UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MODEL-CHECKING DU DÉLAI DANS LES ÉLÉMENTS RÉSEAUX

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAITRISE EN INFORMATIQUE

PAR
SAMI BEN NASR

AVRIL 2011

REMERCIEMENTS

Je tiens de prime abord à remercier M. Roger Villemaire, professeur au Département d'Informatique de l'UQAM, d'avoir dans un premier temps accepté de m'encadrer puis de m'avoir orienté et parrainé dans ce mémoire. C'est sur ses divers conseils et orientations et ses différents encouragements que ce projet a pu voir le jour.

Je remercie également M. Étienne Gagnon, directeur du programme de maîtrise en informatique, qui m'a encouragé et soutenu lors de l'élaboration de ce mémoire ainsi que M. Jean Privat, professeur au Département d'Informatique et Mme Halima Elbiaze, professeure au Département d'Informatique qui ont daigné accepter d'évaluer ce mémoire.

Je remercie également ma femme qui m'a soutenu durant ces derniers mois avec patience. Je remercie aussi mes beaux parents Mme Zohra Chaabane par ses encouragements multiples et M. Slah Eddinne Chaabane qui n'a cessé de me prodiguer conseils et soutiens. Je remercie mon frère Rafik et sa femme qui m'ont été d'un grand secours pendant les moments difficiles.

C'est un grand merci que j'adresse à ma chère mère qui avec son amour et son dévouement pour ma personne a consacré avec le concours de mon père beaucoup de sacrifices pour que je puisse poursuivre mes études universitaires au Canada.

Je ne manque pas de remercier tous ceux qui à un moment donné ont pu m'initier à la connaissance et au savoir à commencer par mes maîtres et instituteurs de l'école primaire, mes professeurs de l'enseignement secondaire au lycée mixte de Gabès-Tunisie, mes professeurs des études supérieures et universitaires à l'UQAM, l'Université de Moncton et l'Institut Supérieur d'Informatique à Gabès-Tunisie.

Comme je remercie tous mes amis qui m'ont aidé à un moment ou un autre que ce soit en Tunisie ou au Canada. Comme je remercie également tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

La responsabilité des routeurs s'engage lorsque les machines hôtes envoient leurs paquets dans le réseau. Les routeurs auront donc la fonction de transmettre ces paquets sur les liens pour les acheminer vers la destination déterminée. Cependant, comme le routeur traite les paquets séparément, la performance du routeur dépend donc du temps de traitement pour chaque paquet. Avec un charge de trafic, il est possible d'optimiser efficacement le traitement des paquets dans le routeur. Notre attention sera portée sur l'évaluation du délai de bout-en-bout dans le réseau *End-to-End*. Ce mémoire propose donc un modèle qui consiste à évaluer et vérifier les délais des paquets dans les routeurs par la méthode de vérification de modèles (Model-Checking).

Mots-clés : vérification de modèles, Model-Checking, réseaux, routeur, délai.

INTRODUCTION

De nos jours, la télécommunication et les réseaux occupent une place très importante dans la vie quotidienne des gens et des entreprises tout en embrassant presque toutes les activités économiques avec ses diverses branches, et tous les aspects culturels de la vie moderne. Cette télécommunication et ces réseaux offrent un service très vaste tant aux secteurs bancaires et industriels que dans d'autres domaines technologiques et scientifiques comme la médecine. En effet, les recherches dans ces domaines ont évolué tout en ayant des objectifs très différents. Ces efforts ont montré que les réseaux et Internet sont des outils nécessaires dans notre vie. Habituellement, la haute utilisation de ces outils peut engendrer une dégradation sur la performance des réseaux. Les métriques comme le débit, le délai et le taux de pertes sont des critères de performance essentiels dans le réseau. Par ailleurs, les méthodes de mesures de ces métriques constituent un grand intérêt pour l'optimisation du réseau *End-to-End*. En effet, notre intérêt s'est porté le délai tout en analysant et évaluant son impact sur l'optimisation du réseau *End-to-End*. Cette métrique sera évaluée avec l'utilisation des méthodes de vérification de modèles (Model-checking).

Globalement, les méthodes de vérification de modèles (Model-Checking) sont pratiquement appliquées dans tous les domaines. Le principe de ces méthodes permet également de vérifier un système réel tout en appliquant les propriétés souhaitées en partant d'une formule mathématique. Lorsque la modélisation du système qu'on veut vérifier est prête, un outil dénommé vérificateur (Model-Checker) va déterminer si les propriétés sont vérifiées par le modèle. Comme l'objectif est de valider le système réel, si aucune erreur n'est décelée par le vérificateur, il n'est pas garanti que le système réel soit correct. Mais si une erreur est mise en évidence par le vérificateur, il faut s'assurer qu'elle doit ressortir et s'afficher dans le système réel. C'est pour cette raison que la vérification de modèles est distinguée comme une méthode capable de détecter les erreurs et pas une méthode qui garantit le bon fonctionnement d'un système réel.

L'idée de notre modèle est née de la question de l'évaluation du délai et des pertes de données dans les éléments des réseaux de télécommunication. En effet, les métriques telles que le débit, le délai et le taux de perte sont devenues des critères de choix lors de la détermination de la qualité service. Ainsi, leurs valeurs définissent le niveau de qualité à satisfaire en ce qui concerne le réseau. Cependant il n'est pas toujours facile d'effectuer des mesures sur des routeurs réels durant leur fonctionnement normal. Les auteurs de l'article (Chertov *et al.*, 2008) ont donc construit un modèle qui permet d'évaluer les délais de traitement dans un routeur Cisco par des méthodes de simulation. Leur modèle est générique et passablement indépendant de la réalisation matérielle du routeur simulé. Pour adapter leur modèle à un routeur réel, il suffit d'ajuster quelques paramètres à partir de quelques mesures sur le routeur réel.

La vérification de modèles est une méthode issue de la logique, qui permet la vérification exhaustive de propriétés. Elle a été largement utilisée tant en vérification de matériel que de logiciel. Dans notre cas, nous l'appliquons à l'évaluation du délai de traitement des paquets dans les routeurs. Ce type d'application présente néanmoins un défi, car normalement la vérification de modèles permet d'établir une propriété logique qui ne peut être que vraie ou fausse. Dans notre cas, il s'agit plutôt d'estimer une valeur (le délai). Nous allons montrer que ceci est possible en adaptant les méthodes de vérification de modèles bornée. Notre objectif est donc tout simplement de montrer que le délai peut être évalué par des méthodes de vérification de modèles.

Ce mémoire est composé de deux parties, la première partie présente l'état de l'art qui est composée par deux chapitres, le premier chapitre présente le rôle des routeurs dans les réseaux et leurs évaluations. Ainsi, on étudie l'acheminement des paquets d'une source à une destination. Le deuxième chapitre présente les méthodes de vérification de modèles les plus utilisées avec les propriétés logiques LTL.

La deuxième partie est composée de trois chapitres, le premier chapitre (chapitre III) présente la source d'inspiration de notre modèle. Le deuxième chapitre (chapitre IV) présente notre modèle qui permet d'évaluer les délais des paquets dans les routeurs

par l'approche de vérification de modèles. Le dernier chapitre (chapitre V) montre tout simplement que le délai peut être évalué par des méthodes de vérification des modèles en interprétant les résultats obtenus.

Enfin, ce mémoire sera corroboré par une conclusion liée à la conception du modèle proposé.

CHAPITRE I

RÉSEAUX ET ROUTEURS

Les réseaux se composent de systèmes informatiques (ordinateurs, serveurs, ...), de liaisons et de routeurs. Les systèmes informatiques (Sanchez, 2003) utilisent les adresses IP (Internet Protocol) (Postel, 1981) pour pouvoir communiquer entre eux. Le protocole IP fournit un service de communication *best-effort* qui est basé sur le mode datagramme. L'information à transmettre se divise en paquets indépendants. Ces paquets ont des entêtes qui contiennent l'information nécessaire au routage. Ainsi, les paquets sont transmis d'un routeur à l'autre en vue d'atteindre leur destination finale. Cependant, la transmission de chaque paquet est indépendante. C'est ainsi que, Internet permet aux systèmes informatiques de partager des ressources de réseau en utilisant la commutation des paquets.

L'objectif à atteindre dans ce chapitre est de présenter le rôle des routeurs dans les réseaux et leurs évaluations. Ainsi, nous allons présenter une étude détaillée sur l'acheminement des paquets d'un port d'entrée à un port de sortie dans un routeur réel. Nous allons par la suite montrer l'utilisation des files d'attente dans les routeurs.

1.1 Rôle du routeur dans le réseau

Le rôle principal du routeur dans le réseau est d'acheminer les paquets d'une source à une destination. Généralement, les machines hôtes (par exemple les serveurs) communiquent entre elles en utilisant des paquets d'information. En particulier, la responsabilité des routeurs (Sanchez, 2003) s'engage justement lorsque les machines hôtes envoient leurs paquets dans le réseau. Les routeurs auront la fonction de transmettre ces paquets sur les liens des réseaux pour pouvoir les acheminer vers la destination adéquate. L'acheminement de chaque paquet est indépendant. En effet, le mode datagramme laisse l'accès

actif aux systèmes informatiques dans le réseau pour pouvoir donner la flexibilité aux utilisateurs d'envoyer des données à tout moment sans avoir l'obligation d'établir une connexion.

De façon générale, l'IP est un protocole simple sans connexion. Ainsi, le service ne peut être que *best-effort*. Un routeur tente donc localement de retransmettre les paquets le plus rapidement possible, mais en cas de congestion, il les laissera tomber. C'est pourquoi l'optimisation du fonctionnement d'un routeur est très importante.

Avec un trafic différent, il est très utile d'optimiser la performance (Sanchez, 2003) des routeurs pour pouvoir acheminer les paquets à leurs destinations. Mais comme le mode datagramme offre un service résistant, les routeurs sont des éléments essentiels qui peuvent s'adapter avec le changement de topologie du réseau tout en acheminant toujours les paquets avec succès à leurs destinations finales. La figure 1.1 illustre l'acheminement des paquets entre les routeurs.

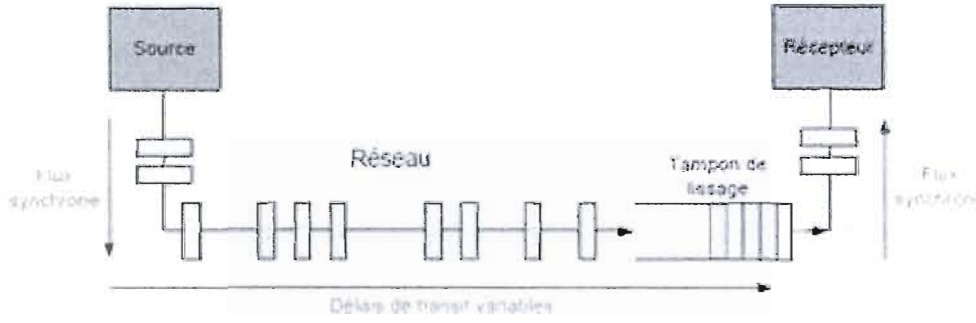

FIGURE 1.1: Acheminement des paquets d'une source à une destination (Ehrensberger, 2006).

Les routeurs (Sanchez, 2003) comportent des étapes nécessaires pour acheminer les paquets à leur destination : Premièrement, le paquet IP contient l'adresse IP de destination. Mais la fonctionnalité interne du routeur en particulier est vraiment la tâche de déterminer à partir de cette adresse IP l'interface de sortie permettant d'atteindre le prochain routeur.

L'information de routage permet également la prise de la décision de routage. En particulier, le routeur a une action essentielle qui s'appelle consultation d'adresse *adress lookup*. Chaque paquet reçu sera acheminé à la sortie indiquée par la table de routage (Huitema, 2000).

Deuxièmement, le routeur doit transmettre le paquet du port d'entrée au port de sortie. Pratiquement, si la route de sortie est libre, le paquet sera transmis sur le lien ; si non, le paquet sera stocké dans une file d'attente temporairement jusqu'à la libération du lien.

Troisièmement, la tâche finale du routeur est la résolution des problèmes de contentions survenus dans les routes de sorties.

1.2 La fonction d'acheminement des paquets dans le routeur réel

Le routeur transfère les paquets entre les réseaux. En particulier, le routeur (Sanchez, 2003) doit retransmettre les paquets qu'il reçoit sur un lien permettant d'atteindre la destination. Le transfert des paquets, s'appelle l'acheminement des paquets (packet forwarding) qui est la tâche principale du routeur. La figure 1.2 illustre l'architecture d'un routeur générique. Fondamentalement, le routeur se compose de ports d'entrée, de ports de sortie et d'une matrice de commutation (switching fabric). Au début, les paquets arrivent sur les ports d'entrée. Ensuite, ils traversent la matrice de commutation (switching fabric) pour atteindre les ports de sortie appropriés. Finalement, les ports de sortie acheminent les paquets vers la destination finale.

Quand les paquets arrivent sur le port d'entrée (Sanchez, 2003), le routeur doit acheminer les paquets vers le prochain routeur ou la destination finale. Précisément, pour acheminer les paquets, le routeur requiert trois tâches critiques :

1. Prendre une décision pour pouvoir acheminer les paquets ; il cherche le prochain routeur dans lequel les paquets vont être envoyés.
2. Commuter les paquets du port d'entrée au port de sortie approprié.

3. Résoudre la contention lors du transfert des paquets sur les ports de sorties.

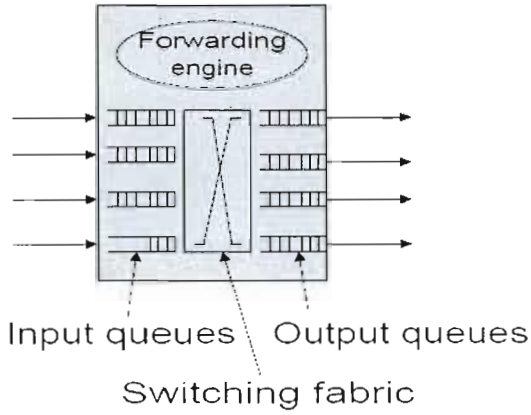

FIGURE 1.2: L'architecture d'un routeur générique (Angrisani *et al.*, 2006).

1.3 L'utilisation des files d'attente dans le routeur

L'information de routage (Sanchez, 2003) est une tâche nécessaire dans le routeur. Elle permet au routeur de décider et d'acheminer les paquets d'une source à une destination. De plus, les paquets seront commutés du port d'entrée au port de sortie tout en les transmettant sur les routes appropriées. Dans certaines situations, les sorties peuvent être occupées lors de la transmission des paquets. Notamment, nous pouvons avoir sur un même port de sortie, des paquets venant de plusieurs entrées différentes. Donc le routeur doit justement résoudre la contention sur les sorties en utilisant, par exemple, des files d'attente.

En général, l'usage des files d'attente permet également de minimiser le taux de perte lors du transfert des paquets sur les ports de sorties. En particulier, avec un charge de trafic, le transfert des paquets du port d'entrée au port de sortie, entraîne une augmentation de taux perte. Par ailleurs, l'utilisation des files d'attente permet d'un côté, de minimiser les paquets engendrés lorsque les routes soient occupées ; et d'un autre coté, permet de multiplexer les paquets provenant de différentes entrées sur une même route de sortie. Donc les files d'attente ont une fonction de multiplexage (Sanchez, 2003) qui peut être

aussi bouleversée par le charge de trafic des différents utilisateurs. Dans ce cas, il faut trouver un moyen pour pouvoir protéger la fonction de multiplexage. Sans aller plus loin, beaucoup des travaux ont été publiés sur l'optimisation des files d'attente et les méthodes de multiplexage dans les routeurs. Notamment les auteurs de l'article (Sanchez, 2003) proposent un mécanisme dénommé MuxQ dont la fonction est d'optimiser l'utilisation des files d'attente de façon efficace tout en protégeant la fonction de multiplexage et tout en fournissant un haut degré d'isolation entre les flux qui partagent la même route de sortie.

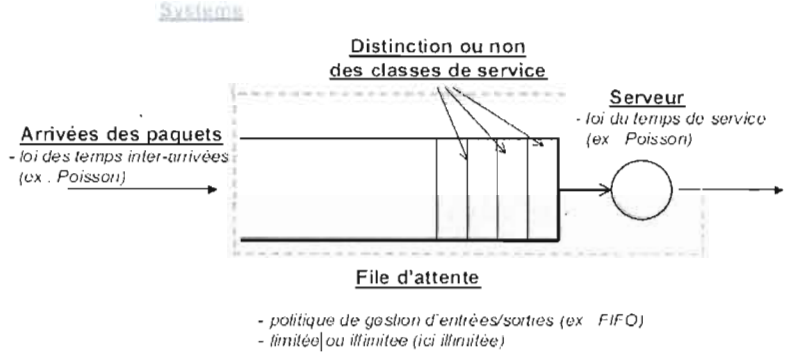

FIGURE 1.3: Modélisation de la file d'attente (source (Vatinlen, 2004)).

En effet, la fonctionnalité du routeur (Vatinlen, 2004) se fait comme suit : Les paquets arrivent sur les ports d'entrées. Si la route est disponible, le paquet sera transféré au port de sortie. Si non il sera mis dans une file d'attente FIFO pour pouvoir attendre son tour. C'est sur cette partie que notre intérêt sera porté dans ce mémoire.

1.4 Les métriques de la qualité de service

L'utilité du réseau IP (Vatinlen, 2004) est de permettre de donner à l'utilisateur une bonne qualité de service lors du transfert des données. En particulier, la qualité peut être déterminée. Plutôt au niveau de l'utilisateur qu'au niveau des gestionnaires du réseau.

En premier lieu, la perception de la qualité de service est sentie par l'utilisateur au

niveau de la facilité de l'utilisation, de la sécurité, de la continuité et de la fiabilité de service en termes d'écoulement du trafic qui se définit par les pertes de données et les délais de transmissions.

En deuxième lieu, du point de vue du gestionnaire des réseaux, la qualité de service est différemment perçue. En effet, celle-ci est ressentie au niveau de la qualité de fonctionnement du réseau qui assure de bonnes communications entre les utilisateurs. Cette dernière englobe la sûreté de fonctionnement, la qualité des ressources et les supports et le délai de transmission.

