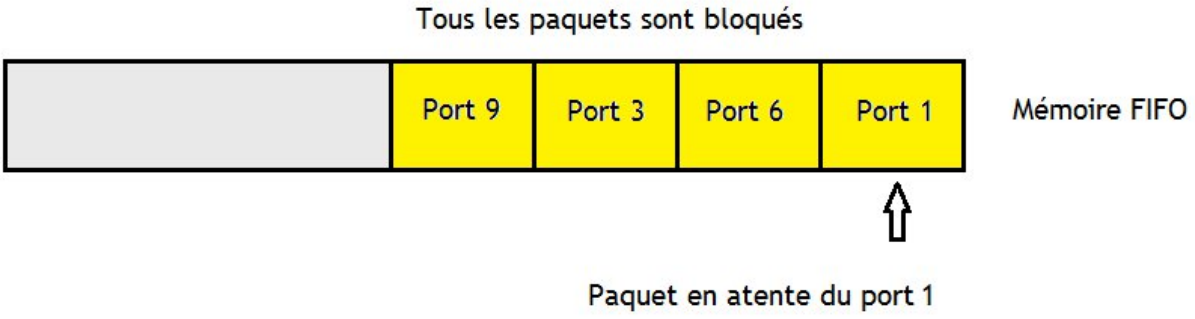

Figure 2.3 – Head of line blocking

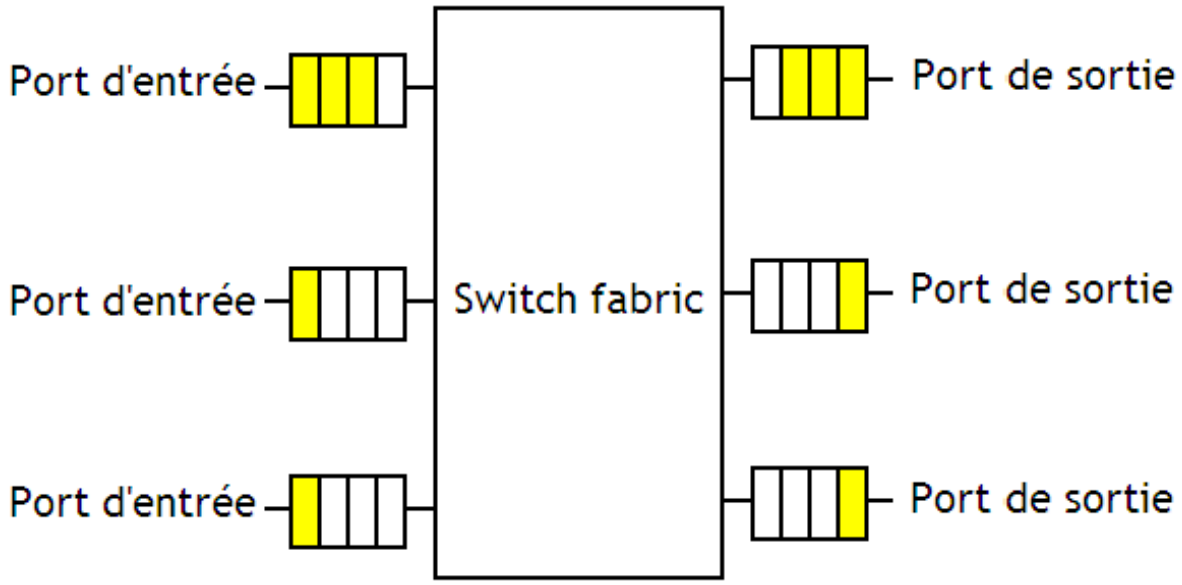

Figure 2.4 – Input-Output Buffering

2.2.4 Buffering interne à la *switch fabric*

Et enfin, certains swicths intègrent les FIFO dans la *switch fabric*.

2.3 Routage

Une *switch fabric* ne suffit pas à créer un switch ou un routeur. En effet, le switch ou le routeur doit déterminer le port de destination en fonction de l'adresse MAC ou IP du paquet. La *switch fabric* doit être secondée par deux circuits :

- un circuit qui va détecter les headers IP ou ARP, et les interpréter : c'est l'**interpréteur de paquet** ;
- et un circuit qui va déterminer le port de destination : le ***forwarding engine***.

L'intérieur d'un switch ressemble donc à ceci (les FIFO d'arbitrage sont intégrées au réseau d'interconnexion) :

Le switch est un matériel qui utilise des adresses MAC, qui servent à identifier des cartes/équipements réseau (et non des ordinateurs, comme les adresses IP) : chaque paquet indique l'adresse MAC du destinataire et de l'émetteur. Pour les routeurs, le routage utilise des adresses IP .

À l'intérieur du *forwarding engine*, on trouve des tables de correspondance qui associent chaque adresse à un port : la **CAM table** pour les adresses MAC, et la table de routage pour les IP. Cette table de correspondance peut être implémenté de deux manières :

- avec un automate, une machine à états finie matérielle ;
- avec une mémoire RAM ou associative.

[[information]] | La dernière solution est la plus utilisée, même si des solutions hybrides sont aussi relativement courantes (on peut notamment citer l'algorithme de routage nommé *Logic-Based Distributed Routing*).

