

UNIVERSITÉ DE MONTRÉAL

VÉRIFICATION ET CONFIGURATION AUTOMATIQUES DE PARE-FEUX PAR
MODEL CHECKING ET SYNTHÈSE DE CONTRÔLEUR

MAJDA MOUSSA
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLOME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
JUN 2014

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

VÉRIFICATION ET CONFIGURATION AUTOMATIQUES DE PARE-FEUX PAR
MODEL CHECKING ET SYNTHÈSE DE CONTRÔLEUR

présenté par : MOUSSA Majda

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

Mme BELLAÏCHE Martine, Ph.D., présidente

Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche

M. CHAMBERLAND Steven, Ph.D., membre et directeur de recherche

M. QUINTERO Alejandro, Doct., membre

REMERCIEMENTS

Tout d'abord, je remercie de tout cœur tous ceux qui, de près ou de loin, m'ont soutenu durant ce travail.

Je tiens à exprimer ma profonde gratitude envers ma directrice de recherche Madame Hanifa Boucheneb sans qui ce travail n'aurait vu son épilogue si tôt. J'ai apprécié sa modestie, son support, et ses conseils qui m'ont permis d'arriver à ce stade. J'ai adoré sa façon de me guider tout en me laissant la liberté de décider.

J'adresse mes plus sincères remerciements à mon co-directeur, Monsieur Steven Chamberland pour sa bienveillance et ses conseils. Un très grand merci à Hakima pour son soutien tout au long de ma maîtrise, ses conseils et critiques constructives qui m'ont été utiles pour l'accomplissement de ce travail.

Un remerciement spécial plein d'amour à ma chère Tunisie de m'avoir fait l'honneur de m'offrir une bourse d'études qui a couvert en intégralité tous les frais liés à mes études et mon séjour.

Je ne manquerai pas de remercier tous mes collègues et amis. Je leur exprime ma profonde sympathie et leur souhaite beaucoup de bien.

Enfin, mon plus gros mot de gratitude est dédié à mes chers parents, mes charmantes sœurs, mon frère et mon fiancé. Merci de votre confiance, merci de m'avoir encouragé et supporté tout le long de mon cheminement.

RÉSUMÉ

Les pare-feux jouent un rôle crucial dans le renforcement de la politique de sécurité d'un réseau. Cependant, leur configuration, qui nécessite souvent l'intervention humaine, est une source majeure de failles de sécurité. Par conséquent, des solutions automatisées sont nécessaires afin de détecter les incohérences de configuration des pare-feux.

Dans ce mémoire, nous proposons des approches d'aide à la configuration des pare-feux, basées sur des techniques formelles comme le model checking et la synthèse de contrôleur. La première approche permet de vérifier par model checking la cohérence des pare-feux vis-à-vis d'un objectif de sécurité et de détecter, le cas échéant, les incohérences. Elle permet notamment de vérifier l'incohérence de croisement de chemins (i.e. si un paquet est rejeté par l'un des pare-feux en direction de sa destination, il ne pourra pas l'atteindre en empruntant un autre chemin).

Nous étendons cette approche, en utilisant SMC-UPPAAL, afin d'étudier les performances du réseau en fonction des paramètres de qualité de service tels que le délai d'acheminement des paquets, le délai d'attente et le taux de perte. Cette extension permet, entre autres, de calculer la probabilité qu'un paquet passant par un nœud malicieux soit accepté, la probabilité qu'un nœud tombe en panne, les taux d'acceptation et de rejet de paquets.

En outre, nous proposons une approche formelle permettant de configurer formellement les pare-feux privés sur un réseau, conformément à un objectif de sécurité donné, en utilisant la technique de synthèse de contrôleur implémentée par l'outil UPPAAL-TIGA. Par ailleurs, pour atténuer le problème d'explosion combinatoire inhérent au model checking et la synthèse de contrôleur, les approches proposées ici sont basées sur des abstractions. Des études expérimentales sont conduites pour démontrer la performance de ces abstractions.

ABSTRACT

Firewalls play a crucial role in the enforcement of network security policies. However, their configuration, which often requires human intervention, is a major source of security vulnerabilities. Therefore, automated solutions are needed in order to detect firewall configuration inconsistencies.

In this master thesis, we propose support approaches of firewall configuration based on formal techniques such as model checking and controller synthesis. The first approach is used to check the firewalls consistency by model checking with respect to a security objective and to detect firewall configuration incoherencies such as cross path incoherence (i.e. if a packet is rejected by a firewall towards its destination, it cannot reach it by taking a different path).

We extend this approach, using SMC UPPAAL to study the network performance according to QoS parameters such as end-to-end time routing, latency and loss rate. This extension allows, inter alia, the computation of the probability to accept a packet passing through a malicious node, the probability that a node fails and packets acceptance or rejection rates.

In addition, we propose another approach to formally configure private firewalls on a network, according to a given security policy, using the controller synthesis technique implemented in the UPPAAL TIGA tool. Furthermore, to alleviate the problem of combinatorial explosion inherent in model checking and controller synthesis, the approaches proposed here are based on abstractions. Experimental studies were conducted to demonstrate the performance of these abstractions.

TABLE DES MATIÈRES

REMERCIEMENTS	iii
RÉSUMÉ	iv
ABSTRACT	v
TABLE DES MATIÈRES	vi
LISTE DES TABLEAUX	ix
LISTE DES FIGURES	x
INTRODUCTION	1
CHAPITRE 1 MECANISMES DE MODELISATION ET VERIFICATION	
FORMELLES	4
1.1 Introduction	4
1.2 Model checking	5
1.2.1 Définition du model checking	5
1.2.2 Extensions du model checking	7
1.2.3 Explosion combinatoire	8
1.3 Formalismes de modélisation	10
1.3.1 Automates temporisés (Timed Automata : TA)	10
1.3.2 Automates temporisés à coût (Priced Timed Automata : PTA)	11
1.3.3 Automates temporisés de jeu (Timed Game Automata, TGA)	13
1.4 Formalisme de spécification	14
1.4.1 La logique temporelle arborescente CTL	15
1.4.2 Extensions de la logique temporelle CTL	17
1.4.3 Classifications des propriétés temporelles	18
1.5 Synthèse de contrôleur	19
1.6 Model checkers	20
1.6.1 Choix du model checker	20
1.6.2 UPPAAL	22
1.6.3 UPPAAL TIGA	26
1.6.4 SMC UPPAAL	27

1.7	Conclusion	29
CHAPITRE 2 SECURITE DES RESEAUX INFORMATIQUES		30
2.1	Introduction	30
2.2	Concepts de base	30
2.2.1	Architecture d'un réseau	31
2.2.2	Menaces réseau	32
2.2.3	Politique de sécurité	33
2.3	Systèmes de détection d'intrusions	34
2.3.1	Approches de détection d'intrusions	34
2.3.2	Types et emplacement des IDS	35
2.4	Pare feu (firewall)	36
2.4.1	Types de pare-feux	36
2.4.2	Emplacement d'un pare-feu	37
2.4.3	Fonctions d'un pare feu	40
2.4.4	Anomalies des pare-feux	40
2.5	Techniques de détection d'anomalies	42
2.5.1	Techniques de détection des anomalies intra-pare-feux :	43
2.5.2	Techniques de détection des anomalies inter-pare-feux	45
2.5.3	Discussion	46
2.6	Renforcement de la politique de sécurité	47
2.6.1	Approche algébrique	47
2.6.2	Autres approches	49
2.6.3	Discussion	51
2.7	Conclusion	51
CHAPITRE 3 MODELISATION Et VERIFICATION FORMELLES D'UN RESEAU INFORMATIQUE A PARE-FEUX		53
3.1	Introduction	53
3.2	Représentation graphique d'un réseau à pare-feux	53
3.3	Modélisation du comportement	56
3.3.1	Automate "Nœud_générateur"	57
3.3.2	Automate "Nœud"	58
3.3.3	Exemple concret	59
3.4	Vérification formelle de la cohérence des pare-feux	62
3.4.1	Propriétés assurant le bon fonctionnement du modèle	62
3.4.2	Cohérence de la politique de sécurité globale du réseau	63

3.4.3	Propriété vérifiant l'incohérence de croisement de chemins	64
3.4.4	Etude expérimentale de performance	66
3.5	Comment atténuer le problème d'explosion combinatoire?	68
3.5.1	Modèle réduit	69
3.5.2	Etude de performance du modèle réduit	70
3.6	Extension quantitative du modèle	72
3.6.1	Paramètres quantitatifs	72
3.6.2	Modèle quantitatif	74
3.6.3	Analyse des propriétés quantitatives	79
3.7	Conclusion	82
CHAPITRE 4 CONFIGURATION FORMELLE D'UN RESEAU INFORMATIQUE		83
4.1	Introduction	83
4.2	Formalisation du problème de renforcement d'une politique de sécurité	84
4.3	Modèle de renforcement	85
4.3.1	Automate "Nœud_générateur"	85
4.3.2	Automate "Nœud"	86
4.3.3	Automate "Propriété"	87
4.4	Configuration formelle des pare-feux	87
4.5	Etude de performance du modèle de renforcement	94
4.6	Comment atténuer le problème d'explosion combinatoire	97
4.7	Conclusion	101
CONCLUSION ET RECOMMANDATIONS		102
LISTE DES RÉFÉRENCES		104

LISTE DES TABLEAUX

Tableau 3.1	Structure d'un paquet.	58
Tableau 3.2	Paquets générés.	61
Tableau 3.3	Propriétés vérifiant le bon fonctionnement du modèle	63
Tableau 3.4	Lexique de variables.	77
Tableau 3.5	Paramètres du réseau à étudier.	79
Tableau 3.6	Propriétés quantitatives	81

LISTE DES FIGURES

Figure 1.1	Procédure du Model Checking.	6
Figure 1.2	Exemple d'automate temporisé.	11
Figure 1.3	Exemple d'automate temporisé à coût.	12
Figure 1.4	Exemple d'automate temporisé de jeu.	14
Figure 1.5	Illustration de certaines formules CTL.	16
Figure 1.6	Synthèse de contrôleur.	20
Figure 1.7	Comparaison des performance de UPPAAL, HYTECH et KRONOS (voir Larsen <i>et al.</i> , 1995).	21
Figure 1.8	Architecture de UPPAAL (voir Bengtsson <i>et al.</i> , 1996).	22
Figure 2.1	Structure d'un paquet.	32
Figure 2.2	Emplacement d'un pare-feu à la frontière.	38
Figure 2.3	Emplacement d'un pare-feu au centre d'un réseau.	39
Figure 2.4	Arbre de politique de sécurité d'un pare-feu.	43
Figure 2.5	Approche algébrique pour la sécurisation des réseaux informatiques.	47
Figure 2.6	Opérateur du monitoring sélectif.	48
Figure 2.7	Exemple de FDD.	50
Figure 3.1	Représentation graphique d'un réseau.	55
Figure 3.2	Automate générateur.	57
Figure 3.3	Automate Nœud.	58
Figure 3.4	Exemple d'un réseau	60
Figure 3.5	Exemple de simulation du modèle.	61
Figure 3.6	Exemple de vérification de modèle	65
Figure 3.7	Temps de vérification de la propriété de croisement de chemins.	67
Figure 3.8	Taux d'utilisation de la mémoire pour la propriété de croisement de chemins.	68
Figure 3.9	Automate réseau.	69
Figure 3.10	Temps de vérification de la propriété de croisement de chemins pour le modèle réduit.	70
Figure 3.11	Taux d'utilisation de la mémoire pour la propriété de croisement de chemin pour le modèle réduit.	71
Figure 3.12	Modèle quantitatif du réseau.	75
Figure 4.1	Problématique.	84
Figure 4.2	Automate "Nœud_générateur" dans le modèle de renforcement.	85

Figure 4.3	Automate Nœud.	86
Figure 4.4	Automate "Propriété".	87
Figure 4.5	Méthodologie de l'algorithme d'extraction des règles.	89
Figure 4.6	Output du vérificateur UPPAAL-TIGA	90
Figure 4.7	Fichier des stratégies gagnantes.	91
Figure 4.8	Illustration de la procédure d'extraction des règles.	93
Figure 4.9	Temps de vérification de la propriété de renforcement.	95
Figure 4.10	Taux d'utilisation de la mémoire pour la propriété de renforcement. . .	96
Figure 4.11	Automate "Réseau" pour le modèle de renforcement	98
Figure 4.12	Temps de vérification pour le modèle de renforcement réduit.	99
Figure 4.13	Taux d'utilisation de la mémoire pour le modèle de renforcement réduit.	100

INTRODUCTION

La sécurité des réseaux informatiques est devenue un enjeu majeur dans la société moderne. En effet, les réseaux informatiques qui sont naturellement vulnérables, ont acquis une place importante dans toutes les activités entrepreneuriales et dans tous les secteurs de la vie : les banques, les assurances, les hôpitaux, etc.

Durant la dernière décennie, nous avons assisté à l'essor phénoménal d'Internet, ce qui a rendu notre monde numérique entièrement connecté permettant ainsi d'accéder à la manne d'information disponible sur les réseaux. Cependant, cette ouverture vers l'extérieur a laissé les portes ouvertes à ceux, qui trouvent un plaisir à saccager les systèmes informatiques, de pénétrer les réseaux privés et y accomplir des actions malicieuses de destruction ou de vol d'informations confidentielles. Par conséquent, il est crucial de mettre en place des barrières pour protéger les données critiques circulant sur les réseaux privés. Pour ce faire, plusieurs solutions ont été déployées telles que : l'authentification, le cryptage, les logiciels anti-virus, les systèmes de détection d'intrusions (IDS), et les pare-feux.

La solution la plus largement adoptée est les pare-feux. En fait, cet outil constitue la pierre angulaire dans la sécurisation des connexions entre les réseaux privés et les réseaux publics. Le rôle principal des pare-feux est de contrôler le flux d'information au sein d'un réseau. Un pare-feu est caractérisé par une configuration à base de règles qui décident de rejeter ou d'accepter les paquets circulant sur le réseau en fonction de leurs attributs (par exemple : source, destination, protocole, port).

L'efficacité des pare-feux dépend de la cohérence de leur configuration. Cependant, la configuration d'un pare-feu nécessite souvent une intervention humaine. Cette intervention peut être à l'origine des vulnérabilités de sécurité majeures. En effet, la configuration des pare-feux est une très lourde tâche qui se complique davantage avec la taille et la topologie du réseau. Par conséquent, elle est sujette à des anomalies qui peuvent entraver la mise en œuvre d'une politique de sécurité globale d'un réseau. Ces anomalies peuvent engendrer des incohérences au sein d'un pare-feu ou des incohérences inter-pare-feux. Pour pallier ces problèmes, plusieurs travaux de recherche visent, en s'appuyant sur des techniques formelles, à fournir des approches d'aide à la configuration de pare-feux. Ces travaux peuvent être classés en deux catégories : i) la vérification de la cohérence des pare-feux, et ii) l'automatisation de la configuration des pare-feux. L'objectif général de notre travail est de proposer des nouvelles approches formelles d'aide à la configuration des pare-feux en tenant compte des deux catégories citées ci-dessus. Nos contributions sont les suivantes :

- Une approche formelle basée sur le model checking est proposée. Elle permet de détecter

les incohérences de configuration inter-pare-feux dans un réseau informatique.

- Pour atténuer le problème d’explosion de l’espace d’états inhérent au model checking, deux abstractions sont proposées et évaluées en termes de complexité en temps et en espace, par rapport à la taille du réseau et son taux de connectivité.
- L’approche proposée est étendue à l’aspect probabiliste permettant ainsi d’étudier le comportement quantitatif du réseau en se basant sur des métriques de qualité de service tels que le délai d’acheminement des paquets de bout en bout, délai d’attente et le taux de perte.
- Une deuxième approche formelle basée sur la technique de synthèse de contrôleur est implémentée. Elle permet de configurer automatiquement les pare-feux dans le réseau.
- Afin de faire face au problème d’explosion combinatoire, deux abstractions de cette dernière approche sont proposées et évaluées en termes de complexité en temps et en espace, en fonction de la taille du réseau et son taux de connectivité.

La suite de ce mémoire est structurée comme suit :

Le chapitre 1 est dédié aux techniques de modélisation et de vérification utilisées dans le cadre de ce mémoire, notamment la technique de model checking et de synthèse de contrôleur.

Le chapitre 2 présente les réseaux informatiques, leurs différentes architectures, leurs composants ainsi que leurs mécanismes de sécurisation les plus répandus : les IDS et les pare-feux. Il s’intéresse particulièrement au problème de configuration des pare-feux. Il passe en revue les différents types d’anomalies qui compromettent la configuration des pare-feux. Il expose les principales techniques proposées dans la littérature de détection d’anomalies et de configuration automatique des pare-feux.

Le chapitre 3 montre comment modéliser et vérifier par model checking, la cohérence des pare-feux d’un réseau, par rapport à des objectifs de sécurité. Il montre également comment évaluer les performances du réseau, comme par exemple, le délai d’acheminement des paquets de bout en bout, le taux de perte de paquets et les délais d’attente. Nous allons y présenter deux approches formelles permettant respectivement d’étudier l’aspect qualitatif et quantitatif d’un réseau informatique.

Le chapitre 4 est consacré à la configuration automatique des pare-feux, en s’appuyant sur la technique de synthèse de contrôleur. Nous y décrivons notre modèle formel qui permet de configurer les pare-feux en déterminant les règles de filtrage de leurs listes de contrôle d’accès.

Les principales contributions de ce mémoire ainsi que quelques perspectives de recherche sont consignées dans la section conclusion. L’annexe reproduit notre article accepté pour

présentation à la 19ième édition de IEEE Symposium on Computers and Communications.
Cet article est constitué exclusivement de la première contribution énumérée ci-dessus.

CHAPITRE 1

MECANISMES DE MODELISATION ET VERIFICATION FORMELLES

1.1 Introduction

Le processus de conception des systèmes passe essentiellement par trois étapes qui sont : la conception, le développement, et la vérification. La dernière étape est très importante car elle permet de détecter les erreurs et de s'assurer du fonctionnement des systèmes avant leurs exploitations. En effet, elle permet la détection des erreurs à faible coût. Les méthodes formelles sont fortement recommandées pour le développement des systèmes logiciels à sécurité critique.

Au cours de la dernière décennie, la recherche poussée dans le domaine des méthodes formelles a pris une place très importante conduisant ainsi au développement de plusieurs techniques de vérification très puissantes. On distingue trois classes de techniques de vérification : i) Les techniques basées sur la preuve de théorème (voir Cook, 1971; Dingel et Filkorn, 1995) qui utilisent un modèle mathématique du système. Le principe d'une telle technique consiste à déterminer un ensemble d'axiomes (théorème de base), et étudier la capacité du modèle mathématique (démonstrateur) à construire une preuve de théorème. ii) Le model checking (voir Baier *et al.*, 2008) est la technique de vérification la plus utilisée. Elle a été appliquée avec succès dans plusieurs domaines notamment le développement des protocoles de communication. iii) La technique de synthèse de contrôleur (voir Vincent, 2001) est la technique de vérification la plus utilisée. Elle a été appliquée avec succès dans plusieurs domaines notamment le développement des protocoles de communication. iii) La technique de synthèse de contrôleur Vincent (2001) consiste à déterminer un programme dont la synchronisation avec un système non totalement spécifié, vérifie les propriétés attendues du système.

Ce chapitre porte sur une étude bibliographique sur le model checking et la synthèse de contrôleur. La section 1.2 décrit la technique du model Checking, ses limites et ses extensions. La section 1.3 présente les formalismes de description de comportement les plus utilisés en entrée des model checkers à savoir les automates temporisés, les automates temporisés à coût et les automates temporisés de jeux. La section 1.4 présente les logiques temporelles qui permettent de spécifier les propriétés attendues d'un système. La section 1.5 présente la technique de synthèse de contrôleur alors que la dernière section 1.6 est dédiée aux différents outils de modélisation et de vérification formelles (model checkers).

1.2 Model checking

Cette section présente la technique de model checking, ses limites et ses différentes extensions.

1.2.1 Définition du model checking

Le model checking (voir Baier *et al.*, 2008; Clarke *et al.*, 1999) est une technique formelle qui permet de vérifier automatiquement le fonctionnement d'un système. Il a en entrée, d'une part, un modèle qui décrit le comportement d'un système et, d'autre part, un formalisme qui spécifie les propriétés attendues du système. Il consiste à explorer l'espace des états du modèle et à vérifier la conformité des évolutions des états vis-à-vis des propriétés attendues. Il permet ainsi de détecter les erreurs de conception avant l'implémentation réelle du système. Ainsi, le modèle checking permet de vérifier formellement si un modèle satisfait sa spécification et fournir, le cas échéant, un contre exemple. Parmi les modèles de description de comportement les plus utilisés, les automates et leurs diverses extensions sont considérés dans ce mémoire. Les propriétés attendues d'un système sont généralement spécifiées par des formules de logiques temporelles (CTL, LTL, CTL*, TCTL, etc.). Ce mémoire s'intéresse à la logique arborescente CTL et ses extensions temporisées et probabilistes. La technique du model checking a été largement appliquée dans plusieurs domaines. Son intérêt a été particulièrement bien montré pour la vérification des protocoles de communication, les applications multimédia, et la sécurité informatique. Le modèle checking a pour avantage de fournir un contre exemple lorsqu'il détecte une violation de la propriété ce qui facilite la compréhension et la correction des erreurs.

Dans la littérature, plusieurs travaux montrent la polyvalence et l'apport indéniable des techniques de vérification dans la conception de systèmes. En effet, à l'aide de cette technique, il a été prouvé l'exactitude d'une variété de protocoles de logiciels (Model Checking de Clarke *et al.* (voir Clarke *et al.*, 1999)), notamment, des protocoles modernes d'e-commerce. Cette technique a également permis de détecter des bugs dans les systèmes de réservation en ligne des compagnies aériennes. Elle a été utilisée également pour vérifier un module d'exécution du contrôleur de la sonde spatiale "Deep Space 1" en identifiant un défaut de conception majeur. Ce défaut n'a pas été détecté pendant la phase de test et a causé un blocage lors d'un vol expérimental de la sonde à 96 millions de kilomètres au-dessus de la Terre.

La vérification de tout système en utilisant la technique de model checking passe par trois étapes comme l'explique la Figure 1.1 :

- **Modélisation** : Cette étape consiste à générer sous forme d'un modèle, une abstraction du système. Ce modèle doit être assez expressif pour pouvoir décrire sans ambiguïté

les spécificités du système. Il doit également s'appuyer sur une sémantique rigoureuse permettant l'exécution et la vérification, à moindre coût, de propriétés à modéliser. Il s'agit de traduire le système sous forme d'une structure de données finie interprétable par un algorithme de model checking. Parmi ces structures de données, on cite : les automates, les réseaux de Petri, les diagrammes binaires de décision (BDD), etc.

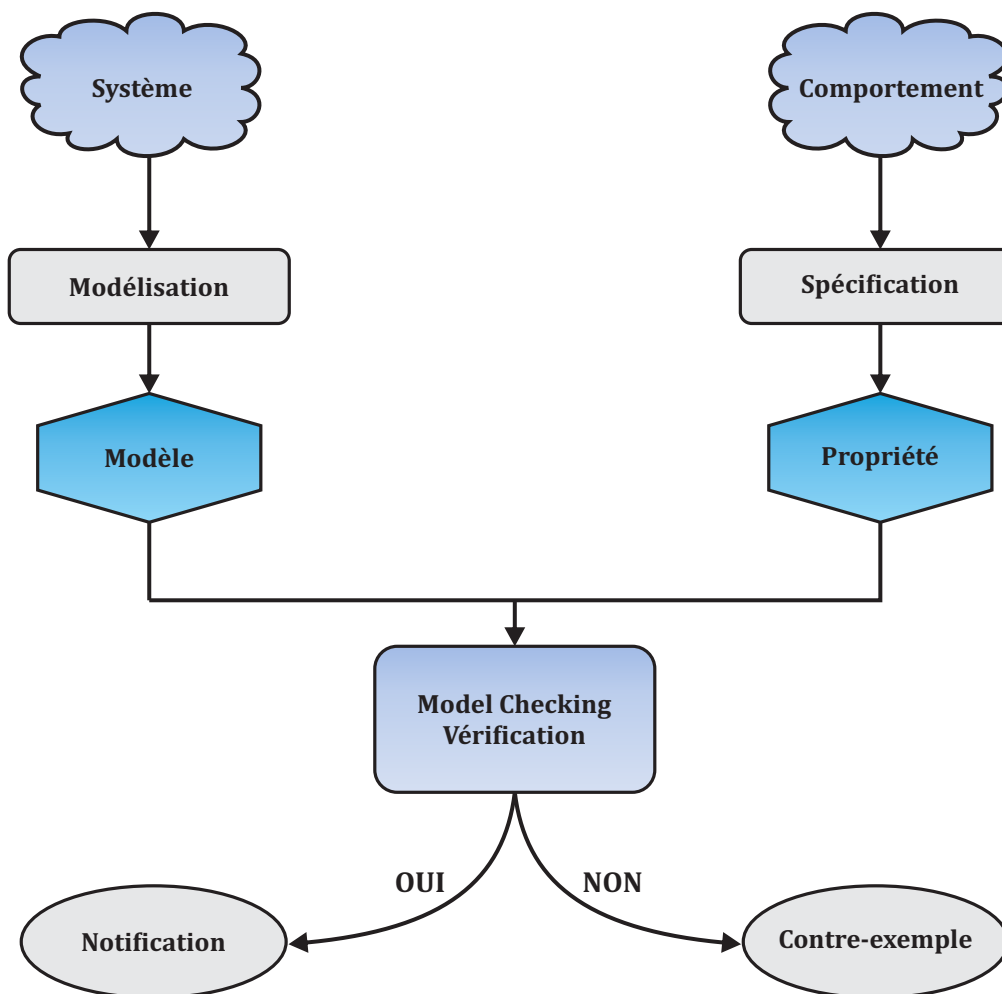


Figure 1.1 Procédure du Model Checking.

- **Spécification** : Cette étape permet de traduire les requis (les propriétés attendues) d'un système dans un formalisme formel. On distingue différents types de requis : la sûreté, la vivacité, l'accessibilité, l'équité et l'absence de blocage, etc. Les logiques temporelles sont les formalismes les plus utilisés pour spécifier des propriétés.

- **Vérification** : Cette étape consiste à mettre en œuvre un algorithme qui prend en entrée un modèle (résultat de l'étape modélisation) et une spécification (résultat de l'étape spécification). Le modèle checker renvoie « vrai » si la spécification est vérifiée et fournit un contre exemple (trace d'erreur), le cas échéant. La vérification d'un modèle se base sur l'exploration de l'espace d'états et peut, donc, se heurter au problème d'explosion combinatoire. Dans ce cas, un retour aux étapes préliminaires (modélisation et vérification) est nécessaire pour modifier le modèle et la spécification afin de réduire le modèle et pallier ce problème majeur. Pour atténuer cette explosion combinatoire d'états, plusieurs techniques de vérification sont proposées. La section suivante discute ce problème et les solutions qui ont été proposées pour y remédier.

1.2.2 Extensions du model checking

Le model-checking s'avère une technique efficace et rigoureuse de vérification automatique de la spécification d'un système. Depuis les années 80, de nombreux outils ont été développés mettant en œuvre cette technique. Au départ, restreint à la vérification de propriétés qualitatives, le model checking s'est peu à peu étendu afin de pouvoir exprimer les aspects quantitatifs nécessaires à l'évaluation des performances des systèmes à savoir l'aspect temporisé (voir Behrmann *et al.*, 2000) et l'aspect probabiliste (voir Norman *et al.*, 2013; Basagiannis *et al.*, 2011). Cette section présente le model checking temporisé qui prend en considération l'aspect temporisé d'un système et le model checking statistique qui traite l'aspect probabiliste.

- **Model checking temporisé** : Le model checking temporisé (voir Bérard *et al.*, 2010) est une extension du model checking classique. Cette nouvelle extension permet de prendre en compte des caractéristiques temporelles durant le processus de vérification. La technique du model checking temporisé permet de vérifier des systèmes temporisés tels que les systèmes temps réel ainsi que des problèmes d'ordonnancement qui traitent explicitement les durées des tâches. Elle permet également de vérifier des propriétés qui incluent de façon explicite des contraintes temporelles, comme, par exemple : "L'alarme se déclenche au plus trois secondes après l'apparition d'un problème". Par conséquent, elle prend en charge des logiques temporelles temporisées (i.e. TCTL, TPTL) faisant intervenir des horloges et des opérateurs pour les manipuler. L'approche du model checking temporisé a été implémentée dans plusieurs outils comme UPPAAL (voir Behrmann *et al.*, 2004), KRONOS (voir Yovine, 1997), HYTECH (voir Henzinger *et al.*, 1997), etc.
- **Model checking statistique (Statistical Model checking : SMC)** : Le modèle

checking statistique (voir David *et al.*, 2011) est une extension du model checking classique basée sur la simulation stochastique. Cette extension a été proposée afin de pouvoir exprimer l’aspect probabiliste nécessaire à l’évaluation des performances de certains systèmes. Le SMC est une approche de vérification automatique des propriétés quantitatives à aspect probabiliste (voir Norman *et al.*, 2013; Basagiannis *et al.*, 2011). La vérification formelle du SMC est appliquée sur un sous-ensemble d’échantillons sous forme des chemins d’exécution. Les échantillons sont définis selon la méthode de Monte Carlo (voir Robert et Casella, 2011). Cette méthode consiste à simuler le modèle plusieurs fois afin de créer un échantillon d’observation suffisant pour estimer la probabilité liée à la propriété en question. Ensuite, par un simple comptage, le SMC estime, avec un certain niveau de confiance, la probabilité de la satisfaction de la propriété envisagée. L’avantage de l’approche statistique réside dans le fait que sa complexité croît proportionnellement avec la taille du modèle. En effet, elle ne stocke pas la totalité du graphe d’états lors de la génération d’un chemin (échantillon), elle stocke uniquement l’état actuel. Ceci réduit considérablement l’utilisation de la mémoire. De plus, cette méthode est applicable à tout processus stochastique et aucune hypothèse de Markov (voir Roblès *et al.*, 2010) n’est nécessaire. Cette approche est implémentée dans de nombreux outils tels que COSMOS(voir Ballarini *et al.*, 2011), PRISM (voir Kwiatkowska *et al.*, 2011), UPPAAL (voir Bulychev *et al.*, 2012), et VESTA (voir ALTurki et Meseguer, 2011).