Les différents types de trafics (Vatinlen, 2004) cohabitant au sein du réseau IP conduisent une complexité supplémentaire dans le routage de l'information. En particulier, chaque type de trafic a ses propres exigences au niveau de la qualité de service. Donc, il est essentiel de classifier les trafics selon leurs exigences pour pouvoir obtenir une bonne qualité de service.

Globalement, les applications critiques (Aweya, 1999) comme la téléphonie IP et les vidéos conférences créent un support qui exige une qualité de service très élevé. Ces applications sont sensibles aux termes du temps d'attente absolu et aux variations de la latence. Au delà du service de *best-effort*, les routeurs commencent à offrir un certain nombre de classes ou de priorités de la qualité de service. Ces priorités sont utilisées pour indiquer le traitement préférentiel d'une classe du trafic sur des autres. Les matrices de commutations (switching fabric) doivent manipuler ces classes du trafic différemment selon leurs conditions de la qualité de service. En particulier, la matrice de commutation est typiquement une matrice qui aura des files d'attente sur chaque port de sortie. Chaque classe de trafic sera attribuée à file d'attente. Ainsi, les files d'attente peuvent être physiquement séparées où une file d'attente peut être divisée logiquement en des files séparées.

En effet, la gestion des files d'attente (Aweya, 1999) se réfère notamment à la politique de rejet pour l'entrée des paquets dans les files d'attente (par exemple, Drop Tail, Drop-From-Front, Random Early Detection (RED), etc.), et à la politique d'ordonnancement

pour les sorties de paquets par les files d'attente (par exemple, priorité stricte, round-robin (WRR), (WFQ), etc.). Cependant, la gestion des files d'attente dans les réseaux IP impliquent deux dimensions qui sont le temps (l'ordonnancement du paquet (packet scheduling)) et l'espace des files d'attente (le rejet du paquet (packet discarding)). Nous voyons donc que la gestion des files d'attente et le support de la qualité de service sont une partie intégrante de la conception de la matrice de commutation (switching fabric).

Les métriques (Vatinlen, 2004) comme le délai de transmission, le débit et les pertes de données permettent de mesurer la qualité de service dans le routeur. Donc avec l'évaluation de ces métriques nous pouvons optimiser la qualité de service. Par ailleurs, les métriques les plus importantes sont :

1. Le délai de transmission : Le délai de transmission fait référence au temps que met un paquet pour être acheminé d'une source vers une destination.
2. Le délai de traitement : Est lié au temps que le routeur doit déterminer pour atteindre le port de sortie approprié afin de transmettre le paquet.
3. Le délai de la file d'attente : Est le temps d'attente de chaque paquet dans la file afin de les transmettre sur le port de sortie approprié. Le délai de la file d'attente dépend aussi de la charge du noeud, de la politique d'ordonnancement des messages dans le noeud. Ce délai est souvent aléatoire mais peut être borné si l'on dispose d'une connaissance suffisante sur le trafic entrant dans le noeud (source/destination).
4. Le débit : La fonctionnalité de débit dans le réseau (Vatinlen, 2004) se représente par la quantité d'informations envoyées pendant le temps émis, comme par exemple nous pouvons définir l'unité de débit en paquets par seconde.
5. Les pertes de données : Les pertes de données (Vatinlen, 2004) constituent un élément essentiel dans le réseau tout en engendrant un impact très important sur le débit. Ces pertes représentent le pourcentage des données perdues lors de leur transfert dans le réseau.

Dans ce chapitre, nous avons mis en évidence un certain nombre de concepts. Ainsi, en premier lieu, nous avons mis l'accent sur le rôle du routeur dans le réseau. Ensuite,

nous avons montré les éléments essentiels du délai dans le routeur qui se compose de trois contributions : délai de la file d'attente, délai de traitement et délai de transmission. Puis, nous avons présenté la fonctionnalité du routeur tout en expliquant les démarches d'acheminement des paquets dans le routeur.

Généralement, le routeur réel a un fonctionnement plus complexe que celui qui est décrit dans ce chapitre, mais que globalement il peut être compris comme un élément contenant une matrice de commutation (switching fabric) et des files d'attente. Nous considèrerons ce point de vue (similairement à l'article (Chertov *et al.*, 2008)) pour évaluer les délais dans les routeurs par la méthode du Model-Checking.

Ce mémoire portera sur l'évaluation du délai dans les routeurs en appliquant les méthodes de vérification de modèles (Model-Checking) que nous allons introduire dans les prochains chapitres.

CHAPITRE II

MODEL-CHECKING

Avant la réalisation d'un projet quelconque, chaque système passe par trois phases essentielles qui sont : la conception, le développement, le test et la validation. Néanmoins, l'étape la plus importante dans un système donné est la vérification et la correction des erreurs. C'est pourquoi, les méthodes formelles ont été intégrées dans le cycle de développement des systèmes afin d'accroître la sûreté et la sécurité dudit système.

Depuis des années, les outils des méthodes formelles ont pris une place très importante dans ce domaine afin de détecter les erreurs et de vérifier les propriétés au niveau logiciels et matériels mais aussi à l'implémentation elle-même. Leurs buts se basent sur la détection des erreurs dans les premières étapes de développement, où la correction de ces erreurs permet de gagner du temps tout en étant à faible coût.

L'objectif de ce chapitre est de présenter une des méthodes de vérification formelles les plus utilisées aujourd'hui, la vérification de modèles (model-cehcking). Aussi, nous allons présenter également les propriétés logiques nécessaires afin de mettre en évidence leurs importances sur des applications de télécommunication.

2.1 Modélisation et vérification

Il existe plusieurs approches (Villemaire, 2009) qui permettent la vérification des systèmes informatiques. Par exemple, une approche associe à chaque procédure des pré-conditions et des post-conditions. D'abord, on suppose que les pré-conditions sont vraies avant l'appel, tout en s'appuyant sur les variables d'entrées du système. Ensuite, on vérifie que les post-conditions sont vraies après l'appel. Cette approche est utilisée de façon informelle où chaque développeur doit programmer de façon à pouvoir justifier que les

post-conditions sont vérifiées. Si nous voulons automatiser cette approche, nous devons vérifier au début, les procédures les plus simples, ensuite nous pouvons considérer des cas plus complexes comme par exemple des sous-programmes récursifs où un programme s'appelle lui-même. Il faut alors utiliser un outil appelé démonstrateur de théorème (theorem prover).

Une autre approche populaire est la vérification de modèles (Model-Checking). Elle représente le système informatique par un ensemble d'états. Le système changeant d'un état à un autre lorsque des actions sont effectuées. Notons que l'état d'un système informatique est le contenu actuel des éléments de mémoire qu'il prend en charge. La vérification de modèles est donc proposée pour explorer cet espace d'états pour pouvoir vérifier et valider un système donné. Notons que dans un système réel, il existe toujours une infinité d'états, par exemple lorsqu'on utilise des structures dynamiques comme les listes, les piles ou les files avec une taille qui n'est pas déterminée. Ceci est dû à l'utilisation de structures de tailles arbitraires, ce qui nous mène à avoir une infinité de possibilités. Il y a un certain nombre de travaux qui ont été publiés sur la vérification de modèles avec un nombre infini d'états mais généralement la majorité des chercheurs considèrent les systèmes avec un nombre fini d'états. Il est alors nécessaire d'approximer le système donné par un modèle fini. En général ceci est réalisé en bornant la taille des structures dynamiques (listes, piles, files) par une constante fixe. En recommençant la vérification avec des tailles croissantes, il est néanmoins possible de découvrir de nombreuses erreurs dans des systèmes intéressants.

Le principe de la vérification de modèles (Baclet, 2005) permet de vérifier les propriétés souhaitées d'un système en partant d'une représentation mathématique. Pour se faire, avant d'entamer la vérification proprement dite du système, nous devons considérer deux points importants qui sont :

- La modélisation du système qu'on désire vérifier.
- La spécification des propriétés souhaitées du système.

Lorsque ces deux étapes seront accomplies, un outil nommé vérificateur (model-checker) va déterminer si les propriétés sont vérifiées par le modèle. Comme l'objectif est

de valider le système réel, si une erreur est découverte par le vérificateur, il faut s'assurer qu'elle peut bien se présenter dans le système réel. Si aucune erreur n'est découverte, il n'est pas assuré que le système réel sera correct. C'est pourquoi la vérification de modèles est souvent présentée comme une méthode pour détecter les erreurs et non comme une méthode assurant un fonctionnement toujours correct.

2.2 Conception des modèles

Avant la réalisation d'un système informatique (une application, logiciel, matériel), la réalisation d'un modèle (Villemaire, 2009) dans un langage formel comme celui de l'outil NuSMV nous permettra de faire l'analyse formelle du système.

La modélisation formelle des systèmes informatiques (E.M. Clarke et Peled, 2000) offre beaucoup d'avantages par rapport à l'implémentation. Premièrement, le modèle présente les spécifications formelles que l'on veut valider, il sera donc plus utile pour vérifier des propriétés du système que l'implémentation réelle. Aussi comme il ne représente que l'aspect que l'on veut vérifier, il pourra normalement être construit plus rapidement qu'une implémentation réelle. De plus, la vérification d'un modèle simplifié nécessite un temps de calcul raisonnable par rapport à celle d'un modèle très fidèle à l'implémentation réelle. D'où, il sera plus judicieux (fructueux) de construire des modèles simples pour pouvoir faciliter la vérification de modèles dans un temps convenable.

Comme le modèle sert à vérifier et non à s'exécuter, il peut néanmoins décrire le comportement du système réel suffisamment fidèlement. En vérification de modèles, le modèle utilise des structures de données avec des tailles déterminées, alors nous obtiendrons toujours un nombre fini d'états ce qui nous mène à avoir un système fini. Par la suite, nous pourrons appliquer les techniques de vérifications nécessaires afin de justifier si le modèle a le même comportement que le système réel. La vérification de modèles permet d'un côté de tester le modèle sur plusieurs tailles différentes pour pouvoir analyser son comportement et comparer les résultats obtenus. Néanmoins le modèle étant plus

simple permet normalement des comportements supplémentaires par rapport au système réel. Ceci n'est pas problématique en soit, car il est toujours possible de raffiner le modèle s'il est trop grossier pour la tâche de vérification.

Il existe plusieurs langages de vérification de modèles comme UPPAAL (Behrmann *et al.*, 2004), HYTECH (Henzinger *et al.*, 1997), KRONO (Bozga *et al.*, 1998). Normalement, chaque outil a son propre langage de description des modèles. Par ailleurs, nous nous intéressons particulièrement au langage de l'outil NuSMV, car c'est celui que nous avons utilisé. L'outil NuSMV est une nouvelle implémentation de l'outil SMV (le premier vérificateur symbolique (McMillan., 1993)). NuSMV fait la vérification de modèles de systèmes à états finis, donc qui n'ont qu'un nombre fini d'états. Avant de présenter les principaux éléments du langage NuSMV, nous allons introduire dans la prochaine section la principale formalisation des systèmes à états finis.

2.3 Structure de Kripke

Nous allons présenter la structure de Kripke qui est la principale formalisation des systèmes à états finis. À l'origine Kripke (E.M. Clarke et Peled, 2000) a introduit ses structures dans le cadre de la sémantique des logiques modales (dont la logique temporelle que nous allons présenter dans la section 2.6 fait partie). Généralement, nous pouvons dire qu'une structure de Kripke est la même chose qu'un système de transition finie. En fait, dès que nous aurons un nombre fini de variables discrètes, il sera possible de représenter le système sous forme d'une structure de kripke.

Une structure de Kripke (E.M. Clarke et Peled, 2000) est formée de :

1. Q un ensemble fini d'états.
2. $I \subseteq Q$ l'ensemble des états initiaux.
3. $T \subseteq Q \times Q$ une relation de transition qui devrait être totale, c'est-à-dire que pour tout $q \in Q$ il existe toujours un $q' \in Q$ tel que $(q, q') \in T$. $Q \times Q$ étant l'ensemble des couples d'états.

4. $L : Q \longrightarrow P(A)$ une fonction qui associe à chaque état $q \in Q$, l'ensemble L(Q) des variables proportionnelles vraies dans q.

Prenant comme exemple le compteur modulo 2 suivant

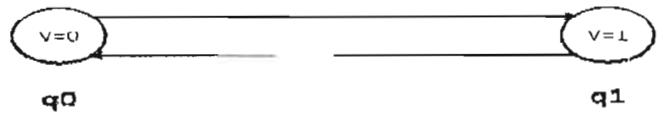

FIGURE 2.1: Compteur modulo 2.

- $Q = \{q0, q1\}$ l'ensemble d'états ={0,1}.
- $I = \{q0\}$ ensemble d'états initiaux.
- $T = \{(q0, q1), (q1, q0)\}$ deux couples, donc deux transitions.
- Pour $v = 0$ on a $L(q0) = \oslash$ = ensemble vide et pour $v = 1$ on a $L(q1) = \{v\}$.

2.4 Langage de NuSMV

2.4.1 Les types de données

Dans cette section, nous allons présenter les principaux types de données du langage de NuSMV. Comme les types de données de NuSMV sont finis, on a nécessairement que chaque système n'aura qu'un nombre fini d'états. Les types de données du langage de NuSMV sont les suivants :

- Les constantes *booléennes* qui correspondent aux valeurs entières 0 et 1 et qui sont équivalentes aux symboles **TRUE** et **FALSE**.
- Les types par *énumération*, spécifiés par toutes les valeurs que le type comporte. Par exemple {open, close, error}, {2, 1, 0}, etc. Tous les éléments d'énumération doivent être uniques. De plus, l'ordre de ces éléments n'est pas important.
- Les *plages d'entiers*, comme 1..5 qui sont tout simplement des ensembles d'entiers (dans notre exemple, l'ensemble contenant les entiers de 1 à 5). Ces nombres entiers

peuvent être dans l'intervalle $[-2^{32}+1$ jusqu'à $1-2^{32}]$, plus exactement, ces valeurs sont équivalentes aux macros C/C++ INT_MIN et INT_MAX.

- Les *mots*, sont employés pour pouvoir modéliser des tableaux booléens (bits) comme par exemple **word[4]** qui représente un mot de quatre bits indicé de 0 à 3 de droite à gauche.
- Les *tableaux*, comme par exemple array 1..4 of open, close, error, qui représente les tableaux de open, close, error numérotés de 1 à 4.

Il est possible de créer des types par énumération en utilisant l'union comme par exemple open, close union error qui est équivalant au type suivant open, close, error.

2.4.2 Les variables

La description du langage NuSMV est un peu spécifique, car le modèle se décrit en NuSMV par la définition de variables initiales et de variables d'entrée. Les variables initiales sont introduites par la notation **VAR** et les variables d'entrée sont introduites par la notation **IVAR**. L'état du modèle est déterminé par les valeurs des variables à cet instant. Les variables initiales et d'entrée se distinguent de la façon suivante :

- Les variables initiales (Villemaire, 2009) servent à décrire le comportement du système en spécifiant leurs valeurs initiales ainsi que leurs évaluations à chaque étape. Nous pouvons donc assigner une valeur initiale à une variable de ce type avec l'instruction (**init(variable)**). Similairement nous pouvons assigner la valeur suivante (celle de l'état suivant) avec l'instruction (**next(variable)**).
- Les variables d'entrée, de leur côté, représentent des actions qui viennent de l'extérieur du système. Il n'est doncpaspossible de leur donner ni valeur initiale, ni valeur suivante. Elles prennent toujours des valeurs quelconques quenousne pouvons pas fixer. Elles modélisent donc bien une entrée venant de l'extérieur et hors du contrôle de notre modèle.

Nous considérons l'exemple suivant (voir la figure 2.2). Ce modèle comporte une seule variable (X) qui est de type entier. La valeur initiale de cette variable est 1 et la valeur suivante s'incrémente chaque fois de 1. Ce qui nous donne toujours 1, 2, 3, 4, 1, 2, 3, 4,1,....etc.

```
-- Compteur modulo 4
MODULE main
VAR
X : 1..4;
ASSIGN
init(X) := 1;
next(X) := (X mod 4) + 1;
```

FIGURE 2.2: Un compteur modulo 4 (compteur4.smv).

2.4.3 Structure des cas (case...esac)

La structure de contrôle principale du langage de NuSMV est une structure de cas (**case...esac**), qui est similaire à celle des langages C et C++. Chaque cas est formé d'une condition booléenne, suivie par ' :' et finalement la valeur retournée.

2.4.4 Les modules

Généralement, le principe de la construction d'un modèle en NuSMV est assez différent du développement d'un programme dans un langage de programmation. Par exemple, bien que la décomposition soit utilisée dans le deux cas, les modules en NuSMV (Villemaire, 2009) permettent également de regrouper les variables initiales, d'entrées et le calcul de leurs valeurs et de leurs états suivants. Un module est donc un type et chaque variable de ce type possède ses propres variables et ses propres calculs. Par ailleurs, dans n'importe quel modèle en NuSMV, il existe un seul **module main** qui est le point de départ de l'exécution du modèle. Ce module n'admet donc pas de paramètre. Donc, un modèle peut se composer de modules simples qui ont des paramètres différents. Par exemple, la figure 2.3 représente un modèle qui regroupe plusieurs modules avec leurs propres variables initiales et leurs propres variables d'entrées.

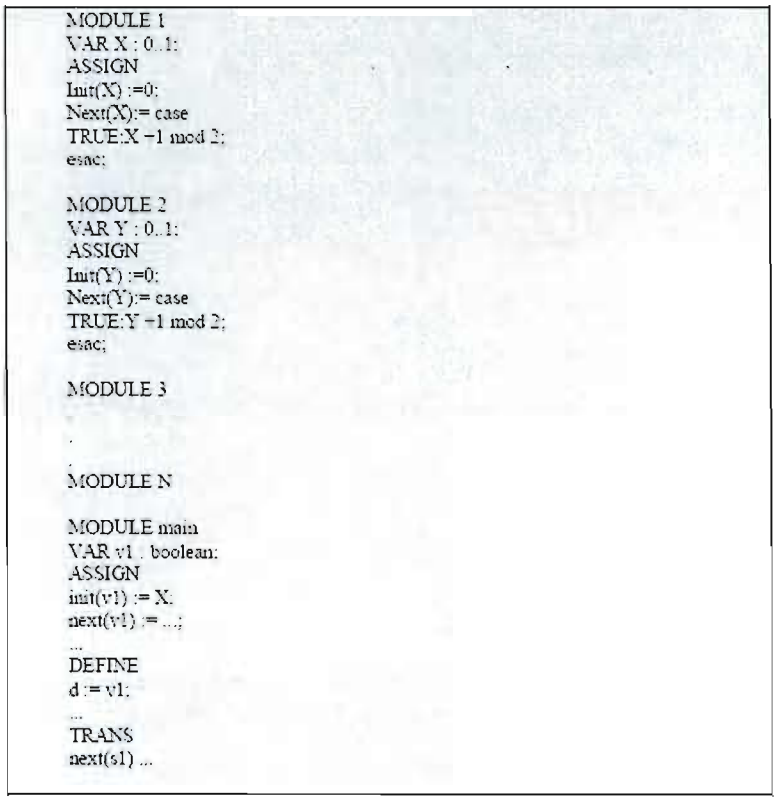

FIGURE 2.3: Conception des modules en NuSMV.