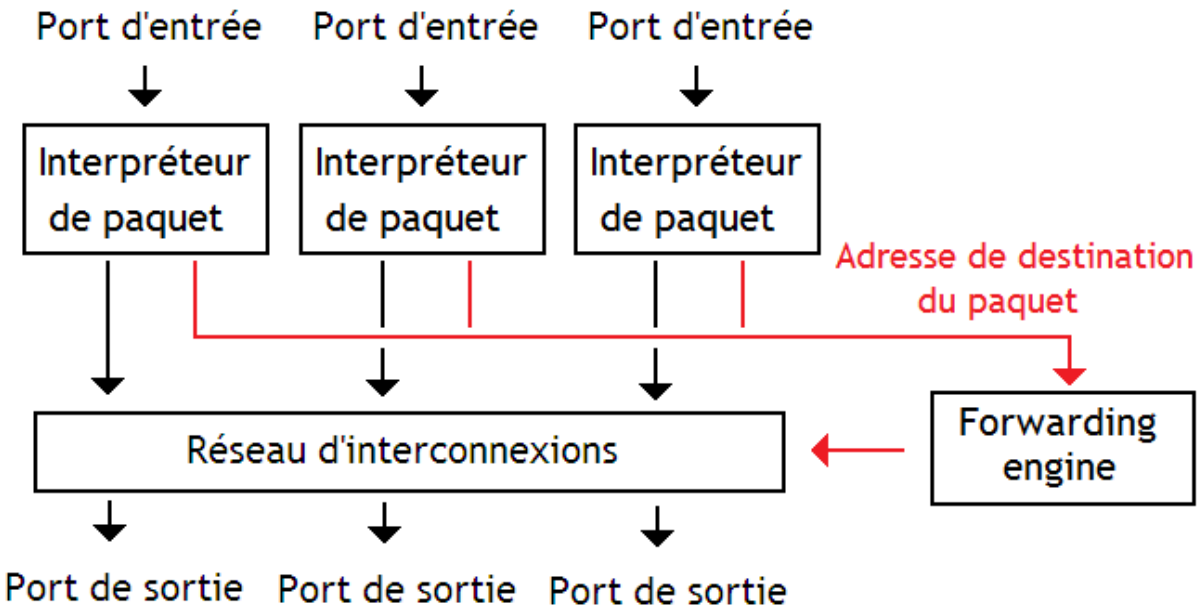

Figure 2.5 – Micro-architecture complète d'un switch/routeur

Pour résumer, un switch ou routeur est composé de plusieurs circuits :

- des circuits pour mettre en attente les paquets reçus et gérer l'arbitrage (FIFO et circuits de sérialisation/dé-sérialisation) ;
- un circuit qui extrait les informations utiles des paquets à transmettre, et qui prend en charge l'extraction de l'adresse de destination ;
- un circuit dédié au routage : le *forwarding engine* ;
- une switch fabric.

Il faut noter que le switch peut être pipeliné, pour gagner en performance : il suffit d'insérer des mémoires tampons entre les différents circuits cités au-dessus.

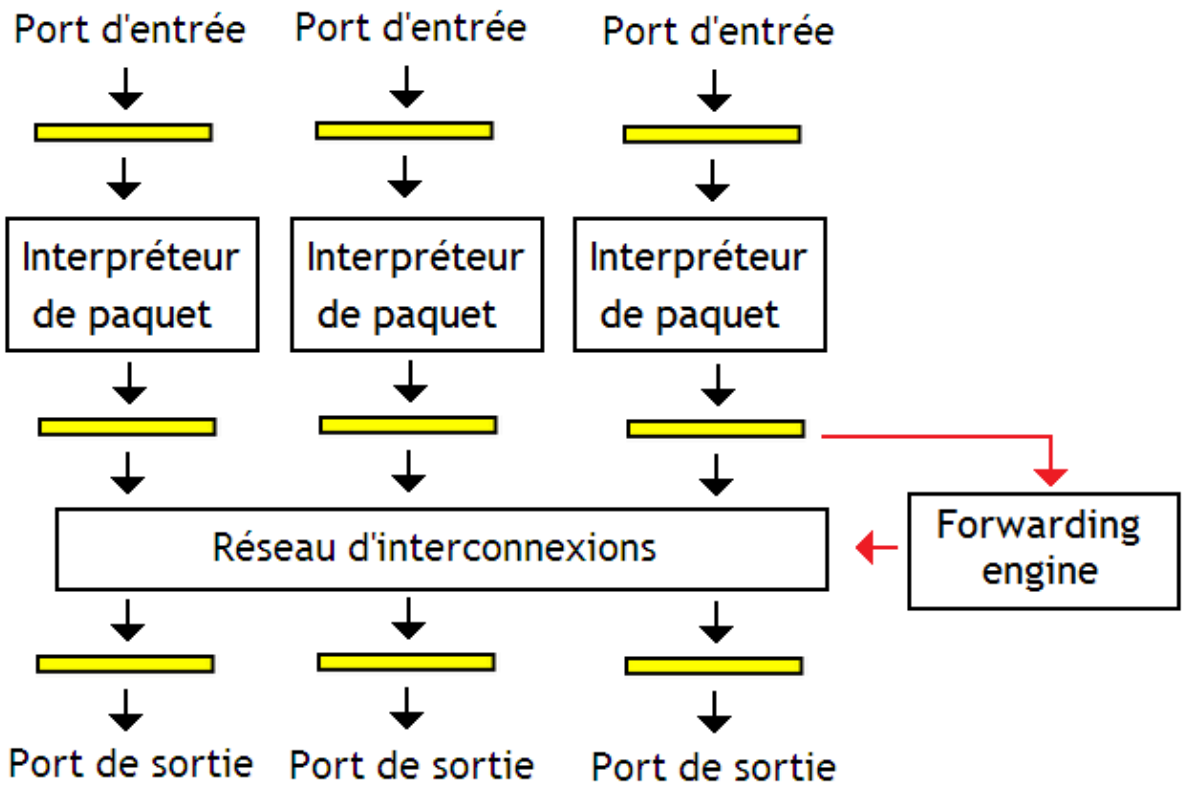

Figure 2.6 – Switch/routeur pipeliné

3 Switchs fabric

Les *switch fabric* peuvent s'implémenter de diverses manières :

— un réseau d'interconnexions configurable selon selon les besoins ;
— une mémoire ;
— un bus.

Certains switchs traitent les ports d'entrée à tour de rôle, l'un après l'autre : on parle de **switchs à partage de temps**, aussi appelés *time-sharing switch* chez les anglo-saxons. D'autres switchs permettent de gérer tous les ports d'entrée à la fois : on parle de **switchs à partage d'espace**, ou *space-sharing switchs* chez les anglo-saxons.

3.1 Switchs fabric à base de bus ou de multiplexeurs

3.1.1 Switchs à base de multiplexeurs

Certains switchs sont crées avec des multiplexeurs et des démultiplexeurs commandés par la CAM table ou la table de routage :