1.2.3 Explosion combinatoire

La principale limitation du model checking est le problème d’explosion combinatoire des états (voir Clarke *et al.*, 2012). De nombreuses recherches ont été conduites pour trouver une solution à ce problème. Ce volet présente quelques solutions (voir Gueffaz, 2012) :

- **L’abstraction** : Comme son nom l’indique, cette technique permet d’abstraire le modèle. L’abstraction du modèle peut se faire en ignorant quelques états du modèle, ce qui réduit le nombre de transitions et par conséquent la taille du modèle. Cette méthode d’abstraction est appelée ”abstraction par restriction”. Il y a aussi l’abstraction par ”fusion d’états”. Cette méthode consiste à regrouper un ensemble d’états ayant des caractéristiques communes en un seul état. Il est également possible d’effectuer une suppression de certaines variables. Cette méthode est appelée ”abstraction par variables”. L’utilisation des automates observateurs constitue une autre méthode d’abstraction qui consiste à utiliser un automate observateur pour restreindre le comportement de l’automate sans toucher à sa structure.

- **Diagrammes de décision binaires (Binary Decision Diagram : BDD)**(voir **Bryand, 2013**) : Les Diagrammes de décision binaires sont des structures de données qui représentent des fonctions booléennes sous forme d'un graphe dirigé acyclique. Ce graphe permet d'évaluer la valeur de vérité de la fonction booléenne en se basant sur les valeurs de vérité de ses variables. Les diagrammes de décision binaires sont très utiles dans la vérification formelle car ils permettent de représenter le comportement d'un système sous forme d'une formule ensembliste.
- **Model checking symbolique (voir Batt et al., 2010)** : Cette approche consiste à représenter symboliquement les états d'un modèle donné par des ensembles plutôt qu'individuellement. Chaque ensemble est représenté généralement par un diagramme binaire de décision. Le principe du model checking symbolique consiste à calculer des points fixes pour déterminer l'ensemble d'états accessibles ainsi que l'ensemble d'états satisfaisant une propriété donnée. Pour calculer les points fixes, le model checking symbolique détermine l'ensemble d'états prédécesseurs et l'ensemble d'états successeurs à un ensemble d'états donné. Il est plus adapté à la vérification des propriétés exprimées en logique temporelle CTL (Computational Tree Logic) puisqu'il calcule l'ensemble d'états d'une façon itérative.
- **Vérification à la volée** : Le principe de cette méthode consiste à calculer l'ensemble d'états accessibles du système et vérifier simultanément la spécification sur cet espace d'états. L'avantage de cette technique est qu'elle permet d'économiser la mémoire dédiée pour la vérification en stockant uniquement les traces parcourues plutôt que la totalité du graphe à explorer. Cette méthode admet deux variantes de vérification : i) L'analyse en avant qui consiste à calculer itérativement les successeurs des états initiaux et vérifier si l'état cible est calculé ou pas. ii) L'analyse en arrière qui consiste à calculer les prédécesseurs des états cibles et vérifier si un état initial se trouve dans l'ensemble calculé.
- **Modèle checking borné (Bounded model checking : BMC)** (voir **Merz et al., 2012**) : Le modèle checking borné est une solution proposée pour atténuer le problème d'explosion combinatoire. Le principe de fonctionnement du BMC consiste à chercher, dans l'espace d'états du modèle, une exécution de longueur k violant la propriété à vérifier. Autrement dit, BMC consiste à chercher un contre-exemple de longueur k . Si la recherche aboutit alors la propriété est non satisfaite et le BMC génère le contre-exemple, sinon il faut augmenter la borne k jusqu'à trouver un contre-exemple sous la forme d'une exécution bornée de longueur k . Le model checking borné ne permet pas de valider une propriété donnée. Il permet plutôt de prouver qu'elle est non satisfaite

en retournant un contre-exemple.

- **Technique d'ordre partiel** : Cette technique consiste à réduire la taille de l'espace d'états à explorer par un algorithme de vérification. Cette réduction est basée sur l'exploitation des différents ordonnancements des événements indépendants et concurrents éliminant ainsi des chemins d'exécutions redondants. Cette technique peut être considérée comme une abstraction par restriction.

1.3 Formalismes de modélisation

Parmi les formalismes de modélisation du model checking, nous nous intéressons, dans ce mémoire, aux automates temporisés ainsi qu'à leurs extensions, notamment les automates temporisés probalistes et automates temporisés de jeu.

1.3.1 Automates temporisés (Timed Automata : TA)

Les automates temporisés (voir Alur, 1999) ont été proposés par R. Alur et D. Dill en 1991. Il s'agit d'automates classiques munis de variables réelles positives ou nulles appelées horloges. Chaque automate possède un nombre fini d'états appelés locations et un ensemble fini de transitions étiquetées par des gardes ainsi que des actions de réinitialisation d'horloges. Chaque location est associée à une contrainte sur les horloges, appelée invariant qui permet de déterminer la durée de séjour dans un état avant de franchir la transition suivante. Dans un état, une horloge évolue de manière continue et synchrone avec le temps. Sa valeur représente le temps écoulé depuis sa dernière mise à zéro. Un automate temporisé admet deux types de transitions : i) Une transition d'action qui consiste à franchir une transition quand sa garde est vraie. ii) Une transition de durée qui consiste à séjourner dans le même état avec l'évolution continue d'horloges en respectant l'invariant.

À titre illustratif, considérons l'automate temporisé simple, de la Figure 1.2, composé d'une horloge x et deux états e_0 et e_1 . L'automate séjourne dans l'état e_0 jusqu'à une unité de temps. Quand l'horloge x est égale à 1, l'automate franchit instantanément la transition dont la garde est satisfaite ($x == 1$) et réinitialise l'horloge x . Puisque qu'il n'y ait pas d'invariant défini sur l'état e_1 , l'automate peut y séjourner au moins jusqu'à ce que la garde de sa prochaine destination soit satisfaite ($x >= 3$). Autrement dit, l'automate doit séjourner au moins trois unités de temps dans l'état e_1 .

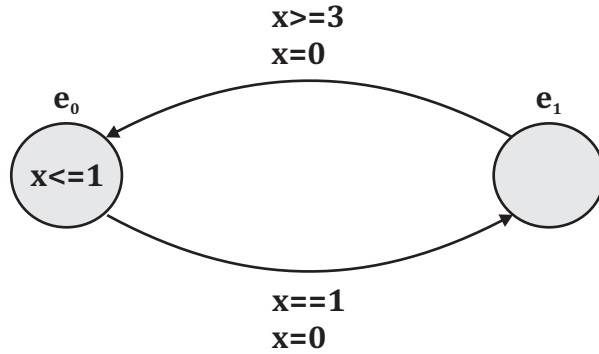


Figure 1.2 Exemple d'automate temporisé.

Formellement, un automate temporisé A est défini par (voir Alur, 1999) :

$A = (Q, q_0, X, \Sigma, I, T)$ où :

1. Q est un ensemble fini et non vide d'états,
2. $q_0 \in Q$ est l'état initial,
3. X est un ensemble fini d'horloges. Soit $C(X)$ l'ensemble de contraintes d'horloges appelées invariants, et 2^X l'ensemble de tous les sous ensembles de X ,
4. Σ est un alphabet fini qui définit un ensemble d'actions,
5. $I : Q \rightarrow C(X)$ est une application qui associe à chaque état un ensemble de contraintes d'horloges de la forme $x \sim c$ où $\sim \in \{<, \leq, =, \geq, >\}$, c est une constante et $x \in X$.
6. $T \subseteq Q \times C(X) \times \Sigma \times 2^X \times Q$ est un ensemble fini de transitions. Chaque élément $t = \langle q, \phi, a, r, q' \rangle \in T$ définit une transition de l'état q vers l'état q' en spécifiant sa garde ϕ , son action a , et le sous ensemble d'horloges r à remettre à zéro suite à son franchissement.

1.3.2 Automates temporisés à coût (Priced Timed Automata : PTA)

Les automates temporisés à coût (voir Behrmann *et al.*, 2005) sont une extension des automates temporisés classiques. Elles permettent une représentation plus efficace de l'aspect quantitatif d'un système temporisé. Les transitions et les états d'un automate temporisé à coût sont étiquetés par des informations additionnelles qui représentent, respectivement, le coût de franchissement d'une transition et le coût par unité de temps de séjour dans un état donné. Ainsi, chaque exécution de l'automate a un coût global, qui est le prix accumulé de tout franchissement d'une transition et tout délai dans les états le long du parcours. Ce modèle est particulièrement pertinent pour la modélisation de la consommation des ressources des systèmes "temps-réel". Ceci est fondamental dans le cadre des systèmes embarqués, qui sont confrontés à plusieurs contraintes de ressources (la bande passante, la mémoire, la consom-

mation d'énergie, etc.).

Considérons l'automate temporisé à coût A de la Figure 1.3. Les états e_0 , e_1 , e_2 et e_3 sont étiquetés par des coûts qui représentent le prix, par unité de temps, du séjour dans la location correspondante. Les transitions qui mènent vers l'état final e_4 sont étiquetées par des coûts qui correspondent respectivement aux prix de franchissement de ces deux transitions. Une exécution possible dans A est $E : (e_0, 1) \rightarrow (e_1, 1) \rightarrow (e_2, 3) \rightarrow (e_4)$. Dans ce cas, l'automate séjourne dans e_0 une unité de temps. Le coût de séjour dans cette location est cinq par unité de temps. L'automate passe, ensuite, à la location e_1 . Aucun coût n'est associé à cette location ou au franchissement de la transition $e_0 \rightarrow e_1$. Ensuite, la transition $e_1 \rightarrow e_2$ est franchie sans aucun coût additionnel. L'automate séjourne dans l'état e_2 trois unités de temps. Le coût de séjour dans cette location est dix par unité de temps, soit $(10 \times 3 = 30)$ unités de temps. Enfin, l'automate arrive à l'état final e_4 en franchissant la transition $e_2 \rightarrow e_4$ avec un coût d'une unité de temps. Par conséquent, le coût total associé à l'exécution E est $5 + 30 + 1 = 36$.

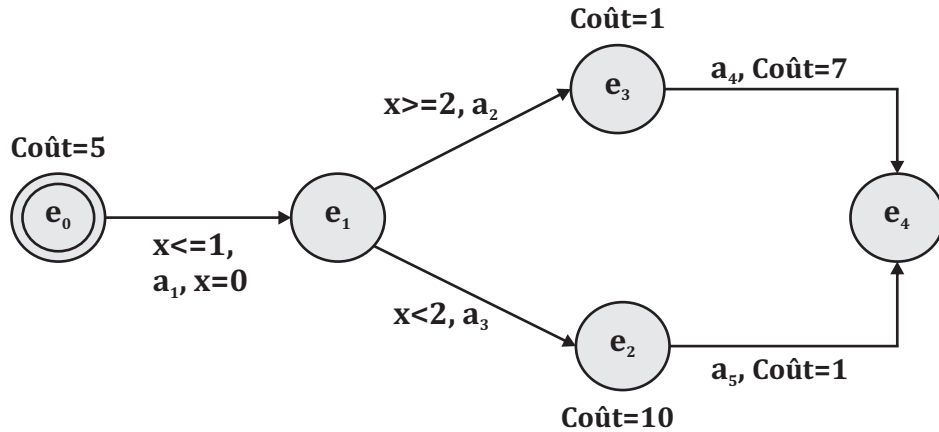


Figure 1.3 Exemple d'automate temporisé à coût.

Formellement, un automate temporisé à coût A est défini par (voir Behrmann *et al.*, 2005) :

$A = (Q, q_0, X, \Sigma, I, T, P)$ où :

1. $(Q, q_0, X, \Sigma, I, T)$ est un automate temporisé,
2. Soit \mathbb{N} l'ensemble des entiers positifs, $P : Q \cup T \rightarrow \mathbb{N}$ est une application qui associe à chaque location un coût de séjour qui quantifie l'évolution d'horloge à cet état, et à chaque transition, un coût de franchissement.

1.3.3 Automates temporisés de jeu (Timed Game Automata, TGA)

Dans les automates temporisés, toutes les transitions franchissables peuvent se produire, sans aucun contrôle, ni restriction supplémentaire. Les automates temporisés de jeu (voir Cassez *et al.*, 2005) permettent de distinguer deux classes de transitions : contrôlables et non contrôlables. Le franchissement d'une transition contrôlable peut être fixé, retardé ou encore bloqué, si besoin est. Par contre, aucun contrôle n'est possible sur les transitions incontrôlables. Les automates temporisés peuvent être considérés comme étant des automates temporisés de jeu où toutes les transitions sont incontrôlables. D'une façon générale, les automates temporisés de jeu permettent de formaliser des problèmes de contrôle des systèmes temporisés par un système de transitions dont l'évolution résulte de l'interaction de plusieurs agents (ou joueurs). Les transitions contrôlables sont associées à des conditions de déclenchement appelées aussi politiques ou stratégies. Le contrôleur recherché est ainsi l'ensemble de stratégies pour un agent (ou une coalition d'agents) permettant de résoudre le problème de contrôle et satisfaire l'objectif de contrôle correspondant quel que soit le comportement des autres agents. Le contrôleur peut être aussi défini comme un autre automate qui est synchronisé avec les transitions contrôlables de l'automate de jeu permettant d'y ajouter les conditions de déclenchement nécessaires afin de satisfaire l'objectif de contrôle. Généralement, pour un système temps réel, les transitions contrôlables sont activées par le contrôleur et les transitions incontrôlables, quant-à-elles, sont activées par l'environnement.

Formellement un automate temporisé de jeu (jeu temporisé) est un 7-uplet $A = (Q, q_0, X, \Sigma_1, \Sigma_2, I, T)$, où :

1. $(Q, q_0, X, \Sigma_1 \cup \Sigma_2, I, T)$ est un automate temporisé, et
2. Σ_1 et Σ_2 sont deux alphabets disjoints, Σ_1 représente l'ensemble des actions incontrôlables et Σ_2 représente l'ensemble des actions contrôlables.

Considérons l'automate temporisé de jeux de la Figure 1.4, où les actions incontrôlables (u_1, u_2, u_3) sont représentées par des traits pointillés et les actions contrôlables $(c_1, c_2, c_3, c_4, c_5)$ sont représentées par des traits pleins. Dans cet exemple, il n'y a pas d'invariant sur l'état initial. L'automate peut alors décider de quitter l'état initial à tout moment. Si le contrôleur veut atteindre l'état "GOAL", il doit :

- à partir de l'état e_0 , exécuter l'action c_1 avant que l'horloge x ne dépasse la valeur 1,
- à partir de l'état e_1 , exécuter l'action c_2 si x atteint la valeur 2,
- à partir de l'état e_2 , effectuer l'action c_3 , et à partir de l'état e_3 , choisir l'action c_5 , (il doit éviter l'action c_4),

Nous venons de définir de façon informelle une stratégie de contrôle qui permet de forcer l'accessibilité de l'état cible "GOAL". Si le contrôleur souhaite jouer l'action c_1 lorsque l'horloge

x est inférieure ou égale à un, il peut franchir alors la transition correspondante uniquement si l'environnement n'a pas effectué auparavant l'action u_2 quand l'horloge x est strictement inférieure à un. Si le contrôleur choisit de quitter l'état initial lorsque x vaut un et réussit à effectuer l'action c_1 , il arrive à l'état e_1 avec x égale à un. Dans ce cas, il peut séjourner dans e_1 jusqu'à ce que l'horloge x soit supérieure ou égale à deux. Par la suite, il peut proposer l'action c_2 qui mène vers l'état cible "GOAL". Si l'environnement choisit à l'état initial un délai plus court que le contrôleur tel que l'horloge x est strictement inférieure à un ($x = 0.5$), alors l'action u_1 sera exécutée et le jeu atteindra l'état e_2 avec l'horloge x est à zéro. Le contrôleur peut, dans ce cas, exécuter les actions c_3 et c_5 afin d'atteindre l'état cible.

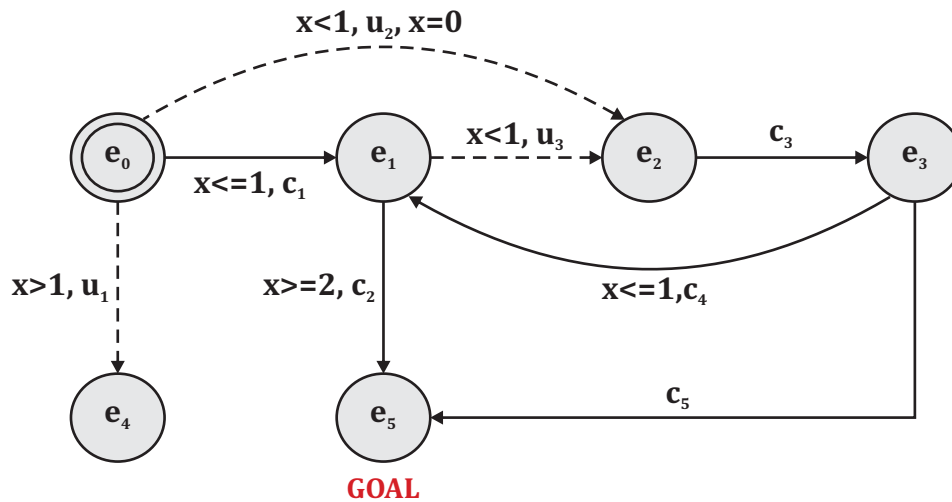


Figure 1.4 Exemple d'automate temporel de jeu.

1.4 Formalisme de spécification

La section précédente a présenté quelques mécanismes utilisés pour mettre en oeuvre la phase modélisation de l'approche de model checking. Dans cette section, nous nous intéressons à la phase spécification. Généralement, le model checking utilise une logique temporelle afin de spécifier les propriétés décrivant les requis du système étudié. Une logique temporelle est un langage de spécification qui permet d'exprimer des propriétés faisant intervenir la notion d'ordonnancement dans le temps. Une logique temporelle permet d'exprimer des propriétés de type "après une instruction i un événement j se produira" en utilisant des opérateurs temporels. On distingue principalement deux classes de logiques temporelles : i) les logiques temporelles linéaires (Linear Temporal Logic : LTL) (voir Rozier, 2011; Vardi, 1996), et ii) les

logiques temporelles arborescentes (Computation Tree Logic : CTL) (voir Kugler *et al.*, 2005). La logique temporelle linéaire permet de décrire le comportement des systèmes qui évoluent linéairement avec le temps. En effet, LTL permet d'exprimer des propriétés portant sur une séquence d'états donnée appelée exécution. Contrairement à LTL qui traite linéairement une seule trace d'exécution à la fois, CTL permet d'analyser plusieurs états futurs à partir d'un état donné du système en considérant plusieurs chemins d'exécution. Dans ce mémoire, nous nous intéressons à la logique temporelle CTL, étant la logique de base supportée par tous les outils que nous avons utilisés. Cette section présente, tout d'abord CTL. Ensuite les différentes classes des propriétés temporelles sont discutées.

1.4.1 La logique temporelle arborescente CTL

La logique temporelle arborescente (voir Kugler *et al.*, 2005) permet d'exprimer des propriétés portant sur les arbres d'exécution qui représentent l'évolution du système. Les formules CTL sont exprimées à l'aide du formalisme suivant :

- **Les propositions atomiques** : p, q, \dots , true, false qui s'interprètent directement sur un état.
- **Les connecteurs booléens** : \neg (négation), \wedge (et logique), \vee (ou logique), \Leftrightarrow (équivalence), \Rightarrow (implication).
- **Les connecteurs temporels** : $EF p \mid EG p \mid EpUq \mid EX p \mid AF p \mid AG p \mid ApUq \mid AX p$ avec E et A sont des quantificateurs de chemins.

La syntaxe d'une formule CTL est alors la suivante :

$\phi, \psi ::= p \mid q \mid \neg\phi \mid \wedge\psi \mid \phi \vee \psi \mid \phi \Leftrightarrow \psi \mid \phi \Rightarrow \psi \mid EF\phi \mid EG\phi \mid E\phi U\psi \mid EX\phi \mid AF\phi \mid AG\phi \mid A\phi U\psi \mid AX\phi$ avec ϕ et ψ sont deux formules CTL.

Les formules CTL peuvent être interprétées comme suit :

- $EF\phi$: Il existe une séquence d'états (opérateur E) qui mène à un état où la propriété ϕ est vérifiée (opérateur F).
- $AF\phi$: Pour toute séquence d'états (opérateur A), il existe un état où la propriété ϕ est vérifiée (opérateur F).
- $EG\phi$: Il existe une séquence d'états (opérateur E) où la propriété ϕ est toujours vérifiée (opérateur G).
- $AG\phi$: Pour toute séquence d'états (opérateur A), la propriété ϕ est toujours vérifiée (opérateur G).

- $AX \phi$: La propriété ϕ est vérifiée sur tous les états (opérateur A) immédiatement successeurs de l'état courant (opérateur X).
- $EX \phi$: Il existe une séquence d'états (opérateur E) où il existe un état dont l'état successeur (opérateur X) satisfait la propriété ϕ .
- $A\phi U \psi$: Pour toute séquence d'états (opérateur A) la propriété ϕ est vérifiée jusqu'à ce que la propriété ψ soit satisfaite.
- $E\phi U \psi$: Il existe une séquence d'états (opérateur E) sur laquelle la propriété ϕ est vérifiée jusqu'à ce que la propriété ψ soit satisfaite.

Certaines formules CTL sont illustrées par la Figure 1.5.

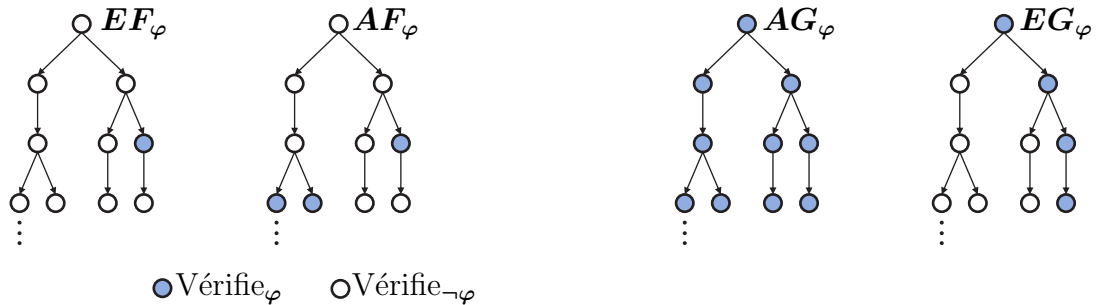


Figure 1.5 Illustration de certaines formules CTL.

Formellement, CTL est interprétée sur les états d'un système de transitions (qui caractérise les arbres issus de ces états).

Définition 1 (Système de transitions étiqueté S (voir Arnold, 1990)) : Un système de transitions S est un tuple $(Q, q_0, A, \rightarrow, L)$ tel que :

- Q est un ensemble (fini) d'états,
- $q_0 \in Q$ est l'état initial,
- A est un alphabet (fini) d'actions,
- $\rightarrow \subseteq Q \times A \times Q$ est la relation de transition de S . On note $q \xrightarrow{a} q'$ si $(q, a, q') \in \rightarrow$,
- $L : Q \rightarrow 2^{AP}$ où AP est un ensemble de propositions atomiques. $L(q)$ donne l'ensemble des propositions que satisfait l'état q .

Définition 2 (Chemin dans un système de transitions) (voir Cassez, 2003) : Un chemin dans un système de transitions est une séquence d'états $\sigma = q_0, q_1, q_2, \dots$ telle que $(q_i, q_{i+1}) \in \rightarrow$ pour tout $i > 0$.

Définition 3 (Sémantique de CTL) (voir Cassez, 2003) : Soit q un état de S . La sémantique de CTL est définie inductivement par :

- $\phi \in AP, q \models \phi \iff p \in L(q),$
- $q \models \phi \vee \phi' \iff q \models \phi \text{ ou } q \models \phi',$
- $q \models \neg\phi \iff q \not\models \phi,$
- $q \models AX\phi \iff \forall q' \text{ tel que } (q, q') \in \rightarrow, q' \models \phi,$
- $q \models EX\phi \iff \exists q' \text{ tel que } (q, q') \in \rightarrow, q' \models \phi,$
- $q \models AX\phi \iff \forall \sigma \text{ un chemin tel que } \sigma(0) = q, \exists j \text{ tel que } \sigma(j) \models \phi,$
- $q \models EF\phi \iff \exists \text{ un chemin } \sigma \text{ tel que } \sigma(0) = q \text{ et } \exists j \text{ tel que } \sigma(j) \models \phi,$
- $q \models AG\phi \iff \forall \sigma \text{ un chemin tel que } \sigma(0) = q, \forall j, \sigma(j) \models \phi,$
- $q \models EG\phi \iff \exists \sigma \text{ tel que } \sigma(0) = q \text{ et } \forall j, \sigma(j) \models \phi,$
- $q \models A(\phi U \phi') \iff \forall \sigma \text{ tel que } \sigma(0) = q, \exists j \text{ tel que } \sigma(j) \models \phi' \text{ et } \forall k \text{ tel que } k < j, \sigma(k) \models \phi,$
- $q \models E(\phi U \phi') \iff \exists \text{ un chemin } \sigma \text{ tel que } \sigma(0) = q, \exists j \text{ tel que } \sigma(j) \models \phi' \text{ et } \forall k \text{ tel que } k < j, \sigma(k) \models \phi.$

1.4.2 Extensions de la logique temporelle CTL

La logique temporelle s'est peu à peu étendue donnant naissance à d'autres logiques temporelles beaucoup plus expressives comme CTL*, TCTL, PWCTL. Dans ce volet, nous mettons l'accent sur les logiques temporelles TCTL (Timed Computation Tree Logic) (voir Alur *et al.*, 1993) et PWCTL (Probabilistic Weighted Computation Tree Logic)(voir Bulychev *et al.*, 2011). Nous discutons uniquement ces deux logiques puisqu'elles sont supportées par les outils que nous avons utilisés dans ce travail, respectivement UPPAAL-TIGA et SMC-UPPAAL.

- **La logique TCTL (Timed Computation Tree Logic) (voir Alur *et al.*, 1993) :**
C'est une extension de la logique temporelle CTL qui permet d'exprimer des propriétés faisant intervenir des quantifications temporelles (i.g. des propriétés de type "un événement i se produit en moins de quatre secondes avant qu'un autre événement j ne soit produit"). La syntaxe de la logique temporelle TCTL se base sur le formalisme de la logique temporelle CTL en utilisant des propositions faisant intervenir des contraintes sur les horloges : $\phi, \psi ::= p \mid \dots \mid \neg\phi \mid \vee \wedge \psi \mid \phi \vee \psi \mid \phi \Leftrightarrow \psi \mid \phi \Rightarrow \psi \mid EF\phi \mid EG\phi \mid E\phi U \psi \mid A\phi U \psi \mid EX\phi \mid AF\phi \mid AG\phi \mid AX\phi \mid x \sim c \mid x - y \sim c$, où ϕ et ψ sont

deux formules CTL, $\sim \in \{<, >, \leq, \geq, =\}$ et $c \in \mathbb{N}$, x et y sont des horloges et p est une proposition atomique.

Par exemple $AG(\text{erreur} \Rightarrow AF(x \leq 3 \wedge \text{alarme}))$ est une formule TCTL qui vérifie le déclenchement éventuel d'une alarme dans un délai inférieur à trois unités de temps en cas d'erreur.

- **La logique PWCTL (Probabilistic Weighted Computation Tree Logic)(voir Bulychev et al., 2011)** : C'est une extension de la logique temporelle TCTL qui permet d'exprimer des propriétés liées à la performance d'un système, telles que la probabilité de satisfaction d'un critère de performance (durée et coût). La syntaxe d'une formule PWCTL est la suivante : $\phi ::= P(F_{C \leq c} \phi) \sim p \mid P(G_{C \leq c} \psi) \sim p$ où C est une horloge, $c \in \mathbb{N}$, ψ est une propriété, $\sim \in \{<, \leq, =, \geq, >\}$, $p \in [0, 1]$, et P est l'opérateur de probabilité permettant d'estimer la probabilité de satisfaction de la propriété envisagée.

1.4.3 Classifications des propriétés temporelles

Il existe cinq grandes classes de propriétés pouvant être exprimées en logiques temporelles (voir Parreaux, 2000) : i) Propriété de vivacité, ii) Propriété de sûreté, iii) Propriété d'équité, iv) Propriété d'atteignabilité et v) Propriété d'absence de blocage. Dans ce volet nous présentons ces différentes classes de propriétés.

- **Propriété de vivacité (liveness property)** : Une propriété de vivacité permet d'exprimer qu'un état ou un événement va forcément se produire dans le futur. Ce type de propriété peut être exprimé en logique LTL en utilisant l'opérateur F (éventuellement) et l'opérateur U (jusqu'à). Cette propriété peut être également exprimée en logique CTL en utilisant l'opérateur AF.

Exemple : $AF \text{ GOAL}$, cette propriété permet de vérifier que le processus va atteindre fatalement son état cible GOAL.

- **Propriété de sûreté (safety property)** : Une propriété de sûreté permet d'exprimer que quelque chose de mauvais n'arrivera jamais. Cette propriété peut être exprimée en utilisant l'opérateur temporel G (toujours).

Exemple : $AG \text{ NOT_FAIL}$, cette propriété permet de vérifier que le processus ne tombe jamais dans un état d'échec.

- **Propriété d'atteignabilité (Reachability)** : Une propriété d'atteignabilité permet d'exprimer qu'une certaine situation peut être atteinte. Cette propriété est moins forte que la vivacité qui exprime qu'une certaine situation doit forcément être atteinte. La

propriété d'atteignabilité peut être exprimée en utilisant l'opérateur temporel EF. Pour vérifier une propriété d'atteignabilité, il suffit de parcourir tout le graphe d'états et s'arrêter lorsqu'un tel état est atteint ou tous les états ont été explorés.

Exemple : EF PANNE, cette propriété exprime que le processus peut tomber en panne.

- **Propriété d'équité (Fairness)** : Une propriété d'équité permet d'exprimer que quelque chose de bon se répète infiniment. L'équité peut être exprimée en utilisant la logique temporelle LTL. Cependant, elle est non exprimable en utilisant la logique temporelle CTL puisque CTL ne supporte pas des opérateurs de type A (FG p). En effet, dans CTL, les opérateurs X, F, G et U doivent être précédés directement de A ou E.
- **Absence de blocage (Deadlock freeness)** : Une propriété d'absence de blocage permet d'exprimer qu'un système ne se bloque pas dans un état où il ne peut plus évoluer (aucune transition vers un autre état n'est possible). Une propriété d'absence de blocage peut être exprimée par la formule CTL : $AG (EX \text{ true})$. Une autre méthode pour conclure quant à l'absence de blocage est de vérifier que les états de blocage ne font pas partie de l'ensemble d'états accessibles.