On peut remarquer les choses suivantes dans le modèle de la figure 2.3.

- La clause **MODULE** spécifie le nom du module ainsi que ses paramètres.
- La clause **ASSIGN** définit les affectations des valeurs des variables qui sont déjà définies pour permettre à NuSMV de s'exécuter.
- La clause **DEFINE** est un type simple de macro qui permet de définir et fixer un identificateur qui n'aura un sens que dans le contexte où le remplacement aura lieu.
- La clause **TRANS** définie la relation des transitions entre les états courants et les états suivants du système.

2.5 Vérification exhaustive de modèles

La vérification de modèles (Villemaire, 2009) est une tâche importante de nos jours qui peut s'appliquer dans tous les domaines de l'informatique. En particulier, la vérification de modèles a été introduite dans le but d'améliorer la qualité des conceptions de systèmes réels dans leurs cycles de vie. La construction d'un modèle fini permet d'approcher des systèmes réels, tout en appliquant par la suite des techniques relativement simples et efficaces de vérification. Bien que la taille d'un modèle en NuSMV soit finie, elle peut être néanmoins très grande.

Un modèle peut donc avoir un grand nombre d'états (des milliers, voire des millions). De plus, une seule modification (ajouter ou supprimer une variable) peut changer le nombre d'états. Par exemple, si on change le type d'une variable ou si on ajoute une autre variable pour rendre le modèle plus flexible pour qu'il corresponde mieux au système réel, on peut facilement doubler le nombre total d'états. Il faut donc toujours rester prudent dans la construction d'un modèle donné, question de limiter sa complexité.

Il n'existe pas de modélisation unique pour un système réel. Il y a toujours plusieurs modélisations valables, en fonction des objectifs de vérification. Bien que la construction d'un modèle avec le moins de variables possibles est une bonne façon de contrôler la complexité, il faut noter que le temps d'exécution des algorithmes de vérification modernes n'est pas directement fonction de ce nombre d'états et qu'il est très difficile, en fait, de prévoir à l'avance le temps d'exécution des algorithmes modernes. Néanmoins les implémentations d'aujourd'hui sont très efficaces et permettent de vérifier rapidement des modèles conséquents.

Remarquons de plus que le modèle peut être lui-même erroné. Si le modèle est erroné alors on ne peut pas s'assurer que sa vérification va être correcte. Il faut donc toujours construire le modèle soigneusement tout en essayant de ne pas oublier un aspect essentiel. Néanmoins, lorsqu'une propriété d'un modèle est fausse, l'outil de vérification nous retournera une trace. En vérifiant si celle-ci peut se produire sur le système réel, on

peut néanmoins s'assurer indirectement que le modèle n'est pas trop simple. Donc, si en général, la vérification de modèles ne permet pas de garantir que le système aura le fonctionnement escompté, elle peut nous permettre de trouver des comportements erronés et d'ainsi corriger la conception.

La description des propriétés à vérifier se fait en NuSMV, comme dans la plupart des autres outils de vérification, à l'aide des logiques temporelles LTL et CTL (Rakkay, 2009), Prior (1967), (Clarke et Emerson, 1982) et (Huth et Ryan, 2004). Comme nous venons de le dire, si NuSMV détecte qu'une propriété n'est pas vérifiée par le modèle, une trace sera produite qui justifie cette affirmation par un contre-exemple. Ce contre-exemple nous donne une explication qui permet de justifier pourquoi le modèle est erroné, tout en nous aidant à repérer l'emplacement de cette erreur (dans le modèle ou dans la formule). Nous allons maintenant nous concentrer dans la section suivante sur la logique temporelle la plus utilisée en vérification de modèles, soit *Linear Temporal Logic* (LTL).

2.6 Logiques temporelles LTL

L'objectif dans cette section est de présenter la logique temporelle (Villemaire, 2009) qui permet de décrire des propriétés de systèmes informatiques en vue de les faire vérifier de manière automatique en utilisant un outil de vérification comme NuSMV. Généralement, avant d'intégrer les techniques de vérifications dans un modèle, il faut que les conditions à vérifier soient formalisées. Pour pouvoir composer des propriétés avec la logique temporelle, nous devons savoir que les formules de base sont des formules atomiques qui permettent la comparaison entre les variables du système. De plus, les formules peuvent être combinées par les connecteurs booléens et les connecteurs temporels. Ces derniers sont justement ceux qui permettent de décrire des propriétés qui dépendent du temps.

La figure 2.4 ci-dessous illustre les connecteurs booléens alors que la figure 2.5 nous montre les connecteurs temporels.

Connecteurs	Nom
!	Négation
&	« et » logique (conjonction)
|	« ou » logique (disjonction)
->	« si... alors » (implication)
<->	« si et seulement si » (double implication)

FIGURE 2.4: Les connecteurs booléens (Villemaire, 2009).

Connecteurs	Étymologie	Sens
X	neXt	à partir de l'état suivant
G	Globally	à partir de tous les états
F	Finally	à partir de certain état
U	Until	. jusqu'à ce que.
V	release	Tant que ... alors...

FIGURE 2.5: Les connecteurs de la logique temporelle LTL (Villemaire, 2009).

Dans ce qui suit, la sémantique informelle qui a été présentée ne permet pas de donner un sens précis à une formule logique. Il faut donc trouver un sens non équivoque qui permet de déterminer le sens d'une formule par l'application d'un algorithme simple. Pour ce faire, comme la formule LTL se construit de manière récursive comme un arbre, alors il faut définir la sémantique de façon, elle aussi, récursive. La figure 2.6 illustre une définition récursive de la relation $t \models P$, qui signifie que la trace t satisfait la formule P. Par exemple $t \models XP$ signifie que le deuxième état de la trace t satisfait la formule P. Comme les formules LTL représentent des propriétés d'une trace, la sémantique de cette logique sera toujours relative à une certaine trace.

Prenons l'exemple $X(P = 0)$. Cette formule exprime que P égale à 0 dans l'état qui suit l'état initial.

Les techniques de vérifications ont été introduites pour pouvoir vérifier des propriétés

t = P (pour P est atomique)	si P est satisfaite dans le premier état de t.
t = ! P	si t = P (P n'est pas satisfaite).
t = P & Q	si t = P et t = Q
t = P | Q	si t = P ou t = Q
t = P -> Q	si t = Q lorsque t = P
t = P <-> Q	t|= P si seulement si t|= Q
t = X P	si t^1 = P
t = G P	si t^i = P, pour tout i = 0,1...
t = F P	si t^i = P, pour (au moins) un i = 0,1...
t = P U Q	s'il existe un i = 0,1... tel que t^j = P pour j < i et t^i = Q
t = P V Q	si pour tout i = 0,1,... tel que t^j = P pour i<j et t^i = Q

FIGURE 2.6: La sémantique de la logique LTL (Villemaire, 2009).

LTL. Mais comment la vérification sera-t-elle intégrée en utilisant un outil de vérification (NuSMV) ?

Comme la propriété est donnée par la spécification d'une formule logique (Villemaire, 2009) sur les traces, l'outil de vérification nous donne tout simplement une réponse « TRUE » ou « FALSE », tout en considérant implicitement toutes les traces possibles. Donc si par exemple l'outil NuSMV affirme que la formule est vérifiée, elle l'est pour toutes les traces. La propriété devrait donc aussi être vérifiée par le système réel. Si NuSMV nous donne une réponse fausse (négative), l'outil va générer une trace qui représente un contre-exemple et qui explique pour quoi la propriété est fausse.

De façon générale, la propriété logique n'a qu'une seule valeur de vérité sur une trace spécifique, mais dans certain cas nous pouvons avoir deux traces l'une vérifiant cette propriété et l'autre l'infirmant. Dans ce cas, l'outil de vérification NuSMV (Villemaire, 2009) affirmera que la formule et sa négation sont fausses et retournera un contre-exemple pour la formule ainsi qu'un autre pour sa négation. Ceci n'est pas contradictoire si on se rappelle qu'une propriété n'est vraie que si elle est vérifiée pour toutes les traces. Donc lorsque nous aurons une propriété vraie pour certaines traces et fausse pour d'autres,

NuSMV va générer un contre-exemple pour la propriété et un autre pour sa négation.

2.6.1 Traces et exemples de formules LTL

L'objectif de la logique LTL est justement de permettre de raisonner sur les séquences d'exécutions d'un modèle. Ce dernier peut avoir plusieurs exécutions différentes formées de suites d'états différents. Comme LTL est interprétée sur une trace spécifique, nous allons présenter dans cette section quelques exemples de formules LTL, ainsi que leur interprétation sur une trace.

Une trace représente une exécution du modèle ou un chemin et se définit par une suite d'états. Par exemple une trace T se compose des états $(e_0, e_1, e_2, e_3, ...e_N)$, débutant par l'état initial e_0. Il sera utile de considérer les traces obtenues débutant sur le N ème état de T. Nous dénoterons donc par Ti la trace $(e_i, e_{i+1}, ...)$ formée des états de T à partir du i ème.Nous allons maintenant présenter quelques formules de la logique temporelle tout en expliquant leur signification.

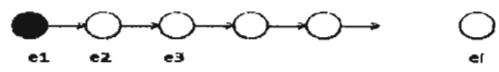

FIGURE 2.7: Une trace.

- La formule *y=1* signifie que y est égal à 1 dans l'état initial (Voir la figure 2.8).

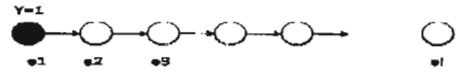

FIGURE 2.8: Exemple pour *y=1*.

- La formule *Xy=1* signifie que y égal à 1 dans l'état suivant qui suit l'état initial(Voir la figure 2.9).

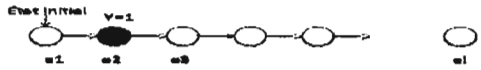

FIGURE 2.9: Exemple pour $Xy{=}1$.

- La formule $Gy{=}1$ signifie que y égal à 1 dans tous les états de la trace (Voir la figure 2.10).

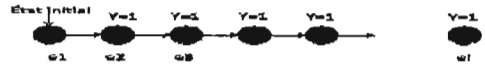

FIGURE 2.10: Exemple pour $Gy{=}1$.

- La formule $Fy{=}1$ signifie qu'il existe quelques états dans la trace où y est égal à 1 (Voir la figure 2.11).

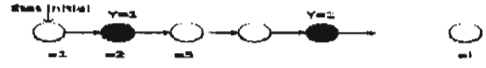

FIGURE 2.11: Exemple pour $Fy{=}1$.

- La formule $y{=}1Uy{=}2$ signifie que y est égal à 1 jusqu'à ce que y devienne égal à 2. Remarquons que si y est déjà égal à 2 dans l'état initial, la formule est satisfaite (Voir la figure 2.12).

2.7 Model-checking symbolique

La vérification de modèles symbolique (symbolic model-checking) est une approche qui se caractérise par la représentation du modèle par des formules de la logique propositionnelle. On a ainsi que l'ensemble des états initiaux est représenté par une formule booléenne et de même pour la relation de transition. Prenons l'exemple dans la figure 2.13 ci-dessous :

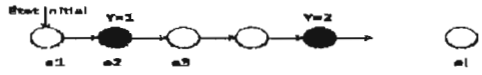

FIGURE 2.12: Exemple pour $y{=}1Uy{=}2$.

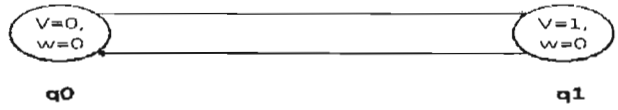

FIGURE 2.13: Représentation booléenne d'un état.

1. L'ensemble des états initiaux $I = Q = \{q0, q1\}$.
2. $T = \{(q0, q1), (q1, q0)\}$ deux couples, donc deux transitions.

L'exemple ci-dessus représente l'ensemble des états **Q** qui contient les deux éléments {*q0* et *q1*}. Particulièrement, l'ensemble des états initiaux **I** contient l'état *q0* et l'état *q1*. La relation de transition **T** est totale car elle contient les couples (*q0*, *q1*) et (*q1*, *q0*).

Pour décrire le système sous forme de formules propositionnelles, nous avons besoin de deux variables propositionnelles P_0 et P_1. P_0 signifie que $v = 0$ et P_1 signifie que $w = 0$. Donc nous obtenons que l'état *q0* est représenté par $\mathbf{Q}(q0) = P_0 \wedge P_1$, alors que *q1* est représenté par $\mathbf{Q}(q1) = \neg P_0 \wedge P_1$. Nous avons donc que l'ensemble des états initiaux est représenté par la formule $\mathbf{I} = \mathbf{Q}(q0) \vee \mathbf{Q}(q1) = ((\neg P_0 \wedge \neg P_1) \vee (\neg P_0 \wedge P_1))$.

Pour décrire la transition entre deux états, il faut introduire pour chaque variable propositionnelle P une nouvelle variable P' qui représente la valeur de P dans l'état suivant. La relation de transition T sera donc représentée par la formule propositionnelle $\mathbf{T} = \bigvee_{(q,q')\in T} \mathbf{Q}(q) \wedge \mathbf{Q}(q)'$. Comme nous avons que les deux transitions suivantes : (*q0*, *q1*) et (*q1*, *q0*), nous aurons donc $\mathbf{T} = (\mathbf{Q}(q0) \wedge \mathbf{Q}(q1)') \vee (\mathbf{Q}(q1) \wedge \mathbf{Q}(q0)')$.

Il existe deux approches algorithmiques (Villemaire, 2009) en vérification symbolique de modèles, qui sont les BDD et la méthode SAT. Un BDD (Velev et Bryant, 2003) est simplement un graphe orienté acyclique dont les noeuds internes indiquent une décision TRUE ou FALSE sur une variable propositionnelle et les feuilles l'évaluation

TRUE ou FALSE. Ce type de structure permet une représentation compacte des formules booléennes. De son côté, la méthode SAT (Clark Barrett *et al.*, 2009) est une méthode algorithme efficace pour trouver une solution à une formule booléenne. Elle peut être utilisée pour résoudre les problèmes sur les formules obtenues lors de la vérification de modèles.

2.8 Diagramme de décision binaire (BDD)

Les algorithmes de la vérification de modèles (vérification formelle) ont été améliorés dans les dernières années notamment les articles (Burch *et al.*, 1990) et McMillan. (1993) qui montrent une nouvelle méthode de la vérification de modèles symbolique. Cette méthode représente implicitement le graphe des états d'un modèle fini sous forme d'une formule de la logique proportionnelle. Elle utilise le codage booléen pour représenter les états de la machine. Par conséquence, la manipulation de cette méthode permet de regrouper l'ensemble des états sous forme d'un simple état tout en explicitant l'énumération des états. Donc, Il s'agit de décrire l'ensemble des transitions par une formule qui exprime la relation entre l'état présent et l'état prochain. Toutes les opérations vont être effectuées en tant que des fonctions booléennes en utilisant les diagrammes de décision binaires (BDD).

De façon générale, les BDD sont une forme canonique pour les formules booléennes plus compacte que la forme normale de la conjonctive ou disjonctive (Velev et Bryant, 2003). Un BDD est une représentation de la formule booléenne sous forme de DAG (Graphe Orienté Acyclique). Il existe des algorithmes efficaces qui permettent de calculer les opérations booléennes sur les BDD (Velev et Bryant, 2003).

Pour réaliser la vérification d'un modèle, on peut donc représenter l'ensemble des états initiaux ainsi que la relation de transition par des BDD. Il est alors possible de vérifier des propriétés LTL et CTL (Biere *et al.*, 1999). Nous allons illustrer le principe de cette vérification sur l'exemple sur la figure 2.14.

La propriété à vérifier est évaluée récursivement par itération en calculant un point fixe. Considérons par exemple, la propriété de sûreté (Villemaire, 2009) **G ! erreur**, qui affirme que la variable booléenne **erreur** est fausse dans tous les états du système. Le calcul s'effectue de la façon suivante. On calcule itérativement l'ensemble des états accessibles, en débutant avec le BDD représentant l'ensemble des états initiaux. À chaque étape, on ajoute au BDD l'ensemble des états qui sont atteints par une transition de plus. Comme nous n'avons qu'un nombre fini d'états, cette procédure finit par se stabiliser et on obtient une représentation de l'ensemble de tous les états accessibles. Si l'intersection de cet ensemble d'états avec l'ensemble des états qui satisfont **erreur** n'est pas vide, on a un état accessible qui viole la condition et donc une trace qui viole **G !erreur**. Sinon, aucun état accessible ne viole la condition et celle-ci est donc vérifiée.

Cette méthode symbolique a été appliquée avec succès à la vérification de plusieurs systèmes complexes (Burch *et al.*, 1990), (McMillan., 1993) et (Vassy, 2004).

2.9 Vérification de modèles bornée (BMC)

La vérification de modèles bornée (Bounded Model-Checking, BMC) est une méthode alternative qui peut éviter l'explosion de l'espace d'état. Cette méthode a été proposée par (Biere *et al.*, 1999), (Burch *et al.*, 1991) et (Clarke *et al.*, 2001). L'idée basique de cette méthode est de chercher un *contre-exemple* sous la forme d'une exécution bornée. Cette méthode cherche donc une exécution d'une longueur bornée m, violant la propriété à vérifier.

Le BMC consiste à construire une formule proportionnelle qui représente une trace débutant dans un état initial et évoluant selon la relation de transition du système. Elle construit également une formule exprimant la violation d'une propriété P dans une de ces étapes. Lorsque la conjonction des deux formules est vérifiée, on a un *contre-exemple* d'une longueur m. Si non (la conjonction n'est pas satisfaite), il faut augmenter la longueur m jusqu'à ce que nous trouvions un *contre-exemple*. Donc, il peut être nécessaire

de faire la vérification avec des bornes m de plus en plus grandes, la méthode est donc incomplète. Elle ne peut donc pas vérifier qu'une propriété est vraie, mais seulement trouver un *contre-exemple* pour une propriété fausse.