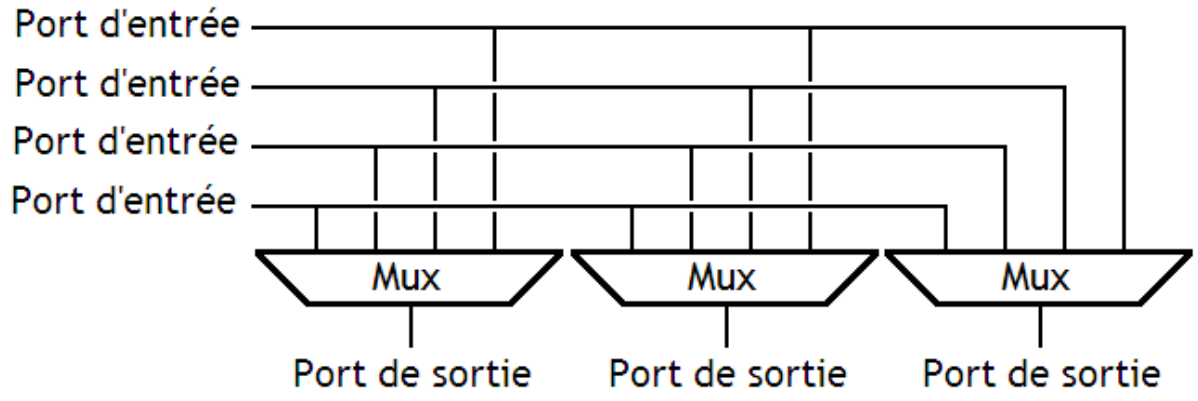

Figure 3.1 – Exemple avec un switch à 4 entrées et 3 sorties

3.1.2 Switchs à base de bus

Pour faciliter l'implémentation du multicast et du broadcast, certains switchs émulent les liaisons point à point entre ports à partir d'un bus ou d'un réseau en anneaux : ce sont les **switchs à média partagés**.

3.1.2.1 Avec partage de temps

L'architecture la plus simple pour ce genre de switch est celle-ci :

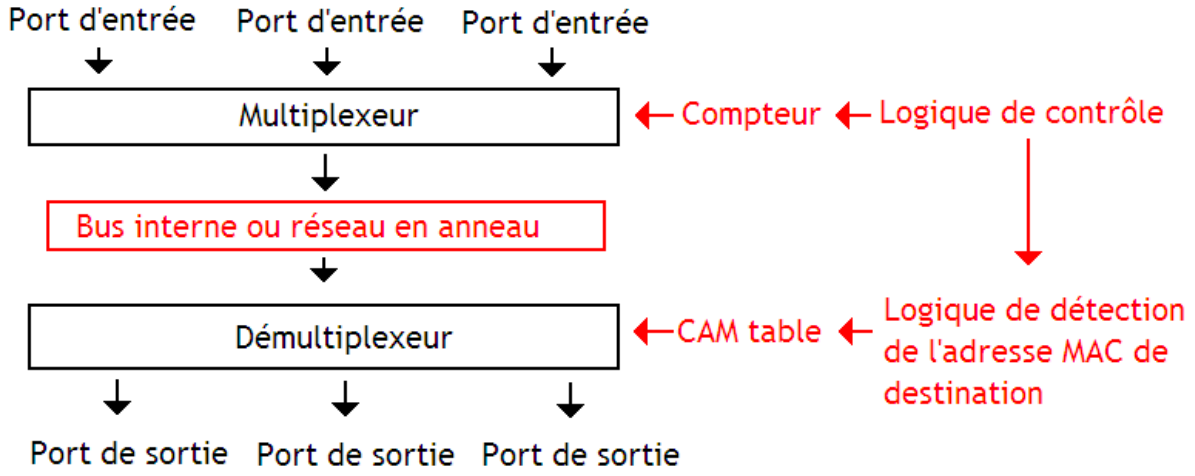

Figure 3.2 – Switch à partage de temps conçu avec des multiplexeurs

Avec cette architecture, implémenter le multicast ou le broadcast est relativement complexe. Pour résoudre ce problème, il suffit de relier chaque port de sortie sur le bus interne directement, sans démultiplexeur. En faisant cela, chaque port de sortie doit filtrer les paquets qui ne lui sont pas destinés. Pour cela, on ajoute un filtreur d'adresse pour comparer l'adresse MAC/IP associé au port (CAM table) et l'adresse MAC de destination : si il y a égalité, alors on peut recopier la donnée sur le port de sortie. Dit autrement, la CAM table est distribuée dans les différents filtreurs d'adresse (ce qui rend sa mise à jour relativement mal-aisée, au passage).

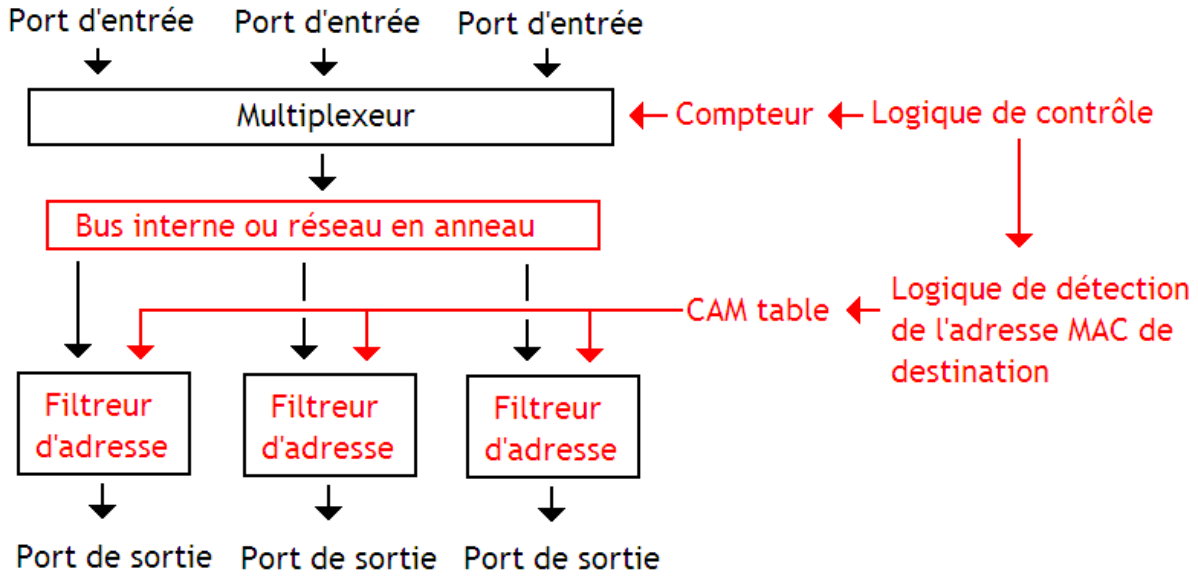

Figure 3.3 – Switch à média partagé

3.1.2.2 Switchs totalement interconnectés

Les switchs vus plus haut sont des switchs à partage de temps (sauf le tout premier). Pour éviter cela, il suffit d'utiliser un bus par port d'entrée, chaque bus ayant ses propres filtreurs d'adresse : on obtient alors un **switch totalement interconnecté**.

Avec ces switchs, il se peut que plusieurs paquets soient destinés à un seul port de sortie. Dans ce cas, ces paquets sont accumulés dans une mémoire tampon en sortie des filtreurs d'adresse : ces switchs utilisent l'*output buffering*. Les sorties des différents filtreurs d'adresse sont donc directement reliés à cette mémoire, et peuvent écrire dedans sans problème. Cela signifie que la mémoire tampon est donc multiports.