1.5 Synthèse de contrôleur

Dans la première partie de ce chapitre, nous avons présenté l'approche de model checking. Dans cette section, nous présentons une approche plus générale appelée synthèse de contrôleur (voir Altisen *et al.*, 2005; Vincent, 2001; Altisen, 2001). Il a été évoqué que le model checking est une méthode de vérification qui intervient uniquement à la fin du développement du programme pour vérifier sa spécification. Généralement cette approche vérifie un modèle complet du système incluant l'environnement à contrôler. Ce système évolue sans influence extérieure. La technique de synthèse de contrôleur quant à elle, fonctionne sur des systèmes ouverts non totalement spécifiés. Le but de cette technique, présentée par la Figure 1.6, est de trouver un contrôleur permettant de forcer le système à faire les choix nécessaires afin de satisfaire une spécification donnée. Plus précisément, le contrôleur permet d'intervenir au moment de la conception pour compléter le système ouvert de sorte qu'il satisfasse ses requis (les propriétés attendues). Dans le cadre de la synthèse de contrôleur, le problème de contrôle se modélise généralement par des automates de jeu où les comportements de deux joueurs y sont représentés : L'environnement qui exécute ses actions conformément aux interactions du système avec son environnement, et le contrôleur qui a la capacité d'établir une stratégie pour forcer le système à satisfaire ses requis.

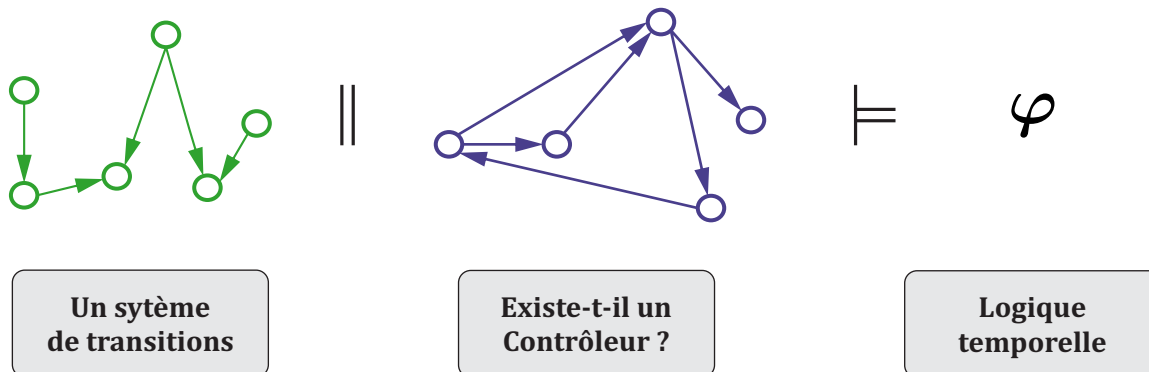


Figure 1.6 Synthèse de contrôleur.

1.6 Model checkers

Les techniques de vérification par model checking sont classées en deux principales catégories [40] :

- **Les modèles de vérification qualitatifs** : Ce sont des outils de vérification de l'aspect qualitatif d'un système donné. Particulièrement, ils abstraient toute information quantitative sur les états du système tels que la durée de séjour dans un état donné et la durée qui sépare un état des autres. Il existe plusieurs modèles de vérification à aspect qualitatif tels que NuSMV (voir Cimatti *et al.*, 2002) , ASTRAL (voir Dang et Kemmerer, 1997), et SPIN (voir Holzmann, 2004).
- **Les modèles de vérification quantitatifs** : Ce sont des outils de vérification permettant de quantifier l'aspect temps en utilisant des variables réelles appelées horloges. Ils sont plus expressifs que les modèles de vérification qualitatifs puisqu'ils permettent de vérifier des propriétés temporisées. Il existe plusieurs modèles de vérification à aspect quantitatif tels que UPPAAL (voir Behrmann *et al.*, 2004), HYTECH (voir Henzinger *et al.*, 1997), KRONOS (voir Yovine, 1997).

1.6.1 Choix du model checker

Dans ce mémoire, une attention particulière a été portée aux modèles de vérification quantitatifs. Kim G et al. (voir Larsen *et al.*, 1995) ont analysé les performances de trois outils de vérification UPPAAL, HYTECH et KRONOS. Leur analyse a été basée sur l'étude du protocole d'exclusion mutuelle de Fisher. Ils ont montré que l'outil UPPAAL est plus puissant que HYTECH et KRONOS. En effet, d'après la Figure 1.7, UPPAAL supporte la

vérification du modèle étudié jusqu'à huit processus contrairement à HYTECH et KRONOS qui échouent au bout de cinq et quatre processus, respectivement. Cette étude a montré également que le temps mis par UPPAAL est considérablement réduit par rapport aux deux autres outils.

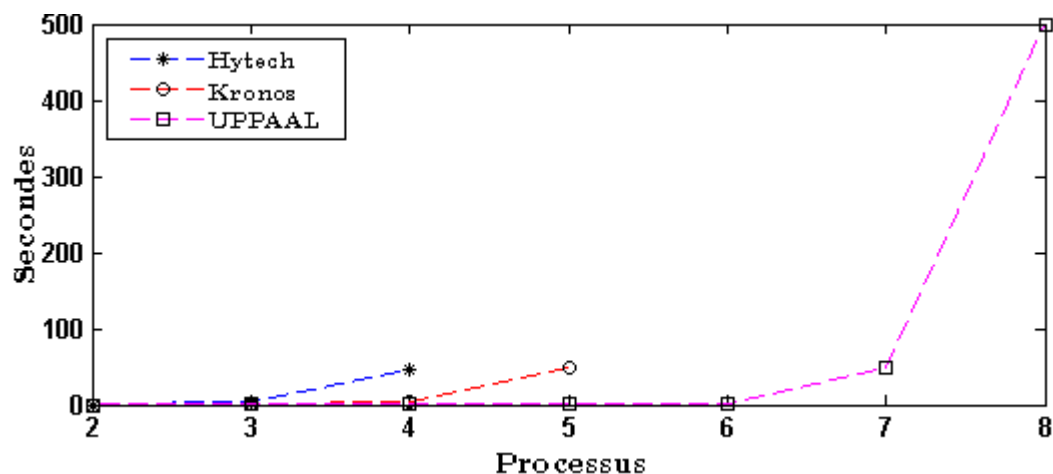


Figure 1.7 Comparaison des performance de UPPAAL, HYTECH et KRONOS (voir Larsen *et al.*, 1995).

Outre sa capacité à traiter des systèmes plus complexes, UPPAAL offre une interface graphique très conviviale qui facilite l'étape de modélisation. D'autre part, il offre un simulateur qui facilite la vérification du bon fonctionnement des automates décrits, avant d'entamer l'étape de vérification. Par ailleurs, UPPAAL offre une suite d'outils qui donnent plusieurs possibilités de vérification telles que l'outil UPPAAL TIGA qui permet de mettre en oeuvre la technique de synthèse de contrôleur afin de forcer le système à satisfaire sa spécification, et l'outil SMC UPPAAL qui permet de vérifier le comportement quantitatif d'un système en utilisant des calculs statistiques et des estimations de probabilité. Cette diversité dans les options de vérification offertes par le logiciel UPPAAL, permet d'étudier le comportement d'un réseau informatique selon plusieurs aspects (aspect qualitatif, probabiliste et correctif). D'après toutes les raisons citées ci-dessus, il est clair que UPPAAL est le model checker le plus adapté à nos objectifs. Dans la section suivante, nous présentons UPPAAL, ainsi que ses extensions TIGA et SMC que nous avons utilisés également dans notre travail.

1.6.2 UPPAAL

Outil UPPAAL

UPPAAL (voir Behrmann *et al.*, 2004) est un outil de modélisation et de vérification de systèmes temps-réel qui peuvent être représentés sous forme de processus non-déterministes et interactifs ayant une structure de contrôle finie et des horloges avec des valeurs réelles. Un modèle d'un système dans UPPAAL (voir Bengtsson *et al.*, 1996) se compose d'un réseau de processus décrits par des automates temporisés étendus communiquant par des canaux ou des structures de données partagées. La conception de l'outil UPPAAL est basée sur l'architecture Client/Serveur, comme le montre la Figure 1.8. Le serveur est représenté par la machine UPPAAL développée en C++ et le client est représenté par l'interface GUI développée en JAVA™. La communication entre la machine UPPAAL et son interface graphique est établie à travers des protocoles internes.

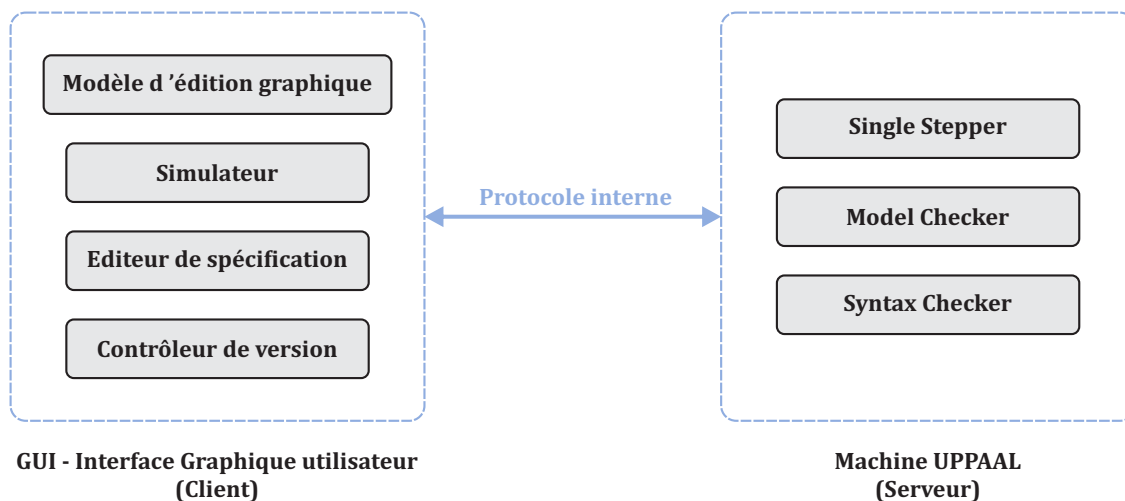


Figure 1.8 Architecture de UPPAAL (voir Bengtsson *et al.*, 1996).

La machine UPPAL assure plusieurs fonctionnalités (voir Bengtsson *et al.*, 1996) :

- **atg2ta** : C'est un compilateur qui traduit la représentation graphique (.atg) d'un réseau d'automates temporisés à une représentation textuelle (.ta).
- **Hs2ta** : Ce programme permet de transformer des automates hybrides linéaires, où la vitesse d'horloges est définie par un intervalle, en réseaux d'automates temporisés.
- **checkta** : Etant donné une représentation textuelle (.ta), ce programme permet d'appliquer une vérification syntaxique de la description du système.

- **verifyta** : Ce programme est le noyau de vérification UPPAAL. Il permet de décider si une spécification donnée est satisfaite ou non et générer un contre exemple dans le cas de violation de la spécification.

Syntaxe

Un modèle UPPAAL est basé sur les automates temporisés. La description d'un modèle se compose de trois parties : i) les déclarations globales et locales, ii) les modèles d'automates, et iii) la définition du système. Ce volet présente la syntaxe de UPPAAL.

- **Déclarations** : Les déclarations peuvent être locales ou globales. UPPAAL permet de déclarer des horloges (type clock), des entiers bornés (type int), des booléens (type bool), des canaux de synchronisation (type chan), des tableaux, des structures de données (type struct), et de définir des types de données (typedef).
- **Locations** : Un automate temporisé est considéré comme un graphe à états. Les locations représentent les états dans ce graphe. Elles se communiquent via les transitions. Un identifiant optionnel peut être associé à chaque location afin de l'identifier. Elles peuvent être étiquetées par des invariants. Un invariant peut être une expression composée d'un ensemble de contraintes sur des horloges, une différence entre des horloges ou des expressions booléennes ne faisant pas intervenir des horloges. UPPAAL distingue trois types de locations :
 - Location initiale : Chaque automate doit avoir une location initiale représentée par un double cercle.
 - Location urgente : La location urgente ne permet pas de faire passer le temps quand le processus s'y trouve. Sémantiquement, une location urgente est équivalente à une location dont l'invariant est $x \leq 0$ où x est une horloge remise à zéro sur toutes les transitions entrantes de la location en question.
 - Location engagée : La location engagée ne permet pas l'écoulement du temps quand un processus s'y trouve. Elles sont utiles pour assurer l'atomicité de la synchronisation entre trois processus ou plus. Elles sont plus prioritaires que les locations urgentes.
- **Transitions** : Les locations sont connectées entre elles par les transitions (edges). Ces dernières sont étiquetées par des sélections, des gardes, des synchronisations et des mises à jour des variables et des fonctions.

- Sélection : Cette option permet d’associer, d’une façon indéterminée, à une variable donnée une valeur dans un intervalle bien déterminé. Les gardes, la synchronisation et les mises à jour peuvent être spécifiées en fonction de la valeur sélectionnée.
- Garde : Une transition est franchie si et seulement si la garde spécifiée est satisfaite. Les gardes expriment des contraintes d’horloges ou de données nécessaires pour franchir une transition. Formellement, les gardes peuvent être spécifiées comme suit : Soient x et y sont deux horloges ou deux variables de données. Une garde est de la forme $x r n$ ou $(x - y) r n$ où $r \in \{\leq, \geq, =, >, <\}$, et n est une expression entière.
- Synchronisation : Les différents automates dans un modèle UPPAAL peuvent se synchroniser via des canaux de synchronisation permettant l’envoi et la réception d’un message. Deux transitions dans deux processus différents peuvent se synchroniser si et seulement si les gardes des deux transitions sont satisfaites et elles admettent deux étiquettes de synchronisation $a_1!$ (envoi) et $a_2?$ (réception) où a_1 et a_2 représentent le même canal de synchronisation. Quand deux processus se synchronisent, les deux transitions sont franchies au même temps. Les mises à jour de la transition portant l’étiquette $a_1!$ se font avant celles de la transition portant l’étiquette $a_1?$. Il existe deux types de canaux de synchronisation : i) les canaux de synchronisation binaires qui permettent de synchroniser deux processus via un seul canal de synchronisation a (au moyen de deux actions $a!$ et $a?$) et ii) les canaux de synchronisation de diffusion (broadcast) qui permettent de synchroniser un processus avec plusieurs autres processus. Une transition avec une étiquette de synchronisation $a!$, avec a est un canal broadcast, peut synchroniser avec chaque transition ayant une étiquette de synchronisation $a?$ et dont les gardes sont satisfaites.
- Mise à jour : L’exécution de la mise à jour sur une transition permet de changer l’état du système. Une mise à jour est une liste d’expressions séparées par des virgules et exécutées dans un ordre séquentiel. L’opérateur d’assignement est « = ».

Vérification UPPAAL

UPPAAL permet de vérifier des propriétés d’accessibilité, de sûreté, de vivacité et d’absence de blocage. Il utilise un sous ensemble étendu de la logique TCTL (Timed Computation Tree Logic). Soient p et q deux propositions atomiques. Les propriétés seront définies comme suit :

- **L'atteignabilité** : Elle peut être exprimée par $A \langle \rangle p$ (tout les chemins d'exécution possibles atteignent éventuellement un état satisfaisant la propriété p).
- **La sureté** : Elle peut être exprimée par $E \square p$ (il existe au moins un chemin dont tous les états satisfont la propriété p) ou par $A \square p$ (tout état accessible satisfait la propriété p).
- **La vivacité** : Elle peut être exprimée par $E \langle \rangle p$ (Il existe au moins un chemin d'exécution qui mène éventuellement à un état satisfaisant la propriété p) ou par $q \rightarrow p$ (q toujours mène vers p : tout chemin qui débute par un état qui satisfait q atteint éventuellement un état qui satisfait forcément p).
- **L'absence de blocage** : Elle est vérifiée en utilisant la proposition "deadlock". Appliqué sur un état, deadlock retourne "vrai" si et seulement si aucune évolution du système n'est possible à partir de cet état. $E \langle \rangle \text{deadlock}$ est utilisé pour vérifier s'il existe un deadlock et $A \square \text{notdeadlock}$ est utilisé pour vérifier qu'il n'y a aucun deadlock dans le système.

UPPAAL offre à l'utilisateur les options de vérification suivantes :

- **Exploration (en profondeur, en largeur ou aléatoire)** : Cette option permet de déterminer dans quel ordre l'espace d'état est exploré. L'exploration de l'espace d'états en largeur-d'abord est généralement l'option la plus efficace lorsqu'il s'agit d'explorer la totalité de l'espace d'états ou de générer des traces courtes ou rapides. L'exploration de l'espace d'états en profondeur-d'abord et l'exploration aléatoire s'avèrent des options plus efficaces que la recherche en largeur lorsqu'il est prévu de trouver un contre exemple.
- **Réduction d'espace d'états (aucune réduction, réduction agressive, réduction conservative)** : Lors de l'exploration de l'espace d'états, UPPAAL essaye d'éviter le stockage de tous les états dans la mémoire afin de garantir la terminaison de la vérification. Cette option permet de déterminer la manière utilisée par UPPAAL afin d'éviter de stocker des états. L'option « Aucune réduction » permet de conserver tous les états, la réduction agressive permet d'éviter de stocker plus d'un état par cycle et la réduction conservatrice évite de stocker les états engagés.
- **Représentation de l'espace d'états (DBM : Difference Bound Matrix, Structure de Donnée Compacte, Sous-Approximation, Sur-Approximation)** : Cette option détermine la façon utilisée pour présenter l'espace d'états. Les DBM sont souvent rapides, mais pour les modèles ayant de nombreuses horloges, ils sont consommateurs en termes de mémoire. La structure de données compacte assure une représentation plus

compacte de l'espace d'états. En particulier, pour les modèles avec plusieurs horloges, cette option permet de réduire considérablement la consommation de mémoire mais elle est plus lente que DBM. Pour les modèles avec peu ou pas d'horloges, cette option permet d'activer d'autres techniques d'économie de mémoire au détriment de la vitesse. La sous-approximation utilise les tables de hachage pour représenter l'espace d'états. Il en résulte une sous-approximation dont le degré peut être réglé en ajustant la taille de la table de hachage qui peut être sélectionnée avec l'option « taille de la table ». La sur-approximation utilise l'approximation des zones. Pour les modèles sans horloges, ce paramètre n'a aucun effet. Quand une représentation approximative est utilisée, UPPAAL ne peut pas faire des conclusions déterministes par rapport à la représentation choisie.

1.6.3 UPPAAL TIGA

UPPAAL TIGA (voir Behrmann *et al.*, 2007) est une extension de UPPAAL permettant la conception des jeux temporisés sous la forme d'un réseau d'automates temporisés de jeu. Cette version dérivée de UPPAAL met en oeuvre le premier algorithme efficace de vérification à la volée pour vérifier des propriétés d'atteignabilité et de sûreté sur des modèles de jeux. L'algorithme proposé par UPPAAL TIGA est une extension symbolique de l'algorithme de vérification à la volée proposée par Liu et Smolka [54] pour les modèles de vérification à temps linéaire des systèmes à états finis. L'algorithme de vérification de UPPAAL TIGA a une complexité linéaire en fonction de la taille des automates du modèle à vérifier.

UPPAAL TIGA applique la technique de la synthèse de contrôleur [55] présentée dans la Figure 1.8. Cette technique permet de forcer le comportement du système afin de satisfaire un objectif de contrôle donné. Particulièrement, cette approche consiste à construire un contrôleur C , de manière à définir le comportement souhaité du système. Ensuite, elle consiste à vérifier si le système contrôlé $C(S)$ satisfait ou non l'objectif de contrôle. UPPAAL TIGA prend en entrée un modèle composé d'un ensemble d'automates temporisés de jeu et une propriété exprimée dans la logique temporelle TCTL. Il fournit en sortie un ensemble stratégies dites "gagnantes" satisfaisant la propriété (le contrôleur).

La syntaxe de UPPAAL TIGA est presque similaire à celle de UPPAAL. Toutefois, UPPAAL TIGA distingue entre deux types de transitions : i) les transitions contrôlables joués par le contrôleur et ii) les transitions incontrôlables joués par l'environnement. On distingue deux objectifs de contrôle différents :

- **Objectif d'accessibilité** : Cet objectif de contrôle cherche la présence d'une stratégie prouvant qu'un ensemble d'états cibles du système (goal states) est accessible quelles que soient les actions prises par l'environnement.

- **Objectif de sûreté** : Cet objectif de contrôle cherche la présence d'une stratégie prouvant qu'un ensemble donné de mauvais états du système (bad states) n'est jamais accessible quelles que soient les actions prises par l'environnement.

Les objectifs de contrôle sont spécifiés en utilisant la syntaxe suivante : *control* : P où P est une propriété TCTL qui exprime soit la sûreté ($A[] p$), soit la vivacité ($A <> p$ ou $A[p_1 U p_2]$).

UPPAAL TIGA permet aussi de déterminer les stratégies gagnantes satisfaisant la propriété dans une durée de temps bien déterminée en utilisant la syntaxe suivante : *control* $_t*(u, g) : A[p U q]$ où seulement les chemins menant vers un état gagnant dans une durée de $u - g$ sont acceptés. Pour obtenir la meilleure stratégie satisfaisant la propriété considérée, on peut commencer par u qui est le temps minimum nécessaire pour atteindre l'état gagnant et $g = 0$. Ensuite, on augmente g jusqu'à ce qu'on atteigne un g' donnant une stratégie gagnante et $g' + 1$ ne le fait pas.

Le noyau de vérification de UPPAAL TIGA est le programme "verifytga". Cette commande permet de générer une stratégie gagnante pour chaque condition de gain donnée, lorsqu'elle est utilisée avec l'option $-t_0$. Une contre-stratégie pour l'environnement est donnée si aucune stratégie n'existe. D'autres options supplémentaires sont aussi disponibles, par exemple : $-c_0$ et $-c_1$ pour contrôler la compacité de la stratégie donnée et $-w_1$ pour générer toutes les stratégies gagnantes possibles satisfaisant une propriété considérée.

1.6.4 SMC UPPAAL

SMC UPPAAL (voir Bulychev *et al.*, 2012) est une nouvelle extension de UPPAAL qui met en oeuvre un model checking statistique. Cet outil a été largement appliqué dans plusieurs domaines de la recherche à savoir le génie logiciel, les systèmes biologiques, les protocoles de communication etc.

La grande utilisation de cet outil est due à plusieurs raisons à savoir : i) il est très simple à mettre en oeuvre, comprendre et utiliser, ii) il ne nécessite qu'un modèle opérationnel du système qui peut être simulé et vérifié avec peu d'efforts de modélisation et de spécification supplémentaires, iii) l'utilisation de la statistique permet de simuler des problèmes indécidables en appliquant des approximations.

Outre sa capacité à gérer les systèmes temporisés, SMC UPPAAL offre la possibilité de comparer deux probabilités sans les calculer. Par contre, les autres modèles de vérification statistiques classiques (voir Younes, 2005; Sen *et al.*, 2005; Katoen *et al.*, 2011) ne sont pas capables de gérer les systèmes temporisés. UPPAAL TIGA est muni d'une large gamme de fonctionnalités pour visualiser les résultats sous la forme de distributions de probabilité,

évolution du nombre d'exécutions en temps borné, calcul d'estimations, etc. Ces résultats sont calculés en utilisant les techniques de simulation Monte Carlo (voir Robert et Casella, 2011) et les tests séquentiels des hypothèses (voir Young et Young, 1998).

Le SMC a trois différences majeures par rapport à UPPAAL classique : i) l'interface utilisateur permet de spécifier des distributions de probabilité, ii) les propriétés à vérifier sont exprimées en PWCTL et iii) le moteur offre un modèle de vérification statistique capable de supporter quatre types de propriétés. Intuitivement, l'exploration du modèle par SMC UPPAAL est limitée par une contrainte d'horloge, une borne sur le temps du modèle ou bien le nombre de transitions discrètes ($\#$). SMC UPPAAL introduit un nouveau opérateur de probabilité noté « **Pr** ». Les quatre types de propriétés supportées par UPPAAL TIGA sont :

- **Propriété quantitative (estimation de la probabilité)** : Cette propriété permet d'estimer la probabilité qu'un événement soit vrai. Pour ce faire, le SMC utilise un algorithme d'approximation qui détermine le nombre d'exécutions nécessaires en fonction de la propriété envisagée. Ensuite, il calcule une estimation de la probabilité p sur cet ensemble d'exécutions, avec un taux de confiance $1 - \alpha$, dans l'intervalle $[p - \varepsilon, p + \varepsilon]$ où ε est un paramètre d'approximation et α est un paramètre de confiance. Particulièrement, cet algorithme calcule le nombre d'exécutions qui satisfont la propriété et divise par le nombre total d'exécutions. L'exemple suivant présente une propriété quantitative permettant de calculer la probabilité qu'un processus tombe en panne :

$$Pr[\# \leq 2000](\langle \rangle \text{proc.panne})$$

- **Propriété qualitative (test d'hypothèse)** : Cette propriété vérifie si la probabilité d'une propriété est inférieure ou supérieure à un seuil bien déterminé. L'exemple suivant présente une propriété qualitative permettant de comparer la probabilité qu'un processus tombe en panne à un seuil bien défini (i.g. 0.5) :

$$Pr[\# \leq 2000](\langle \rangle \text{proc.panne}) \leq 0.5$$

- **Comparaison des probabilités** : Cette propriété compare deux probabilités indirectement sans les estimer. L'exemple suivant permet de comparer les probabilités de défaillance de deux processus $proc_1$ et $proc_2$:

$$Pr[\# \leq 2000](\langle \rangle \text{proc}_1.\text{panne}) \leq Pr[\# \leq 2000](\langle \rangle \text{proc}_2.\text{panne})$$

- **Estimation d'une valeur** : Cette propriété permet d'estimer la valeur d'une expres-

sion en exécutant un certain nombre de simulations. L'exemple suivant permet d'estimer le nombre maximal de pannes d'un processus. La taille de l'échantillon de simulation est bornée par la valeur 26000 et l'horloge x du système est bornée par la valeur 5000 :

$$E[x \leq 5000; 26000](max : nb_panne(proc))$$

1.7 Conclusion

Nous avons consacré ce chapitre aux techniques de vérification qui nous intéressent dans le cadre de ce mémoire, à savoir : le model checking et la synthèse de contrôleur. Nous avons présenté le principe du model checking, ses extensions aux aspects temporels et probabilistes, ainsi qu'une revue de littérature des solutions proposées pour contrecarrer le problème d'explosion combinatoire. Parmi les formalismes utilisés par le model checking, nous avons retenu et présenté les automates temporisés, la logique temporelle TCTL ainsi que leur extension à l'aspect probabiliste. Nous avons ensuite introduit la technique de synthèse de contrôleur qui s'appuie sur les automates temporisés de jeu. Finalement, nous avons décrit les outils de vérification par model checking ou synthèse de contrôleur que nous utiliserons pour implémenter nos approches formelles d'aide à la configuration des pare-feux (UPPAAL, et ses deux extensions UPPAAL TIGA et SMC UPPAAL).

CHAPITRE 2

SECURITE DES RESEAUX INFORMATIQUES

2.1 Introduction

La télécommunication et les réseaux sont devenus des moyens indispensables dans toutes les activités entrepreneuriales et dans tous les secteurs de la vie moderne : les banques, les assurances, les hôpitaux, etc. En dépit des efforts des experts en sécurité, les réseaux informatiques sont sujets à des nombreuses attaques des pirates en tous genres. En effet, depuis l'avènement d'Internet, on lit et entend pratiquement au quotidien des nouvelles à propos des lourdes pertes financières que les entreprises peuvent essuyer à cause des défaillances de sécurité dans leurs systèmes d'informations.

Durant les dernières décennies, plusieurs approches, outils et techniques ont été développés pour améliorer la sécurisation des réseaux informatiques, tels que les systèmes de détection d'intrusions (IDS) (voir Biermann *et al.*, 2001) et les pare-feux (firewalls) (voir Cheswick *et al.*, 2003). Les IDS sont le système d'alarme d'un réseau. Ils permettent de détecter toute activité indésirable et fournir les informations nécessaires aux administrateurs du réseau pour prévenir les intrusions et pouvoir suivre l'attaquant au besoin. Les pare-feux, quant à eux, permettent d'inspecter et filtrer le trafic en s'appuyant sur une politique de sécurité donnée. Les pare-feux ont été largement déployés pour sécuriser les réseaux. Cependant, la configuration de tels outils s'avère une très lourde tâche sujette à nombreuses anomalies qui peuvent entraver la mise en œuvre d'une politique de sécurité sur le réseau.

Le reste de ce chapitre est organisé comme suit : La section 2.2 expose quelques concepts de base sur les réseaux informatiques. La section 2.3 présente les systèmes de détection d'intrusion (IDS) ainsi que leurs différents types. Nous y abordons également les différentes approches de détection d'intrusions. La section 2.4 est dédiée à la présentation des pare-feux ainsi que les différents types d'anomalies qui peuvent compromettre leur bon fonctionnement. La section 2.5 explore quelques techniques de détection d'anomalies liées à la mauvaise configuration des pare-feux alors que la section 2.6 passe en revue quelques solutions proposées dans la littérature pour résoudre les problèmes de configuration de pare-feux.

2.2 Concepts de base

Dans cette section, quelques concepts de base liés à la sécurité informatique sont présentés. Nous donnons un aperçu sur l'architecture des réseaux et les menaces informatiques qui leurs

sont connexes. Enfin, nous explorons la notion de politiques de sécurité.

2.2.1 Architecture d'un réseau

Un réseau informatique se compose d'un grand nombre d'équipements de différents types. On distingue trois types d'éléments réseaux (voir Pujolle et Salvatori, 2010) :

- Les équipements terminaux ou systèmes informatiques tels que les ordinateurs, les stations de travail, les serveurs, les périphériques, etc.
- Les équipements d'interconnexion tels que les routeurs et les commutateurs.
- Les supports de communication tels que les câbles, les fibres optiques et les lignes de transmission.

Un réseau peut avoir plusieurs architectures appelées topologies définissant les connexions entre ses différents équipements. Il existe plusieurs topologies réseaux BILLAMI et BENDAHMANE (2014). Dans ce volet, nous citons les topologies les plus utilisées :

- **Topologie anneau** : Chaque hôte reçoit l'information circulant sur l'anneau et vérifie si l'information lui est destinée. Si c'est le cas, elle la recopie. L'inconvénient d'une telle topologie est le fait que tout le réseau tombe en panne en cas de défaillance de l'un des routeurs dans le réseau.
- **Topologie Étoile** : L'ensemble des stations de travail est connecté à un point central qui aiguille l'information vers son destinataire approprié. L'avantage d'une telle topologie est le fait que la panne d'un nœud ne perturbe pas le fonctionnement global du réseau. Cependant, l'équipement central (concentrateur ou commutateur) qui relie tous les nœuds constitue un point unique de défaillance. Cette technologie est souvent utilisée pour les réseaux téléphoniques.
- **Topologie maillée** : Cette topologie consiste à connecter chaque station à toutes les autres en utilisant des liaisons point à point. En effet, dans un réseau à N hôtes, chaque hôte peut avoir de 1 à $N - 1$ connexions point à point vers plusieurs autres hôtes. Par conséquent, le nombre de liaisons nécessaires est $N * (N - 1)$. Ce nombre croît considérablement avec la taille du réseau.