La figure 2.14 ci-dessous illustre une trace de longueur m bouclant en k. L'équation ci-

FIGURE 2.14: Trace de longueur m bouclant en k (Villemaire, 2009).

dessous montre une formule proportionnelle qui encode une trace (Villemaire, 2009). Soit $\overline{R_1} = P_{11}, ...P_{1n}$ les variables booléennes qui encodent un état du modèle. Ainsi chaque affectation de valeur de vérité aux variables représente un état potentiel du système. Les valeurs représentant des états initiaux sont donc celles rendant $\mathbf{I}(\overline{R_1})$ satisfaite. Donc $\overline{R_1}$ représente l'état initial dans la formule ci-dessous. Comme le codage représente une trace de longueur m, nous aurons besoin d'un compteur i pour pouvoir représenter les états de 1 à m, représentés par $\overline{R_i}$ et $\overline{R_{i+1}}$ $(i = 1, ..., m)$. Ces états sont les états qui viennent après les états initiaux. Nous avons donc bien la représentation suivante pour la trace de longueur m.

$$I(\overline{R_1}) \wedge \bigwedge_{(i=1)}^{(m-1)} T(\overline{R_i}, \overline{R_{i+1}}) \wedge T(\overline{R_m}, \overline{R_k}) \tag{2.1}$$

2.10 La Méthode SAT et l'algorithme DPLL

Les algorithmes de résolution de formules propositionnelles les plus populaires sont basés sur la procédure de Davis-Putnam-Logemann-Loveland (DPLL) (Clark Barrett *et al.*, 2009). Cette procédure est basée sur les algorithmes de recherche «backtracking». En particulier, chaque noeud représente l'affectation d'une variable booléenne. Après

chacune de ces affectations, l'algorithme calcule les implications immédiates (appelées propagations d'unité), qui sont les variables qui ne peuvent plus que prendre une seule des deux valeurs booléennes.

Par exemple, si la décision $A1 = 1$ est effectuée et que la formule $(\neg A1 \vee A2)$ doit être satisfaite, on a immédiatement que $A2 = 1$. Ceci peut d'ailleurs impliquer d'autres affectations. L'application itérative de la propagation d'unités est généralement désignée sous le nom de la propagation de contraintes booléennes (Boolean Constraints Propagations, BCP). Lorsque nous rencontrons une contradiction, l'algorithme revient au dernier choix. Par exemple, si la formule contient également la clause $(\neg A1 \vee \neg A2)$, la décision $A1 = 1$ doit être changée, et les implications de la nouvelle décision doivent être recalculées.

Il existe d'autres méthodes qui permettent de résoudre le problème de satisfiabilité de formules propositionnelles, mais de façon générale, la procédure DPLL est classée aujourd'hui parmi les meilleures solutions. À l'origine, DPLL a été conçu comme algorithme de décision pour la logique de premier ordre, mais a été employé presque exclusivement pour la logique propositionnelle en raison du traitement fortement inefficace des quantificateurs. Pour plus de détails concernant l'algorithme DPLL nous vous invitons à vous référer à (Clark Barrett *et al.*, 2009) et (Baumgartner et Tinelli, 2008).

2.11 Comparaison des BDD et de SAT

Le BMC et les BDD sont deux méthodes différentes qui se basent sur la vérification symbolique. En effet, il est souvent possible de réaliser une vérification avec la méthode du BMC et pas avec les BDD, et parfois vice-versa. En fait, l'inconvénient principal du BMC est que la méthode cst incomplète, car elle ne peut donner qu'un contre-exemple. Notamment, l'absence d'erreur après l'exécution d'une longueur bornée m ne prouve pas l'absence de *contre-exemple*. La seule solution offerte par le BMC consiste à vérifier le modèle sur une longueur plus grande.

Pour chaque formule logique il y a un et un seul BDD. De plus, on peut vérifier

l'équivalence logique entre les BDD en temps constant. Le problème essentiel est donc de produire le BDD. Néanmoins, la taille d'un BDD est fortement dépendante de l'ordre choisi pour les variables. Il y a même des formules qui sont toujours représentées par des « gros » BDD quelque soit l'ordre choisi. En pratique, l'obstacle principal est donc de réussir à construire le BDD sans déborder de la mémoire, ce qui parfois impossible. C'est pourquoi cette approche est moins populaire aujourd'hui.

2.12 Applications : vérifications des protocoles de communications

D'après les recherches qui ont été faites, la vérification de modèles (vérification formelles) (Villemaire, 2009) a bien montré son utilité particulièrement pour vérifier des systèmes réels qui ont plusieurs comportements possibles. Notamment, les techniques de vérifications formelles ont été largement appliquées en télécommunication. En particulier sur les protocoles de communication, multimédia, sécurité informatique et les gestionnaires de l'énergie. Par exemple, les auteurs de l'article (Duflot *et al.*, 2010), ont étudié et analysé le protocole de réseau sans-fil IEEE 802.3 (CDMA/CD). Pour pouvoir considérer des aspects comme les contraintes temps réel ainsi que la retransmission après un temps aléatoire, ils ont étendu la vérification de modèles à des systèmes probabilistes, un sujet que nous ne couvrirons pas dans ce mémoire. On peut tout de même noter que leur approche intègre la vérification symbolique de modèles aux méthodes numériques et par simulation.

D'un autre coté, les auteurs de l'article (Artho *et al.*, 2007) utilisent la vérification de modèles pour pouvoir vérifier et contrôler les opérations d'entrée/sortie (Input and Output) dans un réseau de communication. Ils utilisent tout simplement une approche qui permet d'exécuter un processus simple de télécommunication sur un outil de vérification de modèles. En particulier, ils ont appliqué les techniques de vérifications sur les protocoles de l'envoie et la réception des messages pour pouvoir visualiser les différentes erreurs.

Par ailleurs, les techniques de vérifications ont été appliquées aussi sur les conceptions des composants électroniques ; par exemple les auteurs de l'article (Alur *et al.*, 1997) décrivent une application sur un outil de la vérification de modèles qui permet d'analyser en temps réel le défi d'un logiciel de la conception des systèmes téléphoniques Lucent Technologies' 5ESS.

Sans aller plus loin sur l'explication de ces articles, il existe beaucoup de travaux qui ont été faits en vérification de modèles dans tous les domaines. En particulier, les travaux suivants (Bérczes *et al.*, 2007), (Musuvathi et Engler, 2004), (Kwiatkowska *et al.*, 2002), (Artho et Garoche, 2006) et (Yuhong Yan et Cordier, 2009) présentent plusieurs applications de la vérification de modèles dans le domaine des réseaux. C'est d'ailleurs pour cette raison que nous nous sommes intéressés à appliquer la vérification de modèles aux télécommunications.

CHAPITRE III

MODÈLES DE ROUTEURS

Les métriques telles que le débit, le délai et le taux de perte sont devenues des critères de choix lors de la détermination de la qualité service. Ainsi, leurs valeurs définissent le niveau de qualité à satisfaire en ce qui concerne le réseau.

Bien que l'importance des métriques et des méthodes de mesures (Angrisani *et al.*, 2006) dépend de l'application étudiée, des éléments comme le délai et, la disponibilité de la bande passante suscitent beaucoup d'intérêt en raison du rôle qu'ils jouent dans l'optimisation du réseau *end-to-end.* Ceci est surtout vrai dans les cas où l'on requiert des applications en temps réel.

Dans notre cas, nous nous intéressons à l'évaluation du délai dans les routeurs. Pour se faire, nous avons revu la littérature, en particulier les articles (Angrisani *et al.*, 2006), (Papagiannaki *et al.*, 2003) et (Zerfos *et al.*, 2003). Par exemple les auteurs des articles (Angrisani *et al.*, 2006) et (Papagiannaki *et al.*, 2003) présentent une méthode d'évaluation du délai sur un routeur en fonctionnement normal tout en proposant une mesure bien efficace. L'article (Zerfos *et al.*, 2003), présente un simulateur dénommé «DIRAC» basé sur les routeurs du réseau sans-fil pour pouvoir faciliter l'implémentation et l'évaluation de divers protocoles de communication. L'architecture de ce simulateur se compose en deux parties : «Router Core» et «Router Agent», dont un ayant le rôle de partager le sous-réseau sans fil entre les sous-réseaux «subnet» et le second va se charger de donner l'accès à la station de base.

En plus des méthodes précédentes qui permettent de mesurer les délais dans les routeurs, plusieurs travaux ont plutôt utilisé des modèles de routeurs pour évaluer des métriques comme le délai.

D'après nos connaissances, les études les plus récentes sur les modèles de routeur ont été présentées dans les articles suivants : (Hohn *et al.*, 2004) et (McKeown, 1999). Par exemple, les auteurs de l'article (Hohn *et al.*, 2004) présentent une méthode qui permet de visualiser le comportement de tous les paquets traversant un simple routeur dans un temps déterminé tout en détaillant les statistiques nécessaires pour calculer les délais. Aussi ils ont créé une file d'attente virtuelle tout en ajoutant un délai constant pour

chaque paquet qui rentre dans la file d'attente pour pouvoir analyser le comportement d'acheminement des paquets d'une source à une destination. Ainsi, Ils se sont basés sur l'article (McKeown, 1999) qui donne une architecture détaillée sur la création des files d'attentes virtuelle FIFO de sorties. Ainsi un algorithme permet de configurer le « crossbar switch ».

Dans ce qui suit, les auteurs de (Chertov *et al.*, 2008) ont construit un modèle simple qui permet d'évaluer les comportements des réseaux et précisément les routeurs Cisco. Les chercheurs de (Chertov *et al.*, 2008) ont construit un outil de profilage qu'ils ont utilisé sur des tableaux de paramètres de routeur dans un intervalle de temps. Ils génèrent chaque fois un flux TCP et UDP tout en calculant avec un simulateur nommé «QSim» (Chertov *et al.*, 2008) les délais de traitement des paquets et les pertes de données tout en comparant les résultats obtenus de leur modèle par rapport aux routeurs réels. Ils ont constaté par la suite que leur modèle peut se rapprocher approximativement à des routeurs Cisco 3660 et Cisco7206VXR. La différence essentielle entre les articles (Hohn *et al.*, 2004), (McKeown, 1999) et (Chertov *et al.*, 2008), est que le modèle de (Hohn *et al.*, 2004) présente une méthodologie simple qui permet de calculer les délais des paquets traversant un routeur simple d'une source à une destination. Ces mesures sont appliquées sur le temps du traitement de chaque paquet traversant un routeur avant de les transmettre sur les files d'attente virtuelles FIFO de sorties. Ainsi, ils se sont fondés sur l'article (McKeown, 1999) qui présente une architecture détaillée sur la création des files d'attente virtuelles FIFO. Par contre, les auteurs de (Chertov *et al.*, 2008) présentent un modèle simple qui est basée sur un simulateur nommé «QSim» qui permet de calculer les délais du traitement de chaque paquet ainsi que les données engendrées tout en comparant leurs résultats obtenus par rapport à des routeurs réels.

Par ailleurs, nous avons constaté que l'article (Chertov *et al.*, 2008) est très représentatif, il a d'ailleurs été la source d'inspiration pour notre modèle que nous allons décrire dans le prochain chapitre.

3.1 Idée générale et objectif

Dans l'article (Chertov *et al.*, 2008), les chercheurs présentent un modèle basé sur les mesures des routeurs et d'autres dispositifs d'acheminement (forwarding). Ils ont simulé deux différents routeurs (Cisco3660 et Cisco7206VXR) avec une variation de trafic en utilisant une méthode de modélisation. Pour construire leur modèle indépendamment d'un routeur réel, les auteurs ont constaté que la nécessité de la connaissance de paramètres spécifiques aux routeurs est indispensable. Ils ont construit un outil de profilage qu'ils

ont utilisé sur des tableaux de paramètres de routeur dans un intervalle de temps donné. D'après les résultats de leurs simulations, les auteurs ont conclu que la performance de leur modèle se rapproche aux routeurs Cisco. Ils ont constaté aussi que la compacité des tableaux de paramètres et la simplicité de leur modèle leur permettent d'employer ce dernier sur des simulations de haute-fidélité. En effet, pour construit leur modèle, les auteurs se sont basés sur le modèle de l'article (Hohn *et al.*, 2004). Ainsi :

- Ils ont créé un modèle des files d'attentes virtuelles de sorties (VOQ) basé sur un certain nombre d'aspects importants comme : la taille de la file d'attente et le nombre de serveurs utilisés.
- Ils ont conçu et développé un processus de dérivation basé sur des mesures simples telles que le BBP (Chertov *et al.*, 2007) « Black-Box Profiler » afin de prendre les mesures des données pour chaque routeur.
- Un BBP est un SMP PC qui sert à générer une simulation de trafic et qui enregistre les paquets transmis.
- Ensuite, ils ont comparé les mesures de leur modèle par rapport à des mesures réelles sur des routeurs Cisco3660 et Cisco7206VXR. Ils ont constaté que le comportement de leur modèle est très proche de celui des routeurs Cisco.

3.2 Modèle indépendamment d'un routeur réel

En construisant leur modèle, les auteurs ont constaté qu'il était difficile de développer un modèle de routeur en raison de la multitude des architectures possibles et des performances requises à sa conception. Le routeur réel doit traiter des paquets de contrôle comme les ARP et ICMP ainsi que des paquets de cheminement tel que BGP, OSPF et RIP. Ces paquets peuvent avoir un impact profond sur l'acheminement (forwarding).

Notons que certains paquets peuvent être retardés ou perdus lors du transfert d'une source à une destination et que les routeurs peuvent avoir des interfaces avec différentes vitesses de trafic.

Ceci étant dit, on peut trouver un certains nombres de points communs entre les routeurs qui peuvent faciliter leur construction. Parmi ces éléments on peut citer :

1. Les paquets peuvent être perdus (dropped) ou retardés dans un routeur.
2. Les routeurs ont un certain nombre d'interfaces d'entrée/sortie.
3. Les routeurs peuvent avoir des files d'attente /tampons supplémentaires.
4. Le flux des paquets dans un routeur est complexe et peut avoir plusieurs files d'attente et serveurs.
5. Les paquets peuvent être divisés en petites unités dans le routeur.

Pour faciliter la construction de leur modèle, les auteurs ont pris en considération certains éléments :

1. Ils n'ont pas modélisé le contrôle de trafic (OSPF/BGP/ARP/ICMP, etc.).
2. Ils ont supposé que toutes les interfaces d'entrée/sortie ont les mêmes performances.
3. Ils ont supposé que les paquets de données traitées sont égaux (aucune Qualité de Service).

Puis, les auteurs se sont basés sur les similarités des routeurs qui sont décrites ci-dessus dans le modèle et ils ont commencé par décrire la fonctionnalité de leur modèle. Ainsi :

1. Chaque port de sortie a une taille de file d'attente fixe de Q emplacements (slots).
2. Chaque paquet rentre au système par un des N ports d'entrées.
3. Ce paquet sera classifié par un classificateur qui va déterminer la file d'attente de sortie appropriée QueueN.
4. Chaque file d'attente QueueN est bornée par une taille de Q emplacements (slots).
5. Un paquet peut occuper plus d'un emplacement, d'où, les auteurs ont constaté qu'ils ont besoin d'un tableau QueueRep qui représente le nombre d'emplacements occupés par un paquet.
6. Chaque paquet sera traité par un des M serveurs, afin d'être transmis par un des N ports de sorties.

7. Les M serveurs servent à traiter les paquets afin de déterminer et transmettre ces derniers sur les N ports de sorties appropriés en utilisant le délai moyen de traitement. Ceci nécessite l'utilisation d'un tableau DelaytTbl qui représente le délai de traitement dans les serveurs pour un paquet de taille s.
8. Les paquets peuvent être servis simultanément par différents serveurs. Ces derniers permettent de minimiser la taille de buffer de paquets avant d'être transmis sur le N ports de sorties.

Les auteurs ont constaté qu'il est difficile de déterminer le nombre exact de serveurs. Ainsi dans certains cas, les paquets ont besoin de plus d'un serveur afin d'être traités. À ce moment-là, les auteurs ont créé un autre tableau ServReq qui représente le nombre de serveurs nécessaires pour traiter les paquets de différentes tailles. Notons que, les tailles des files d'attentes décroisent selon le nombre de serveurs déclarés dans le modèle.

Finalement, les auteurs ont utilisé un algorithme qui permet de calculer les paramètres suivants DelayTbl, Q, M, QueueReq, et ServReq. D'après l'analyse de cet algorithme, le modèle est basé sur les délais des paquets et sur les pertes de données.

3.3 Modèle de fidélité

Pour évaluer leur modèle, les auteurs ont conçu un simulateur « QSim » (leur modèle) qui contient plusieurs files d'attentes et plusieurs serveurs. Ce simulateur utilise cet algorithme pour calculer les paramètres du modèle et comparent chaque fois les résultats trouvés par rapport aux délais et aux pertes de données déjà déterminées.

Dans le modèle de fidélité, le simulateur « QSim » a permis de modéliser les routeurs Cisco3660 et Cisco 7206VXR tout en comparant les délais de paquets ainsi que les pertes de données.

Par exemple, « QSim » a permis de mettre en évidence que le routeur Cisco7206VXR ne peut pas transmettre 64 bits sur 4 interfaces et de prévoir avec beaucoup d'exactitude

les délais de paquets dans la file d'attente lorsqu'il y aura une combinaison de deux flux de données sur un seul port de sortie.

Pour vérifier l'efficacité de leur modèle, les auteurs ont développé plusieurs scénarios complexes :

- Les auteurs ont généré avec NS2 un trafic de 30 TCP et 5 UDP sur chaque noeud d'entrée qui sont (node0, node1, node3) destiné au node2. Ils ont essayé d'exécuter cette expérimentation sur les deux routeurs Cisco pendant un temps de 200 secondes dans la réalité, puis ils ont refait l'expérience dans leur simulateur en utilisant les paramètres de leur modèle. Les résultats trouvés montrent que leur modèle permet de minimiser la perte de données en comparaison avec les routeurs réels. (Donc ici, on a une différence entre le modèle et les routeurs).
- Chaque fois, que les auteurs génèrent un trafic TCP et UDP sur les routeurs Cisco en utilisant les paramètres de leur modèle, ils utilisent enfin « QSim » pour calculer et comparer les délais et les données perdues déjà déterminées.
- D'après les expériences, les auteurs conclurent que leur modèle est raisonnablement précis et très proche à un routeur réel mais n'est pas toujours parfait particulièrement dans les cas où le fond de panier (backplane) est dominant.