3.1.2.3 Switch Knockout

Utiliser une mémoire multiports reliée à autant de filtreurs d'adresse est clairement un challenge quand le nombre de ports de sortie est trop grand. De plus, ce n'est utile que quand tous les paquets reçus sur les ports d'entrée doivent être routés sur un seul port de sortie. Dans la majorité des cas, seule une faible quantité de ports d'entrée veut router un paquet sur le même port de sortie.

Dans ces conditions, certains switchs ne permettent de router qu'un nombre limité de paquets sur le même port en même temps pour économiser du circuit. On obtient alors un **Knockout switch**. Le principe est très simple : les filtreurs d'adresse associés à un port de sortie sont suivis par un concentrateur, qui sélectionne L paquets parmi les N filtreurs d'adresse. Seuls ces N paquets sont envoyés dans les FIFO du port de sortie. Si jamais le nombre de paquets destinés au port de sortie dépasse la limite L, seuls L paquets seront conservés et les autres seront perdus.

3.2 Switchs fabric à mémoire partagée

Certains switchs replacent le bus interne par une mémoire RAM : ce sont les **switchs à mémoire partagée**. Ces switchs permettent de gérer l'arbitrage directement dans la *switch fabric* : la mémoire centrale stocke les FIFO d'arbitrage des ports d'entrée.

Le switch présenté dans le schéma ci-dessus peut être transformé en switch sans partage de temps avec une mémoire multiport : il suffit d'avoir autant de ports d'écriture que de ports d'entrée, et d'adapter les circuits de gestion de la mémoire. Dans ce qui va suivre, mes schémas utiliseront un switch à partage de temps.

3.2.1 Linked-list switch

Reste que ces FIFOs peuvent s'implémenter de différentes manières. Une première méthode mémorise les FIFOs dans des listes doublement chainées dans la RAM. Les circuits qui gèrent la mémoire partagée doivent gérer eux-même les listes chainées, ce qui fait qu'ils contiennent deux registres par liste : un pour l'adresse de la tête de la liste, et un pour le dernier élément. Il y a un circuit pour l'ajout, et un autre pour le retrait des paquets.

En plus de cela, les circuits de gestion de la mémoire doivent allouer dynamiquement les nœuds de la liste et libérer la mémoire. La mémoire utilise des Bytes démesurément grands, capables de mémoriser un paquet de plusieurs centaines de bits sans problèmes. Ainsi, le switch a juste

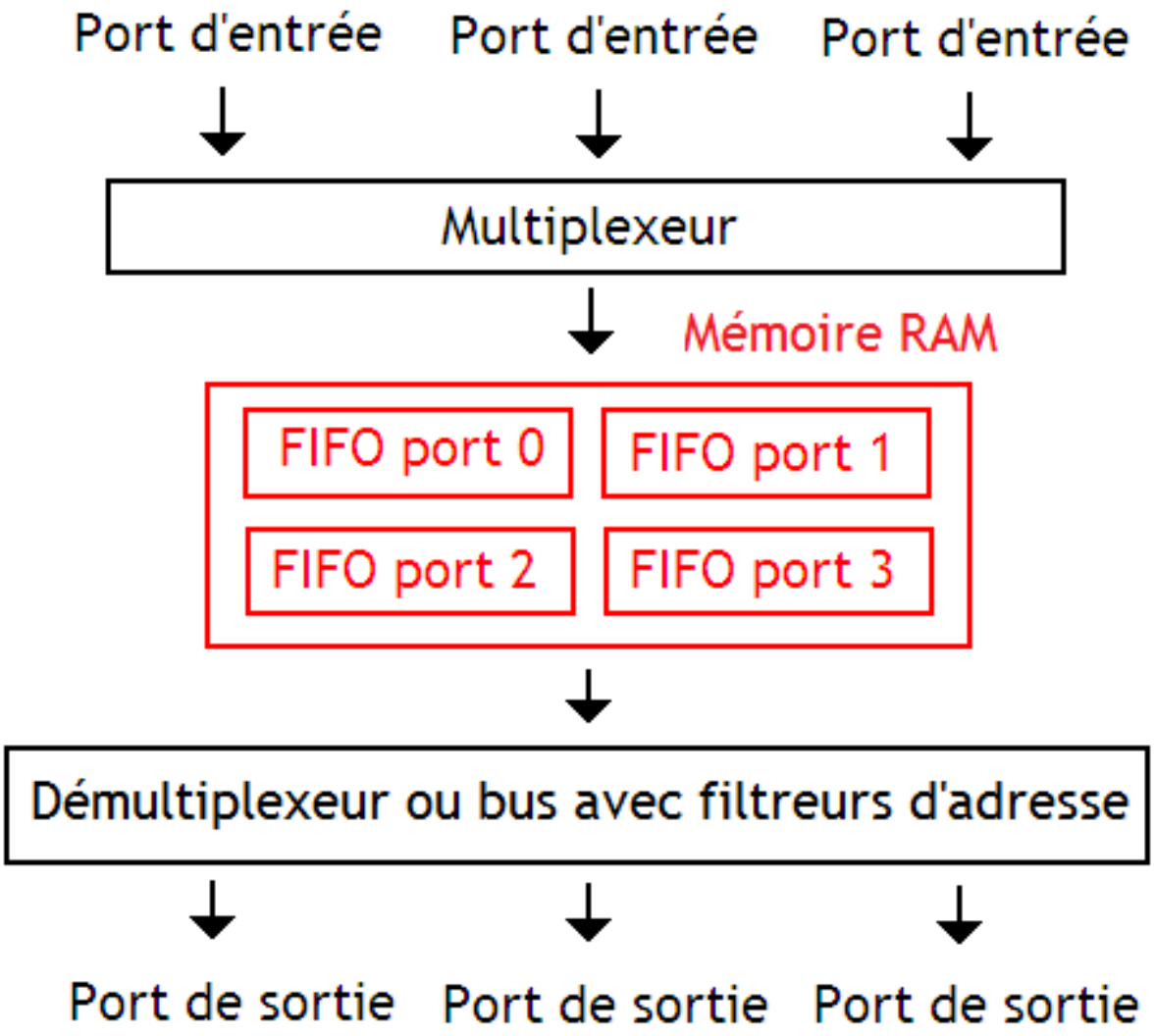

Figure 3.4 – Switch à mémoire partagée

besoin de mémoriser quels sont les Bytes libres et les Bytes occupés dans une mémoire annexe : la ***free list***, souvent implémentée avec une FIFO.