Les équipements terminaux communiquent entre eux moyennant les adresses IP (Internet protocol). Lors d'une communication entre deux machines, le protocole IP découpe chaque message transmis en paquets indépendants. Ces paquets sont transmis séparément, ce qui permet aux utilisateurs réseau de partager les ressources réseau via Internet en utilisant la

commutation des paquets. L'en-tête d'un paquet IP comprend les informations utiles pour la reconstitution du message et pour l'inspection et le filtrage des paquets par les pare-feux. Ainsi, pour bien comprendre le principe de fonctionnement d'un pare-feu, il est important d'être bien familiarisé avec la structure d'un paquet IP.

La Figure 2.1 présente l'entête d'un paquet IP qui contient des champs de longueur fixe ou variable. Chaque champ joue un rôle bien spécifique. On s'intéresse particulièrement à l'adresse IP d'origine et l'adresse IP de destination qui indiquent respectivement la source et la destination du paquet. On s'intéresse également au champ protocole qui indique le type de protocole encapsulé dans le paquet IP et au champ des données dont les 8 premiers octets contiennent les numéros de ports utilisés pour TCP ou UDP.

VERS	IHL	TOS	TOTAL LENGTH	
IDENTIFICATION			FLAGS	FRAGMENT OFFSET
TTL	PROTOCOL		HEADER CHECKSUM	
Adresse IP d'origine				
Adresse IP de destination				
OPTION				Remplissage
Données				
...				

Figure 2.1 Structure d'un paquet.

2.2.2 Menaces réseau

De nos jours, les réseaux informatiques sont de plus en plus grands, plus complexes et leur architecture est de plus en plus distribuée. Ils sont ainsi potentiellement vulnérables aux attaques extérieures. De ce fait, les efforts des experts réseaux ont été multipliés pour assurer les propriétés fondamentales de la sécurité informatiques à savoir :

- **La confidentialité** : Cette propriété assure que l'information n'est accessible que par les sujets autorisés.
- **L'intégrité** : Cette propriété assure que toute information ne peut pas être modifiée ou détruite de façon non autorisée.

- **La disponibilité** : Cette propriété garantie que les données sont disponibles sans délai ou dégradation, au moment où elles sont sollicitées.

Afin de faire face aux attaques et mettre des dispositions préventives basées sur les propriétés précédentes, il est nécessaire de connaître les principaux types d'attaques. Il existe quatre catégories principales d'attaques (voir Whitman et Mattord, 2011) :

- **Attaque d'accès** : Cette attaque viole la propriété de confidentialité. C'est le fait qu'une personne non autorisée essaye d'accéder à l'information circulant sur le réseau.
- **Attaque de modification** : Cette attaque viole la propriété d'intégrité. C'est le fait qu'une personne non autorisée essaye de modifier l'information circulant sur le réseau.
- **Attaque par saturation (le déni de service)** : Cette attaque viole la propriété de disponibilité. Les attaques par saturation consiste à bloquer l'accès aux sites en rendant indisponible le service.
- **La répudiation** : Cette attaque consiste à pouvoir nier avoir reçu ou émis un message en donnant de fausses informations.

2.2.3 Politique de sécurité

Une politique de sécurité est un document qui établit un ensemble de règles. Ces règles reflètent les actions et les objectifs qu'une entreprise s'est posés en matière de sécurité. Un pare-feu peut incarner une politique de sécurité sous forme de règles de filtrage. Ces règles sont prescrites dans des listes appelées les listes de contrôle d'accès *ACLs*. La mise en œuvre d'une politique de sécurité dépend de la philosophie de sécurité envisagée par l'administrateur. Il existe deux philosophies de sécurité (voir Nouali-Taboudjemat, 1999) :

- **Tout ce qui n'est pas explicitement permis est interdit** : Toutes les actions autorisées doivent être implémentées explicitement au cas par cas. Toute action non implémentée sous forme d'une règle est automatiquement refusée.
- **Tout ce qui n'est pas explicitement interdit est permis** : Toutes les actions interdites (dangereuses) doivent être décrites explicitement au cas par cas. Toute action non implémentée sous forme d'une règle est automatiquement acceptée.

Une politique de sécurité globale est un ensemble de politiques de sécurité indépendantes mais cohérentes. Une politique de sécurité peut être trop permissive. Une telle politique suit généralement la première philosophie en spécifiant une liste assez large d'actions autorisées. Dans ce cas, cette politique risque de présenter une faiblesse de sécurité au niveau du système

qui l'incarne. Une politique de sécurité peut être, au contraire, trop restrictive. Une telle politique suit généralement la deuxième philosophie en spécifiant une liste assez large d'actions interdites. Dans ce cas, elle risque de devenir inapplicable à cause des règles trop strictes.

2.3 Systèmes de détection d'intrusions

Un système de détection d'intrusions (Intrusion Detection System, IDS)(voir Biermann *et al.*, 2001) est un périphérique destiné à analyser l'activité du réseau afin de détecter toute activité malveillante ou suspecte dans les plus brefs délais. Particulièrement, les IDS ont pour objectif d'analyser le système ou le trafic réseau contre des patrons d'attaques enregistrés dans leurs journaux internes. Ces patrons d'attaques sont utilisés pour identifier d'éventuelles violations et de déclencher une alarme lorsqu'une violation est détectée.

Vu l'utilité des IDS sur le plan pratique, des études poussées ont été élaborées pour améliorer leur performance, conduisant ainsi à différents types d'IDS (voir Mé et Michel, 2001) dont chacun est applicable dans un contexte d'application approprié. Parmi les différents types d'IDS, on cite les HIDS (Host based Intrusion Détection Systems) qui décident en se basant sur des informations trouvées dans des machines hôtes et les NIDS (Network-Based Intrusion Détection Systems) qui décident en se basant sur des informations qui circulent dans le réseau. D'autres IDS plus avancés permettent de prévenir les intrusions en reconfigurant dynamiquement les listes de contrôle d'accès (ACL) des routeurs et les politiques de sécurité des pare-feux afin d'exclure les paquets suspects. Cette nouvelle génération d'IDS, qui combine les fonctionnalités des IDS et des pare-feu, est appelée "Système de prévention d'Itrusions" (Intrusion Prevention Systems, IPS).

Dans ce qui suit, nous présentons, en premier lieu, les deux grandes approches d'IDS les plus utilisées : l'approche par scénarios et l'approche comportementale. Ensuite, nous énumérons les différents types d'IDS.

2.3.1 Approches de détection d'intrusions

La détection des intrusions par les IDS se base généralement sur deux approches différentes (voir Mechri, 2007) :

- **Approche par scénarios** : Cette approche utilise une base de données de signatures documentant toutes les signatures des motifs d'attaques connus. L'IDS analyse le trafic réseau, le compare avec les signatures qui se trouvent dans la base de données et alerte les administrateurs de sécurité avant que l'intrusion détectée ne se produise. L'analyse du trafic se fait en utilisant plusieurs mécanismes. Parmi lesquels, on peut citer l'analyse de transition d'états, réseaux de neurones, reconnaissance de forme, etc. Cette

approche est simple et facile à implémenter. Cependant, elle ne permet de détecter que les attaques dont les signatures se trouvent dans la base de connaissance. Par conséquent, Cette dernière doit être mise à jour régulièrement pour augmenter l'efficacité de l'approche.

- **Approche basée sur le comportement** : Cette approche consiste à analyser le comportement actuel du système contre son profil habituel. Toute déviation ou action inhabituelle par rapport à ce profil constitue un signe d'une activité malveillante. Ainsi, une alerte est déclenchée afin de signaler la présence d'un nouveau comportement suspect. L'avantage d'une telle approche est qu'elle permet la détection d'un grand nombre d'intrusions, ainsi que la détection de nouveaux types d'attaques. Cependant, cette approche peut engendrer beaucoup de fausses alertes car elle est très sensible à toute déviation par rapport au comportement habituel du système. Les modèles de détection d'intrusions par approche comportementale utilisent généralement des modèles statistiques, tels que le modèle statistique développé par Denning dans (voir Denning, 1987).

2.3.2 Types et emplacement des IDS

Plusieurs types de systèmes de détection d'intrusions ont été développés depuis les années 80. Chacun a ses propres fonctions, caractéristiques et champs d'application. L'emplacement d'un IDS dans un réseau dépend fortement du type de protection qu'il offre. Ainsi, on peut distinguer deux grands types d'IDS à savoir :

- **NIDS (Network-Based Intrusion Detection Systems)** (voir Dagorn, 2006) : Ce sont des IDS réseau permettant d'analyser et d'interpréter le trafic de tout un réseau afin de repérer des signatures d'attaques déjà connues à différents endroits sur le réseau ou des anomalies dans les entêtes des paquets IP. Toutefois, un NIDS doit toujours disposer d'une bande passante suffisante pour l'analyse continue de l'ensemble de paquets, et il doit être aussi bien positionné pour garantir son efficacité. L'emplacement d'un IDS réseau avant un pare-feu permet de détecter des paquets malveillants conçus pour outrepasser la politique de sécurité du pare-feu. Ceci permet de contourner les attaques dans les plus brefs délais et d'empêcher tout dommage engendré. Un IDS réseau peut aussi être placé après un pare-feu pour protéger un réseau privé donné en signalant uniquement les attaques qui tentent d'y entrer. L'emplacement d'un NIDS après un pare-feu permet, par exemple, d'empêcher des attaques portées par un client d'un réseau privé virtuel VPN (Virtual Private Network) contre un autre réseau VPN protégé par un NIDS. Il existe plusieurs exemples de NIDS à savoir : NetRanger(voir Gleichauf et Teal, 1996), NFR (voir Ranum, 2001), SNORT(voir Green et Roesch,

2003), etc.

- **HIDS (Host based Intrusion Détection Systems)** (voir Dagorn, 2006) : Ce sont des IDS permettant d'examiner le fonctionnement d'une machine donnée et signaler tout comportement malveillant pouvant compromettre sa sécurité. Ces IDS peuvent utiliser plusieurs sources pour fournir des informations sur l'activité d'une machine (i.g. les logs, les traces d'audit du système d'exploitation). Les HIDS sont généralement placés sur les serveurs. Ils sont considérés plus complets et plus efficaces que les NIDS puisqu'ils implémentent un système de protection pour chaque hôte du réseau. En plus, un HIDS permet de détecter des attaques sur un trafic chiffré ce qui n'est pas possible avec un NIDS. Cependant, les HIDS sont très consommateurs en termes de ressources et sont plus vulnérables aux attaques par dénié de service. Un exemple de HIDS est l'outil Trip-wire (voir Kim et Spafford, 1994).

2.4 Pare feu (firewall)

Les pare-feux (voir Cheswick *et al.*, 2003) sont des mécanismes de sécurité efficaces pour protéger un réseau interne contre toute attaque extérieure. En effet, un pare feu est un dispositif de sécurité conçu pour examiner constamment le trafic entrant et sortant sur le réseau. Il joue le rôle d'une barrière de sécurité qui bloque ou laisse passer les paquets en fonction des règles de sécurité établies. De plus, un pare-feu permet de journaliser le trafic réseau en enregistrant les tentatives de connexions afin de pouvoir mieux l'analyser. La mise en œuvre d'un pare-feu nécessite un certain nombre de choix faits par l'administrateur réseau : type du pare-feu, l'emplacement du pare-feu, la politique de sécurité, et le coût financier du pare-feu. Les pare-feux sont configurés en utilisant des listes de contrôle d'accès ACLs qui stipulent l'ensemble des règles de filtrage transcrivant le fonctionnement souhaité par l'utilisateur. Ces règles peuvent accepter ou rejeter les paquets selon les informations encapsulées dans les entêtes des paquets IP. Par exemple, la règle *"access – list 101 permit TCP host 10.1.1.2 host 172.16.1.1 any 23"* accepte tous les paquets du protocole TCP dont l'adresse IP source est 10.1.1.2 et l'adresse IP destination est 172.16.1.1 et utilisant l'importe quel port source et le port destination 23.

2.4.1 Types de pare-feux

Essentiellement, il existe trois catégories de pare-feux (voir Whitman et Mattord, 2011) :

- **Pare-feu sans état (stateless firewall)** : Ce pare-feu applique un mode de filtrage statique. Il permet d'inspecter les paquets qu'il reçoit pour valider les règles de la

politique de sécurité. Ces règles sont basées sur l'en-tête du paquet IP (l'adresse IP de source et de destination, le protocole, le port de source et de destination pour les protocoles TCP et UDP). Il traite chaque paquet indépendamment des autres et décide, en fonction des règles stipulées par la politique de sécurité, de bloquer ou laisser passer le paquet. Cependant, ce type de pare-feu est incapable de détecter plusieurs attaques potentielles telles que l'usurpation de l'adresse IP (IP Spoofing), et certaines attaques par déni de service (DoS) qui nécessitent une solution dynamique en se basant sur l'analyse du trafic sur le réseau.

- **Pare-feu à état (stateful firewall) :** Ce pare-feu applique un mode de filtrage dynamique. Il a été conçu pour remédier aux limites des pare-feux sans état. En effet, il inclut un mécanisme de reporting permettant d'établir des rapports sur le trafic et conserver les tentatives de connexions. Cette fonction permet de renforcer la capacité d'un pare-feu à détecter les attaques. Ce pare-feu opère au niveau de la couche réseau. Par conséquent, il est incapable d'œuvrer sur les failles logicielles au niveau de la couche application.
- **Pare-feu applicatif :** Ce type de pare-feu incarne une politique de sécurité beaucoup plus stricte qu'avec un pare-feu à filtrage de paquets. Il opère au niveau de la couche application. Ce pare-feu empêche tout trafic qui ne respecte pas les politiques de sécurité spécifiées. Par exemple le pare-feu d'application de proxy permet d'accepter seulement les applications dont le proxy est configuré conformément à la politique de sécurité.

2.4.2 Emplacement d'un pare-feu

L'emplacement d'un pare-feu (voir Fall, 2010) est l'une des tâches importantes d'un administrateur réseau. Ce choix dépend de la fonction du pare-feu. Ce dernier peut être placé à la frontière de sorte qu'il protège le réseau local de toutes les connexions venant d'Internet. La situation correspondante est présentée par la Figure 2.2.

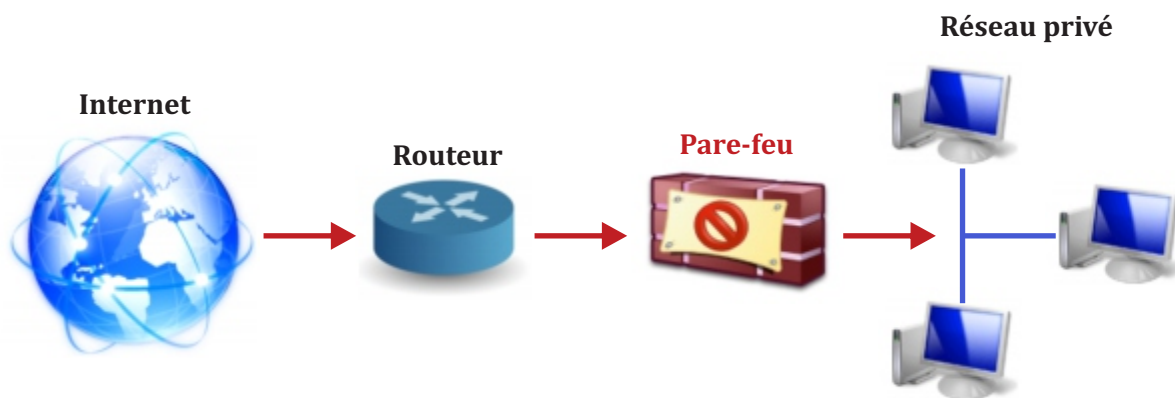


Figure 2.2 Emplacement d'un pare-feu à la frontière.

Un pare-feu peut aussi être placé à l'intérieur d'un réseau pour le diviser en plusieurs sous-réseaux et contrôler les différents accès entre eux. La disposition correspondante est représentée par la Figure 2.3.

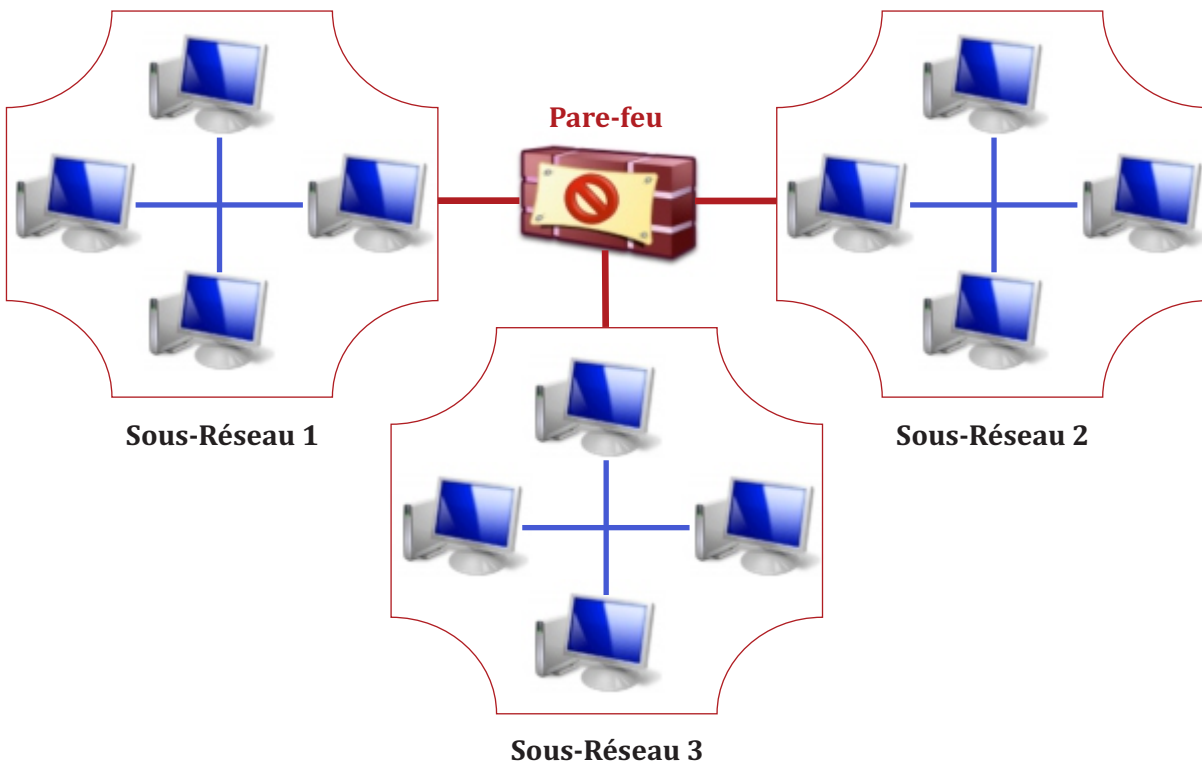


Figure 2.3 Emplacement d'un pare-feu au centre d'un réseau.

2.4.3 Fonctions d'un pare feu

Les fonctions du pare-feu les plus répondues sont (voir Tom, 2005) :

- **Blocage de trafic entrant en fonction de l'origine et de la destination** : Il s'agit de contrôler le trafic entrant d'un réseau et empêcher certains nœuds extérieurs de se connecter à un réseau local.
- **Blocage de trafic sortant en fonction de l'origine et de la destination** : Il s'agit de contrôler le trafic sortant d'un réseau en direction d'internet, et notamment éviter que les utilisateurs accèdent à certains sites inappropriés.
- **Blocage de trafic en fonction du contenu** : Un pare-feu peut inspecter le contenu du paquet IP en utilisant par exemple un scanner de virus intégré. Il peut aussi intégrer un filtre pour empêcher les emails indésirables.
- **Établir des rapports sur les trafics et l'activité du pare-feu** : Un pare-feu doit incorporer un mécanisme de reporting. Ce mécanisme permet d'établir des rapports sur son activité et les archiver dans un journal pour pouvoir l'examiner ultérieurement.

2.4.4 Anomalies des pare-feux

La configuration d'un pare-feu est une tâche fondamentale pour assurer la sécurité des infrastructures réseau. Cependant, cette tâche est complexe et sujette à plusieurs anomalies. Dans cette section, nous étudions les différents types d'anomalies de configuration de pare-feux classifiés comme suit (voir Yuan *et al.*, 2006) :

- **Violation de la politique de sécurité** : La violation de la politique de sécurité consiste à autoriser une action interdite ou rejeter une action permise contrairement à ce qui a été défini sur les listes de contrôle d'accès. Cette violation peut mettre en cause la propriété de confidentialité (accès non autorisé) et la propriété de disponibilité (blocage non désiré).
- **Incohérences intra-pare-feu** : Les incohérences intra-pare-feu sont des révélateurs d'une mauvaise configuration. La détection de ces incohérences est basée uniquement sur les fichiers de configuration d'un pare-feu. On distingue 3 types d'incohérences :
 - **Ombfrage (shadowing)** : Il s'agit de masquer l'effet d'une règle par une autre qui la précède. Par exemple :

R1 : accept udp any 192.168.1.0/24

R2 : deny udp 172.16.1.0/24 192.168.1.0/24

La règle R1 accepte tous les paquets supposés être refusés par la règle R2. Par conséquent, la règle R2 ne sera plus exécutée. On dit que R2 est ombragée par R1.

- **Généralisation** : Il s'agit de définir une règle comme un sous ensemble d'une autre ayant une décision différente. Par exemple :

R2 : deny udp 172.16.1.0/24 192.168.1.0/24

R3 : accept udp 172.16.1.0/24 any

La règle R2 exclut un sous ensemble de paquets supposés être acceptés par la règle R3. On dit que R3 est une généralisation de la règle R2.

- **Corrélation** : On parle d'une corrélation lorsque une règle se croise avec des règles précédentes mais elle définit une action différente. Par exemple :

R1 accept udp any 192.168.1.0/24

R4 deny udp 10.1.1.0/24 192.168.0.0/16

Les règles R1 et R4 se croisent en udp 10.1.1.0/24 192.168.0.0/24. C'est-à-dire, le paquet udp 10.1.1.0/24 192.168.0.0/24 appartient au domaine de décision des deux règles R1 et R4. Cependant, ces deux règles ont des actions différentes. On dit que R1 et R2 sont corollaires.

- **Incohérence inter-pare-feux** : Tous les pare-feux contrôlant la communication sur un réseau sont indépendamment configurés. Cependant la sécurité de bout en bout d'un réseau (end-to-end security behavior) dépend forcément de la configuration de la politique de sécurité globale de tous les pare-feux déployés sur le réseau. Par exemple, soient les deux pare-feux P1 et P2. P1 est en amont du flux et P2 en aval du flux.

P1 :

1. deny tcp any 10.0.0.0/8
2. accept tcp any any
3. deny udp any 192.168.0.0/16

P2 :

1. accept tcp any any
2. accept udp 172.16.0.0/16 192.168.0.0/16

La règle 2 du pare-feu P2 accepte tous les paquets dont l'adresse IP source est 172.16.0.0/16 et l'adresse IP destination est 192.168.0.0/16. Ces paquets sont déjà refusés par la règle 3 du pare-feu P1 en amont du flux. D'où, il y aura un problème de connectivité (c'est-à-dire les paquets en question ne peuvent pas atteindre leur destination).

- **Incohérences de croisement de chemin (Cross-path inconsistencies)** : On parle de ce type d'incohérences lorsque l'un des paquets en transit sur le réseau est rejeté en empruntant un chemin et accepté en empruntant un autre. Par conséquent, un paquet ayant l'accès au réseau peut être rejeté en empruntant un autre chemin à cause d'un changement de routage et vis versa. Ce type d'incohérences est un cas particulier de l'incohérence inter-pare-feux.
- **Inefficacité** : Un pare-feu est dit efficace lorsque sa configuration assure les intentions de l'administrateur en mettant en œuvre un nombre minimum de règles. l'inefficacité d'une configuration d'un pare-feu se résume en deux aspects : la redondance et la verbosité.
 - **La redondance** : La redondance est la présence des règles pouvant être supprimées sans modifier la décision du pare-feu pour tous les paquets. La redondance peut réduire le nombre total des règles et par conséquent la consommation de la mémoire. Par exemple :

R5 accept tcp 10.0.0.0/8 any

R6 accept tcp 10.2.1.0/24 any

La règle R5 couvre tous les paquets qui correspondent à la règle R6 en exerçant la même action. On dit que R6 est une règle redondante.

- **La verbosité** : On parle de verbosité lorsqu'un ensemble de règles peut être réduit en un ensemble comprenant un nombre inférieur de règles. Par exemple, Les règles R7, R8, R9 et R10 peuvent être remplacées par la règle R11.

R7 deny udp 10.1.1.0/26 any

R8 deny udp 10.1.1.64/26 any

R9 deny udp 10.1.1.128/26 any

R10 deny udp 10.1.1.192/26 any

R11 deny udp 10.1.1.0/24 any

2.5 Techniques de détection d'anomalies

La détection des anomalies de configuration des pare-feux fait l'objet de nombreux travaux de recherche. Dans cette section, nous présentons un survol de ces travaux qui sont répertoriés principalement en deux grandes classes : i) les techniques de détection des anomalies intra-pare-feu qui permettent de découvrir les erreurs de configuration de la politique de sécurité au sein d'un pare-feu dans un contexte centralisé et ii) les techniques de détection des anomalies

inter-pare-feux qui permettent de détecter les incohérences entre les différentes règles de filtrage d'un ensemble de pare-feux déployés dans un contexte distribué.

2.5.1 Techniques de détection des anomalies intra-pare-feux :

Al-Shaer et al. ont proposé dans (voir Al-Shaer et Hamed, 2004) l'un des premiers outils dédiés à l'analyse des règles de filtrage au sein d'un pare-feu. Ils sont basés sur la notion des diagrammes de décision binaires (BDD) afin de modéliser les règles d'un pare-feu. La politique de sécurité d'un pare-feu est alors représentée par un diagramme appelé "arbre de politique". Comme montre la Figure 2.4, chaque nœud de cet arbre est un champ de filtrage et les branches partant de chaque nœud sont étiquetées par les valeurs possibles associées à chaque champ. Le parcours de l'arbre en profondeur permet une lecture rapide et claire des règles de filtrage de la politique du pare-feu et facilite la détection d'éventuelles anomalies (redondance, généralisation, corrélation, etc.).

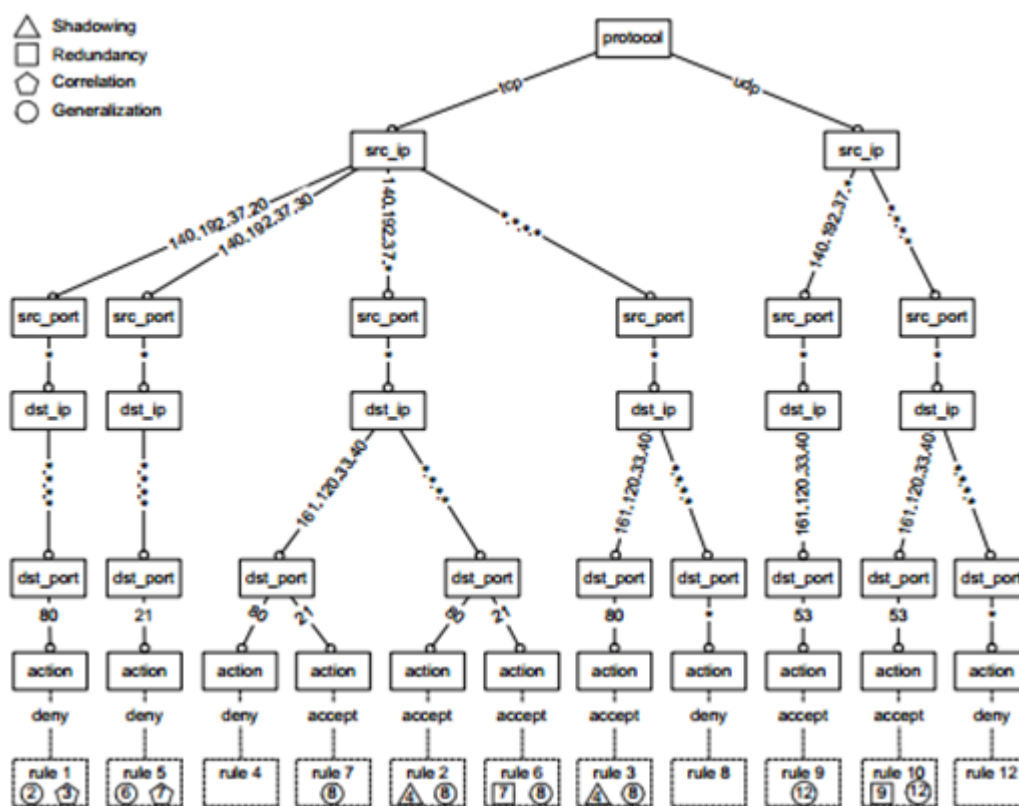


Figure 2.4 Arbre de politique de sécurité d'un pare-feu.

En se basant sur le même principe, Yuan et al. ont proposé dans (voir Yuan *et al.*, 2006) un outil de modélisation et d'analyse des pare-feux appelé FIREMAN. Cet outil est capable

d'analyser automatiquement et d'une façon statique les règles d'un pare-feu en utilisant un algorithme. Cet algorithme permet de parcourir la liste de contrôle d'accès du pare-feu règle par règle et s'assurer que chaque nouvelle règle concerne des paquets non encore couverts. Cette approche est basée sur le model checking symbolique et utilise les diagrammes de décision binaires (BDD) pour représenter et traiter les règles de pare-feu. L'efficacité de l'approche a été validée par une étude expérimentale. En effet, elle permet de détecter des anomalies dans un pare-feu à 800 règles en moins de 3 secondes. Cet outil permet également de détecter des anomalies de configuration inter-pare-feux.

Dans (voir Kotenko et Polubelova, 2011), Kotenko et al. ont proposé un algorithme pour la vérification de la configuration des pare-feux qui permet aussi de détecter des incohérences intra-pare-feux. Cette approche a été mise en œuvre en utilisant le model checker SPIN.

Khorchani et al. dans (voir Khorchani *et al.*, 2012) ont suggéré une logique permettant de spécifier des modèles d'anomalies de configuration. Cette logique est appelée logique de visibilité (visibility logic, VL). Pour vérifier la configuration des pare-feux, les auteurs ont implémenté un modèle de vérification basé sur la logique VL permettant de détecter des modèles d'anomalies intra-pare-feux. Ils ont prouvé l'efficacité de leur approche en élaborant une étude expérimentale. L'avantage de cette approche réside dans le fait qu'elle offre à l'utilisateur la possibilité de spécifier le modèle d'anomalie qu'il cherche à détecter.