3.4 Conclusion

Nous pourrons conclure que les résultats expérimentaux de ce chapitre montrent qu'on peut approximer un routeur assez bien avec un modèle et que nous allons voir au chapitre suivant justement en développer un, mais dans l'optique de la vérification de modèles.

CHAPITRE IV

MODÈLE-CHECKING DU DÉLAI

L'objectif de ce chapitre est de présenter un modèle qui permet d'évaluer les délais des paquets dans les routeurs par l'approche de vérification de modèles (model-checking). Après avoir étudié de près la littérature (Chertov *et al.*, 2008), (Hohn *et al.*, 2004), (McKeown, 1999), (Angrisani *et al.*, 2006), (Papagiannaki *et al.*, 2003) et (Zerfos *et al.*, 2003), nous nous sommes inspirés du modèle de l'article (Chertov *et al.*, 2008) qui évalue les délais de traitements dans les routeurs. Notre modèle sera donc composé de N ports d'entrées, un classificateur, des files d'attente de sorties, des serveurs et N ports de sorties. Chaque paquet qui rentre au système sera classifié selon la file d'attente de sortie appropriée. Ensuite, le paquet sera traité par un serveur. Enfin, il sera transmis sur un port de sortie.

La différence entre les modèles pour des outils de simulation tels que OPNET (OPNET, 1986) et OMNET++ (OMNeT++, 1992) ou encore QSim (Chertov *et al.*, 2008) et notre modèle, est qu'en vérification de modèles, on vérifie des propriétés logiques en procédant par vérifications exhaustives plutôt que par simulation. Dans notre cas, ceci permet la vérification et l'évaluation du délai dans tous les états de la trace en donnant chaque fois la valeur du délai du pire cas possible. Alors que, la simulation permet de générer des statistiques en produisant une suite de paquets et en évaluant leur traitement dans le modèle de routeur.

4.1 Le délai dans les routeurs

L'objectif de notre travail est de construire un modèle qui permettra d'évaluer les délais des paquets traversant un routeur d'une source à une destination. Sachons que le délai de routeur se compose de trois éléments qui sont : le délai de transmission, le délai de traitement et le délai de la file d'attente. Notons que dans notre modèle, nous nous

concentrons sur le délai de la file d'attente. La vérification formelle de notre modèle sera donc basée sur l'évaluation du délai des paquets dans les files d'attente en fonction du débit d'entrée. Rappelons que la différence entre la simulation et notre approche est qu'en vérification de modèles, on calcule le pire délai possible c'est-à-dire le délai maximal alors qu'en simulation, on calcule le délai moyen.

4.2 Analyse et architecture du modèle

Globalement, la structure de notre modèle suit celle de l'article (Chertov *et al.*, 2008). Les différences essentielles avec le nôtre sont : la méthode de génération du trafic, l'implémentation dans un langage restrictif où les structures sont finies et bornées (voir chapitre II) et l'évaluation qui tient compte uniquement du délai de la file d'attente. Notre modèle suit donc la structure générale illustrée à la figure 4.1. Il génère tout d'abord des paquets sur les ports d'entrées qui seront classifiés sur des files d'attente. Ensuite, des serveurs traitent les paquets et diminuent la taille des files. Enfin, les paquets seront transmis sur les ports de sorties appropriés.

Dans cette section, nous allons décrire les composantes principales de notre modèle.

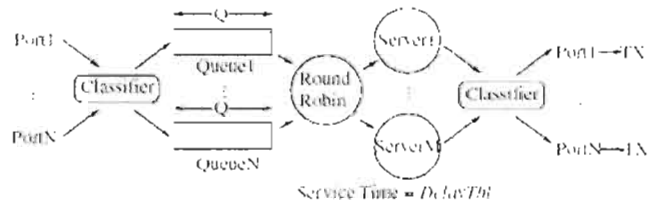

FIGURE 4.1: Présentation du modèle de routeur (Chertov *et al.*, 2008).

4.2.1 Génération du trafic

En simulation, la génération du trafic se fait aléatoirement sur les ports d'entrées avec différentes distributions. Alors qu'en vérification, nous devons concevoir un processus

déterministe, fini et borné. De plus, la vérification de modèles (model-checking) est une vérification exhaustive, donc le modèle tiendra toujours compte du pire cas possible. Si nous modélisons la génération de trafic en donnant chaque fois une borne au débit sur les ports d'entrées, le pire cas possible sera le débit maximal. Normalement ce dernier donnera lieu aux débordements des files d'attente et donc à un délai maximal qui ne sera fonction que de la taille des files d'attente. Ce qui n'est pas satisfaisant.

Il nous faut donc trouver une méthode déterministe qui permettra de générer un débit variant autour d'une moyenne. Pour résoudre ce problème nous nous sommes inspirés de (Soviani *et al.*, 2009) où les auteurs considèrent une architecture de pipeline qui se compose par des modules électroniques simples. Ces modules sont reliés par des files d'attente FIFO d'entrées/sorties. Ceci permet de minimiser le couplage entre les modules. Chaque module écrit avec un rythme différent sur la file d'attente pour pouvoir transmettre les informations d'un module à un autre. Si on peut estimer les débits d'entrée et de sortie d'une file, il devrait être possible d'estimer quelle est la longueur minimale de la file assurant qu'il n'y ait pas de perte. L'objectif des auteurs consiste justement à trouver une approche qui détermine les longueurs minimales des files d'attente FIFO en fonction des débits d'entrée et de sortie en utilisant la vérification de modèles bornée (bound-model-checking). Pour arriver à cela, les auteurs ont conçu une méthode efficace qui permet de générer automatiquement une suite de paquets sur les interfaces d'entrées. Ainsi, ils ont développé un algorithme dont la fonction est de calculer les longueurs minimales des files d'attente FIFO qui permet d'achever un débit maximal à la sortie des files et qui permet aussi de ne pas avoir des pertes sur ces dernières. Cette méthode produit successivement des motifs de longueurs fixes (L) qui déterminent l'arriver des paquets sur les interfaces d'entrées. Ces motifs se divisent sur un certain nombre des cycles (L) et qui peuvent être représentés par une suite binaire de 0 et 1. Ici 1 représente l'arrivée d'un paquet dans la file d'attente FIFO et 0 représente l'absence d'arrivée d'un paquet dans la file FIFO. Notons, que le débit d'entrée dépend du nombre des 1 dans le motif. Par exemple, le motif 1010 correspond à un débit de 50% de la capacité maximale.

Nous avons appliqué une approche similaire à celle de l'article (Soviani *et al.*, 2009)

tout en construisant un système qui permet de fixer et comptabiliser un nombre déterminé de paquets arrivant dans un port d'entrée afin d'être transmis sur la file d'attente correspondante.

Pour développer cette approche en NuSMV, nous avons réalisé un module générateur dont la fonction principale est de produire périodiquement des motifs de longueurs fixes (immobiles) qui déterminent l'arriver des paquets sur les interfaces d'entrées. Ainsi à chaque cycle, la présence d'un 1 dans le motif déterminera l'arrivée d'un paquet alors que celle d'un 0, l'absence d'arrivée d'un paquet. De plus, à la fin de la lecture du motif, un nouveau motif (un pour chaque file de sortie) sera généré de façon non-déterministe.

De plus, nous utilisons toujours un nombre fixe de 1 dans les motifs d'un même modèle. Ainsi avec une longueur de motif de L et un nombre de 1 par motif de U, nous aurons un débit moyen de U/ L (que nous exprimerons normalement en pourcentage). Finalement comme le choix des motifs consécutifs est non-déterministe, nous aurons une certaine variation autour de cette moyenne, en fonction de la recherche d'un contre-exemple par l'outil de vérification de modèles.

Notons que :

- Tous les motifs ont le même nombre des cycles (L).
- Chaque file d'attente possède son propre motif.

Par exemple, pour un débit d'entrée égal à 25% et une longueur des motifs L = 4, nous pourrons prendre les quatre motifs de la figure 4.2 ci-dessous. Ainsi sur chaque motif nous aurons quatre cycles dont un contient un nombre binaire égal à 1 et qui représente un paquet qui va être rentré dans la file d'attente (c'est-à-dire (1 paquet) pour les 4 cycles de la longueur de motif (L)), par contre les autres contiennent des nombres binaires égal à 0 et qui représentent aucune arrivée de paquet dans la file d'attente.

Pour un débit d'entrée égal à 50% avec une longueur de motif L = 4, nous aurons deux arrivées de paquets par motif (c'est-à-dire (2 paquets) pour les 4 cycles de la longueur de motif (L)). Notons que dans ce cas, nous pouvons avoir six motifs comme indiqué dans

la figure 4.3 ci-dessous. Par ailleurs, le choix final du débit moyen utilisé dans un modèle particulier se fera en fonction du nombre de file de sortie et du nombre de serveurs, comme nous le verrons plus loin.

	C4	C3	C2	C1
Motif1=1	1	0	0	0
Motif1=2	0	1	0	0
Motif1=3	0	0	1	0
Motif1=4	0	0	0	1

FIGURE 4.2: Exemple d'un débit d'entrée de 25 %.

	C4	C3	C2	C1
Motif1=1	1	1	0	0
Motif1=2	0	1	1	0
Motif1=3	0	0	1	1
Motif1=4	1	0	0	1
Motif1=5	0	1	0	1
Motif1=6	1	0	1	0

FIGURE 4.3: Exemple d'un débit d'entrée de 50 %.

4.2.2 Les files d'attente

Notre deuxième composante est d'intégrer une file d'attente à chaque port de sortie. Mais comment peut-on trouver une méthode qui nous permet de modéliser les files d'attente en vérification de modèles (model-checking) tout en permettant une vérification de

modèles efficace ?

Une évaluation exhaustive des différentes méthodes d'implémentation des files dans des modèles de vérification formelle ainsi que leurs impacts sur la performance de la vérification a été faite par (Junttila et Dubrovin, 2008). Dans cet article, les auteurs ont comparé trois différentes approches usuelles d'implémentation de files d'attente et les ont mis-en-oeuvre en NuSMV.

Rappelons (voir chapitre II) que la complexité de la vérification de modèles est essentiellement fonction de la structure de la relation de transition du modèle. Il est donc possible qu'un choix judicieux de mise-en-oeuvre des files d'attente permette une vérification beaucoup plus efficace.

Pour développer cette idée en NuSMV, les auteurs de l'article (Junttila et Dubrovin, 2008) encapsulent chaque file d'attente derrière une interface unifiée qui permet d'accéder au contenu de la file d'attente. Ainsi ils auront la possibilité d'expérimenter avec différents codages de la file tout en gardant les mêmes codages du système.

Pour construire l'interface (Junttila et Dubrovin, 2008), les auteurs supposent que la file d'attente suit une loi FIFO (first-in, first-out). L'interface est composée de deux variables booléennes et deux pointeurs. Les deux variables booléennes déterminent quelle(s) opération(s) parmi enfiler et défiler seront effectuée au prochain cycle. Par contre les deux pointeurs indiquent la position du premier élément (pour défiler) et celle du prochain emplacement libre (pour enfiler). En outre, le client est à même de connaître de prime abord l'état de la file, par exemple si elle est vide ou pleine. Pour plus de détails, nous vous invitons à consulter les articles suivants (Gary, 2003), (Sheeran *et al.*, 2000) et (Een et Sörensson, 2003).

Maintenant, nous allons présenter les différentes approches qui ont été proposées dans l'article (Junttila et Dubrovin, 2008) pour réaliser la file d'attente.

1. Approche de décalage : Cette approche est basée sur le décalage des éléments de la file lors du défilement. Lors de l'enfilement, il suffit d'insérer dans le prochain

emplacement libre (dans la mesure où la file n'est pas pleine). L'indicateur de début de file est ainsi ici toujours la première position de la file.

2. Approche cyclique : Cette approche se base sur le fait que les deux pointeurs qui indiquent la position du premier élément est celle du prochain emplacement libre se déplacent de façon cyclique. Lorsqu'on enfile, le pointeur de fin s'incrémente de un et lorsqu'on défile, le pointeur de tête s'incrémente de un.
3. Approche linéaire : Cette approche se fonde sur deux pointeurs comme la méthode cyclique mais elle est basée sur l'approche des fonctions non-interprétées (UIF) (voir section 3.2 page 297 de l'article (Junttila et Dubrovin, 2008)). Comme cette approche n'est pas supportée par NuSMV, nous n'irons pas plus avant dans cette direction.

Remarquons qu'en principe, la complexité de la relation de transition sera moindre si la méthode de mise-en-oeuvre déplace les éléments de la file le moins possible. A priori l'approche «shifting» qui ne déplace jamais les éléments après leurs insertions devrait être la plus efficace en vérification de modèles. Néanmoins les auteurs de (Junttila et Dubrovin, 2008), ont déterminé que les performances des différentes approches sont essentiellement les mêmes, donc que le choix d'une méthode de mise-en-oeuvre des files n'est pas critique pour la vérification de modèles. Notre choix s'est donc portée sur l'approche de décalage «shifting», car elle est naturelle et s'implémente assez facilement en NuSMV, comme nous allons le voir.

Pour la réalisation en NuSMV nous avons réalisé les files d'attente comme des tableaux de type : *array*. Nous pouvons de plus distinguer trois cas possibles, ce qui est utile vu que l'implémentation en NuSMV, se base sur des cas (condition « case »). Les trois cas sont :

1. Enfiler seulement.
2. Défiler seulement.
3. Défiler et enfiler.

1. Enfiler seulement : Si la file d'attente n'est pas pleine et qu'un paquet est reçu sur l'interface, le pointeur de fin sera incrémenté de un (déplacement vers la gauche dans la figure 4.4 ci-dessous).

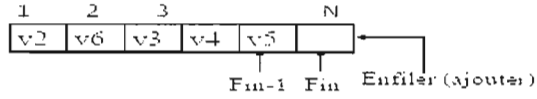

FIGURE 4.4: File d'attente dans le cas enfiler seulement.

2. Défiler seulement : Si la file d'attente n'est pas vide et qu'elle est servie par un serveur, tous les éléments de la file sont décalés de un vers la gauche (voir la figure 4.5).

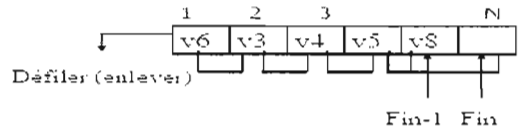

FIGURE 4.5: File d'attente dans le cas défiler seulement (v2 = défiler, v8= enfiler).

3. Défiler et enfiler : Si la file d'attente est servie par un serveur et que simultanément un paquet est reçu sur l'interface, tous les éléments de la file sont décalés de un vers la gauche alors que le paquet arrivé est inséré à droite à la position fin-1. La position de fin ne change donc pas.

Par ailleurs, l'implémentation par un langage de modélisation et vérification comme le NuSMV est un peu difficile par rapport à d'autres langages de programmation comme le C++ et le Java, vu que la description de ce langage est un peu limitée et restrictive, ce qui nous oblige de s'orienter chaque fois au manuel de NuSMV pour pouvoir faciliter l'exécution en vérification de modèles bornée (bound-model-checking).

En effet, nous avons rencontré quelques difficultés pour remplir les files d'attente par des variables d'entrée (paquets) lors de description en NuSMV, vu que ce dernier

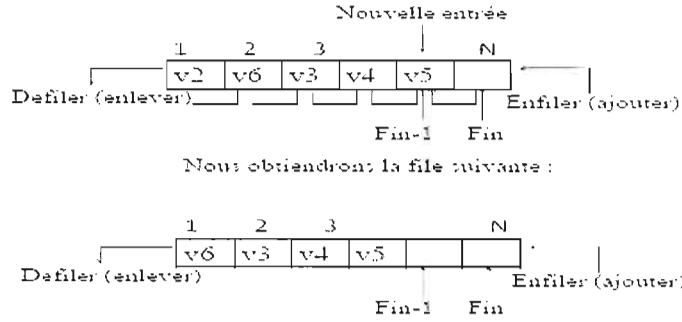

FIGURE 4.6: File d'attente dans le cas défiler et enfiler (v2=défiler, v5=enfiler).

n'accepte pas certaines expressions de tableaux comme par exemple tab[v] avec v une variable, ni les expressions récursives. Nous avons dû conformément au manuel NuSMV (voir (Cavada *et al.*, 2007)) version 2.4 page 21 utiliser les macros et le préprocesseur M4 qui nous permettent de générer tout le code nécessaire au développement de notre modèle.

4.2.3 Les serveurs

La responsabilité des serveurs est de prendre les paquets des files d'attente venant de l'interface d'entrée et les transmettent aux ports de sorties. Alors, la grande question qui se pose dans cette partie est comment trouver une approche pour pouvoir traiter et diminuer les tailles des files d'attente afin de transmettre les paquets aux ports de sorties appropriés ?

Pour se faire, nous avons constaté que les serveurs de l'article (Chertov *et al.*, 2008) traitent les paquets des files d'attente simultanément et parallèlement selon un processus « Round-Robin ». Dans notre modèle, nous avons suivi cette approche. Rappelons qu'un processus de « Round-Robin » assure une répartition de charge équitable entre le traitement des files. Les serveurs ne traitent jamais une file F de nouveau avant d'avoir traité toutes les files qui étaient non-vides au moment du dernier traitement de F. Par exemple avec deux serveurs, le processus de fonctionnement sera comme suit :

- Lorsque le premier serveur traite une file d'attente, le deuxième servira la prochaine file non-vide
- Les serveurs font leurs choix de files à servir simultanément pour pouvoir traiter des files d'attente différentes. Ainsi, nos serveurs traitent les paquets parallèlement avec équité (une file ne pouvant être bloquée indéfiniment par le traitement des autres).

4.3 Les paramètres du modèle

Rappelons que notre modèle NuSMV, consiste en un générateur de trafic, des files d'attente qui suivent une loi FIFO et des serveurs « Round-Robin » qui traitent les paquets venant des files d'attente.