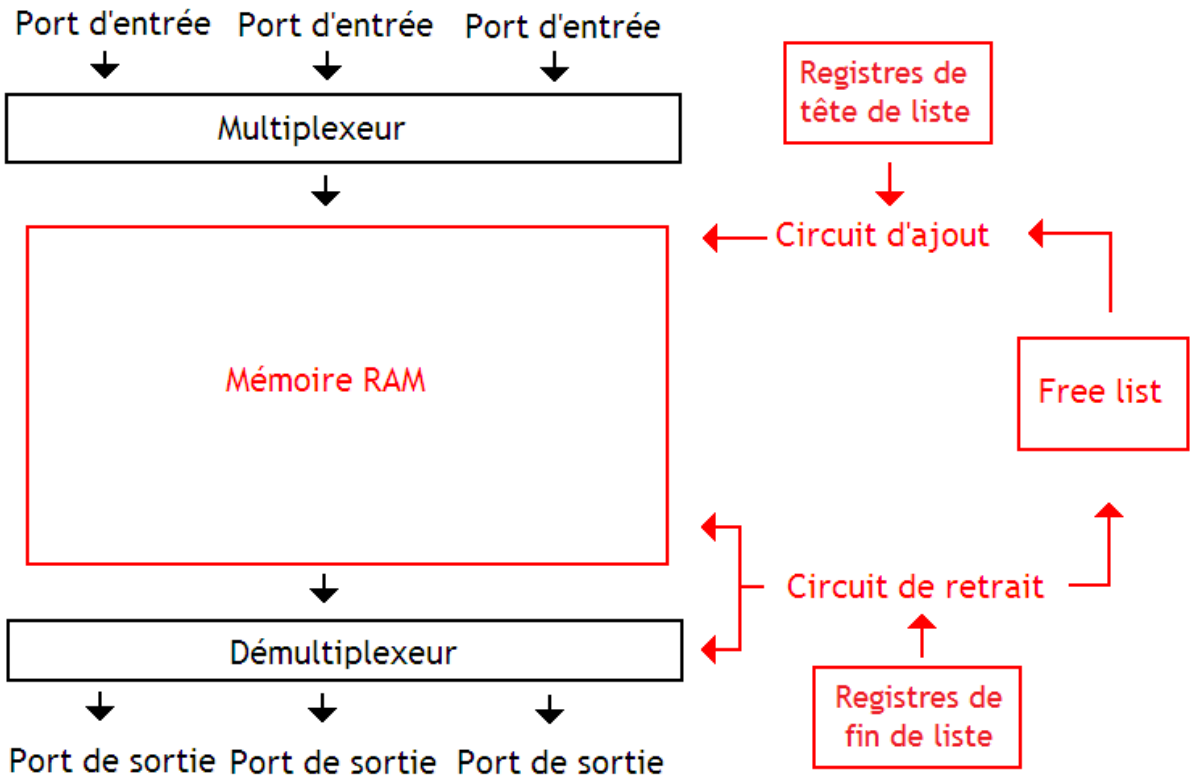

Figure 3.5 – Linked-list switch

3.2.2 FIFO switch

La seconde méthode :

- réserve la mémoire centrale pour le stockage des paquets ;
- et maintient l'ordre d'envoi des paquets dans des FIFO, qui mémorisent les adresses des paquets dans la mémoire centrale.

La taille fixe des FIFOs ne permet pas d'allouer toute la mémoire pour seulement quelques ports. Mais cette organisation permet de gérer facilement le multi-cast et le broadcast : on peut facilement ajouter un paquet dans toutes les FIFO, simultanément avec un circuit d'ajout conçu pour.

3.2.3 CAM switch

Une autre solution, nettement plus simple, consiste à remplacer la mémoire RAM du switch par un cache (une mémoire associative, plus précisément). Chaque ligne de cache mémorise :

- un paquet ;
- le numéro de port de sortie ;
- l'ordre d'envoi du paquet, afin de simuler une FIFO

Pour mémoriser l'ordre d'envoi, le switch attribue un numéro de séquence à chaque paquet : plus ce numéro est grand, plus le paquet est ancien.

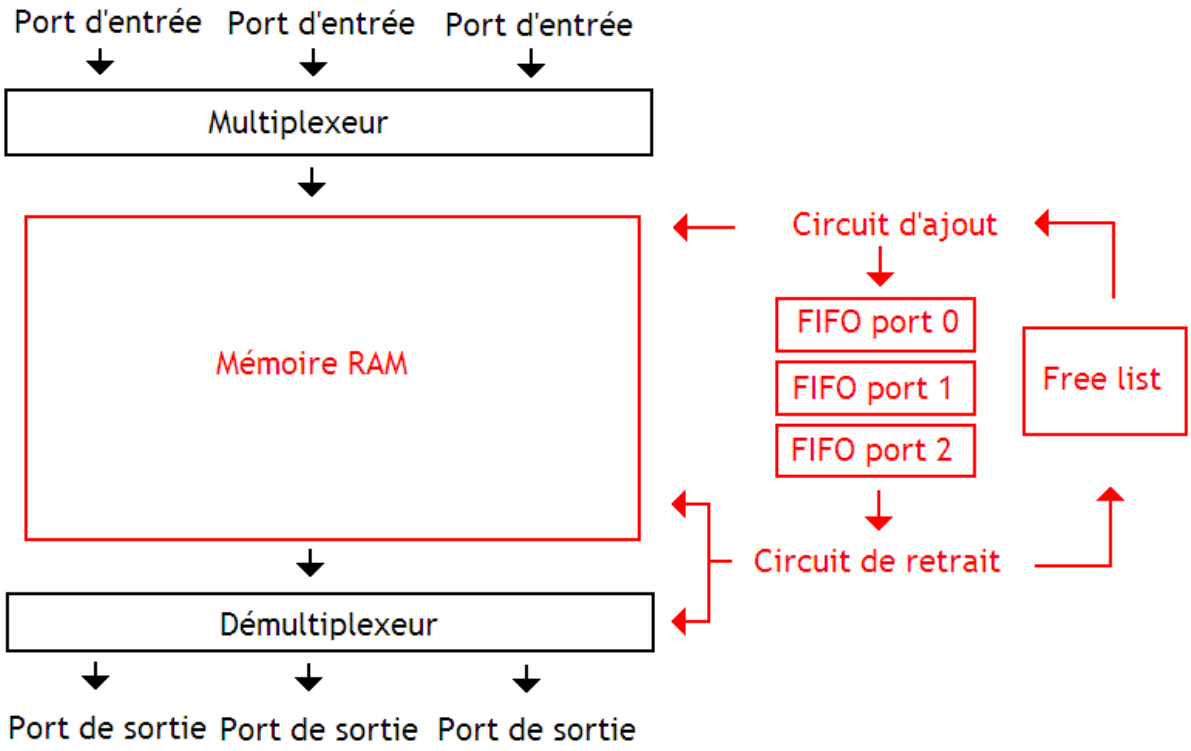

Figure 3.6 – FIFO switch

3.3 Switchs à base de réseau d'interconnexions

3.3.1 Switch crossbar

Le switch crossbar utilise un réseau d'interconnexion dont les fils sont organisés en lignes et en colonnes. A l'intersection de chaque ligne et de colonne, on trouve :