Dans (voir Gawanmeh et Tahar, 2011), Gawanmeh et al. ont présenté un modèle formel basé sur la méthode de restriction de domaine afin d'analyser les règles intra-pare-feux. L'approche est implémentée en utilisant la technique Event-B (voir Abrial, 2010) qui est une méthode formelle pour la modélisation des opérations contrôlées par des gardes. L'approche proposée permet de vérifier formellement la cohérence des règles de configuration au sein d'un pare-feu. Ce travail offre la possibilité de caractériser la configuration des pare-feux à différents niveaux d'abstraction.

La contribution de Capretta et al. (voir Capretta *et al.*, 2007) a été proposée, également, pour résoudre le problème des conflits intra-pare-feux. Elle consiste à développer un programme de détection de conflit et de prouver son exactitude en utilisant l'assistant de preuve Coq. L'avantage de cette approche est qu'elle permet de détecter les anomalies de configuration de pare-feux ayant un nombre très large de règles.

Beaucoup d'autres travaux ont proposé des techniques de détection d'anomalies intra-pare-feux. Par exemple, la contribution (voir Eronen et Zitting, 2001) présente un système expert permettant la vérification de la fonctionnalité des règles de filtrage par l'exécution des requêtes. Dans (voir Mayer *et al.*, 2000), les auteurs ont développé un outil d'analyse de pare-feux permettant d'effectuer des requêtes appropriées sur un ensemble de règles de filtrage et découvrir d'éventuelles anomalies dans la politique avant ou après qu'elle soit déployée.

Cependant, l'utilisateur de cet outil nécessite une haute expertise pour pouvoir écrire les requêtes appropriées.

L'outil Margrave (voir Nelson *et al.*, 2010), a été aussi une contribution importante dans le domaine. Cet outil permet à un utilisateur d'écrire des requêtes dans un langage du premier ordre pour détecter des anomalies intra-pare-feux. Il permet d'identifier le comportement de pare-feu par rapport à des règles spécifiques, et le vérifier par rapport aux objectifs de sécurité.

Par ailleurs, Abbes et al. (voir Abbes *et al.*, 2008) ont proposé une méthode pour détecter la redondance des règles de filtrage au sein d'un pare-feu, ils ont classé les règles en fonction de leurs champs de contrôle afin de déterminer les règles redondantes.

Ben Youssef et al. (voir Ben Youssef *et al.*, 2009) ont proposé également une méthode pour vérifier si un pare-feu réagit correctement par rapport à une politique de sécurité spécifiée dans un langage déclaratif de haut niveau.

2.5.2 Techniques de détection des anomalies inter-pare-feux

Contrairement à la détection des anomalies intra-pare-feux, assez peu de travaux ont abordé le problème de conflits inter-pare-feux. La majorité de ces travaux ont utilisé la technique de model checking.

El-Atawy et al. (voir El-Atawy et Samak, 2012) ont modélisé un réseau sous forme d'un ensemble de politiques de sécurité qui régissent chaque élément du réseau (i.g. les routeur, les pare-feux, etc.). Le modèle proposé permet de vérifier les propriétés envisagées par rapport à la configuration courante en utilisant la technique de model checking basée sur la logique temporelle CTL. La vérification des propriétés se fait par rapport à chaque paquet individuellement en raisonnant sur le chemin emprunté par le paquet en question (de la source à la destination).

Un travail similaire a été présenté par Jeffrey et al. (voir Jeffrey et Samak, 2009). Ce travail a proposé un modèle pour la configuration des règles de pare-feux dans un contexte distribué. Ils ont concentré leur analyse formelle sur deux propriétés : i) l'atteignabilité (pour chaque règle, il existe au moins un paquet qui la viole, donc un pare-feu qui contient des règles inatteignables est pare-feu mal configuré, et ii) acyclicité (aucun paquet ne repasse par un pare-feu qu'il a déjà visité). Ils ont montré expérimentalement que l'utilisation des solveurs de satisfiabilité SAT dans l'analyse des politiques de sécurité des pare-feux est plus efficace que les BDD notamment lorsque la taille du problème augmente. En effet, le model checking borné basé sur les SAT est considéré comme une bonne alternative à la vérification de modèle basée sur la BDD. Il est plus approprié pour trouver des bogues dans les systèmes infinis. Son idée de base consiste à rechercher un contre-exemple sur les traces d'états dont

la longueur est bornée par un certain entier k . Si aucun contre-exemple n'est trouvé alors k est augmentée jusqu'à ce qu'un contre-exemple soit trouvé ou la limite de la mémoire dédiée pour la vérification soit atteinte.

Matousek et al. (voir Matousek *et al.*, 2008), ont élaboré une analyse dynamique des réseaux avec des différentes topologies permettant de vérifier des propriétés de sécurité. Ils ont utilisé une logique du premier ordre pour représenter les éléments de filtrage de paquets et l'ensemble de règles de configuration dans le réseau. Le model proposé a fourni une analyse dynamique et générale des propriétés. Cependant, le travail manque une étude empirique évaluant l'efficacité de l'approche.

Aussi, Chaure a proposé dans (voir Chaure, 2010) un outil basé sur deux approches permettant de détecter respectivement des anomalies de configuration inter-pare-feux et intra-pare-feux. Cependant, il a fourni une étude empirique prouvant uniquement l'efficacité de son approche de détection d'anomalies intra-pare-feux.

Hallé et al. (voir Halle *et al.*, 2013) ont présenté également une approche permettant de détecter des anomalies de configuration inter-pare-feux. Ils ont utilisé le model checking LTL pour spécifier et vérifier leur modèle.

2.5.3 Discussion

Les techniques de détection d'anomalies de configuration de pare-feux ont été répertoriées selon le type d'anomalies auquel elles s'appliquent à savoir : les techniques de configuration intra-pare-feux et les techniques de configuration inter-pare-feux.

Nous avons remarqué que la majorité des travaux, présentés dans la littérature, ont abordé le problème de la détection des incohérences intra-pare-feux. En fait, ils se sont placés dans un contexte centralisé et ont proposé des approches formelles pour détecter les conflits entre les règles de filtrage au sein d'un pare-feu.

Certes, l'étude des incohérences intra-pare-feux permet d'améliorer la performance d'un réseau de communication. Cependant, dans les grands réseaux, plusieurs pare-feux sont souvent déployés ensemble pour mettre en œuvre une politique de sécurité globale. Le déploiement d'un ensemble de pare-feux, dans un contexte distribué, peut engendrer des incohérences ayant des conséquences graves sur le réseau et ses utilisateurs. Toutefois, dans la littérature, il y a peu de contributions qui ont fourni des évaluations utiles permettant d'aider les administrateurs de réseau à corriger les incohérences inter-pare-feux. La majorité de ces travaux propose des solutions très formelles et théoriques avec un niveau d'abstraction très élevé, et ne permettent pas de combiner des différents aspects (i.g. qualitatif, quantitatif).

Dans ce contexte, notre travail vise à contourner ces problèmes. En fait, nous proposons, dans la première partie de ce mémoire, deux approches formelles complémentaires qui per-

mettent de vérifier la cohérence des pare-feux vis-à-vis d'un objectif de sécurité et d'évaluer la performance du réseau en se basant sur les paramètres de qualité de service (i.e. le délai d'acheminement de paquets de bout en bout, le taux de perte, le délai d'attente). Nous proposons un modèle basé sur des contraintes réelles. L'utilisation du modèle checking UPPAAL a donné à notre solution son aspect pratique. En fait, UPPAAL permet de suivre clairement l'acheminement des paquets entre les différents nœuds du réseau et d'avoir des informations pertinentes sur le type et l'emplacement des incohérences inter-pare-feux à l'aide des contre-exemples.

2.6 Renforcement de la politique de sécurité

Depuis quelques décennies, des nombreuses approches ont été proposées afin de renforcer la politique de sécurité dans les réseaux. Dans la présente section, nous passons en revue quelques unes de ces approches.

2.6.1 Approche algébrique

T.Mechri a proposé dans (voir Mechri, 2007) une approche algébrique basée sur des méthodes formelles pour la configuration automatique et la sécurisation des réseaux informatiques conformément à une politique de sécurité bien déterminée. Cette approche consiste à définir une nouvelle algèbre de processus (Calculus for Monitored Network, CMN) pour modéliser le réseau et une logique propositionnelle (LM) pour spécifier des politiques de sécurité. L'objectif est de définir un opérateur de renforcement noté \otimes permettant de générer, à partir d'un réseau et une politique de sécurité donnée, une autre version sécuritaire du réseau. La Figure 2.5 illustre l'objectif de l'approche.

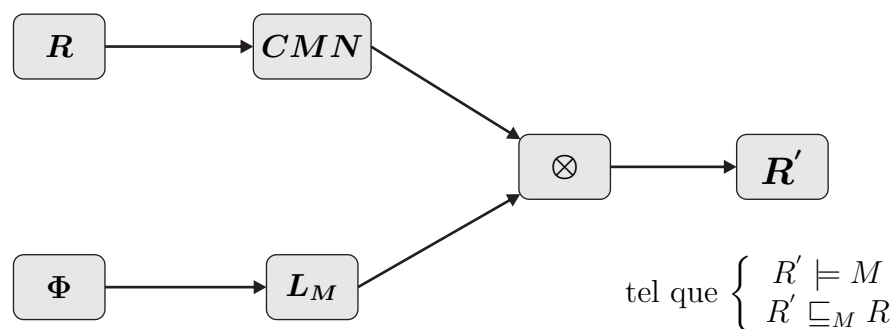


Figure 2.5 Approche algébrique pour la sécurisation des réseaux informatiques.

Une version améliorée de cet opérateur est obtenue par le développement d'une technique de monitoring sélectif permettant de contrôler l'insertion des moniteurs dans les composantes du réseau. Le nouvel opérateur de renforcement noté \otimes_s consiste à générer automatiquement, à partir d'un réseau R , une politique de sécurité M et un ensemble de composantes du réseau S , un nouveau réseau R' respectant M dans lequel les moniteurs sont placés uniquement sur les éléments de S . La Figure 2.6 présente un schéma illustratif du principe de fonctionnement de l'opérateur \otimes_s .

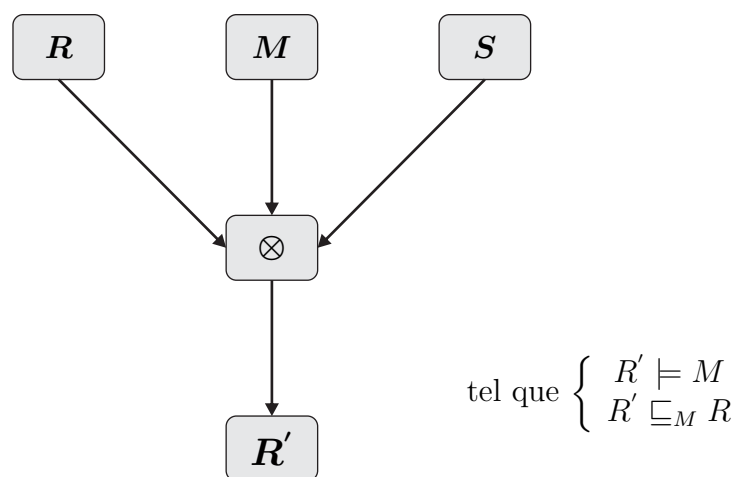


Figure 2.6 Opérateur du monitoring sélectif.

L'approche proposée par Mechri est une approche rigoureuse. Cependant, le fait d'ajouter un moniteur dans toutes les composantes du réseau affecte certainement l'efficacité de l'approche. En plus l'intervention humaine pour choisir les emplacements des moniteurs peut engendrer des problèmes.

Une approche similaire est proposée par M.N.D. Fall dans (voir Fall, 2010). Elle consiste à modéliser un réseau informatique par un graphe étiqueté par des pare-feux contrôlant le trafic entre ses différentes entités. L'auteur a proposé une logique propositionnelle pour spécifier la politique de sécurité. Il a fourni une extension du langage de spécification des pare-feux (firewall language, FL) proposé par Mechri. Le principe de renforcement de la politique de sécurité est semblable à celui présenté par T.Mechri (Figure 2.5). En effet, le but est de définir un opérateur de renforcement permettant de générer une configuration sécuritaire et optimale d'un réseau. La valeur ajoutée de ce travail, par rapport au travail T. Mechri, est l'introduction de la notion du coût relatif à la qualité de service d'un réseau afin de déterminer le coût d'une configuration. Cependant, la définition du coût d'une configuration est basée

uniquement sur le nombre de règles dans un pare-feu. Cette définition est loin d'être pratique et ne faisant pas intervenir concrètement les coûts matériels liés à une configuration (i.g. le coût de câblage, le coût de mise en œuvre, le coût de réparation, etc.).

Les deux contributions présentées, ci-dessus, manquent une étude de cas réel qui projette leurs approches sur le plan pratique.

2.6.2 Autres approches

D'autres approches ont été proposées afin de renforcer la sécurité sur les réseaux informatiques à savoir :

- Firmato (voir Bartal *et al.*, 1999) : Cet outil est basé sur un modèle entité-relation permettant la description des réseaux. Les auteurs ont proposé un langage appelé MDL (Model Definition Language) permettant la spécification des propriétés de sécurité. Le principe de fonctionnement de Firmato repose sur la notion de rôle qui présente la topologie du réseau selon les fonctionnalités des hôtes qui la composent. L'outil permet tout d'abord de définir la spécification de la politique de sécurité en MDL, ensuite, il utilise un analyseur pour en générer un modèle entité-relation. Ce modèle qui définit la topologie du réseau en tant qu'un rôle, sera transformé automatiquement par un compilateur en un ensemble de fichiers de configuration de pare-feux. Cependant, l'algorithme proposé ne s'appuie pas sur des techniques formelles de validation et la spécification de politique de sécurité nécessite une bonne maîtrise du langage MDL. De plus, l'utilisation de la notion de rôle dans la définition de la topologie du réseau implique une explosion du nombre de rôles avec l'augmentation de la taille de topologie.
- Mirage (voir Garcia-Alfaro *et al.*, 2011) : C'est un outil de gestion pour l'analyse et le déploiement des politiques de sécurité sur les composantes de réseaux (i.g. les pare-feux, systèmes de détection d'intrusion, les routeurs VPN, etc.). Cet outil est implémenté en java. Il permet d'analyser et de raffiner les configurations déjà déployées sur les composantes de réseau. Mirage fournit une analyse intra-composants qui permet de détecter et de corriger les incohérences au sein d'un seul composant et une analyse inter-composants, pour détecter et corriger les incohérences entre les configurations des différents composants déployés dans un contexte distribué. Cependant cet outil ne permet pas de gérer la mise à jour des configurations de composants réseau en se basant sur les configurations souhaitées et réelles. De plus il se limite uniquement à l'analyse de pare-feux sans état. Les auteurs dans (voir Garcia-Alfaro *et al.*, 2013) ont proposé une extension à Mirage permettant de traiter les pare-feux à état.

- Diagramme de décision d'un pare-feu (firewall decision diagram, FDD) : C'est une approche proposée par Gouda et al. dans (voir Gouda et Liu, 2004), permettant la conception et l'ordonnancement des listes de contrôle d'accès dans un pare-feu. Cette approche consiste à modéliser la liste de règles d'un pare-feu par un FDD. Ce FDD sera ensuite réduit et simplifié en utilisant un ensemble d'algorithmes dédiés à cet effet. Le FDD réduit est cohérent, complet et consistant. Un exemple de FDD est présenté par la Figure 2.7. Chaque nœud correspond à un champ de contrôle et chaque branche désigne la valeur du champ correspond. Cette approche permet de renforcer la politique de sécurité au sein d'un pare-feu.

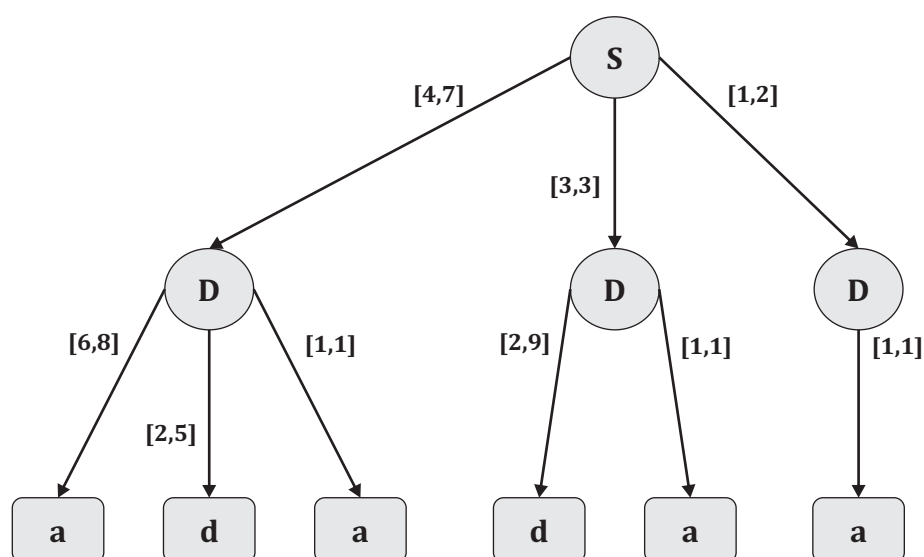


Figure 2.7 Exemple de FDD.

- D'autres solutions ont été explorées. Dans (voir Guttman, 1997), Guttman a proposé une approche appelée le filtrage des postures. Elle consiste à générer des filtres de paquets incarnant une politique de sécurité donnée. L'auteur a introduit un nouveau langage basé sur LISP pour définir une politique de filtrage. Cependant, cette contribution n'assure pas une séparation complète entre la politique de sécurité et la topologie du réseau, ceci rend difficile la réutilisation de la politique de sécurité ainsi définie. En outre, cette approche nécessite une intervention manuelle afin d'introduire des optimisations à la configuration générée. Par ailleurs, elle ne semble pas se prêter facilement à être utilisée avec des pare-feux réels puisque la configuration d'un pare-feu se fait indépendamment des autres.

Dans (voir Liu et Gouda, 2009), Liu et al. ont proposé un langage structuré SFQL (Structured Query Language Firewall). Ce langage ressemble au langage SQL et permet la spécification des requêtes dans les pare-feux. Le but de leur contribution est de donner un théorème permettant de guider le développement des algorithmes dédiés au traitement des requêtes dans les pare-feux.

Toujours, dans le but de renforcer la sécurité sur les pare-feux, Verma et al. (voir Verma et Prakash, 2005) ont proposé un outil appelé FACE qui aide à l'analyse et à la configuration des pare-feux distribués. FACE permet aux administrateurs réseau d'analyser et générer automatiquement des configurations pour tous les pare-feux du réseau en spécifiant la politique de filtrage et le modèle de menace visé.

2.6.3 Discussion

Plusieurs travaux ont été proposés, dans la littérature, afin de renforcer la politique de sécurité dans un réseau de communication. Le but est de trouver une solution permettant de détecter et de corriger les anomalies de configuration qui entravent une mise en œuvre efficace et optimale d'une politique de sécurité. Ci-dessus, nous avons passé en revue quelques unes de ces solutions. Nous avons présenté les techniques qui ont été utilisées par les différents chercheurs dans le domaine ainsi que les avantages et les inconvénients de chaque approche.

À l'instar des travaux existants, nous abordons, dans la deuxième partie de ce mémoire, une nouvelle approche d'aide à la configuration des pare-feux permettant de forcer un réseau à satisfaire un objectif de sécurité global. Dans ce cadre, un modèle abstrait est implémenté en utilisant l'outil UPPAAL TIGA. Cet outil permet d'implémenter un algorithme de vérification à la volée afin de synthétiser les stratégies gagnantes permettant de renforcer la propriété envisagée. Ces stratégies gagnantes présentent l'ensemble de règles définissant les *ACLs* des pare-feux. Ce sont les règles nécessaires pour le contrôle du réseau par rapport à la politique de sécurité.

L'approche, que nous proposons dans ce contexte, n'a pas été proposée auparavant. Les anciens travaux permettent généralement de raffiner les règles d'une configuration déjà déployée sur le réseau. Par contre, notre approche permet de synthétiser les règles nécessaires et suffisantes pour assurer l'objectif de contrôle des pare-feux initialement non configurés déployés sur un réseau de communication.

2.7 Conclusion

Dans ce chapitre, nous avons présenté un bref aperçu sur quelques concepts de base liés à la sécurité informatique. Les différentes topologies d'un réseau informatique (i.g. anneau,

bus, étoile, maillé, etc.) ont été également présentées. Après avoir exposé les menaces informatiques, nous avons introduit la notion de politiques de sécurité dans les réseaux.

Dans un deuxième volet, nous avons passé en revue deux principales techniques utilisées pour la sécurisation de réseaux informatiques : i) les systèmes de détection d'intrusions IDS et ii) les pare-feux. Étant l'objet de notre travail, nous avons exploré les différents types et fonctionnalités des pare-feux ainsi que leurs principales anomalies de configuration.

Un autre volet de ce chapitre a été consacré à la discussion des différentes techniques de détection d'anomalies de configuration ainsi que plusieurs solutions proposées dans la littérature afin de corriger ces anomalies et renforcer automatiquement une politique de sécurité dépourvue de toute incohérence. La discussion de ces différentes techniques nous a permis de repérer les limites de chacune et nous a donné une base fertile pour lancer notre travail de recherche. Dans les deux chapitres qui suivent, nous présentons nos contributions.

CHAPITRE 3

MODELISATION Et VERIFICATION FORMELLES D'UN RESEAU INFORMATIQUE A PARE-FEUX

3.1 Introduction

Dans le chapitre précédent, nous avons présenté les anomalies de configuration des pare-feux distribués qui peuvent entraver la solution de sécurité envisagée au sein d'un réseau informatique. Nous avons également discuté plusieurs travaux dans la littérature qui ont pris en charge cette problématique.

Dans le chapitre présent, nous proposons un modèle formel permettant de vérifier qualitativement la consistance d'un réseau à pare-feux de bout en bout (end-to-end security behavior) en découvrant les erreurs de configuration de la politique de sécurité globale du réseau. Notre approche formelle s'appuie sur la technique de model checking. Elle a été implémentée en utilisant l'outil de modélisation et de vérification UPPAAL.

Dans ce chapitre, nous proposons, également, un deuxième modèle formel permettant de vérifier quantitativement le comportement d'un réseau informatique en se basant sur des paramètres de QoS à savoir, le délai d'acheminement des paquets de bout en bout, le délai d'attente et le taux de perte. Cette deuxième approche est basée sur le model checking statistique et elle a été implémentée en utilisant SMC UPPAAL.

Le reste du chapitre est organisé comme suit : La section 3.2 donne une représentation graphique d'un réseau informatique à pare-feux. La section 3.3 présente le modèle qualitatif qui permet de vérifier la cohérence des pare-feux. La section 3.4 est consacrée à la vérification formelle de notre modèle. Nous y présentons également une étude expérimentale permettant d'étudier la performance de l'approche. La section 3.5 décrit une nouvelle abstraction du modèle permettant d'atténuer le problème d'explosion combinatoire alors que la section 3.6 présente l'extension quantitative de notre approche qualitative.

3.2 Représentation graphique d'un réseau à pare-feux

Un réseau informatique est constitué d'un ensemble de nœuds interconnectés. La communication entre les différents nœuds est contrôlée par des pare-feux. Chaque pare-feu contrôle la réception du nœud dont il est responsable et incarne une politique de sécurité. Une politique de sécurité d'un pare-feu est une liste de règles ordonnées. Ces règles reflètent les objectifs et les mesures prises par un expert de sécurité afin d'assurer la sécurité sur un réseau. La

mise en œuvre d'une politique de sécurité dépend de la philosophie de la sécurité adoptée par l'administrateur du réseau. Nous rappelons qu'il existe deux différentes philosophies :

- Tout ce qui n'est pas explicitement autorisé est interdit : toutes les actions autorisées doivent être mises en œuvre au cas par cas.
- Tout ce qui n'est pas explicitement interdit est autorisé : toutes les actions interdites doivent être décrites au cas par cas.

Les règles de filtrage d'un pare-feu sont définies dans une liste spécifique appelée liste de contrôle d'accès (*ACL*). Chaque règle est définie comme suit :

$$\mathit{rule} : \langle \mathit{Condition} \rangle \Rightarrow \langle \mathit{Decision} \rangle$$

où "*Condition*" est une expression booléenne qui associe à chaque paquet une valeur booléenne (vrai ou faux) en se basant sur un ensemble de paramètres de filtrage. Les paramètres de filtrage les plus couramment utilisés sont : le type de protocole, l'adresse IP source, le port source, adresse IP destination et le port destination. La décision d'une règle appartient à l'ensemble $\{\mathit{accept}, \mathit{deny}\}$ dépendamment de la valeur de vérité de la condition, où "*accept*" et "*deny*" signifient respectivement que le paquet en question est accepté ou refusé par le processus de filtrage.

Un réseau à pare-feux peut être spécifié à l'aide d'un graphe étiqueté. Chaque état, dans le graphe, représente un nœud ou éventuellement une abstraction d'un sous réseau et chaque arête est étiquetée par un pare-feu qui contrôle le trafic entre ses deux extrémités dans une direction spécifique.

Formellement, un réseau représenté par un graphe étiqueté est défini par un tuple $G = (N, F, R, A, L)$, où :

- $N = N_0, N_1, \dots, N_n$ est un ensemble d'identificateurs de nœuds ($n + 1$ est le nombre de nœuds),
- $F = F_0, F_1, \dots, F_n$ est un ensemble d'identificateurs de pare-feux,
- A est une fonction partielle $N \times N \rightarrow F$, qui associe, à tout couple de nœuds connectés, un pare-feu,
- $R = R_0, R_1, \dots, R_k$ est un ensemble d'identificateurs de règles ($k + 1$ est le nombre de règles),
- Soit 2^R un ensemble de tous les sous-ensembles de R . L est une fonction $F \rightarrow 2^R$ qui associe à un pare-feu un sous-ensemble de règles appartenant à 2^R .

À titre illustratif, la Figure 3.1 montre un exemple d'un réseau à pare-feux représenté par graphe étiqueté, où $N = \{N_0, N_1, N_2, N_3, N_4\}$, $F = \{F_0, F_1, F_2, F_3, F_4\}$, A et L sont définies par :

- $A(N_1, N_0) = A(N_2, N_0) = A(N_3, N_0) = F_0$,
- $A(N_0, N_1) = A(N_2, N_1) = F_1$,
- $A(N_0, N_2) = A(N_1, N_2) = F_2$,
- $A(N_0, N_3) = A(N_4, N_3) = F_3$,
- $A(N_3, N_4) = F_4$
- $L(F_0) = R_1, R_2$
- $L(F_1) = R_3, R_4$
- $L(F_2) = R_2$
- $L(F_3) = R_1, R_5$
- $L(F_4) = R_1$

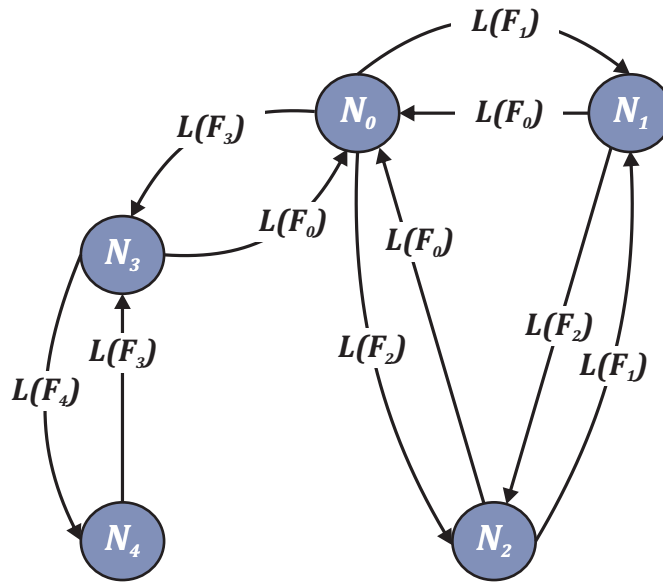


Figure 3.1 Représentation graphique d'un réseau.

Chaque nœud N_j peut recevoir un paquet d'un autre nœud N_i si et seulement s'il existe un arc reliant N_i et N_j . Le pare-feu $F_j = A(N_i, N_j)$ accepte le paquet considéré en se basant sur sa liste de contrôle d'accès $L(F_j)$. La décision d'un pare-feu suit la loi de "first match".

En effet, la liste $L(F_j)$ est explorée en ordre décroissant du haut vers le bas jusqu'à atteindre une règle acceptant le paquet. Dans le cas où il n'existe pas une règle de correspondance, le paquet est rejeté.

En général, les nœuds d'un réseau sont connectés selon différentes topologies (anneau, bus, étoile et maillée). Dans ce travail, nous avons opté pour la topologie maillée puisqu'elle est la plus utilisée dans les grands réseaux de communication tels qu'Internet. Nous considérons que les nœuds d'un réseau sont de deux types :

- Réseau privé ou nœuds privés où seuls les personnes autorisées peuvent accéder au réseau (i.g. le réseau domestique).
- Réseau public ou nœuds de transit où il n'existe presque pas de restrictions sur l'accès au réseau (i.g. les fournisseurs de service Internet comme Bell).

Les nœuds privés peuvent générer et envoyer des paquets vers d'autres nœuds privés via le réseau public. Un réseau maillé est caractérisé par sa taille qui représente le nombre d'hôtes interconnectés et son taux de connectivité défini par la formule suivante :

$$\mathbf{Taux_de_connectivit} = \mathbf{nb_liens/n * (n - 1)}$$

où nb_liens est le nombre de connexions dans le réseau et n est le nombre de nœuds dans le réseau.

3.3 Modélisation du comportement

Notre objectif consiste à élaborer une étude comportementale d'un réseau informatique à pare-feux. Cette étude consiste à modéliser le réseau par n automates étendus, où n est le nombre de nœuds du réseau. Ainsi, chaque nœud est représenté par un automate étendu paramétré par un numéro d'identification unique. Nous avons appliqué la technique de model checking pour vérifier certaines propriétés liées à la cohérence des pare-feux distribués.

Nous rappelons qu'un automate étendu est un système de transitions dont les transitions sont annotées par des sélections, des gardes, des signaux de synchronisation et des blocs d'actions. Un état est défini par les valeurs courantes de toutes les variables du système ainsi que les invariants sur les horloges définis sur cet état. Une transition franchie à partir d'un état, si sa garde est satisfaite. Le franchissement d'une transition implique l'exécution de son bloc d'actions permettant ainsi de changer l'état du système. Deux transitions étiquetées par des signaux de synchronisation $c!$ et $c?$ sur un canal commun c doivent synchroniser.

Dans notre modèle formel, nous avons utilisé deux types d'automates : i) un automate "Noeud_générateur" dédié à la spécification du réseau et la génération des paquets, ii) un

automate "Noeud" permettant de modéliser les nœuds du réseau.