Les paramètres principaux utilisés dans notre modèle sont donc :

- Le nombre de ports d'entrée et de sortie N (un entier) ;
- Les ports d'entrée (i1, i2,...,iN) ;
- Les files d'attente (f1, f2,..., fN) ;
- Le nombre de serveurs S ;
- Les serveurs (serveur1, serveur2,..., serveurS) ;
- Un compteur de temps global nommé horloge dont la fonction est de comptabiliser les délais des paquets dans les files d'attente.

4.4 Implémentation du modèle en NuSMV

Dans cette section, nous décrierons la structure et le fonctionnement de notre modèle NuSMV.

L'étude et l'analyse des intervenants potentiels de ce modèle ainsi que les différents scénarios possibles et le recueil des informations sur le domaine ont permis de distinguer quatre modules potentiels.

Le processus de fonctionnement de notre modèle se fait comme suit : D'abord, à chaque cycle, des paquets rentrent par les ports d'entrée et sont classifiés sur les files d'attente correspondantes. En fait plutôt que de mettre un paquet dans une file, on y insère plutôt la valeur de l'horloge globale qui est elle incrémentée à chaque cycle. Finalement lorsqu'un paquet est traité par un serveur, le système détermine le délai en soustrayant de la valeur actuelle de l'horloge celle qui est défilée.

Les modules NuSMV sont donc les suivants :

- Module générateur : Ce module génère les motifs d'arrivées des paquets.
- Module file : Ce module contient les files d'attente de sortie réalisées sous forme des tableaux et un classificateur qui détermine sur quelle file d'attente mettre les paquets.
- Module serveur : Ce module contient des serveurs. Chaque serveur indique quelle file d'attente sera traitée au prochain tour.
- Module main : Ce module est le programme principal. Il contient en plus de l'instanciation de tous les autres modules et l'horloge globale du système.

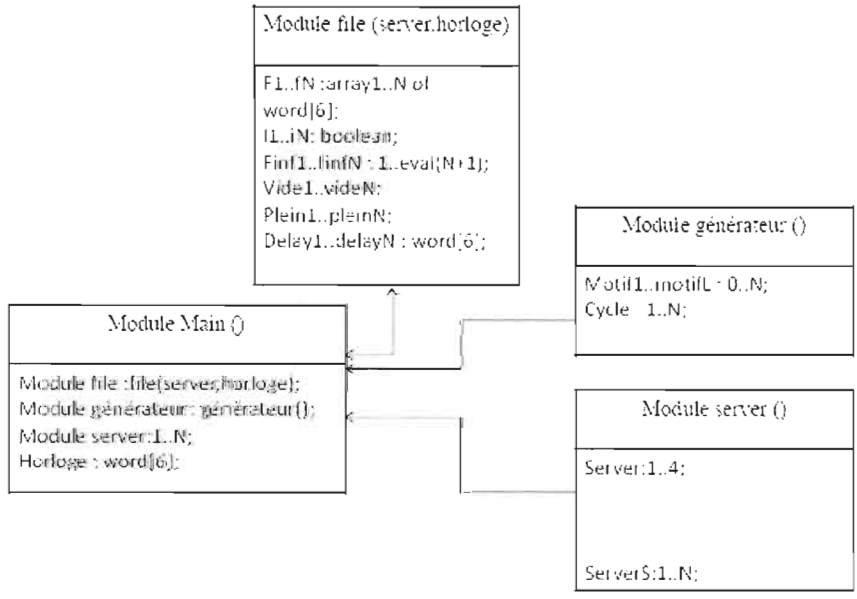

FIGURE 4.7: Conception du modèle.

La figure 4.7 ci-dessous, nous montre le diagramme de classe globale de notre modèle qui est constitué de cinq modules (où classes). Ce diagramme de classe constitue une ébauche du schéma de la base de notre modèle en représentant les objets qui vont intervenir au niveau des fonctionnalités de notre modèle.

4.5 Conclusion

Dans ce chapitre nous avons présenté :

- L'architecture et l'analyse de notre modèle.
- La définition détaillée de l'approche choisie pour le codage de la file d'attente, ainsi que sa justification.
- Détaillé l'implémentation en cinq modules NuSMV, de notre modèle.

Dans le chapitre suivant, nous allons mettre en place des propriétés logiques de vérification qui nous permettent d'évaluer les délais des paquets dans les files d'attente afin de produire des données expérimentales.

CHAPITRE V

RÉSULTATS EXPÉRIMENTAUX

5.1 Méthode d'évaluation

Après la présentation de la structure de notre modèle et le processus de son fonctionnement, notre objectif sera donc d'évaluer les délais des paquets dans les files d'attente. Pour se faire, nous devrons répondre à certaines questions importantes telles que :

1. Calculer le délai dans les files d'attente par une propriété logique (section 5.2).
2. Concevoir une méthodologie générale pour pouvoir trouver le délai maximum (section 5.3).
3. Évaluer la performance de la méthode en fonction des paramètres les plus importants (section 5.5, 5.6 et 5.7).
4. Évaluer la méthode sur des exemples représentatifs (section 5.8).

5.2 Propriétés LTL

Dans cette section, nous présentons la propriété logique qui permet d'évaluer et de calculer les délais des paquets dans les files d'attente. Ensuite, nous lancerons l'exécution de notre modèle pour voir si la propriété est satisfaite ou non.

Rappelons que la vérification de modèles nécessite une propriété en logique temporelle, par exemple LTL. En effet, la signification de la propriété LTL sur la figure 5.1 ci-dessous, permet de vérifier si le délai est inférieur ou égal à 1 cycle dans tous les états de la trace.

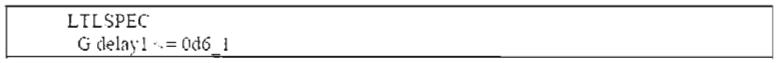

FIGURE 5.1: Formule LTL.

De façon concrète, la propriété LTL permet de borner le délai sur une valeur déterminée. Nous dénoterons cette borne par D et nous considèrerons les formules *LTL G delay <= D*, avec différentes valeurs de D. L'exécution en vérification de modèles revient justement de chercher un contre-exemple. Dans notre cas ceci revient à exhiber une trace dans laquelle la variable delay prend une valeur plus grande que D. Ainsi si la vérification de modèles bornée (BMC) trouve un contre-exemple, une trace où le délai maximal est supérieur à D sera produite. S'il n'y a pas de contre-exemple, comme la vérification de modèles bornée fait la recherche pour une longueur de trace fixée K, il est possible qu'un contre-exemple existe pour une borne plus grande (cette question sera traitée plus en détail à la section suivante 5.4).

5.3 Méthodologie de l'évaluation du délai

Nous allons maintenant présenter notre approche d'évaluation du délai des paquets dans les files d'attentes. Notre méthodologie est basée sur deux paramètres importants : D (la borne de délai apparaissant dans la formule LTL, par exemple la valeur égale à 1 dans la figure 5.1) et K (la longueur du contre-exemple cherché) qui nous permettent de vérifier et d'évaluer un délai borné D de façon efficace.

De façon concrète, si lors de la découverte d'un contre-exemple avec une certaine valeur de D, on recommence la vérification avec D augmenté de 1, il est possible d'estimer correctement le délai maximal. Néanmoins, la situation n'est pas si évidente lorsque nous n'arriverons pas à trouver un contre-exemple. Ici aussi la longueur K de la trace peut être encore augmentée de 1. Mais en pratique, s'il n'y a toujours pas de contre-exemple, il faudra arrêter la vérification sur une certaine valeur de K, que nous allons choisir assez

grande par rapport aux longueurs des traces des contre-exemples trouvés précédemment.

Par ailleurs, l'objectif est de trouver le plus petit D pour lequel aucun contre-exemple n'existe. Comme nous l'avons vu, il est possible d'incrémenter D de 1, mais il est possible en fait de trouver le délai maximal beaucoup plus rapidement. Nous avons donc plutôt utiliser la méthode suivante qui nous permet d'estimer le délai maximal :

Au début, nous choisissons un délai borné D avec une longueur de la trace K. Ensuite, nous lancerons l'exécution de notre modèle en vérification de modèles bornée. Si nous ne trouverons pas de contre-exemple, nous allons augmenter la borne K, car il peut y avoir un contre-exemple pour une longueur de trace K plus élevée. Évidemment il est possible qu'aucun contre-exemple n'existe. Nous allons donc devoir choisir une valeur maximale de K à partir de laquelle la recherche d'un contre-exemple est abandonnée.

Dans le cas où nous aurons un contre-exemple, nous allons extraire de la trace le délai maximal Dmax y apparaissant. Comme la trace est un contre exemple, on a nécessairement que Dmax est supérieur à D. De plus, cette trace montre déjà que le délai peut être égal à Dmax, nous relancerons donc la recherche d'un contre-exemple, mais cette fois-ci avec $D = Dmax + 1$. Ceci nous permettra de minimiser le nombre total de valeur de D différentes utilisées et donc de réduire le temps total de vérification.

5.4 Évaluation expérimentale du délai

Dans cette partie, nous allons expliquer comment nous pourrons produire les données expérimentales à partir de vérification de modèles (model-checking). L'idée est tout simplement d'évaluer le délai sur plusieurs cas différents (modèles). Cependant, certains cas offrent beaucoup de serveurs par rapport au débit d'entrée et devrait donc donner des délais très petits alors que dans d'autres cas le petit nombre de serveurs devrait permettre des délais plus importants. Donc il s'agit de couvrir plusieurs cas différents. Nous allons donc présenter :

1. Premier cas (modèle), le test sera de fixer le débit d'entrée égale à 25% en utilisant

un serveur.

2. Deuxième cas (modèle), le test sera de fixer le débit d'entrée égale à 25% en utilisant deux serveurs.
3. Troisième cas (modèle), le test sera de fixer le débit d'entrée égale à 50% en utilisant un serveur.
4. Quatrième cas (modèle), le test sera de fixer le débit d'entrée égale à 50% en utilisant deux serveurs.

Par ailleurs, l'évaluation du délai des paquets dans les files d'attente sera appliquée sur tous les cas ci dessus. Ainsi dans ces expériences, nous allons comparer et interpréter tous les résultats obtenus par rapport au nombre de serveurs utilisés et par rapport l'augmentation de débit d'entrée.

5.4.1 Contre-exemple généré par NuSMV

La vérification de modèles a été introduite dans ce contexte pour vérifier et évaluer le délai des paquets dans les files d'attente. Nous avons mis en place une propriété logique LTL (figure 5.1 ci-dessus) qui nous permet de faire ceci. Par ailleurs, l'objectif de cette expérience est d'expliquer également une trace retournée par NuSMV.

Pour analyser le fonctionnement de notre méthode et mettre en évidence son utilité, nous allons effectuer un test qui consiste à vérifier un délai borné. Notamment, l'exemple de la figure 5.2 consiste à vérifier, si le délai D est toujours inférieur ou égal à 1cycle (G *delay* $<= 1$) avec une trace de longueur K égale à 10 ($K = 10$).

Après l'exécution en vérification de modèles, la figure 5.2 ci-dessous nous montre que la propriété LTL est fausse (ligne1, 2 voir figure 5.2) et que le délai n'est pas toujours inférieur ou égal à 1 tout en produisant une trace avec un contre-exemple (ligne3-ligne16 voir figure 5.2). En effet, la trace nous indique qu'il existe un état de la trace dans lequel cette borne est dépassée, la valeur maximale obtenue étant égale à 8 cycles (ligne 9 voir figure 5.2). Donc la valeur du délai maximal est au moins égale à 8 cycles (0d6_8).

Remarquons que, si la longueur de la trace K est trop petite, il se peut très bien que nous ne trouverons pas de contre-exemple.

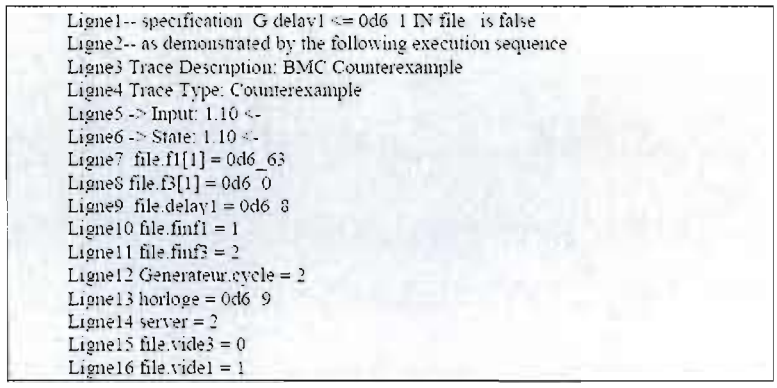

FIGURE 5.2: Un exemple de vérification du délai.

5.4.2 Robustesse de la méthodologie d'évaluation du délai

Dans cette section, nous allons mettre en évidence la robustesse de notre méthode tout en appliquant les tests nécessaires pour pouvoir évaluer les délais des paquets dans les files d'attente. De façon générale, le temps de vérification symbolique est très difficile à prévoir a priori. De petites modifications à un modèle ou à une propriété peuvent souvent faire varier le temps de vérification de façon considérable. Comme nous utilisons une méthodologie itérative qui nécessite plusieurs vérifications indépendantes, nous pouvons nous demander si cela n'aura pas comme effet de nécessiter un temps de vérification qui pourrait devenir prohibitif. Par ailleurs, nous avons deux paramètres fondamentaux qui sont D et K. Nous allons donc tout d'abord déterminer de façon expérimentale leurs influences respectives sur le temps de calcul du système.

5.5 Dépendance de temps de calcul par rapport au délai D

Le but de cette section est d'évaluer l'influence de la borne de délai D sur le temps de calcul. Donc le contexte expérimental dans cette partie sera d'évaluer l'influence de D sur le temps de calcul pour chacun de nos quatre cas.

5.5.1 Débit d'entrée égal à 25% avec un serveur

Notre premier test sera de fixer le débit d'entrée égal à 25% avec un serveur. Pour ce faire, nous avons effectué plusieurs scénarios de vérifications formelles indépendantes. Nous avons augmenté chaque fois la valeur du délai borné D tout en fixant la longueur de la trace K à 25. À chaque exécution effectuée en NuSMV, nous calculons chaque fois le temps de calcul du système.

Pour K=25 sur la figure 5.3 ci-dessous, nous montre d'un côté, une courbe presque constante du temps de calcul en fonction de D pour une trace fixe de longueur K=25 avec des valeurs entre 1,2s et 0,8s. En résumé, nous pouvons supposer que la courbe de délai D en fonction du temps de calcul est une courbe essentiellement constante.

Par ailleurs, nous avons pensé de refaire le même test tout en augmentant la longueur de la trace K de 25 jusqu'à 100 pour voir si toutes les courbes sont les mêmes ou non ? Et d'après la figure 5.3, nous avons constaté que les courbes pour (K=25, K=50, K=75 et K=100), sont des courbes presque constantes. Ainsi que le temps de calcul du système varie essentiellement indépendamment du délai D.

5.5.2 Débit d'entrée égal à 25% avec deux serveurs

Après les tests que nous avons effectués en utilisant un serveur et un débit d'entrée égal à 25%. Nous allons refaire les mêmes scénarios en utilisant dans ce cas deux serveurs et le même débit d'entrée, pour mettre en relief la rigueur de notre méthode tout en comparant les deux scénarios et en dégageant la différence qui les caractérise.

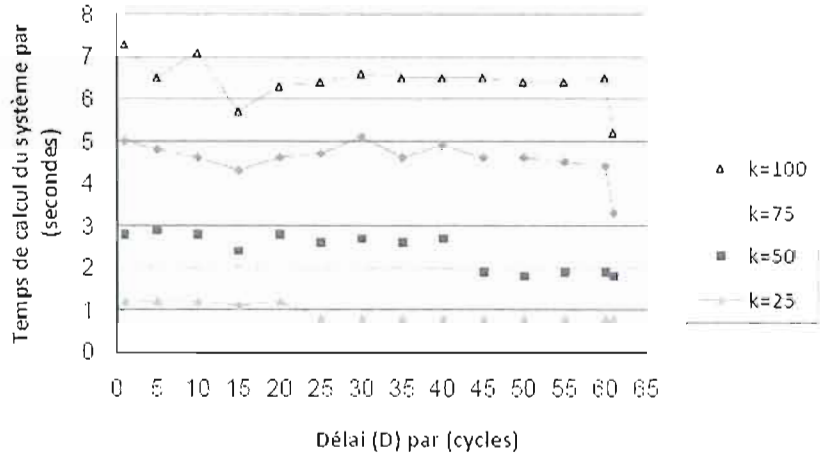

FIGURE 5.3: Courbe de délai en fonction du temps de calcul du système en utilisant un débit d'entrée égal à 25% et un serveur « Round-Robin ».

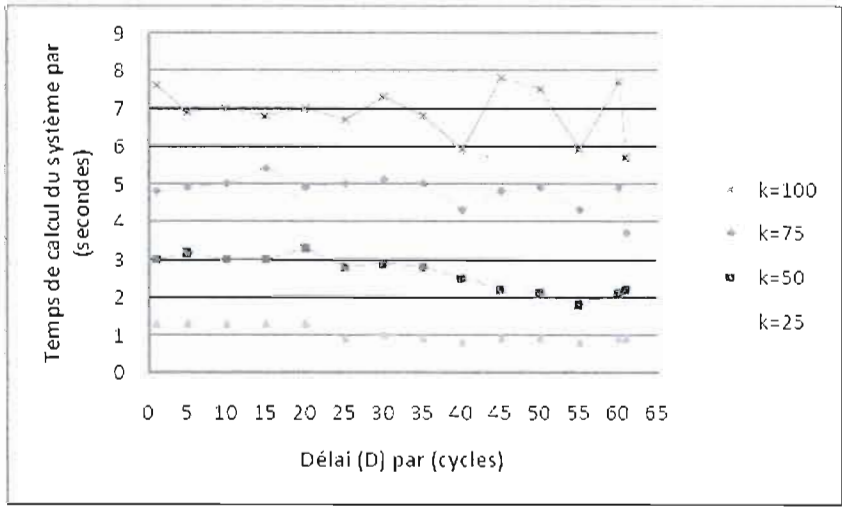

FIGURE 5.4: Courbes des délais en fonction du temps de calcul du système en utilisant un débit égal à 25% et deux serveurs « Round-Robin ».

D'après les figure 5.3 et 5.4 nous remarquons que le temps de calcul du système varie ici aussi essentiellement indépendamment de D. Naturellement, plus on augmente la longueur de la trace K plus le temps de calcul augmente.