— soit un interrupteur qui relie la ligne et la colonne ;
— soit des FIFO d'arbitrage à pour gèrer l'*Head-Of-Line Blocking*.

Le nombre d'interrupteurs/FIFO est de N * M pour un switch à N ports d'entrée et M ports de sortie : pour les switchs qui ont un grand nombre de ports d'entrée et de sortie, cela devient rapidement impraticable. Pour éviter cela, les switchs à haute performance réduisent la taille des FIFO intégrées dans le crossbar, mais rajoutent de grosses FIFO sur les ports d'entrée : les simulations montrent que le rapport entre performance et nombre de portes logiques utilisées est meilleur avec cette technique.

3.3.1.1 Switch CLOS

On peut utiliser plusieurs switchs crossbar pour former un switch plus gros. Cette méthode donne ce que l'on appelle un **réseau CLOS**. Un réseau CLOS est conçu à partir de trois couches de réseaux crossbar : une couche d'entrée, une couche intermédiaire, et une couche de sortie.

Il est aussi possible d'utiliser plus de trois couches. Par exemple, on peut créer un switch CLOS à partir de plusieurs couches de crossbar 2 - 2 : on parle alors d'un réseau de Benes.

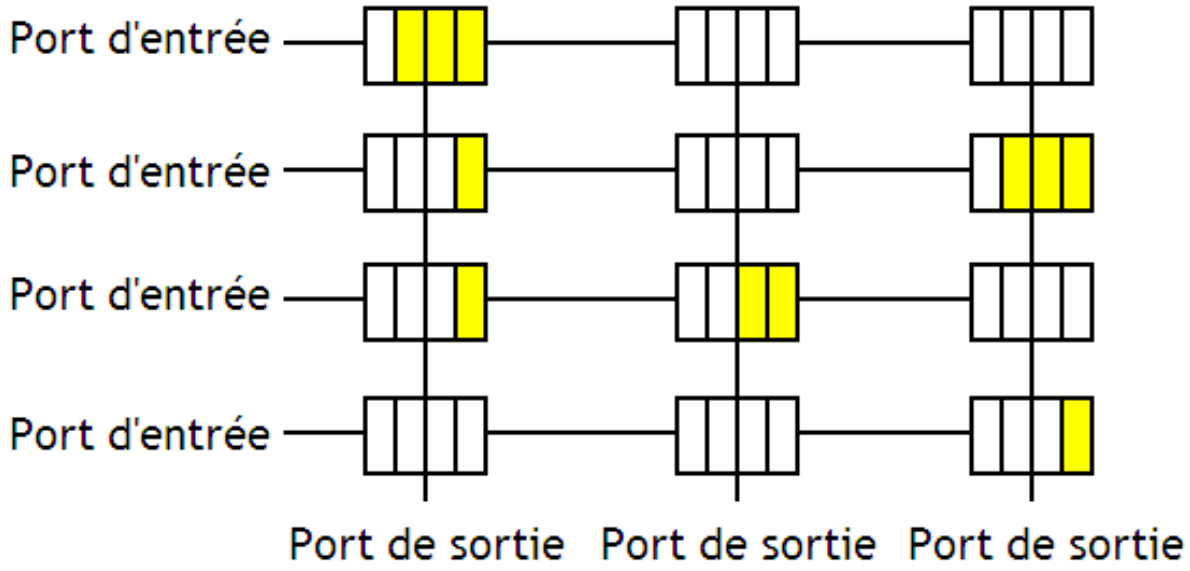

Figure 3.7 – Switch Crossbar avec arbitrage intégré

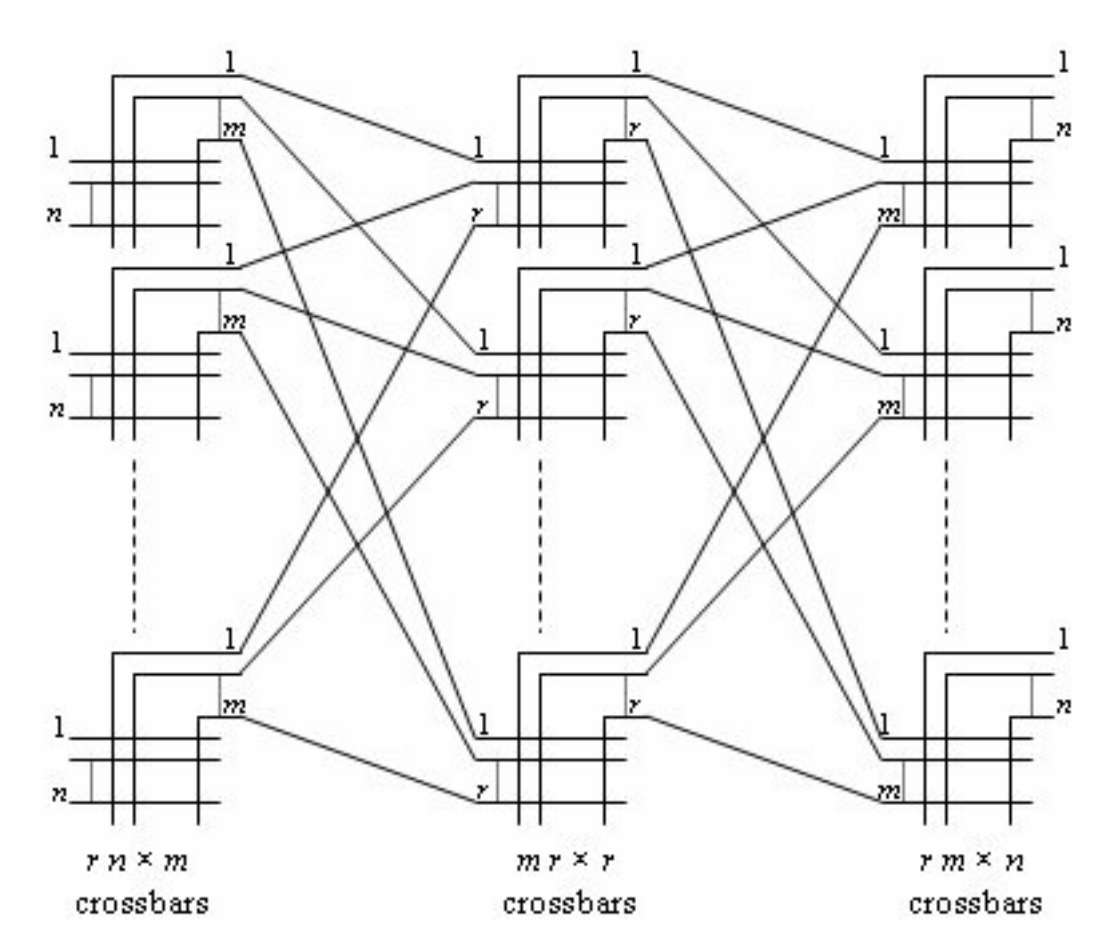

Figure 3.8 – Switch CLOS

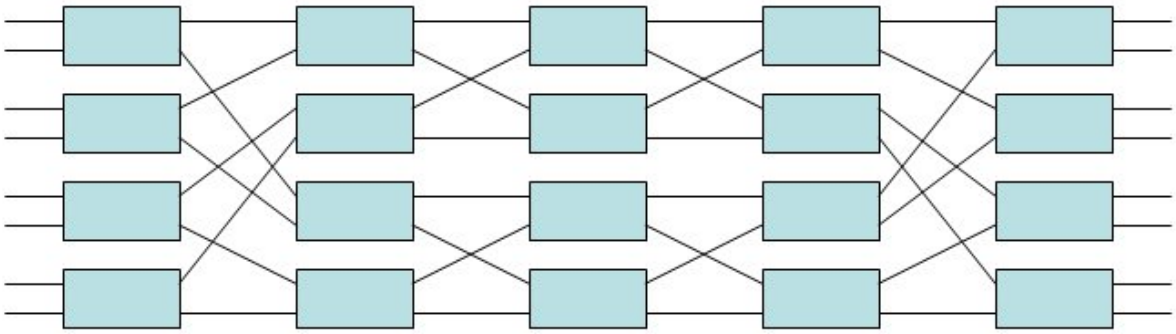

Figure 3.9 – Switch de Benes

3.3.2 Switch de Banyan

Dans ce qui va suivre, nous allons nous intéresser aux **switchs de Banyan**.

3.3.2.1 Briques de base

Les versions les plus simples des switchs de Banyan sont conçues à partir de switchs à deux entrées et deux sorties, qui ressemblent à ceci :

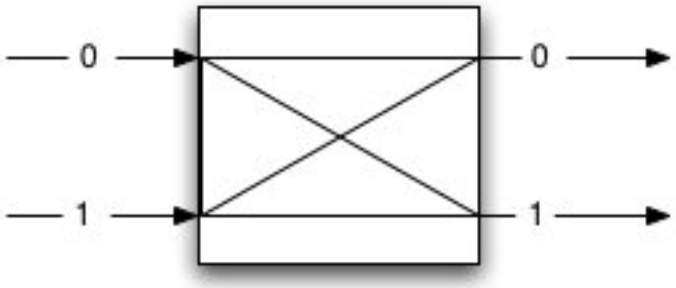

Figure 3.10 – Switch de Banyan à deux entrées et deux sorties