3.3.1 Automate "Nœud_générateur"

L'automate, présenté dans la Figure 3.2, permet de spécifier la topologie du réseau en se basant sur sa taille et son taux de connectivité. La procédure *define_topology()* crée initialement un réseau ayant une topologie en anneau en fonction de la taille du réseau (nombre de nœuds). Ensuite, en fonction du taux de connectivité du réseau défini comme une variable globale, cette procédure calcule et ajoute les connexions afin de concevoir la topologie maillée choisie.

La procédure *define_topology()* permet également de spécifier les règles de configuration des différents pare-feux déployés sur le réseau. Elle définit la fonction partielle *A* qui associe, à toute paire de nœuds connectés, le pare-feu correspondant.

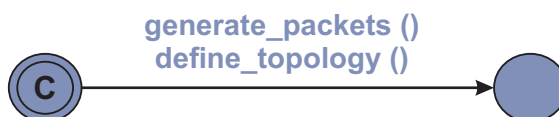


Figure 3.2 Automate générateur.

L'automate "Nœud_générateur" permet aussi de définir une liste exhaustive et figée contenant tous les types de paquets non redondants pouvant être générés en fixant les sources et les destinations dans le réseau à contrôler. Cette liste est générée à l'aide de la fonction *generate_packets()*. La structure d'un paquet est représentée dans le tableau 3.1. Les attributs *ip_src*, *ip_dest*, *port* et *protocol* sont des données statiques relatives aux paquets en transit et les attributs *état*, *NC*, *nœuds_visités* et *nb_v* sont des données dynamiques qui changent en fonction du parcours de paquets dans le réseau.

Tableau 3.1 Structure d'un paquet.

Attributs	Definition
int num_paquet	L'identifiant du paquet.
int ip_src	L'adresse IP source du paquet.
int ip_dest	L'adresse IP destination du paquet.
int protocol	Le protocole (TCP ou UDP).
int port	Le port (80 ou 25).
int etat	L'état courant du paquet (0 à l'état initial, 1 si le paquet est acheminé vers la destination avec succès, -1 si le paquet est rejeté et -2 s'il est bloqué dans un nœud terminal).
int NC	Le nœud courant du paquet.
int noeuds_visites [nb_nodes]	La liste des nœuds visités par le paquet.
int nb_v	Le nombre des nœuds visités.

3.3.2 Automate "Nœud"

L'automate, illustré par la Figure 3.3, modélise les nœuds du réseau. Il est paramétré par un identifiant unique "nid". Ainsi, chaque nœud est représenté par un automate étendu. La communication entre les automates est assurée par les canaux de synchronisation $m[nid]?$ et $m[nid]!$.

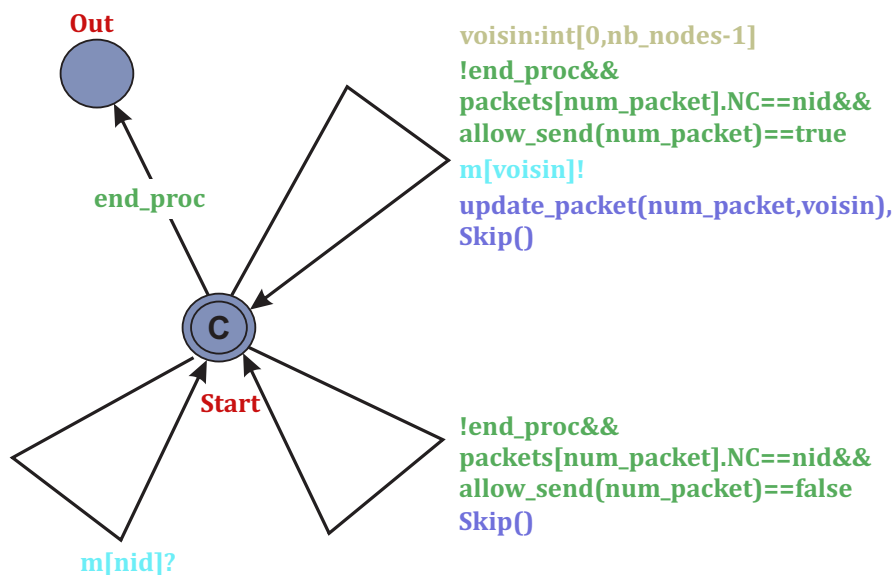


Figure 3.3 Automate Nœud.

Le trafic entre deux extrémités dans une direction spécifique (source-destination) est

contrôlé par des transitions étiquetées par des gardes. Ces gardes sont les règles des pare-feux prescrites dans les *ACLs* et spécifiées par la fonction *L*. La décision prise par un pare-feu (*accept* ou *deny*) concernant un paquet donné est définie par une fonction basée sur des conditions (*if/else*). Notez que, nous pourrions également représenter la décision des pare-feux à l'aide d'autres algorithmes tels que les diagrammes de décision (voir Liu, 2008).

Un paquet est envoyé à partir de son emplacement courant vers un de ses nœuds voisins choisi, aléatoirement, parmi ceux qui existent. L'automate responsable de cet envoi est celui dont le *nid* est égal à l'emplacement courant *NC* du paquet en transit.

Le paquet est envoyé si et seulement si la fonction *allow_send()* retourne "*true*". Cette fonction permet d'envoyer le paquet à sa prochaine destination si et seulement si : i) il n'a déjà pas été rejeté ou bloqué dans un nœud terminal (un nœud qui n'admet aucune transition vers un autre nœud), ii) il n'a pas encore atteint sa destination finale et iii) il y a une connexion dans le réseau entre son emplacement courant et sa prochaine destination.

Si la fonction *allow_send()* est à "*true*", la procédure *update_paquet()* est appelée pour mettre à jour le statut du paquet en fonction des règles du pare-feu qui contrôle le trafic entre la position actuelle et la prochaine destination du paquet.

Si la fonction *allow_send()* est à "*false*", le paquet sera abandonné et la procédure *Skip()* sera appelée pour choisir le prochain paquet à analyser parmi ceux qui i) n'ont pas été encore rejetés, ii) n'ont pas atteint leurs destinations finales et iii) ne sont pas bloqués dans un nœud terminal.

Si le traitement de tous les paquets est fini, *skip()* met *end_proc* à 1 et l'état *out* sera activé.

3.3.3 Exemple concret

La Figure 3.4 présente un exemple d'un réseau composé de nœuds publics (nœuds bleus) et de nœuds privés (nœuds orangés). Les liens entre les nœuds sont bidirectionnels. La communication entre les nœuds est contrôlée par des pare-feux présentés sur la figure par leurs *ACLs*. Les listes de contrôle d'accès *ACLs* contrôlent la réception de chaque nœud. Elles sont utilisées, par exemple, pour restreindre certaines destinations. Ce réseau est composé de sept nœuds. Selon notre approche, il est alors modélisé par sept automates. Généralement, le nombre de paquets générés par l'automate générateur est défini par la formule suivante :

$$nb_paquets = nb_private * nb_port * nb_protocol$$

où *nb_private* est le nombre de nœuds privés, *nb_port* est le nombre de ports et *nb_protocol* est le nombre de protocoles.

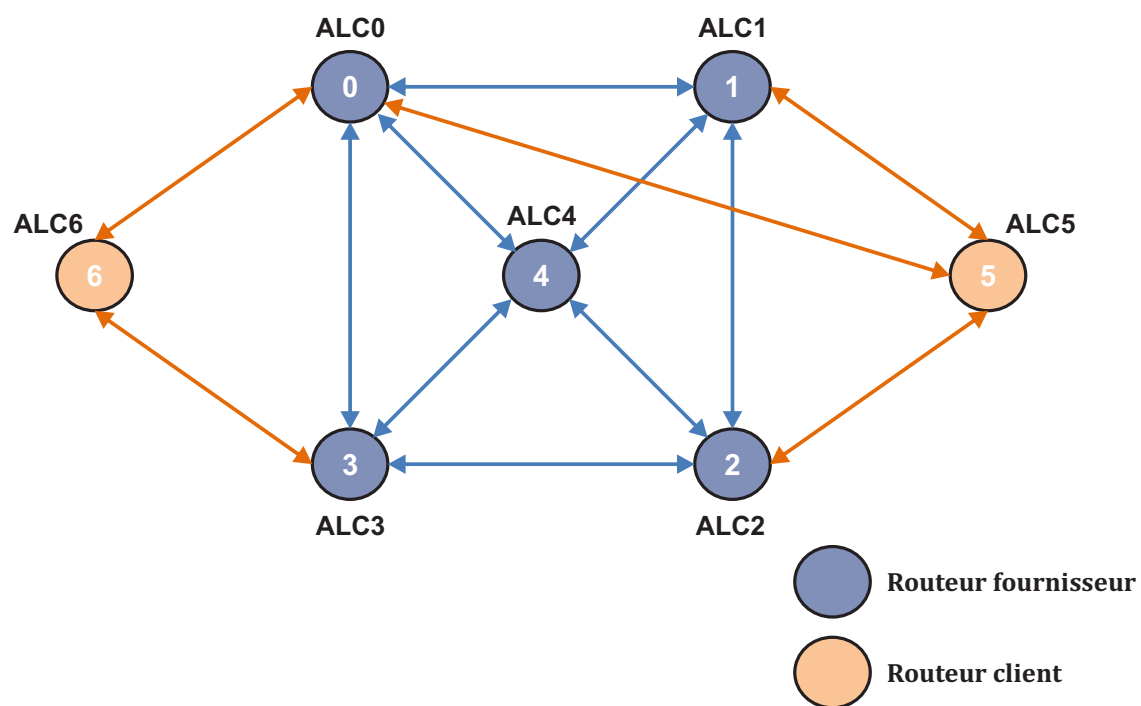


Figure 3.4 Exemple d'un réseau

Ce réseau contient deux nœuds privés. Si on ne considère que les deux ports 80 et 25 et les deux protocoles UDP et TCP, alors le nombre de paquets générés par l'automate "Noeud_gnrateur", pour le modèle du réseau ci-dessus, est huit paquets. Ces huit paquets (voir tableau 3.2) représentent toutes les classes des paquets possibles et deux à deux différents.

Tableau 3.2 Paquets générés.

Attributs/paquets	0	1	2	3	4	5	6	7
Source	5	5	5	5	6	6	6	6
Destination	6	6	6	6	5	5	5	5
Port	80	80	25	25	80	80	25	25
Protocole	TCP	UDP	TCP	UDP	TCP	UDP	TCP	UDP

La Figure 3.5 présente un exemple de simulation du réseau ci-dessus, dans le simulateur UPPAAL. Le simulateur affiche l'ensemble de paquets en transit ainsi que leurs données statiques et dynamiques. Par exemple, le paquet 0 a été directement rejeté (*état*=-1) par le pare-feu responsable de sa destination 6. Le seul nœud visité par ce paquet est son nœud source 5. Ainsi, le nombre de nœuds visités est 1.

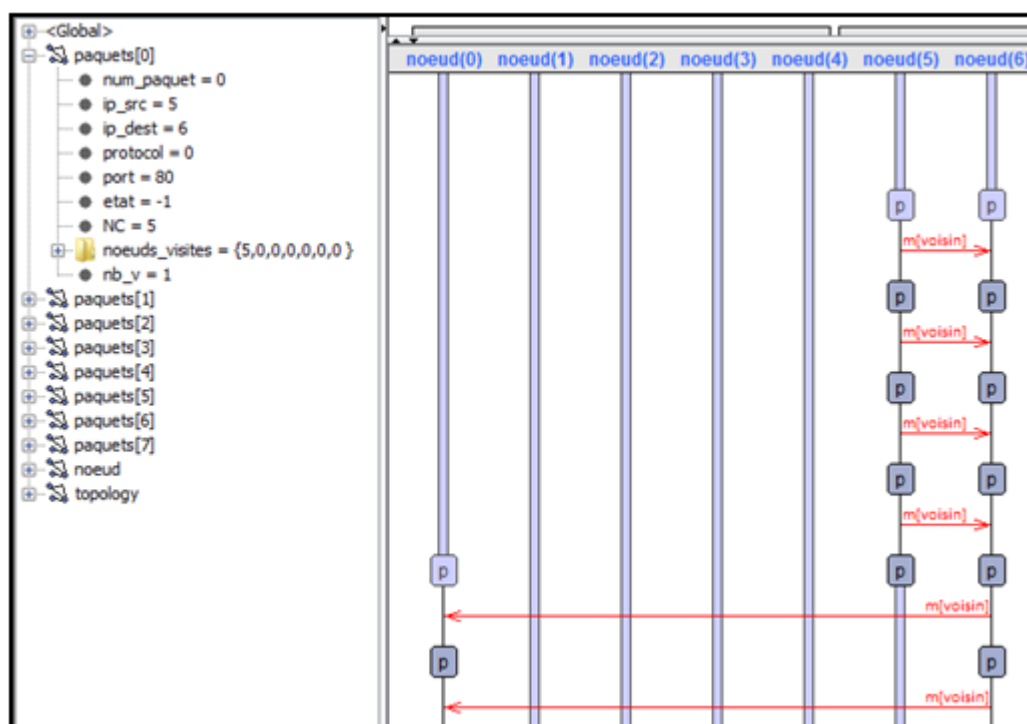


Figure 3.5 Exemple de simulation du modèle.

3.4 Vérification formelle de la cohérence des pare-feux

Dans la section précédente, nous avons présenté notre modèle qui décrit le fonctionnement d'un réseau informatique à pare-feux. Le but de cette section est de vérifier et valider le comportement du modèle tout en détectant les erreurs de configuration distribuées qui peuvent entraver la sécurité du réseau.

Pour ce faire, nous avons déterminé un ensemble de propriétés permettant d'analyser le comportement du réseau. Les propriétés ont été exprimées en logique temporelle à la UPPAAL. Nous rappelons que A et E sont des quantificateurs de chemin, $A[] p$ signifie que tous les états accessibles à partir de l'état actuel satisfont la propriété p (y compris l'état actuel) et $E \langle \rangle p$ signifie qu'il y a une évolution conduisant à un état qui satisfait la propriété p .

Les propriétés sont répertoriées en 3 classes : i) les propriétés assurant le bon fonctionnement du modèle, ii) la propriété de la cohérence de la politique de sécurité globale du réseau et iii) la propriété de croisement de chemins. Dans ce qui suit, nous analysons ces 3 classes de propriétés.

3.4.1 Propriétés assurant le bon fonctionnement du modèle

Ce groupe de propriétés a été envisagé pour valider le bon fonctionnement du modèle. Le tableau 3.3 présente la syntaxe ainsi que les résultats de vérification du modèle vis-à-vis de ces propriétés. La propriété P_1 montre que le modèle ne comporte pas des deadlocks sauf au cas où il a fini le traitement des paquets qu'il a générés. La propriété P_2 stipule que si l'état *out* est activé alors tous les paquets sont traités avec succès (acceptés, rejetés ou bloqués dans un nœud terminal). L'activation de l'état *out* annonce la fin d'exécution du modèle. La propriété P_3 prouve la capacité du modèle à faire transiter les paquets qu'il génère entre les nœuds. Les propriétés P_4 et P_5 quant à elles, vérifient la capacité de modèle à utiliser les listes de contrôle d'accès en vue de rejeter les paquets suspects et accepter ceux qui ont accès selon les règles établies. Les deux propriétés P_6 et P_7 font preuve que le modèle est capable d'acheminer les paquets générés à leurs destinations.

Tableau 3.3 Propriétés vérifiant le bon fonctionnement du modèle

	Propriétés	Syntaxe	Résultats
P_0	L'absence des deadlocks	$A[] \textit{not deadlock}$	Non Satisfaite
P_1	L'absence des deadlocks avant la terminaison du traitement de tous les paquets.	$A[](\textit{deadlock imply end_proc} == \textit{true})$	Satisfaite
P_2	L'état "out" est activé alors le modèle a fini le traitement des paquets.	$A[](\textit{forall}(i : \textit{int}[0, nb_p - 1])(\textit{end_proc} == \textit{true imply packets}[i].\textit{isrejected} \parallel \textit{packets}[i].\textit{isblocked} \parallel (\textit{packets}[i].\textit{isaccepted})))$	Satisfaite
P_3	Un paquet peut transiter par un nœud intermédiaire.	$E \langle \rangle \textit{exists}(i : \textit{int}[0, nb_p - 1])(\textit{packets}[i].\textit{nb} > 2)$	Satisfaite
P_4	Un paquet parmi les paquets générés peut être rejeté par l'une des listes d'accès.	$E \langle \rangle \textit{exists}(i : \textit{int}[0, nb_p - 1])\textit{packets}[i].\textit{isrejected}$	Satisfaite
P_5	Un paquet parmi les paquets générés peut être accepté par l'une des listes d'accès.	$E \langle \rangle \textit{exists}(i : \textit{int}[0, nb_p - 1])(\textit{packets}[i].\textit{isaccepted})$	Satisfaite
P_6	Un paquet peut se boquer dans un nœud terminal.	$E \langle \rangle \textit{exists}(i : \textit{int}[0, nb_p - 1])(\textit{packets}[i].\textit{isblocked})$	Satisfaite
P_7	Les paquets générés peuvent être acheminés sans blocage à leurs destinations.	$E \langle \rangle (\textit{exists}(i : \textit{int}[0, nb_p - 1])(\textit{packets}[i].\textit{isaccepted} \&\& \textit{packets}[i].\textit{ip_dest} == \textit{packets}[i].\textit{NC}))$	Satisfaite

3.4.2 Cohérence de la politique de sécurité globale du réseau

La cohérence de la politique de sécurité globale du réseau est une propriété fondamentale dans la spécification de notre modèle. Cette propriété a été exprimée en utilisant la syntaxe suivante :

$$E \langle \rangle \textit{exists}(i : \textit{int}[0, nb_p - 1]) \textit{packets}[i].\textit{isrejected imply node}(\textit{ip_dest}).\textit{decision}(i) == -1)$$

La satisfaction de cette propriété permet de montrer que si un paquet devrait être accepté par sa destination, il ne doit jamais être rejeté par un nœud de transit sur un chemin qui mène à cette destination. Cette propriété permet de vérifier qu'il n'existe pas des règles en aval du flux qui sont ombragées par d'autres en amont du flux.

3.4.3 Propriété vérifiant l'incohérence de croisement de chemins

La propriété de croisement de chemins permet de vérifier qu'un paquet peut être soit constamment rejeté soit constamment accepté. C'est à dire, si un paquet est rejeté par l'un des pare-feux en direction de sa destination, il ne pourra pas l'atteindre en empruntant un autre chemin. La vérification de cette propriété est faite en deux étapes :

1. La vérification qu'un paquet i donné est rejeté en empruntant l'importe quel chemin. La syntaxe d'une telle propriété est :

$$A \langle \rangle (end_proc == true \ \&\& \ packets[i].isrejected) \text{ où } i < nb_p$$

2. La vérification qu'un paquet i donné est accepté ou bloqué dans un nœud terminal en empruntant l'importe quel chemin. La syntaxe d'une telle propriété est :

$$A \langle \rangle (end_proc() == true \ \&\& \ (packets[i].isaccepted \ || \ packets[i].isblocked)) \text{ où } i < nb_p$$

La satisfaction de la propriété de croisement de chemins pour un paquet i donné engendre la satisfaction de l'une des deux propriétés, citées ci-dessus, et la non-satisfaction de l'autre. Afin de montrer la capacité du modèle à détecter une telle incohérence, nous avons intégré cette anomalie dans la configuration des règles de filtrage. Le résultat prouve que le modèle est capable de détecter l'incohérence de croisement de chemins. La Figure 3.6 représente un exemple illustrant comment notre approche peut détecter l'incohérence de croisement de chemins.

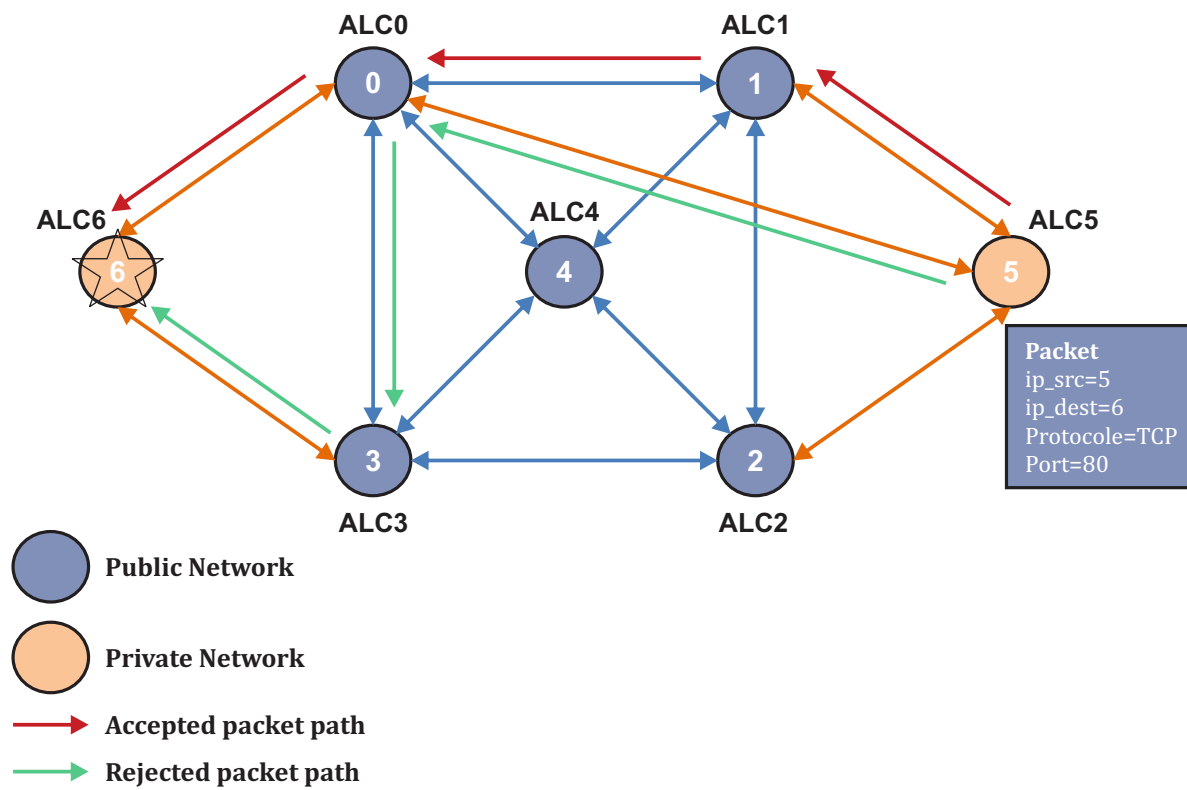


Figure 3.6 Exemple de vérification de modèle

En effet, un paquet envoyé par le nœud source 5 peut prendre des chemins différents pour atteindre son nœud destination 6 (i.g. 5-1-0-6, 5-0-6, 5-1-2-3-6, etc.). La technique de model checking permet d'explorer tous les chemins qui lient les deux nœuds 5 et 6 en vérifiant que l'état du paquet (accepté ou rejeté) est le même quel que soit le chemin emprunté.

3.4.4 Etude expérimentale de performance

Dans cette section, nous discutons la performance de notre approche. Nous rappelons que notre modèle décrit chaque nœud dans le réseau par un automate avec un seul état et trois arcs : i) le premier arc envoie les paquets, ii) le deuxième arc reçoit les paquets envoyés par les autres automates et iii) le troisième arc détermine le prochain paquet à traiter.

Les mesures ont été effectuées en utilisant la version 4.0.13 de UPPAAL installée sur une machine fedora19 (Core i5 RAM 8GO). Nous avons utilisé, en conjonction avec UPPAAL, l'utilitaire memtime1.3 qui permet de déterminer le temps de vérification et le taux d'utilisation de mémoire. Notre étude expérimentale a été réalisé uniquement pour la propriété de croisement de chemins :

A <> (end_proc == true && paquets[i].isrejected) o i est un identificateur d'un paquet

La Figure 3.7 et la Figure 3.8 décrivent la performance du modèle respectivement en termes du temps de vérification et de la mémoire consommée.

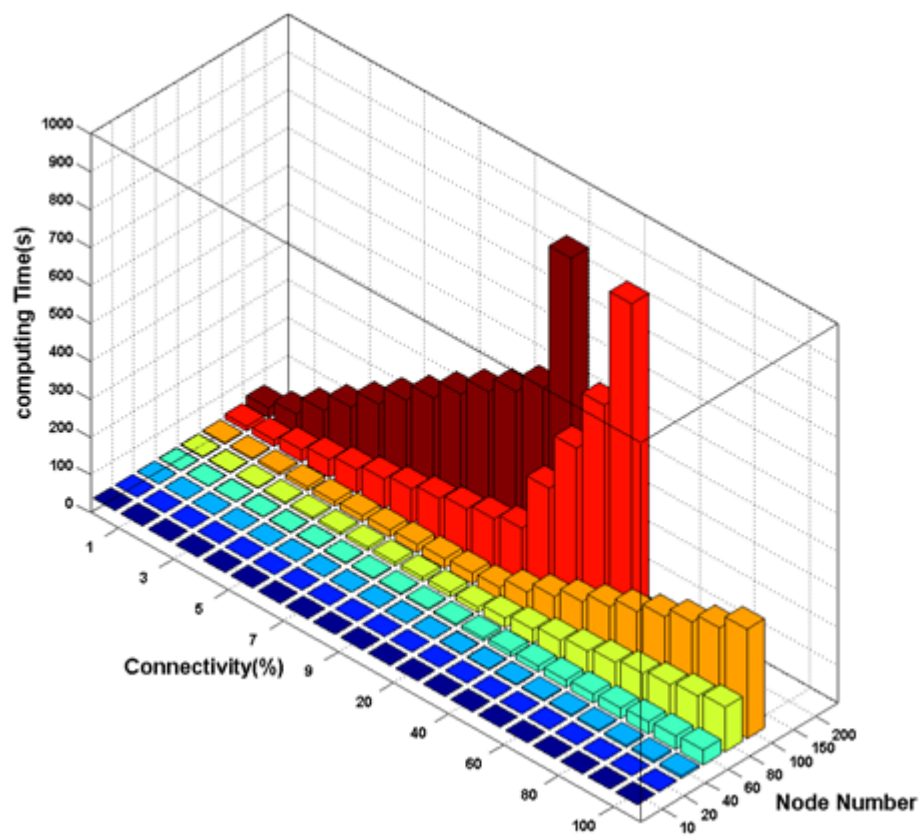


Figure 3.7 Temps de vérification de la propriété de croisement de chemins.

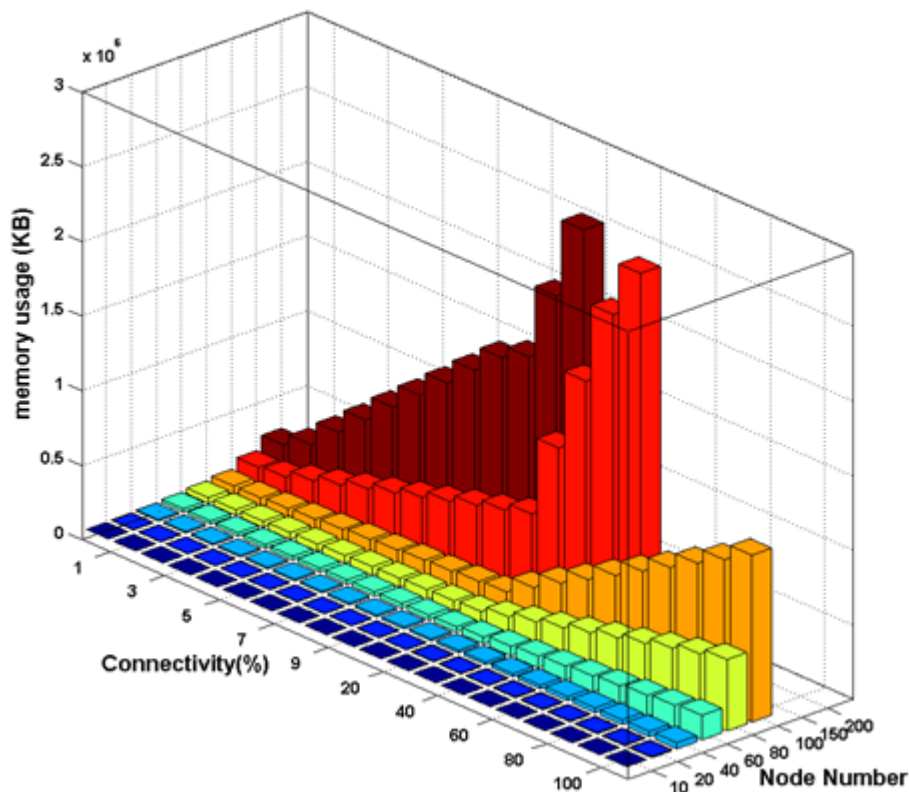


Figure 3.8 Taux d'utilisation de la mémoire pour la propriété de croisement de chemins.

Ces deux figures montrent que le modèle peut vérifier des réseaux de 100 nœuds fortement connectés (100% de connectivité) en 200 s avec un taux d'utilisation de mémoire voisin de 1 GB. Notre modèle peut également vérifier des réseaux de 200 nœuds avec un taux de connectivité de 20% en seulement 700 s et en consommant environ 2,5 Go de mémoire. Ces statistiques montrent la capacité du modèle à gérer des grands réseaux avec un taux de connectivité relativement élevé. Cependant, notre modèle est limité par le problème d'explosion combinatoire. En effet, au delà de 200 nœuds, le modèle n'est plus capable de vérifier la propriété considérée. Dans la section suivante, nous proposons une solution à ce problème.

3.5 Comment atténuer le problème d'explosion combinatoire ?

Comme toute solution formelle basée sur la technique de model checking, notre approche se heurte au problème d'explosion combinatoire. Pour contourner ce problème, nous avons limité l'étendue des paramètres du modèle tels que le nombre de nœuds clients, le nombre de protocoles et le nombre de ports. Nous avons aussi généré une liste statique comportant tous les types de paquets possibles en précisant les sources et les destinations dans le réseau à

vérifier. Cette liste est utilisée par tous les nœuds du modèle. De même, Nous avons conservé, dans une seule structure de données, les informations de parcours liées à chaque paquet au lieu de les faire voyager d’un nœud à un autre. Nous avons aussi mis en œuvre un automate permettant de définir la topologie d’un réseau d’une façon statique pour éviter de traiter parallèlement plusieurs exemplaires d’un même réseau. Finalement nous avons utilisé l’option ”committed” pour les différentes locations des automates dans le modèle.

Ces mesures et ces abstractions ont permis d’atténuer énormément le problème d’explosion combinatoire donnant ainsi les résultats discutés dans la section précédente. Cependant, ce problème persiste pour les grands réseaux fortement connectés. Par conséquent, une nouvelle abstraction du modèle est proposée. La description et l’évaluation de cette abstraction sont présentées dans la section suivante.