5.5.3 Débit d'entrée égal à 50% avec un serveur

Pour tester la performance de notre modèle et montrer la robustesse de notre méthode, nous allons essayer de varier le débit d'entrée de 25 % à 50% tout en appliquant les mêmes scénarios relatifs à un serveur. La figure 5.5 ci-dessous nous montre l'évaluation des délais en fonction du temps de calcul en utilisant un débit d'entrée égal à 50% et un serveur.

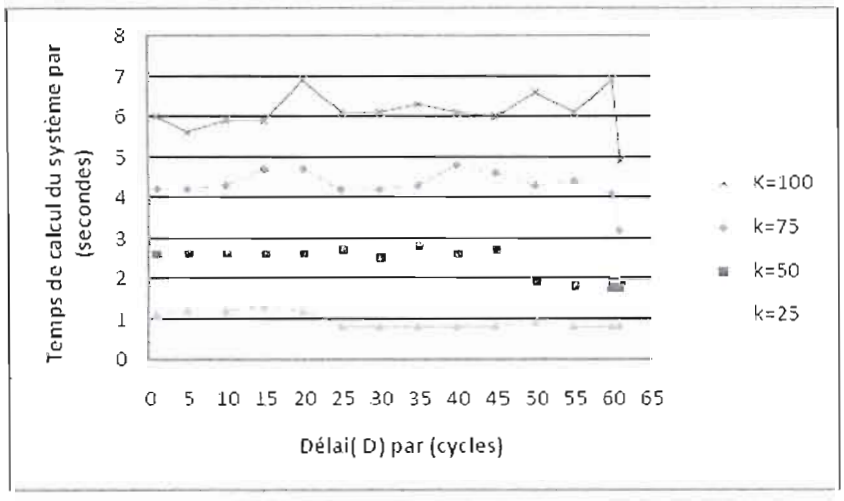

FIGURE 5.5: Courbes des délais en fonction du temps de calcul du système en utilisant un débit égal à 50% et un serveur « Round-Robin ».

5.5.4 Débit d'entrée égal à 50% avec deux serveurs

Dans ce modèle, nous allons refaire les mêmes tests avec un débit d'entrée égal à 50% tout en utilisant deux serveurs.

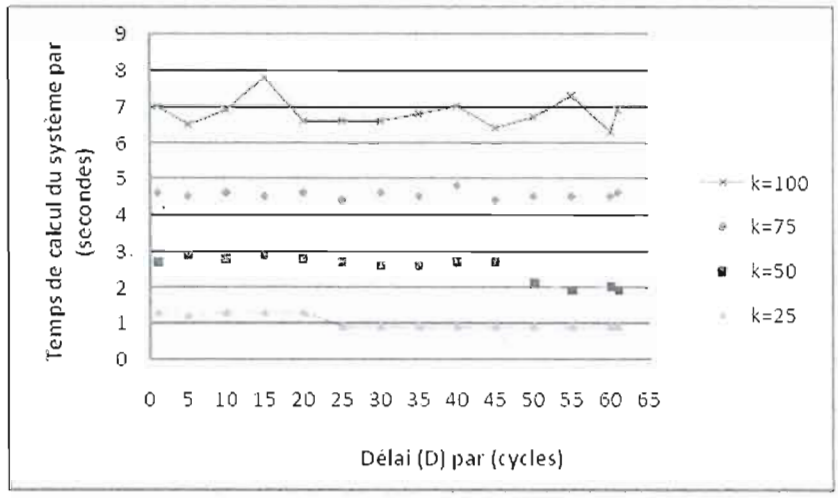

FIGURE 5.6: Courbes des délais en fonction du temps de calcul du système en utilisant un débit égal à 50% et deux serveurs « Round-Robin ».

5.5.5 Interprétation des résultats obtenus du temps de calcul par rapport à D

Les figures 5.3, 5.4, 5.5 et 5.6, présentent toutes le même comportement. Ainsi on peut supposer que le temps de calcul varie essentiellement indépendamment de D. Donc la performance totale de la méthode ne sera fonction que du choix des longueurs K utilisées lors de la vérification. Il est donc utile d'analyser maintenant la variation du temps de calcul en fonction de la longueur de la trace K.

5.6 Dépendance de temps de calcul par rapport à la longueur de la trace K

L'idée est d'évaluer un délai borné D sur différente longueur de la trace pour pouvoir évaluer comment le temps de calcul varie en fonction de la longueur de la trace K. Donc le contexte expérimental dans cette partie sera d'évaluer le temps de calcul en fonction des longueurs de la trace toujours sur nos quatre cas.

5.6.1 Débit d'entrée égal à 25% avec un serveur

La méthode que nous allons utiliser pour pouvoir évaluer le temps de calcul en fonction des longueurs de la trace sera comme suit :

D'abord, nous allons borner le délai de la figure 5.1 D sur une valeur déterminée. Ensuite, nous lancerons chaque fois l'exécution de notre modèle en vérification de modèles sur des différentes longueurs de la trace K de K=5 jusqu'à K=200. Nous mesurerons aussi le temps de calcul du système pour chaque valeur de K. Finalement, nous saisirons les données relatives de ce test sur un tableau et nous tracerons la courbe nécessaire. La figure 5.7 montre une courbe essentiellement linéaire du temps de calcul en fonction de K. Évidemment, lorsque la longueur de la trace K augmente, le temps de calcul augmente.

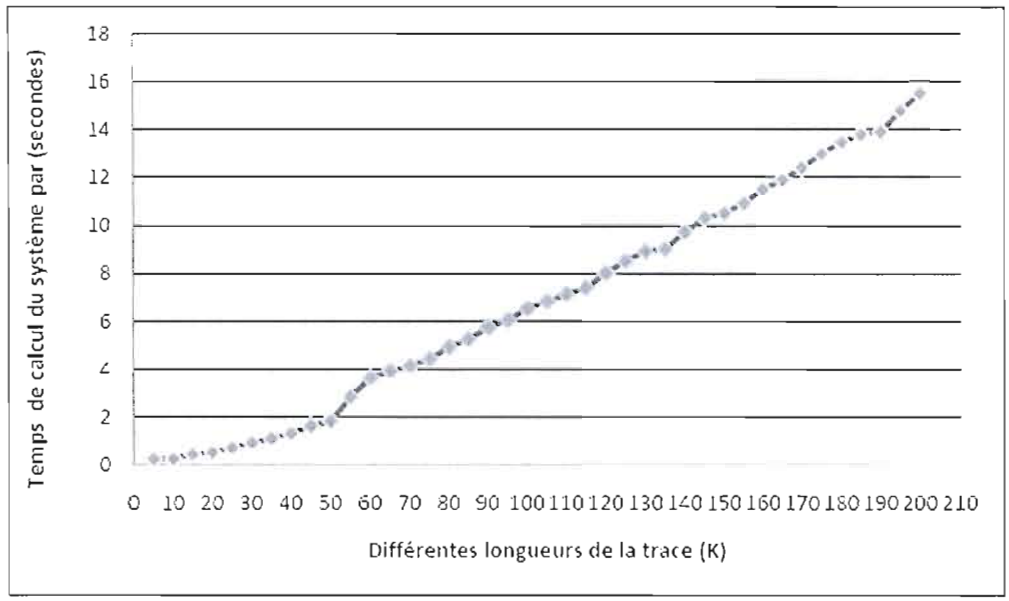

FIGURE 5.7: Le temps de calcul en fonction de K en utilisant un débit égal à 25% et un serveur « Round-Robin ».

5.6.2 Débit d'entrée égal à 25% avec deux serveurs

Dans ce test, nous allons refaire la même expérience mais, avec un débit d'entrée égal à 25% et deux serveurs. L'utilité de ce test sera de prévoir s'il y a un changement au niveau de comportement du modèle lorsque nous augmentons le nombre de serveurs.

La figure 5.8 montre une courbe essentiellement linéaire du temps de calcul en fonction de K. Ce qui mène à conclure que la variation du nombre de serveurs ne changera pas le comportement du modèle, du moins avec un débit de 25%.

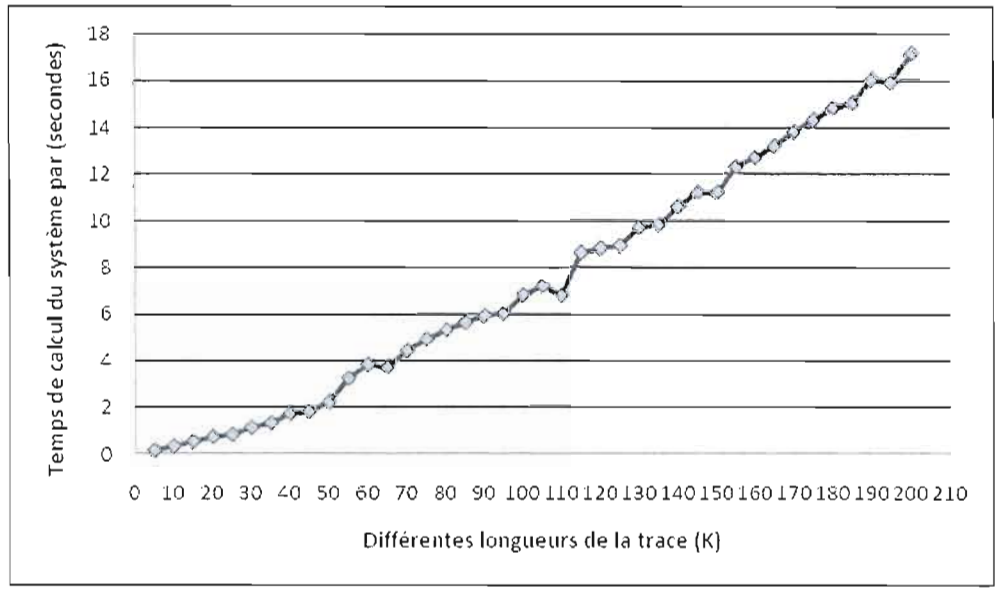

FIGURE 5.8: Le temps de calcul en fonction de K en utilisant un débit égal à 25% et deux serveurs « Round-Robin ».

5.6.3 Débit d'entrée égal à 50% avec un serveur

Dans ce scénario, nous allons appliquer la même méthode que nous avons utilisée sur les cas précédent tout en augmentant cette fois le débit d'entrée de 25% à 50% pour pouvoir comparer la différence entre l'augmentation du débit d'entrée et l'augmentation

du nombre de serveurs. Par ailleurs, la figure 5.9 ci-dessous montre aussi une courbe essentiellement linéaire du temps de calcul en fonction de K. Ceci étant dit, que l'augmentation du débit d'entrée ne changera pas aussi fondamentalement le comportement du modèle.

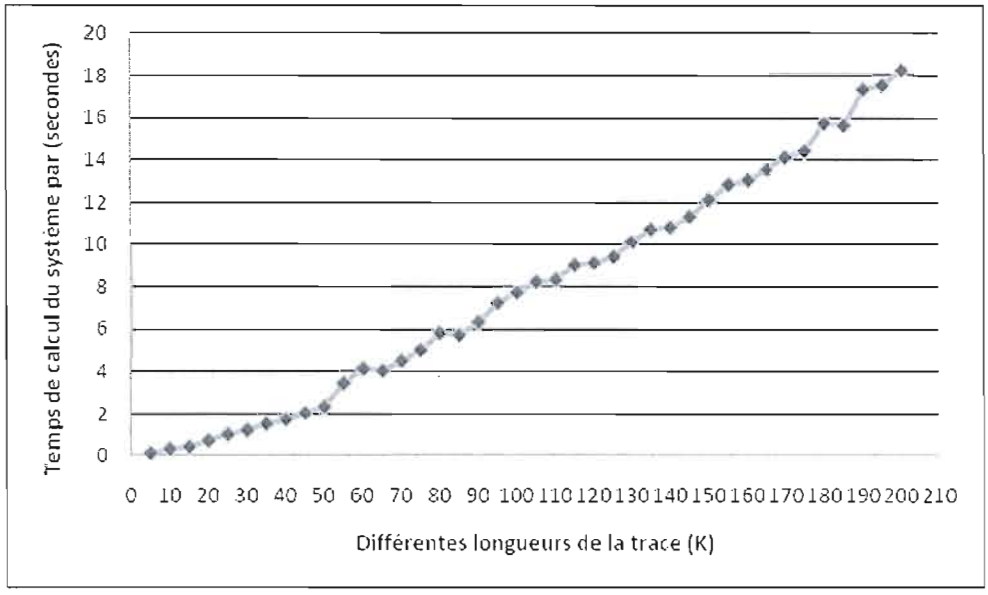

FIGURE 5.9: Le temps de calcul en fonction de K en utilisant un débit d'entrée égal à 50% et un serveur « Round-Robin ».

5.6.4 Débit d'entrée égal à 50% avec deux serveurs

Dans ce modèle, nous allons appliquer les mêmes scénarios utilisés sur les autres cas tout en augmentant le débit d'entrée de 25% à 50% et le nombre de serveurs de un à deux serveurs. ¸Ainsi que, la figure 5.10 a bien montrée que même, lorsque nous augmentons le débit d'entrée et le nombre de serveurs. La courbe du temps de calcul en fonction de K reste toujours une courbe essentiellement linéaire.

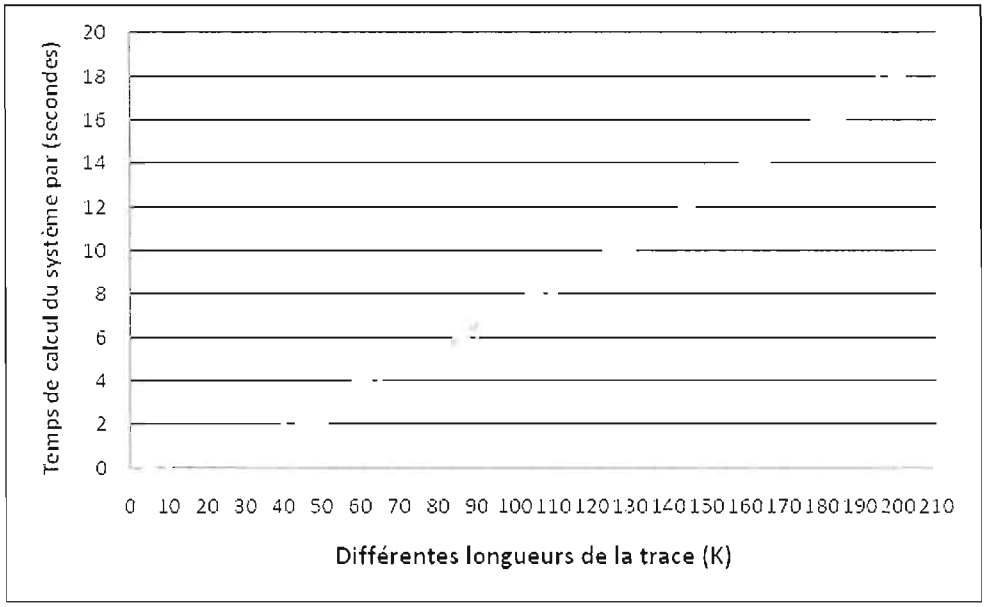

FIGURE 5.10: Le temps de calcul en fonction de K en utilisant un débit d'entrée égal à 50% et deux serveurs « Round-Robin ».

5.6.5 Interprétation des résultats obtenus du temps de calcul par rapport à K

Dans la section précédente (voir section 5.5.1), nous avons expliqué notre méthode qui permet d'évaluer le délai des paquets dans les files d'attente. Ainsi nous avons montré la dépendance du temps de calcul par rapport à D et K. Après les tests que nous avons effectués sur la variation du débit d'entrée et sur le nombre de serveurs. Nous constatons que le temps de calcul varie en fonction de la longueur de la trace K. Aussi, les figures 5.3, 5.4, 5.5 et 5.6 ci-dessus, ont bien justifié que le temps de calcul du système varie indépendamment de D. Ainsi les figures 5.7, 5.8, 5.9 et 5.10 ci-dessus ont montré que les courbes sont essentiellement linéaires. Cela, nous indique que la variation du débit d'entrée et la variation du nombre de serveurs ne changera pas le comportement de notre modèle. Par ailleurs, nous pouvons dire que le temps de calcul à chaque itération sera en fonction de la longueur K et que comme celle-ci augmente, ce temps de calcul augmente aussi normalement. Donc bien que l'on n'a pas d'évaluation a priori du temps total de

calcul, le temps de calcul de chaque itération donne à l'usager une indication du temps nécessaire pour la prochaine itération. Ceci peut l'aider à décider quand arrêter une vérification qui tarde à se compléter.

5.7 Interprétation globale des résultats obtenus du temps de calcul par rapport à D et K

De façon globale, le temps d'exécution en vérification de modèles est souvent difficile à prédire, une petite modification appliquée à un modèle peut entrainer une grande variation du temps de vérification. En général, d'après les tests effectués sur l'évaluation du délai D des paquets dans les files en fonction du temps de calcul, nous pouvons conclure que le temps de calcul varie essentiellement indépendamment de D. De plus, le temps de calcul augmente essentiellement linéairement en fonction de la longueur de la trace K.

5.8 Évaluation du délai maximal

L'objectif dans cette partie est de déterminer le temps global de calcul pour pouvoir estimer le délai maximum (Dmax) des paquets dans les files d'attente tout en appliquant la méthode sur tous nos cas.

Pour arriver à cela, nous devons rappeler notre méthode. D'abord, nous allons commencer à chercher le délai maximal avec une borne $D = 3$. Nous lancerons l'exécution de notre modèle en vérification de modèles avec une trace de $K = 10$ états (c'est la valeur par défaut de NuSMV). Dès que nous aurons une trace avec un contre-exemple, nous conserverons le délai maximal Dmax ainsi que le temps de calcul. Si nous n'avons pas de contre-exemple, nous augmentons la valeur de K jusqu'à trouver un contre-exemple ou atteindre la valeur $K = 23$. Enfin, si nous trouvons un contre-exemple, nous relancerons l'exécution avec $D = Dmax + 1$. Ainsi, c'est lorsque nous ne trouverons pas de contre-exemple avec $K = 100$ que nous pourrons estimer que le délai maximal est égal à la valeur actuelle de D.

TABLE 5.1: Le temps total de calcul du système pour trouver le délai maximum des paquets dans la file d'attente en utilisant un débit d'entrée égal à 25% et un serveur « Round-Robin ».