On peut adapter les switchs de Banyan pour qu'ils n'utilisent pas des switchs 2 - 2, mais 3 - 3, ou 4 - 4, voire plus : on parle alors de **switchs de Banyan augmentés**. Qui plus est, ces briques de base peuvent contenir des FIFO et de la logique de traitement, comme pour les switch Crossbar.

3.3.2.2 Switch de Banyan

Ces switchs sont empilés pour former des couches, chaque couche possédant autant de sorties et d'entrée qu'il y en a dans le switch de Banyan final : si un switch a 8 entrées et 8 sorties, alors chaque couche aura 8 sorties et 8 entrées (et cela marche aussi avec 4, 16, etc). Le nombre de couches est égal au logarithme du nombre de ports d'entrée/sortie N, ce qui fait que le nombre de switchs de base utilisé est proportionnel à $N \times \log N$: on économise donc des portes logiques comparé à un crossbar.

Le switch de Banyan le plus simple est composé de deux couches, et possède donc 4 entrées et 4 sorties. Il ressemble à ceci :

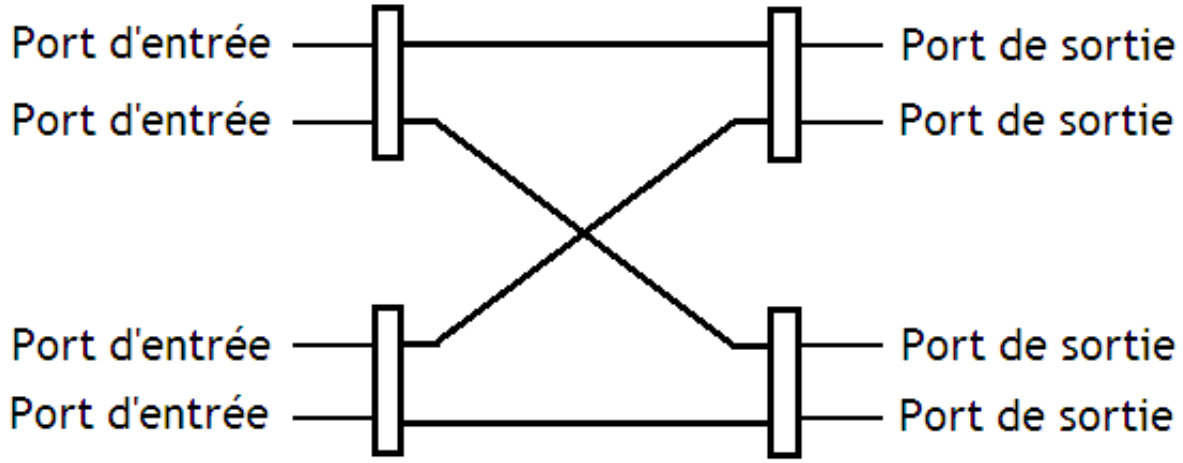

Figure 3.11 – Premier exemple

Pour un switch à 8 entrées et 8 sorties, voici ce que cela donne :

Et ainsi de suite…

En guise d'exercice, regardez bien les schémas, et essayez d'en déduire la logique de construction. Vous verrez, rien de bien sorcier. Et non, je ne pose pas cet exercice parce que j'ai la flemme de vous l'expliquer…

Blague à part, il existe différents types de réseaux de Banyan, qui portent les doux noms d'Omega, d'Alpha, etc. Ces réseaux se construisent simplement : si on prend un switch à N entrées (et autant de sorties), alors chaque couche doit réaliser une permutation des entrées et sorties, permutation qui doit respecter certaines propriétés pour tenir compte du fait que les switchs de base ont deux entrées et deux sorties. Le principe de création dépend donc des permutations utilisées sur chaque couche.