3.5.1 Modèle réduit

Dans le nouveau modèle, nous spécifions le réseau avec un seul automate présenté dans la figure 3.9. Cet automate gère l’évolution des paquets d’un nœud à un autre en abstrayant les communications effectives des différents nœuds via les canaux de synchronisation.

La procédure *configure_network()* permet de spécifier le réseau à vérifier et génère une liste exhaustive contenant tous les types de paquets non redondants. Cette procédure permet également de déterminer les règles dans les *ACLs* des pare-feux implantés sur le réseau.

La prochaine destination d’un paquet en transit est choisie aléatoirement en utilisant l’option de sélection. Une fois la prochaine destination de paquet est déterminée, la procédure *update_packets()* est appelée pour mettre à jour le statut du paquet en fonction de la décision des règles du pare-feu responsable.

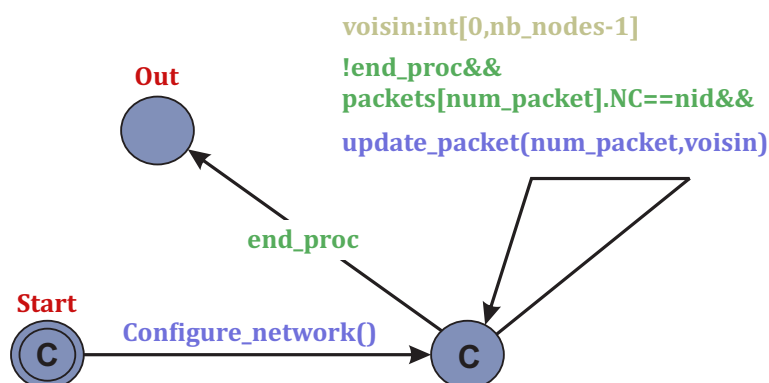


Figure 3.9 Automate réseau.

3.5.2 Etude de performance du modèle réduit

Nous étudions, maintenant, la performance de notre nouveau modèle. La Figure 3.10 et la Figure 3.11 montrent la performance du modèle, respectivement, en termes du temps de vérification et de la quantité de mémoire consommée. Ces deux figures montrent que le modèle peut vérifier un réseau de 150 nœuds fortement connectés (i.e. le taux de connectivité est à 100%) en uniquement 3 s avec un taux d'utilisation de mémoire de 4 MB. De plus, notre modèle peut également vérifier les réseaux de 200 nœuds, avec un taux de connectivité de 80% en seulement 4 s et en consommant environ 7 Mo de mémoire.

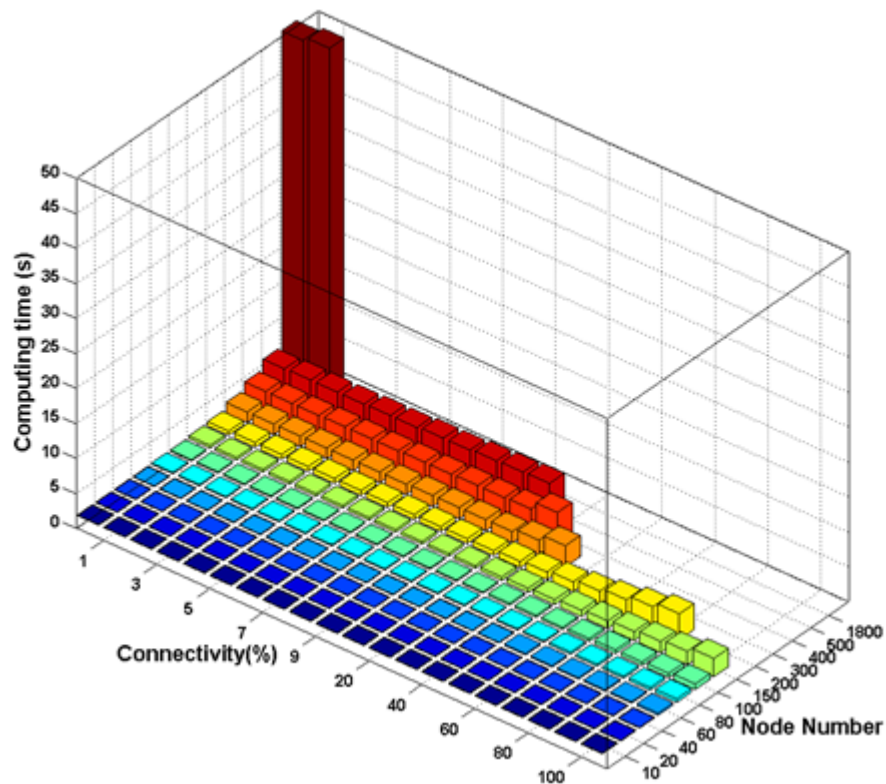


Figure 3.10 Temps de vérification de la propriété de croisement de chemins pour le modèle réduit.

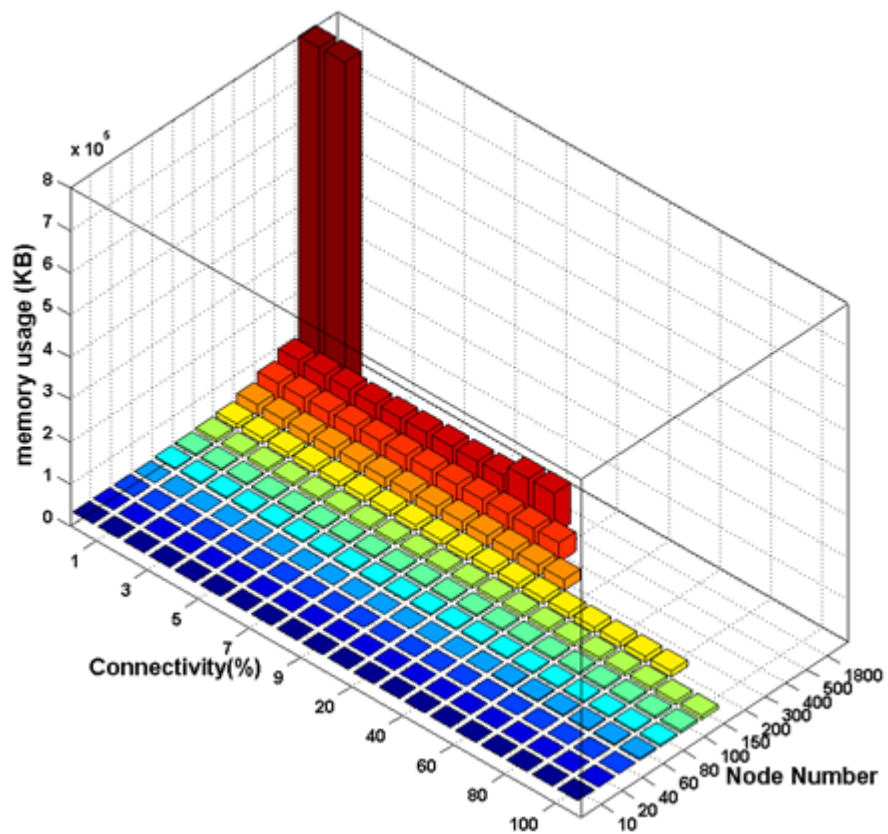


Figure 3.11 Taux d'utilisation de la mémoire pour la propriété de croisement de chemin pour le modèle réduit.

En comparaison avec le premier modèle, nous constatons une nette diminution dans le taux de consommation de mémoire, ainsi que dans le temps de calcul nécessaire à la vérification. En effet, contrairement à la première spécification, ce nouveau modèle peut gérer des réseaux composés de 500 nœuds avec un taux de connectivité de l'ordre de 20% dans un délai assez réduit et avec une utilisation de mémoire assez faible. Par ailleurs, ce modèle peut atteindre jusqu'à 1800 nœuds avec un faible taux de connectivité.

Contrairement à la première abstraction qui se base sur l'évolution des nœuds, cette deuxième abstraction se base sur l'évolution des paquets. Ceci nous a permis de réduire le nombre d'automates de $(n + 1)$ automates où (n) est le nombre de nœuds) à un seul automate. Par conséquent, le nombre d'entrelacements a été diminué de manière spectaculaire au cours du processus de vérification. De ce fait, l'espace d'états a été énormément réduit et le problème d'explosion combinatoire a été grandement atténué. La seule limitation qui confronte toujours notre spécification est une limite logicielle. En fait, UPPAAL ne peut pas gérer des nombres entiers supérieurs à 32767. Par conséquent, UPPAAL est incapable de gérer certaines topologies où le nombre de connexions entre les différents nœuds dépasse cette limite.

3.6 Extension quantitative du modèle

Dans cette section, nous étudions l'aspect quantitatif d'un réseau informatique. Notre intérêt porte sur l'évaluation de la performance du réseau en se basant sur des métriques comme le délai d'acheminement des paquets de bout en bout, le délai d'attente et le taux de perte. L'analyse de ces métriques permet d'évaluer et d'optimiser le réseau tout en assurant une meilleure qualité de service. Notre étude quantitative a été élaborée en utilisant SMC UPPAAL. Nous présentons tout d'abord, notre modèle formel. Ensuite, nous analysons le comportement du modèle en spécifiant un ensemble de propriétés quantitatives liées à la qualité de service du réseau.

3.6.1 Paramètres quantitatifs

Afin d'étudier le comportement quantitatif d'un réseau informatique, le modèle qualitatif présenté précédemment a été étendu à deux nouveaux aspects : i) l'aspect temporel qui fait intervenir des contraintes de temps afin de quantifier, par exemple, les délais d'attente et les délais d'acheminement de paquets et ii) l'aspect probabiliste qui fait intervenir les notions de coût et de probabilité afin de quantifier par exemple le taux de perte de paquets et le taux de pannes de nœuds. Par conséquent, nous avons introduit l'ensemble de paramètres quantitatifs suivant :

- **Horloges** : Les horloges ont été introduites afin de mesurer les délais séparant les différentes actions dans le modèle. Elles évoluent de manière continue et synchrone avec le temps.
- **Taux d'horloges** : Les taux d'horloges ont été introduits pour modéliser les coûts de séjour dans les différents états du modèle. En effet, les taux d'horloges permettent de quantifier l'évolution d'horloges dans un état. Ces taux peuvent être spécifiés par un entier positif. Lorsqu'un taux d'horloge est à 0, ceci signifie que cette horloge est stoppée jusqu'à ce que le processus quitte l'état correspondant.
- **Délais probabilistes** : Les délais probabilistes permettent de synchroniser les différents processus dans le modèle. En fonction de son état courant, chaque automate choisit un délai de séjour. Le processus de plus bref délai est choisi pour exécuter une de ses prochaines transitions dont la garde est satisfaite. S'il y en a plusieurs processus dont les délais sont égaux, alors un parmi eux sera choisi, aléatoirement, selon une distribution uniforme. D'une manière générale, le choix du délai, par un processus, est déterminé selon les règles suivantes :
 - Si l'état courant du processus admet un invariant borné, alors le délai est pris selon une distribution uniforme jusqu'à la borne définie.
 - Si l'invariant est absent, le délai est choisi selon une loi exponentielle en utilisant un taux λ déterminée à l'état courant (λ est défini dans l'onglet intensité exponentielle de l'interface de modélisation dans SMC UPPAAL). La probabilité de quitter l'état courant est alors : $Pr(partir \text{ aprs } t) = 1 - e^{-\lambda t}$, où $e = 2,718281828 \dots$, t est le temps et λ est le taux exponentiel fixé. La densité de probabilité de la distribution exponentielle est $\lambda e^{-\lambda t}$. Donc, λ désigne intuitivement la densité de probabilité de quitter à l'instant zéro (délai nul). L'augmentation du taux λ , diminue le délai et vice versa.
- **Probabilités de franchissement** : Les transitions, dans notre modèle, peuvent être associées à ce qu'on appelle des probabilités de franchissement. Après sa détermination, le processus de plus bref délai tente à franchir une transition. Son choix est basé sur les deux règles suivantes :
 - Si la transition a des branches probabilistes alors la probabilité de suivre une branche i est déterminée par le rapport w_i/W , où w_i est le poids de la branche i et W est la somme de tous les poids des branches émises de la transition considérée : $W = \sum_j w_j$.

- Sinon, le processus choisit une transition selon une distribution uniforme, c'est-à-dire, il choisit, aléatoirement, l'une des transitions dont les gardes sont satisfaites.

3.6.2 Modèle quantitatif

Dans cette section, nous décrivons notre modèle formel dédié à l'analyse du comportement quantitatif d'un réseau informatique. Ce nouveau modèle se compose de n automates temporisés probabilistes (PTA, Priced Timed Automata) où n est le nombre de nœuds dans le réseau. Chaque automate, appelé aussi processus, est paramétré par un identifiant unique "*nid*". Les automates se communiquent par une synchronisation binaire.

Nous rappelons que le trafic entre deux processus (nœuds) dans une direction spécifique (source-destination) est contrôlé par des transitions étiquetées par des pare-feux. Les états peuvent être aussi annotés par des taux d'horloges qui permettent de préciser le taux d'évolution du temps dans l'état en question par rapport à une horloge donnée.

La Figure 3.12 décrit l'automate modélisant un nœud du réseau et le Tableau 3.4 présente le lexique de variables utilisé dans notre modèle. Si un automate est choisi comme le processus de plus bref délai, la procédure *ready_to_send()* sera appelée afin de vérifier si le processus en question est prêt à envoyer l'un des paquets qui se trouvent dans sa liste d'attente.

Chaque nœud a une probabilité de défectuosité définie comme une variable globale. Cette probabilité peut être calculée en fonction de plusieurs facteurs à savoir l'âge du nœud, son nombre de pannes, la quantité de données qui y transite, etc. Selon cette probabilité, le nœud peut passer à un état de panne temporaire ou un état "*pré-envoi*".

Chaque nœud a un délai de rétablissement pendant lequel il peut être réparé s'il tombe temporairement en panne. Si le délai de séjour, dans un état de panne temporaire, dépasse le délai de rétablissement "*delai_retab*" du nœud en question, l'automate franchit alors une transition probabiliste. Cette transition permet de décider si l'automate considéré doit passer à un état de panne permanente ou bien un état de remplacement où le nœud correspondant est remplacé par un autre nœud fonctionnel.

La décision prise à ce niveau dépend de la probabilité "*rep_prob*" qui présente la probabilité de remplacement du nœud considéré. Cette probabilité peut être calculée en fonction de plusieurs facteurs comme l'âge du nœud, son nombre de pannes, la quantité du matériel mise à disposition, etc.

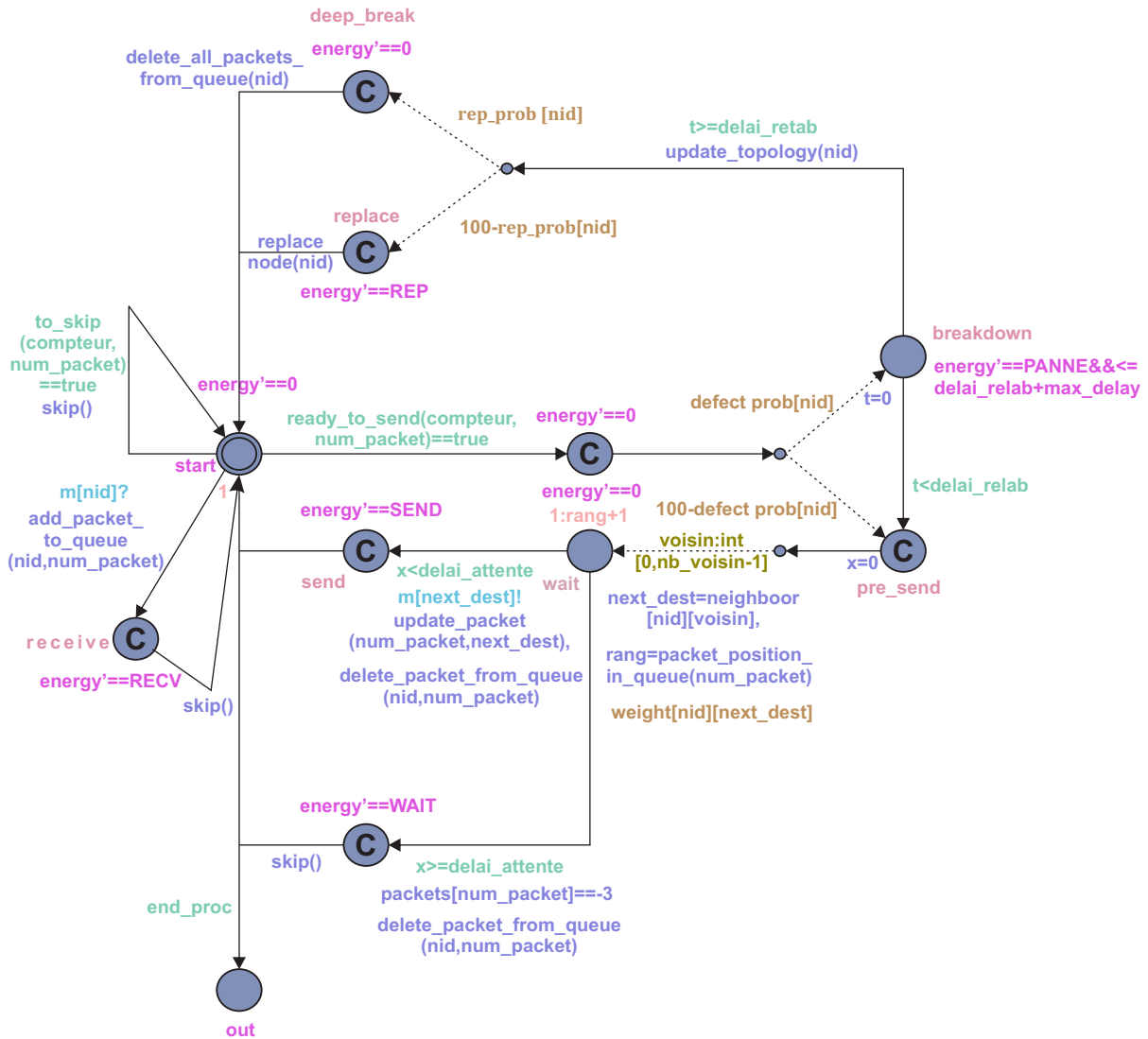


Figure 3.12 Modèle quantitatif du réseau.

Le passage du nœud à un état de panne permanente ou un état de remplacement engendre l'exécution de la fonction *update_topology()* qui permet de supprimer le nœud en question ainsi que toutes les connexions qui le relie avec les autres nœuds voisins.

Dans le cas où le nœud tombe en panne définitivement, la procédure *delete_all_packets_from_queue()* sera appelée pour supprimer tous les paquets se trouvant dans la liste d'attente du nœud en question en mettant à jour leur statut (les paquets sont désormais perdus).

Dans le cas où le nœud est à un état de remplacement alors la procédure *replace_node()* sera appelée pour introduire un nouveau nœud dans la topologie du réseau ayant les mêmes caractéristiques que celui supprimé. Cette procédure établit également des nouvelles connexions vers les autres nœuds voisins ayant les mêmes caractéristiques que celles supprimées auparavant.

Tableau 3.4 Lexique de variables.

TYPES	Variables	Définitions
Horloges	energy	Quantifie le coût temporel dans le réseau.
	x	Quantifie le délai d'attente du paquet dans sa liste d'attente.
	t	Quantifie le délai de séjour du nœud dans un état de panne temporaire.
Entiers	Voisin	Désigne le prochain nœud de transit choisi.
	RECV	Coût temporel relatif à une action d'envoi dans le réseau.
	SEND	Coût temporel relatif à une action de réception dans le réseau.
	REP	Coût temporel relatif à une action de remplacement d'un nœud défectueux par un autre nœud fonctionnel dans le réseau.
	WAIT	Coût temporel relatif à une action d'attente d'un paquet dans le réseau.
	Délai_attente	Désigne le délai d'attente maximal d'un paquet dans sa liste d'attente à partir duquel le paquet en question sera perdu.
	Délai_retab	Désigne délai de rétablissement maximal pendant lequel un nœud peut être réparé s'il tombe en panne temporairement.
Tableaux	Rep_prob[nb_nodes]	Désigne les probabilités de remplacement de chaque nœud dans le réseau.
	Defect_prob[nb_nodes]	Désigne les probabilités de défectuosité de chaque nœud dans le réseau.
Matrices	Neighbor[nb_nodes][nb_voisins]	Contient les voisins de chaque nœud en se basant sur la topologie di réseau à étudier.
	Weight[nb_nodes][nb_nodes]	Désigne les coûts des différentes connexions directes dans le réseau à étudier.

Le passage par un état de panne temporaire ou un état de remplacement augmente, respectivement, l'énergie cumulative du réseau "energy" par la valeur *PANNE* ou *REP*. *PANNE* représente le coût temporel relatif à une situation de panne temporaire et *REP* représente le coût temporel relatif au remplacement d'un nœud. Le passage par un état de panne permanente n'engendre aucun coût temporel (à ce niveau le modèle arrête l'horloge "energy" : $energy = 0$).

Si le nœud réussit de passer à l'état de *pré_envoi*, l'automate correspondant franchit une transition probabiliste lui permettant de choisir le prochain voisin. Les voisins de chaque nœud sont enregistrés dans une matrice "neighbor". La probabilité de choisir le prochain nœud de transit est déterminée en fonction du coût associé à la transition qui interconnecte le nœud courant et le nœud choisi. Les coûts de toutes les connexions sont définis dans la matrice "weight". Ils peuvent être déterminés en fonction de plusieurs facteurs matériels (i.g. le coût de câblage, de mise en œuvre, réparation, etc.) et logiciels (i.g. la configuration du pare-feu, nombre de règles dans les ACLs, etc.).

Après avoir choisi le prochain nœud voisin, l'automate passe à un état d'attente où le délai d'attente x est déterminé selon la loi exponentielle en utilisant un taux λ égal à $1/rang + 1$ où *rang* définit le rang du paquet dans la liste d'attente du nœud courant. Ainsi, le paquet, dont le rang est le plus élevé, a le délai d'attente le plus grand.

Cependant, le temps de séjour x dans l'état "wait" est contrôlé par une variable du réseau appelée "delai_attente" qui définit le temps d'attente maximale dans un état "wait". En effet, si le temps de séjour x dépasse *delai_attente*, il va y avoir les actions suivantes : i) Le paquet en transit est perdu, ii) La procédure *delete_packet_from_queue()* est appelée pour supprimer le paquet de la liste d'attente du nœud courant, iii) L'énergie commutative de système est augmentée par la valeur *WAIT* qui représente le coût temporel relatif à l'action d'attente correspondante et iv) La procédure *skip()* est appelée pour choisir le prochain paquet à traiter.

Si, en contre partie, le temps du séjour dans l'état d'attente est inférieur strictement au "delai_attente" défini par le système, alors le modèle exécute les actions suivantes : i) Le paquet en question est envoyé au nœud voisin choisi, ii) La procédure *updatepacket()* est alors appelée pour mettre à jour le statut du paquet en question, iii) La procédure *delete_packet_from_queue()* est appelée pour supprimer le paquet de la liste d'attente du nœud expéditeur, iv) La procédure *add_packet_to_queue()* est appelée pour ajouter le paquet à la liste d'attente du nœud récepteur, et v) L'énergie commutative de système est augmentée par les valeurs *SEND* et *RECV* qui représentent, respectivement, les coûts temporels relatifs à l'action d'envoi et de réception dans le réseau.

Notons que nous avons utilisé l'automate, présenté dans la figure 3.2, afin de spécifier la

topologie du réseau à étudier et de générer la listes de paquets circulant sur le réseau.

3.6.3 Analyse des propriétés quantitatives

Nous avons implémenté et vérifié notre modèle en utilisant le model checking statistique SMC UPPAAL. En effet, SMC UPPAAL supporte les automates temporisés probabilistes. Il combine les fonctionnalités classiques du model checking et la technique de simulation (Échantillonnage de Monte Carlo) afin de soutenir l'analyse quantitative des systèmes temporisés et limiter le problème d'explosion combinatoire.

L'échantillonnage est effectué en fonction de la distribution de probabilité définie par les automates probabilistes. Les paramètres de réglage ϵ (paramètre d'approximation) et α (le paramètre de confiance) peuvent être utilisés pour spécifier la confiance statistique du résultat.

Dans cette section, nous présentons et interprétons les propriétés quantitatives que nous avons spécifiées afin d'étudier le comportement du réseau. Ces propriétés sont résumées dans le Tableau 3.6. Le paramètre d'approximation ϵ et le paramètre de confiance (paramètre des faux négatifs) α sont tous les deux mis à 0.01 avec un taux de confiance ($1 - \alpha$) de 99%. Nous avons vérifié nos propriétés sur un réseau composé de 5 nœuds avec un taux de connectivité égale à 100 %. Le tableau 3.5 résume la configuration du réseau considéré.

Tableau 3.5 Paramètres du réseau à étudier.

Paramètres du réseau	Valeurs
Nombre de nœuds	5
Taux de connectivité (%)	100
Rep_prob[nb_nodes] (%)	0, 1, 1, 50, 1
Defect_prob[nb_nodes] (%)	0, 1, 1, 50, 1
SEND (unité de temps)	30
RECV (unité de temps)	35
WAIT (unité de temps)	5
PANNE (unité de temps)	3
PANNE (REP (unité de temps)	3
Délai_attente (unité de temps)	10
Délai_retab(unité de temps)	20
Max_delay (unité de temps)	10
Probabilité de connexions (weight[5][5])	weight = 6,6,6,6,6,6,6,6,...

Les propriétés quantitatives que nous avons vérifiées sont :

- La propriété P_1 calcule la probabilité qu'un paquet passant par un nœud malicieux soit accepté. Cette propriété est très intéressante puisqu'elle évalue la configuration

du réseau et permet de prendre des mesures pour sa reconfiguration si la probabilité calculée est supérieure à un seuil bien déterminé. Dans le réseau décrit ci-dessus, le taux d'acceptation d'un paquet i passant par le nœud malicieux 3 est en moyenne 35%. Cependant, une telle probabilité ne peut pas être significative sans la comparer à un taux fixe défini généralement par un expert.

- La propriété P_2 permet de comparer la probabilité précédente à un seuil défini normalement par l'administrateur réseau (i.g. 50%). Le résultat expérimental montre que dans le réseau, le taux d'acceptation d'un paquet passant par le nœud malicieux 3 est strictement inférieur à 50% avec un taux de confiance de 99%. Cette probabilité est en fait rassurante par rapport à ce que nous avons défini comme seuil (50%). Ainsi le réseau étudié ne nécessite aucune reconfiguration éventuelle.
- La propriété P_3 estime la probabilité qu'un nœud tombe en panne. Dans notre étude de cas, le taux de défektivité du nœud 3 est en moyenne 50%. Cette valeur correspond bien à notre configuration initiale.
- La propriété P_4 vérifie la tolérance du réseau aux pannes. C'est à dire, elle permet de déterminer la capacité du réseau à acheminer un paquet à sa destination en cas de panne au niveau d'un certain nombre de nœuds. Les expérimentations montrent que notre réseau n'est pas tolérant aux pannes. En effet, la panne des deux nœuds de transit (nœud 1 et nœud 2) réduit le taux d'acceptation de paquets à 1%.
- La propriété P_5 calcule le taux de perte des paquets dans le réseau. Dans cette étude de cas, cette probabilité est de 65% en moyenne. Cette probabilité paraît très élevée et révèle une mauvaise configuration du réseau. Cependant, seul l'administrateur du réseau peut juger la signification d'une telle probabilité en la comparant avec un seuil qu'il définit en se basant sur les données du réseau.

Tableau 3.6 Propriétés quantitatives

Propriétés	Syntaxe	Résultats
P_0 $i < nb_p$	$\Pr[energy \leq 5000](\langle \rangle end_proc \ \&\& \ exists(j : int[0, nb_nodes - 1]) (paquets[i].noeuds_visites[j] == 3) \ \&\& \ paquets[i].isaccepted)$	$[0.332512, 0.432506]$
P_2 $i < nb_p$	$\Pr[energy \leq 5000](\langle \rangle end_proc \ \&\& \ exists(j : int[0, nb_nodes - 1]) (paquets[i].noeuds_visites[j] == 3) \ \&\& \ paquets[i].isaccepted) \ \>= \ 0.50.49$	$\Pr(\langle \rangle \dots) <= 0.49$
P_3 $i < nb_nodes$	$\Pr[energy \leq 5000](\langle \rangle end_proc \ \&\& \ ni.isdesabled)$	$i = \{0, 2, 4\},$ $[0, 0.0973938]$ $i = 4,$ $[0.459921, 0.559832]$
P_4 $i < nb_p$	$\Pr[energy \leq 5000](\langle \rangle end_proc \ \&\& \ n1.isdesabled \ \&\& \ n2.isdesabled \ \&\& \ paquets[i].isaccepted)$	$[0, 0.0973938]$
P_5	$\Pr[energy \leq 5000](\langle \rangle \ exists(i : int[0, nb_p - 1]) paquets[i].islost \ \ paquets[i].isblocked)$	$[0.620252, 0.719973]$
P_6	$\Pr[energy \leq 5000](\langle \rangle \ exists(i : int[0, nb_p - 1]) paquets[i].islost \ \ paquets[i].isblocked) \ \>= \ 0.6$	$\Pr(\langle \rangle \dots) \ \>= \ 0.51$
P_7	$\Pr[energy \leq 5000](\langle \rangle \ exists(i : int[0, nb_p - 1]) paquets[i].ip_src == 4 \ \&\& \ (paquets[i].islost \ \ paquets[i].isblocked))$	$[0.56013, 0.660073]$
P_8	$\Pr[energy \leq 5000](\langle \rangle \ forall(i : int[0, nb_p - 1]) paquets[i].isaccepted)$	$[0.242346, 0.343354]$
P_9	$\Pr[energy \leq 5000](\langle \rangle \ forall(i : int[0, nb_p - 1]) paquets[i].rejected)$	$[0.412606, 0.446064]$

- La propriété P_6 permet de comparer la probabilité précédente à un seuil défini par l'administrateur sans avoir à la calculer. Cette propriété permet de prendre des mesures pour la réparation du réseau au cas où cette probabilité dépasse le seuil défini (i.e. 50%). Dans notre étude de cas, Nous fixons le seuil à 50%. Le taux de perte dans le réseau est strictement supérieur à 50%. Cette valeur est inquiétante et nécessite l'intervention de l'administrateur du réseau afin de régler les problèmes techniques et réduire les blocages dans les nœuds terminaux.
- La propriété P_7 permet de calculer le taux de perte d'une classe de paquets bien déterminée. Dans notre cas, cette propriété permet de calculer le taux de perte des paquets qui proviennent du nœud source 4. Cette probabilité est de 60% en moyenne. Pour cette probabilité, aussi, seul l'administrateur peut décider si la situation requiert une intervention humaine pour réparer les problèmes techniques, s'il y en a, ou de réduire les blocages dans les nœuds terminaux en mettant à jour les tables de routage.
- Les propriétés P_8 et P_9 permettent d'étudier respectivement, la permissivité et la restrictivité du réseau analysé. En effet, P_8 calcule la probabilité d'acceptation de paquets dans le réseau et P_9 calcule la probabilité du rejet des paquets dans le réseau. Si la probabilité d'acceptation (ou de rejet) dépassent un seuil défini par un expert, le réseau est alors considéré permissif (ou restrictif) et sa reconfiguration est requise afin de

diminuer sa permissivité (ou sa restrictivité).