Propriété(D) à vérifier	Délai max trouvé	Temps de calcul/s
1	2	0,6
3	5	3,9
Le temps total de calcul (s)	-	4,5

TABLE 5.2: Simulation d'un délai borné supérieur ou égal à 5 cycles avec un débit d'entrée égal à 25% et un serveur « Round-Robin ».

Propriété	Description de la trace	Différente longueur de la trace(K)	Temps de calcul/s
$D >= 6$	-no counterexample	25	1,5
$D >= 6$	-no counterexample	30	1,7
$D >= 6$	-no counterexample	35	2,1
$D >= 6$	-no counterexample	50	3,3
$D >= 6$	-no counterexample	100	8,4

5.8.1 Débit d'entrée égal à 25% avec un serveur

Les tableaux 5.1 et 5.3 ci-dessous montrent comment nous pourrons déterminer le temps global pour pouvoir estimer le délai maximum. Pour pouvoir estimer le délai maximum des paquets dans la file d'attente, nous devrons appliquer la méthode qui est décrite ci-dessus. D'après le tableau 5.1, le délai maximum est égal à 5 cycles pour un débit égal à 25% et un serveur. Ceci étant dit, que nous avons essayé de chercher un délai borné $D >= 6$ qui supérieur à 5 cycles sur plusieurs longueurs de la trace K qui sont plus grandes que celle que nous avons trouvées mais nous n'arriverons pas à trouver un contre-exemple. Nous pouvons donc estimer que le délai maximum est le délai précédent que nous avons trouvé $Dmax = 5$ cycles. Ainsi le tableau 5.2, justifie bien l'efficacité de notre méthode, notamment que nous avons essayé de déterminer un autre délai maximum qui est supérieur à 5 cycles sur plusieurs longueurs de la trace K mais nous n'arriverons pas à trouver un contre-exemple.

TABLE 5.3: Le temps total de calcul du système pour trouver le délai maximum des paquets dans la file d'attente en utilisant un débit égal à 25% et deux serveurs « Round-Robin ».

Propriété(D) à vérifier	Délai max trouvé	Temps de calcul/s
1	2	0,6
Le temps total de calcul (s)	-	0,6

5.8.2 Débit d'entrée égal à 25% avec deux serveurs

Le tableau 5.3 nous montre un délai max =2 cycles pour un débit égal à 25% et deux serveurs. De façon concrète, nous avons essayé de vérifier le délai ($D = 6$ cycles pour débit=25% et 1 serveur) ainsi le délai ($D = 3$ cycles pour un débit=25% et 2 serveurs) sur différentes longueurs de la trace K et nous n'arriverons pas à trouver un contre-exemple alors nous pourrons conclure que le délai maximum est de 5 cycles pour un débit égal à 25% en utilisant un serveur. Par contre lorsque nous utilisons le même débit avec deux serveurs nous aurons un délai maximum égal à 2 cycles.

Par ailleurs, nous pouvons conclure que la variation du nombre de serveurs ne changera pas le comportement de notre modèle mais il diminue le délai maximum ainsi que le temps total de calcul. Notamment, le tableau 5.3 montre qu'il y a une diminution sur le délai maximum de 5 cycles à 2 cycles ainsi que sur le temps total du calcul où la valeur de ce dernier a été diminuée de 4,5s à 0,6s.

Dans le cas où le débit d'entrée est égal à 25% avec deux serveurs, il peut sembler étonnant que la valeur du délai maximal soit de 2 cycles. On pourrait plutôt s'attendre à une valeur égale à 0 cycle. Mais remarquons que dans notre modèle un paquet est d'abord mis dans la file, donc si la file était vide, c'est seulement à ce moment-là qu'elle devient non-vide. Ainsi elle ne pourra obtenir le serveur qu'à l'état suivant. Finalement encore un autre état est nécessaire pour qu'elle soit servie. Ce qui donne exactement un délai de 2 cycles. Un modèle différent pourrait être développé, mais notre objectif dans ce mémoire est de montrer qu'étant donné un modèle, on peut évaluer le délai

TABLE 5.4: Le temps total de calcul du système pour trouver le délai maximum des paquets dans la file d'attente en utilisant un débit égal à 50% et un serveur « Round-Robin ».

Propriété(D) à vérifier	Délai max trouvé	Temps de calcul/s
1	3	0,5
4	6	0,7
7	8	1,6
9	11	2,8
12	13	6,2
14	15	8,6
Le temps total de calcul (s)	-	20,4

avec l'approche de vérification de modèles. Comme nous n'utilisons rien de vraiment spécifique à notre modèle, notre approche est plutôt générique et pourrait être appliquée avec d'autres modélisations.

5.8.3 Débit d'entrée égal à 50% avec un serveur

Pour pouvoir tester la performance de notre modèle, nous avons varié le débit d'entrée de 25% à 50% tout en appliquant la même méthode d'évaluation du délai pour pouvoir estimer le délai maximal en utilisant un serveur.

Et d'après le tableau 5.4, nous avons obtenu un délai maximum qui est égal à 15 cycles avec un temps total de calcul égal à 20,4s. Pour mettre en évidence l'utilité de cette valeur, le tableau 5.5 montre un essai pour trouver un autre délai maximum Dmax qui est supérieur ou égal à 16 sur plusieurs longueurs de traces. Et d'après les tests effectués, nous n'arriverons pas à trouver un contre-exemple.

5.8.4 Débit d'entrée égal à 50% avec deux serveurs

Le tableau 5.6 montre une évaluation de délai maximal importante qui est dûe à l'augmentation de nombre de paquets arrivant aux interfaces d'entrée. Ceci nous mène à

TABLE 5.5: Simulation d'un délai borné supérieur ou égal à 16 cycles avec un débit d'entrée égal à 50% et un serveur « Round-Robin ».

Propriété	Description de la trace	Différente longueur de la trace(K)	Temps de calcul/s
$D >= 16$	-no counterexample	25	1,7
$D >= 16$	-no counterexample	30	1,9
$D >= 16$	-no counterexample	35	2,3
$D >= 16$	-no counterexample	50	3,8
$D >= 16$	-no counterexample	100	9,4

TABLE 5.6: Le temps total de calcul du système pour trouver le délai maximum des paquets dans la file d'attente en utilisant un débit égal à 50% et deux serveurs « Round-Robin ».

Propriété(D) à vérifier	Délai max trouvé	Temps de calcul/s
1	2	0,4
3	6	0,7
7	8	1,5
Le temps total de calcul (s)	-	2,6

avoir des files d'attentes pleines et des débordements qui se produisent. Deplus, le tableau 5.6 montre un délai maximum de 8 cycles avec un temps de calcul du système égal à 2,6s.

D'après les tableaux 5.4 et 5.6, nous avons remarqué que le délai maximum et le temps total de calcul diminuent en fonction du nombre de serveurs. Notamment, le tableau 5.6 montre que le délai maximum a été diminué de 15 à 8 cycles. Ainsi, le temps total du calcul de 20,4s à 2,6s. Ceci étant dit, nous pouvons conclure que plus on augmente le nombre de serveurs plus le délai maximum et le temps total diminuent.

De façon concrète, à chaque cycle égal à 4, le modèle génère deux paquets à chaque interface d'entrée. Comme nous avons un serveur qui traite chaque fois un paquet d'une file d'attente non-vide, à la prochaine génération de deux nouveaux paquets, nous aurons un paquet restant sur chaque file d'attente avec deux paquets (F=3 paquets). Après un nouveau cycle qui est égal à 4, les files d'attente deviennent pleines et des débordements se

produisent. Donc il sera très utile d'augmenter le nombre de traitement des files d'attente tout en employant deux serveurs qui traitent à la fois deux files d'attente différentes soient non-vides dans un cycle égal à 4. Ceci, nous permet de diminuer la valeur du délai maximal ainsi le temps total du calcul.

5.9 Conclusion

Cela nous amène à conclure ce qui suit :

- La variation de débit d'entrée, a montré que le temps de calcul du système varie toujours essentiellement indépendamment de D ainsi que le temps de calcul augmente de façon essentiellement linéaire en fonction des différentes tailles de la trace K.
- L'estimation du délai maximal se fait, lorsque nous n'arriverons pas à trouver des contre-exemples et que nous avons essayé plusieurs longueurs qui sont plus grandes que celles que nous avons trouvées. Dans ce cas il faut s'arrêter quelque part tout en estimant que le délai maximal est le délai dernier que nous avons vérifié.

CONCLUSION

Aujourd'hui, Internet coiffe presque toutes les activités de la société moderne tout en occupant une place prépondérante de la vie quotidienne des personnes ainsi que tous les secteurs de l'économie de la société actuelle. Cependant, la télécommunication et les réseaux utilisent le protocole d'Internet (IP) comme un support de communication pour partager leurs ressources dans le réseau. Ce dernier est composé par des systèmes informatiques (serveurs, ordinateurs,...), des routeurs et des liaisons. En particulier, les routeurs jouent un rôle très important dans Internet. Ils permettent également de partager les données (paquets) entre une source et une destination. En effet, la fonction de l'acheminement des paquets dans le routeur se fait d'abord, par l'arrivée des paquets sur les ports d'entrée. Puis, ces paquets traversent la matrice de commutation (switching fabric) pour atteindre les ports de sortie appropriés. Enfin, les ports de sortie acheminent les paquets vers la destination adéquate. Notons que les paquets se traitent séparément dans le routeur. Par ailleurs, avec un charge de trafic sur les ports d'entrée, nous pouvons avoir des contraintes sur la performance des réseaux. Particulièrement, les indicateurs comme le délai, le débit et le taux de pertes sont devenus des critères importants dans les réseaux. Donc, les méthodes de mesures de ces indicateurs génèrent un grand intérêt pour l'optimisation du réseau *End-to-End*. C'est pour cette raison que nous nous sommes intéressés à évaluer et à vérifier les délais par la méthode de vérification de modèles (Model-Checking).

De façon générale, la construction d'un modèle comme le nôtre permet de le vérifier facilement par rapport à un système réel ; cependant comme le modèle sert à vérifier et non à exécuter, nous pourrons dire qu'il peut s'approcher approximativement à des systèmes réels. Cela étant dit, comme le modèle utilise des structures de données avec des tailles déterminées. Donc nous obtiendrons toujours un nombre fini d'états ; ce qui nous mène à avoir un système fini. La vérification de modèles permet d'un côté de tester le modèle sur plusieurs tailles différentes pour pouvoir visualiser son comportement et comparer les résultats obtenus. Et d'un autre côté, elle le laisse plus flexible qu'un système réel en

lui permettant des comportements supplémentaires. Et comme notre modèle permet de vérifier et d'évaluer les délais des paquets dans un routeur, en estimant chaque fois le délai maximal des files d'attente, nous pourrons dire que notre modèle peut s'approcher approximativement à un routeur réel. Cependant, la partie principale de notre modèle est la gestion du choix de serveurs. Donc le processus de fonctionnement de ces derniers était comme suit :

- Lorsque le premier serveur traite une file d'attente non-vide, le deuxième servira la prochaine file non-vide. C'est-à-dire nous aurons chaque fois deux serveurs qui traitent à la fois parallèlement deux files d'attente non-vides et qui fonctionnent avec un processus « Round Robin ».
- Les serveurs font leurs choix simultanément pour pouvoir traiter des files d'attentes non-vides. Ils ne traitent jamais dans un cycle déterminé la même file d'attente non-vide.

Concrètement, le fait que nos serveurs traitement une fois une file d'attente non-vide dans un cycle déterminé, c'est-à-dire que nous sommes toujours contraints de trouver un seul délai par cycle et pas deux délais dans un même cycle. Alors il n'est pas toujours évident d'établir un lien entre les cycles et des secondes d'exécution du délai sur un routeur réel.

Par ailleurs, la vérification de modèles permet de vérifier formellement un modèle d'un système réel. Cette approche est normalement utilisée pour déterminer des erreurs de comportement à un stade précoce de la conception. Dans ce mémoire on montre qu'il est possible d'appliquer cette approche à la vérification de propriété quantitative, comme le délai. Il s'agit donc de déterminer le délai maximal pour un certain modèle de routeur. Concrètement, nous avons appliqué les techniques de vérifications à un modèle simple, mais que celui-ci est similaire au modèle de l'article (Chertov *et al.*, 2008) qui a été montré comme simulant assez fidèlement à des routeurs réels.

BIBLIOGRAPHIE

ALUR, R., JAGADEESAN, L. J., KOTT, J. J. et OLNHAUSEN, J. E. V. (1997). Model-checking of real-time systems : A telecommunications application - experience report. *In Proceedings of the 19th International Conference on Software Engineering*, pages 514–524.

ANGRISANI, L., VENTRE, G., PELUSO, L. et TEDESCO, A. (2006). Measurement of processing and queuing delays introduced by an open-source router in a single-hop network. *IEEE Transactions on, Instrumentation and Measurement*, 55(4):1065–1076.

ARTHO, C., BIERE, A., SHIBAYAMA, E. et HONIDEN, S. (2007). Efficient model checking of applications with input/output. *Lecture Notes in Computer Science*, pages 515–522.

ARTHO, C. et GAROCHE, P.-L. (2006). Accurate centralization for applying model checking on networked applications. *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 177–188.

AWEYA, J. (1999). Ip router architectures : An overview. *Journal of Systems Architecture*, 46:483–511.

BACLET, M. (2005). Applications du model-checking à des problèmes de vérification de systèmes sur puce. *Ph.D. thèse, Institut de Recherche en Informatique de Toulouse.*

BAUMGARTNER, P. et TINELLI, C. (2008). The model evolution calculus as a first-order dpll method. *Artif Intell*, 172:591–632.

BEHRMANN, G., DAVID, A. et LARSEN, P. K. (2004). A tutorial on uppaal. *LNCS, Formal Methods for the Design of Real-Time Systems (revised lectures)*, 3185:200–237.

BIERE, A., CIMATTI, A., CLARKE, E. M. et ZHU, Y. (1999). Symbolic model checking without bdds. *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207.

BOZGA, M., DAWS, C., MALER, O., OLIVERO, A., TRIPAKIS, S. et YOVINE, S. (1998). Kronos : A model-checking tool for real-time systems. *Dans Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, FTRTFT '98, pages 298–302, London, UK: Springer-Verlag.

BÉRCZES, T., GUTA, G., KUSPER, G., SCHREINER, W. et SZTRIK, J. (2007). Analyzing a proxy cache server performance model with the probabilistic model checker prism. *Supported by the Austrian-Hungarian Scientific/Technical Cooperation, Contract HU 13/2007.*

BURCH, J., CLARKE, E., MCMILLAN, K., DILL, D. et HWANG, L. (1990). Symbolic model checking : 1020 states and beyond. *Logic in Computer Science. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439.

BURCH, J. R., CLARKE, E. M. et LONG, D. E. (1991). Symbolic model checking with partitioned transition relations. pages 49–58. North-Holland.

CAVADA, R., C., ALESSANDRO, K., GAVIN, O., EMANUELE, P., MARCO, R. et MARCO (2007). User manual. *NuSMV v2.4 Tutorial, http ://nusmv.fbk.eu/NuSMV/tutorial/index.html.*

CHERTOV, R., FAHMY, S. et SHROFF, N. (2007). A black-box router profiler. *IEEE Global Internet Symposium*, pages 37–42.

CHERTOV, R., FAHMY, S. et SHROFF, N. (2008). A device-independent router model. *IEEE INFOCOM. The 27th Conference on Computer Communications.*, pages 1642–1650.

CLARK BARRETT, Roberto Sebastiani, A., SANJIT, T. et CESARE (2009). Handbook of satisfiability. *IOS Press*, page 980.

CLARKE, E., BIERE, A., RAIMI, R. et ZHU, Y. (2001). Bounded model checking using satisfiability solving. *Formal Methods in System Design*, pages 7–34.

CLARKE, E. M. et EMERSON, E. A. (1982). Design and synthesis of synchronization skeletons using branching-time temporal logic. *Dans Logic of Programs, Workshop*, pages 52–71, London, UK. Springer-Verlag.

DUFLOT, M., KWIATKOWSKA, M., NORMAN, G., PARKER, D., PEYRONNET, S., PICARONNY, C. et SPROSTON, J. (2010). Handbook on industrial critical systems, chapitre 7 : Practical applications of probabilistic model checking to communication protocols. *IEEE Computer Society Press*. To appear.

EEN, N. et SÖRENSSON, N. (2003). Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science 89No. 4*, pages 541–638.

EHRENSBERGER, J. (2006). Chapitre1 : Performances et qos. *Note de cours, Haute École d'Ingénierie et de Gestion du Canton de Vaud.*

E.M. CLARKE, O. G. et PELED, D. A. (2000). Model checking. *MIT Press, Cambridge, MA.*

GARY, P. Z. (2003). Dirac : A software-based wireless router system. *In Proc. of ACM MOBICOM'03*, pages 230–244.

HENZINGER, T. A., HO, P.-H. et WONG-TOI, H. (1997). Hytech : A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463.

HOHN, N., VEITCH, D., PAPAGIANNAKI, K. et DIOT, C. (2004). Bridging router performance and queuing theory. *In Proc of ACM SIGMETRICS*, pages 355–366.

HUITEMA, C. (2000). Routing in the internet. *Prentice Hall PTR, 2 edition.*

HUTH, M. et RYAN, M. (2004). Logic in computer science - modelling and reasoning about systems. *2nd edition, Cambridge.*

JUNTTILA, T. et DUBROVIN, J. (2008). Encoding queues in satisfiability modulo theories based bounded model checking. *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 290–304.

KWIATKOWSKA, M. Z., NORMAN, G. et SPROSTON, J. (2002). Probabilistic model checking of the ieee 802.11 wireless local area network protocol. *Proceedings of the Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 169–187.

MCKEOWN, N. (1999). The islip scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on, Networking*, 7(2):188–201.

MCMILLAN., K. (1993). Symbolic model checking. *Kluwer Academic Publishers, Boston.*

MUSUVATHI, M. et ENGLER, D. R. (2004). Model checking large network protocol implementations. *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, 1:12–12.

OMNET++ (1992). http ://www.omnetpp.org/.

OPNET (1986). Network modeling and simulation environment.

PAPAGIANNAKI, K., MOON, S., FRALEIGH, C., THIRAN, P. et DIOT, C. (2003). Measurement and analysis of single-hop delay on an ip backbone network.