Mais ces switchs ne sont pas sans défauts. En effet, ces switchs sont dits **bloquants** : il se peut que deux paquets destinés à des ports de destination différents ne puissent pas voir leurs demandes assouvies simultanément. Cela arrive quand deux paquets veulent sortir sur le même port d'un des switch 2 - 2, d'où conflit.

3.3.2.3 Switch Batcher-Banyan

Cependant, on peut éviter ces conflits internes en triant les paquets suivant leur port de destination avec un algorithme bien précis. En conséquence, certains switchs de Banyan sont précédés d'un circuit de tri conçu avec des switchs 2 - 2, empilés en couches, ainsi qu'avec de la logique de configuration composée de comparateurs : l'ensemble forme un **switch Batcher-Banyan**.

3.3.2.4 Switch Sunshine

Le switch de Banyan, et ses dérivés ont un défaut : il ne peut router qu'un seul paquet à la fois sur une sortie. Pour router plusieurs paquets sur une seule sortie, on est obligé d'utiliser plusieurs réseaux de Banyan. Plus exactement, il faut utiliser autant de réseaux de Banyan qu'il y a de ports d'entrée/sortie. Chaque réseau est alors relié à la mémoire FIFO utilisé pour l'*output buffering*.

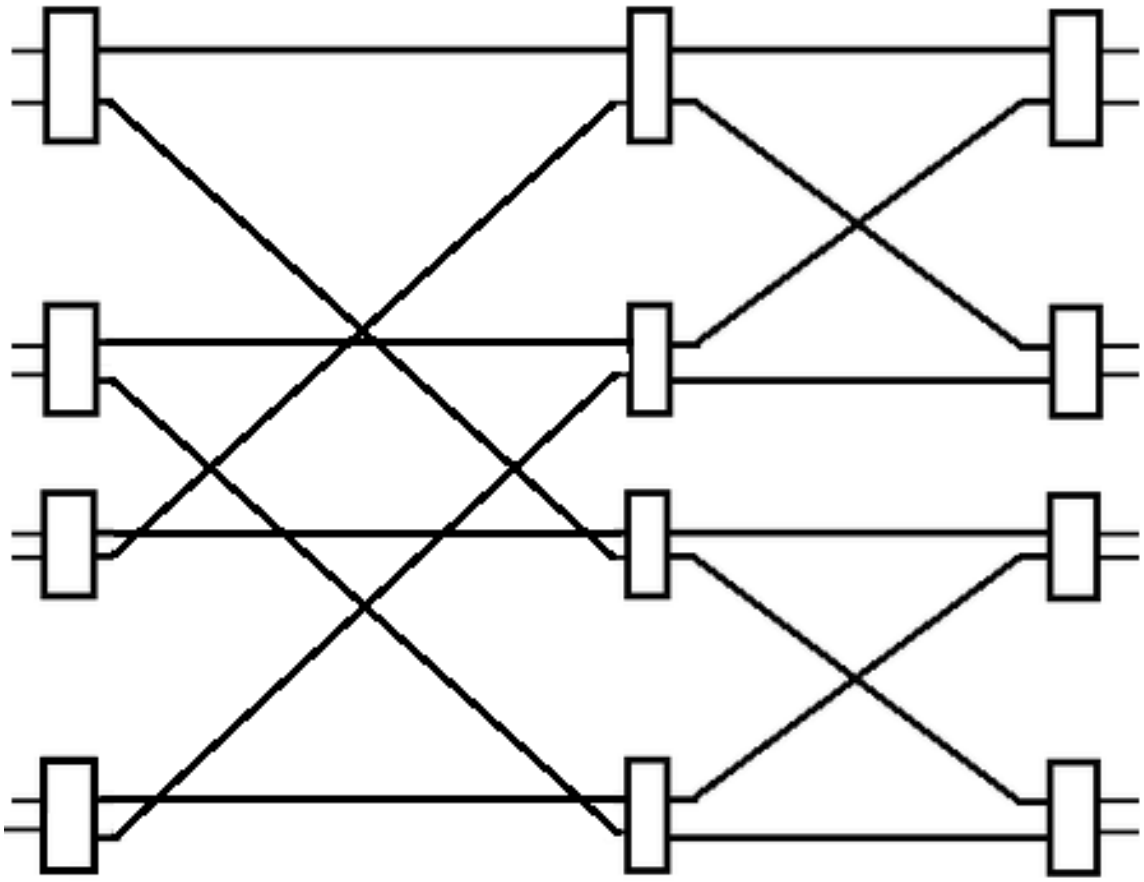

Figure 3.12 – Deuxième exemple

Mais, encore une fois, il est rare que tous les ports d'entrée doivent transmettre un paquet sur un unique port de sortie. Dans cette situation, il vaut mieux concevoir un switch capable de gérer un nombre de paquets concurrents sur le même port plus petit. Ce nombre sera noté L. Ainsi, on peut alors se contenter d'utiliser seulement L réseaux de Banyan, chacun étant couplé à un tampon de sortie pour l'*output buffering*.

Fait intéressant, cela peut aussi s'appliquer avec un switch Batcher-Banyan : le réseau de Banyan et les buffers de sortie sont dupliqués en L exemplaires, mais le circuit de tri n'a pas besoin de l'être. Dans ces conditions, le circuit de tri est relié à un répartiteur qui répartit les paquets sur les réseaux de Banyan adéquats en fonction de leur adresse de destination.

Mais que faire des paquets en trop ? On pourrait les oublier et les jeter une bonne fois pour toute. Mais les **switch Sunshine** décident de simplement mettre en attente ces paquets, afin de les renvoyer une fois que les réseaux de Banyan le permettent. Ces paquets sont mis en attente dans une FIFO, et sont renvoyés au cycle suivant en entrée du trieur de paquet.