3.7 Conclusion

Dans la première partie de ce chapitre, nous avons proposé une approche formelle qui permet de vérifier qualitativement le comportement de bout en bout de la sécurité dans un réseau (end-to-end security behavior). Plus précisément, cette approche permet de vérifier la cohérence des pare-feux distribués en identifiant les anomalies inter-pare-feux, en particulier, l'incohérence de croisement de chemins. L'approche proposée est basée sur la technique de model checking et elle a été implémentée en utilisant l'outil UPPAAL.

Afin d'atténuer le problème d'explosion combinatoire, deux abstractions ont été proposées et évaluées afin d'étudier l'efficacité de l'approche en fonction de la complexité du temps et d'espace. Dans la deuxième partie de ce chapitre, nous avons présenté une extension quantitative du modèle qualitatif permettant d'optimiser le comportement du réseau du bout en bout en se basant sur les métriques de qualité de service d'un réseau tels que le délai d'acheminement des paquets de bout en bout, délai d'attente et le taux de perte. Cette extension supporte deux nouveaux aspects : i) l'aspect temporisé qui fait intervenir des contraintes de temps et ii) l'aspect probabiliste qui fait intervenir les notions de coût et de probabilité. Cette deuxième approche s'appuie sur le model checking statistique et elle a été implémentée en utilisant l'outil SMC UPPAAL.

Dans le chapitre suivant, nous proposons une nouvelle approche permettant de configurer automatiquement les pare-feux surveillant un réseau privé.

CHAPITRE 4

CONFIGURATION FORMELLE D'UN RESEAU INFORMATIQUE

4.1 Introduction

La sécurité des réseaux informatiques constitue un défi majeur dans notre société moderne. Dans le deuxième chapitre, nous avons présenté les mécanismes et dispositifs de sécurité les plus déployés dans les réseaux informatiques. Parmi ces dispositifs de sécurité, nous nous intéressons aux pare-feux et plus précisément à la configuration des pare-feux. Cette tâche très complexe, est souvent source d'anomalies qui peuvent compromettre les objectifs de sécurité d'un réseau.

Dans le troisième chapitre, nous avons montré comment utiliser la technique de model checking pour détecter les erreurs de configuration des pare-feux, vis-à-vis d'une politique de sécurité du réseau.

Dans ce chapitre, nous proposons une approche formelle d'aide à la configuration automatique de pare-feux. Étant donnés, un réseau et un objectif global de sécurité, il s'agit d'utiliser la synthèse de contrôleur pour synthétiser des listes *ACLs* pour certains pare-feux, afin de forcer le réseau à satisfaire l'objectif de sécurité. La configuration ainsi synthétisée pourrait être, par la suite, vérifiée formellement par l'approche de model checking présentée dans le chapitre 3.

Pour ce faire, nous proposons de modéliser le réseau par un ensemble d'automates de jeux où les actions contrôlables concernent le choix de règles de configuration des pare-feux. Nous utilisons l'outil UPPAAL-TIGA pour synthétiser, si elles existent, des stratégies gagnantes permettant de renforcer la politique de sécurité du réseau. Ces stratégies sont analysées et filtrées automatiquement pour en extraire une configuration optimale des pare-feux du réseau.

Ce chapitre est organisé comme suit : la section 4.2 formalise le problème de synthèse de contrôleur. La section 4.3 présente notre modèle formel permettant de vérifier s'il est possible de configurer les pare-feux de telle manière à forcer les objectifs de sécurité d'un réseau. La section 4.4 décrit la méthodologie d'extraction des règles de configuration des pare-feux. La section 4.5 analyse les performances de notre approche alors que la section 4.6 discute les limites de notre approche puis propose une nouvelle abstraction au modèle qui améliore considérablement les performances.

4.2 Formalisation du problème de renforcement d'une politique de sécurité

Étant donné, un réseau composé d'un ensemble de nœuds contrôlés par des pare-feux et une politique de sécurité globale fixée par l'administrateur du réseau, notre objectif est de forcer la satisfaction de la politique de sécurité, en renforçant le contrôle au niveau des pare-feux.

Afin de mettre en œuvre notre objectif, nous avons fixé les hypothèses suivantes :

- **Hypothèse 1** : Les pare-feux contrôlant les nœuds du réseau public sont tous préconfigurés (*ACLs* sont définies auparavant).
- **Hypothèse 2** : Nous nous intéressons à la configuration des pare-feux qui contrôlent les nœuds du réseau privé.
- **Hypothèse 3** : Les listes de contrôle d'accès au niveau des nœuds du réseau privé sont supposées vides initialement.

Pour vérifier si un tel renforcement est possible, nous utilisons la technique de synthèse de contrôleur implémentée dans l'outil UPPAAL-TIGA. La Figure 4.1 montre le schéma fonctionnel de notre approche de renforcement. Les entrées sont un réseau R , défini par sa taille t , son taux de connectivité c , ses nœuds privés et ses pare-feux, et une politique de sécurité P , exprimée par une formule de logique temporelle.

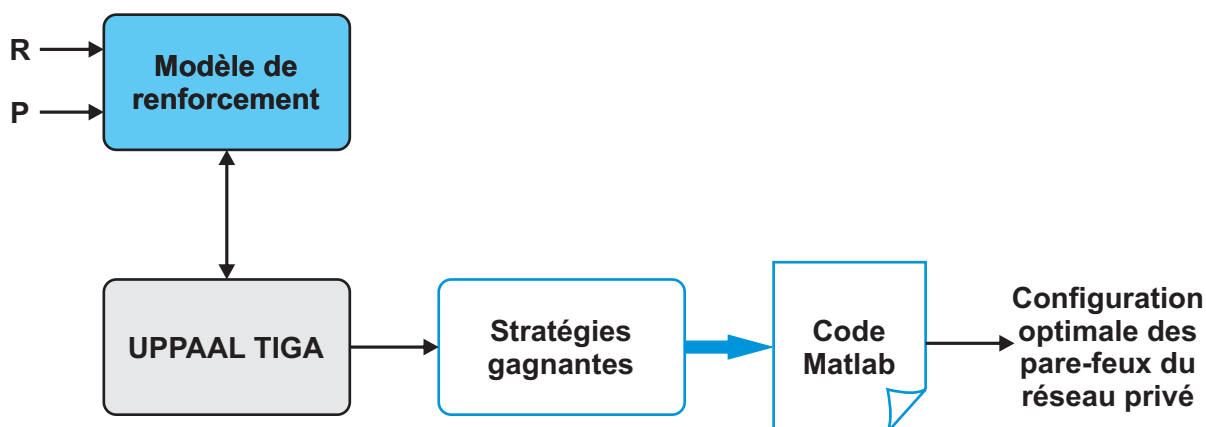


Figure 4.1 Problématique.

Le modèle de renforcement est composé d'automates de jeu où le choix des règles, à appliquer au niveau des pare-feux privés, est contrôlable.

UPPAAL TIGA permet déterminer s'il existe, au moins, une stratégie pour choisir ces règles de manière à satisfaire l'objectif de sécurité P . Si une telle stratégie gagnante existe,

UPPAAL-TIGA fournit, sous forme brute, toutes les stratégies gagnantes. Ces stratégies gagnantes présentent l'ensemble de règles définissant les *ACLs* des pare-feux privés. Ce sont les règles nécessaires pour le contrôle du réseau par rapport à la politique de sécurité *P*.

Le output de UPPAAL TIGA est analysé, en utilisant Matlab, afin d'extraire l'ensemble de règles nécessaires et suffisantes pour le contrôle du réseau qui résulte en une configuration optimale des pare-feux du réseau privé.

4.3 Modèle de renforcement

Notre modèle de renforcement est une extension du modèle qualitatif que nous avons présenté dans la première partie du chapitre 3. Il consiste en : i) un ensemble d'automates de jeu, un automate de jeu par nœud, représentant chacun le comportement d'un nœud identifié par un numéro d'identification unique "*nid*", ii) un automate "*Noeud_gnrateur*" qui a pour rôle de créer les structures de données qui caractérisent le réseau (i.e. la topologie, les types de paquets en transit, les listes de contrôle d'accès des pare-feux publics), et iii) un automate "*Propriet*" qui permet de modéliser l'objectif de contrôle. Il a pour rôle de transformer l'objectif de sécurité en un objectif d'accessibilité.

4.3.1 Automate "Nœud_générateur"

L'automate "*Noeud_gnrateur*", présenté dans la Figure 4.2, permet de générer, à l'aide de la fonction *generate_packets()*, une liste exhaustive et figée contenant tous les types de paquets deux à deux différents pouvant être générés en fixant les sources et les destinations dans le réseau à contrôler.

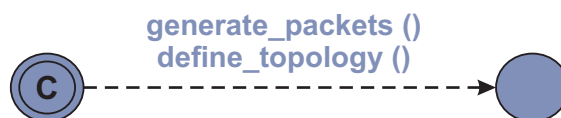


Figure 4.2 Automate "Nœud_générateur" dans le modèle de renforcement.

De plus, cet automate permet de fixer la topologie du réseau en fonction de sa taille et son taux de connectivité en utilisant la procédure *define_topology()*. Cette procédure crée initialement un réseau en anneau en fonction de la taille du réseau (nombre de nœuds). Ensuite, la procédure *define_topology()* calcule et ajoute les connexions, conformément au taux de connectivité correspondant. La transition qui relie les deux états de l'automate est une transition incontrôlable exécutée par l'environnement.

La procédure *define_topology()* permet aussi de configurer les pare-feux publics en déterminant les listes de contrôle d'accès correspondantes.

4.3.2 Automate "Nœud"

Chaque nœud dans le réseau est modélisé par un automate de type "Noeud" présenté dans la Figure 4.3. Cet automate décrit les échanges de paquets entre un nœud et ses voisins. Un paquet est envoyé à partir de son nœud courant vers un nœud voisin, choisi aléatoirement. Cette action d'envoi est modélisée par un arc non contrôlable. La réception d'un paquet envoyé dépend de la destination du paquet en question. Si la destination choisie fait partie de l'ensemble de nœuds publics, la procédure *public_update()* est appelée pour mettre à jour l'état du paquet en fonction de la décision des règles qui contrôlent le trafic entre l'emplacement courant et la prochaine destination publique du paquet en question. Cette action est incontrôlable. Si la destination choisie fait partie de l'ensemble de nœuds privés, la procédure *private_update()* est appelée pour mettre à jour l'état du paquet en fonction de la décision des règles choisies par l'option "select". C'est à ce niveau que le contrôleur a le pouvoir de choisir la règle à appliquer au paquet courant afin de forcer la politique de sécurité souhaitée du réseau.

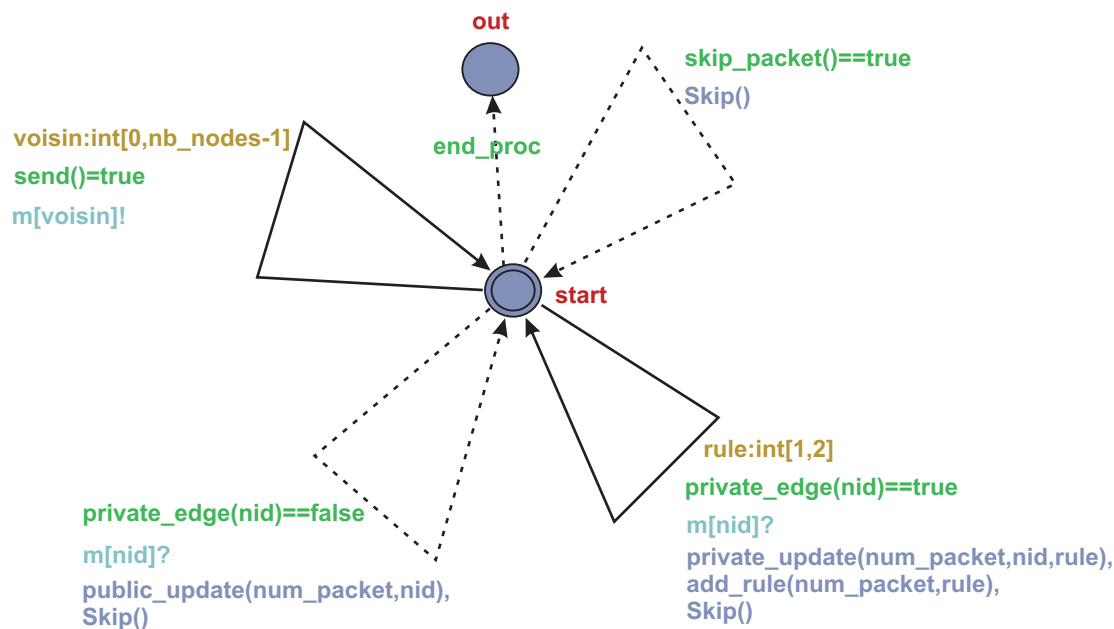


Figure 4.3 Automate Nœud.

4.3.3 Automate "Propriété"

L'automate *Propriet*, présenté dans la Figure 4.4, spécifie l'objectif global de sécurité dans un réseau. Nous supposons que la politique de sécurité à renforcer est de type *deny* qui permet de rejeter un ensemble de paquets bien déterminé. Par exemple, considérons l'objectif global de sécurité suivant :

***P* : rejeter tous les paquets dont l'adresse IP source est 128.0.0.0**

Dans ce cas, la fonction *prop()* prend en argument l'information 128.0.0.0, détermine l'ensemble de paquets visés puis vérifie s'ils satisfont la propriété (s'ils sont rejetés). La transition qui mène à l'état « Win » doit être contrôlable car nous voulons forcer la satisfaction de la propriété.

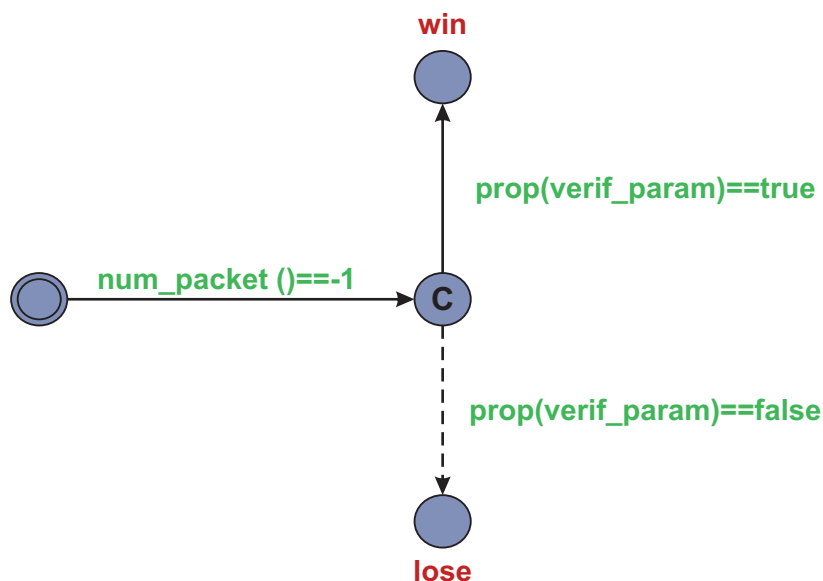


Figure 4.4 Automate "Propriété".

4.4 Configuration formelle des pare-feux

Dans la section précédente, nous avons décrit notre modèle qui permet de renforcer une politique de sécurité sur un réseau informatique. Dans cette section, nous montrons comment utiliser ce modèle pour configurer les pare-feux du réseau privé conformément à un objectif de sécurité global du réseau.

Pour forcer notre modèle à satisfaire l'objectif de contrôle global défini par l'automate Propriété, nous avons utilisé la propriété suivante :

Control : A <> Propriété.win

Notre modèle stocke l'ensemble de règles qu'il a défini pour renforcer la politique de sécurité, dans une structure de données appelée "rule". Cette structure de données est un tableau de taille égale au nombre de paquets (chaque règle décide par rapport à un paquet). Cependant, ces règles correspondent à l'ensemble de règles appliquées pour tous les paquets qui sont visés ou non par la propriété fixée au départ. En effet, un paquet qui est hors du domaine de la propriété envisagée peut être indifféremment accepté ou refusé par le pare-feu de sa destination en lui appliquant une règle de type *accept* ou *deny*.

Par exemple, si la politique de sécurité du réseau veut écarter tous les paquets qui viennent de la source 5, alors les règles associées à cet ensemble de paquets dans le tableau "rule" vont être toutes de type *deny* quelle que soit la stratégie. Par contre, la décision prise à l'égard d'un paquet qui vient par exemple de la source 4, en cas où il échappe du réseau public, peut être indifféremment *accept* ou *deny* et ne contredit d'aucune façon l'objectif de sécurité global du réseau. Par conséquent, l'ajout de cette règle dans la configuration du pare-feu privé responsable n'a aucune importance et ne fait qu'alourdir le coût de la configuration du pare-feu.

Notre but est de garder uniquement les règles nécessaires et suffisantes pour forcer le réseau à satisfaire la propriété. Pour ce faire, nous avons extrait toutes les stratégies gagnantes permettant de satisfaire la politique de sécurité en utilisant la ligne de commande *verifytga* avec l'option de vérification $-w_1$ et nous avons implémenté un algorithme d'extraction des règles en utilisant MATLAB. Cet algorithme permet i) d'analyser et de présenter sous forme lisible l'ensemble de stratégies gagnantes, ii) de réduire le nombre de stratégies en gardant que les stratégies non redondantes, et iii) de déterminer les règles nécessaires et suffisantes à configurer sur les pare-feux privés.

La Figure 4.5 présente la méthodologie de l'algorithme d'extraction des règles.

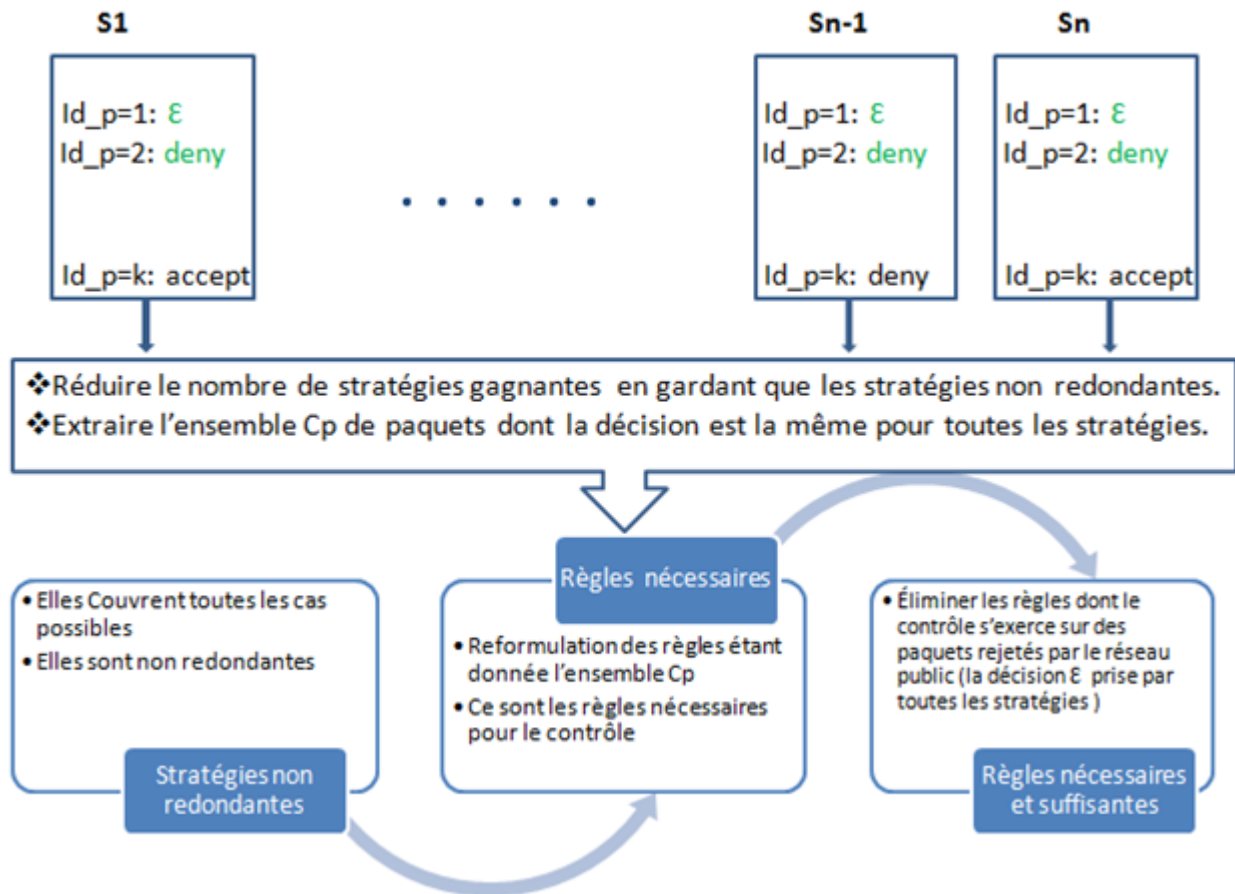


Figure 4.5 Méthodologie de l'algorithme d'extraction des règles.

L'utilisation de la commande en ligne *veriflytga* avec l'option d'extraction $-w_1$ extrait toutes les stratégies gagnantes possibles en explorant la totalité du graphe d'états. L'outil génère, alors, un très grand fichier contenant toutes les stratégies gagnantes permettant de satisfaire l'objectif de sécurité fixé ainsi que des informations redondantes sur la totalité de l'espace d'états (i.g. les valeurs de toutes les variables du système à chaque état et pour chaque processus). À titre illustratif, la Figure 4.6 présente une partie de ce fichier.

```

Microsoft windows [version 6.1.7601]
copyright (c) 2009 Microsoft Corporation. Tous droits r, serv, s.

C:\uppaal-tiga-0.16\bin-win32>
C:\uppaal-tiga-0.16\bin-win32>options for the verification:
Generating some trace
Search order is breadth first (UPPAAL), automatic (TIGA)
Using conservative space optimisation
Seed is 1384187428
State space representation uses minimal constraint systems
-[2K
Verifying property 1 at line 6
-- Throughput: 352 states/sec, Size: 12 states, Load: 0 states-[K -- Throughput: 12885 states/sec, Size: 856 states, Load: 542 stati
%[KPreparing: 29%-[KPreparing: 30%-[KPreparing: 31%-[KPreparing: 32%-[KPreparing: 33%-[KPreparing: 34%-[KPreparing: 35%-[KPreparing: 3
85%-[KPreparing: 86%-[KPreparing: 87%-[KPreparing: 88%-[KPreparing: 89%-[KPreparing: 90%-[KPreparing: 91%-[KPreparing: 92%-[KPreparing:
Note: The 'strategy' is not guaranteed to be a strategy.

strategy to win:

State: *( noeud_generateur._id2 propriety._id7 noeud(0).out noeud(1).out noeud(2).start noeud(3).start noeud(4).out noeud(5).start
eseau[2][6]=-1 reseau[3][0]=-1 reseau[3][1]=-1 reseau[3][2]=2 reseau[3][3]=-1 reseau[3][4]=-1 reseau[3][5]=-1 reseau[3][6]=-1 reseau
s_visites[1]=1 paquets[1].noeuds_visites[2]=0 paquets[1].noeuds_visites[3]=0 paquets[1].noeuds_visites[4]=0 paquets[1].noeuds_visite
=0 paquets[4].port=80 paquets[4].etat=1 paquets[4].NC=5 paquets[4].noeuds_visites[0]=6 paquets[4].noeuds_visites[1]=3 paquets[4].noe
paquets[6].nb_v=3 paquets[7].num_paquet=7 paquets[7].ip_src=6 paquets[7].ip_dest=5 paquets[7].protocol=1 paquets[7].port=25 paquets[
when you are in true, take transition propriety._id7->propriety.win { prop() == 1, tau, 1 }

State: ( noeud_generateur._id2 propriety._id7 noeud(0).start noeud(1).start noeud(2).out noeud(3).start noeud(4).out noeud(5).start
eseau[2][6]=-1 reseau[3][0]=-1 reseau[3][1]=-1 reseau[3][2]=2 reseau[3][3]=-1 reseau[3][4]=-1 reseau[3][5]=-1 reseau[3][6]=-1 reseau
s_visites[1]=1 paquets[1].noeuds_visites[2]=0 paquets[1].noeuds_visites[3]=0 paquets[1].noeuds_visites[4]=0 paquets[1].noeuds_visite
=0 paquets[4].port=80 paquets[4].etat=-1 paquets[4].NC=2 paquets[4].noeuds_visites[0]=6 paquets[4].noeuds_visites[1]=3 paquets[4].no
paquets[6].nb_v=4 paquets[7].num_paquet=7 paquets[7].ip_src=6 paquets[7].ip_dest=5 paquets[7].protocol=1 paquets[7].port=25 paquets[
when you are in true, take transition propriety._id7->propriety.win { prop() == 1, tau, 1 }

State: ( noeud_generateur._id2 propriety._id7 noeud(0).out noeud(1).out noeud(2).start noeud(3).start noeud(4).out noeud(5).out noe
eau[2][6]=-1 reseau[3][0]=-1 reseau[3][1]=-1 reseau[3][2]=2 reseau[3][3]=-1 reseau[3][4]=-1 reseau[3][5]=-1 reseau[3][6]=-1 reseau[4
visites[1]=1 paquets[1].noeuds_visites[2]=0 paquets[1].noeuds_visites[3]=0 paquets[1].noeuds_visites[4]=0 paquets[1].noeuds_visites[
paquets[4].port=80 paquets[4].etat=-1 paquets[4].NC=2 paquets[4].noeuds_visites[0]=6 paquets[4].noeuds_visites[1]=3 paquets[4].noeud
uets[6].nb_v=4 paquets[7].num_paquet=7 paquets[7].ip_src=6 paquets[7].ip_dest=5 paquets[7].protocol=1 paquets[7].port=25 paquets[7].
when you are in true, take transition propriety._id7->propriety.win { prop() == 1, tau, 1 }

State: ( noeud_generateur._id2 propriety._id7 noeud(0).out noeud(1).start noeud(2).start noeud(3).out noeud(4).start noeud(5).out no
eseau[2][6]=-1 reseau[3][0]=-1 reseau[3][1]=-1 reseau[3][2]=2 reseau[3][3]=-1 reseau[3][4]=-1 reseau[3][5]=-1 reseau[3][6]=-1 reseau
s_visites[1]=1 paquets[1].noeuds_visites[2]=0 paquets[1].noeuds_visites[3]=0 paquets[1].noeuds_visites[4]=0 paquets[1].noeuds_visite
=0 paquets[4].port=80 paquets[4].etat=1 paquets[4].NC=5 paquets[4].noeuds_visites[0]=6 paquets[4].noeuds_visites[1]=3 paquets[4].noe
paquets[6].nb_v=3 paquets[7].num_paquet=7 paquets[7].ip_src=6 paquets[7].ip_dest=5 paquets[7].protocol=1 paquets[7].port=25 paquets[
when you are in true, take transition propriety._id7->propriety.win { prop() == 1, tau, 1 }

State: ( noeud_generateur._id2 propriety._id7 noeud(0).start noeud(1).start noeud(2).out noeud(3).out noeud(4).start noeud(5).start
reseau[2][6]=-1 reseau[3][0]=-1 reseau[3][1]=-1 reseau[3][2]=2 reseau[3][3]=-1 reseau[3][4]=-1 reseau[3][5]=-1 reseau[3][6]=-1 reseau
ds_visites[1]=1 paquets[1].noeuds_visites[2]=0 paquets[1].noeuds_visites[3]=0 paquets[1].noeuds_visites[4]=0 paquets[1].noeuds_visite
l=0 paquets[4].port=80 paquets[4].etat=-1 paquets[4].NC=2 paquets[4].noeuds_visites[0]=6 paquets[4].noeuds_visites[1]=3 paquets[4].n
0 paquets[6].nb_v=4 paquets[7].num_paquet=7 paquets[7].ip_src=6 paquets[7].ip_dest=5 paquets[7].protocol=1 paquets[7].port=25 paquet
when you are in true, take transition propriety._id7->propriety.win { prop() == 1, tau, 1 }

```

Figure 4.6 Output du vérificateur UPPAAL-TIGA

L'algorithme MATLAB, que nous avons implémenté, permet, tout d'abord, d'analyser l'output du vérificateur en extrayant, dans un autre fichier, les informations pertinentes et non redondantes des stratégies gagnantes sous forme lisible. Chaque stratégie est alors représentée sous forme d'une combinaison de règles appliquées pour chaque paquet. La Figure 4.7 présente une partie du fichier généré par MATLAB contenant l'ensemble de stratégies gagnantes sous forme concise et lisible.

```
-----  
*Strategy 1  
-----
```

```
id_paquet=0 rules[0]= $\epsilon$   
id_paquet=1 rules[1]= $\epsilon$   
id_paquet=2 rules[2]= $\epsilon$   
id_paquet=3 rules[3]= $\epsilon$   
id_paquet=4 rules[4]= $\epsilon$   
id_paquet=5 rules[5]= $\epsilon$   
id_paquet=6 rules[6]= $\epsilon$   
id_paquet=7 rules[7]= $\epsilon$ 
```

```
-----  
*Strategy 2  
-----
```

```
id_paquet=0 rules[0]=deny  
id_paquet=1 rules[1]= $\epsilon$   
id_paquet=2 rules[2]= $\epsilon$   
id_paquet=3 rules[3]= $\epsilon$   
id_paquet=4 rules[4]=accept  
id_paquet=5 rules[5]= $\epsilon$   
id_paquet=6 rules[6]= $\epsilon$   
id_paquet=7 rules[7]=accept
```

```
-----  
*Strategy 3  
-----
```

```
id_paquet=0 rules[0]=deny  
id_paquet=1 rules[1]=deny  
id_paquet=2 rules[2]= $\epsilon$   
id_paquet=3 rules[3]=accept
```

Figure 4.7 Fichier des stratégies gagnantes.

Les règles sont établies en se basant sur trois différents types de décisions : i) la décision ϵ signifie que le paquet en question a été rejeté par le réseau public et le réseau privé n'a aucun contrôle dessus, ii) la décision *deny* implique que le paquet considéré doit être rejeté par l'un des pare-feux du réseau privé, et iii) la décision *accept* implique que le paquet considéré peut être accepté par l'un des pare-feux privés sans contrarier l'objectif de sécurité global du réseau.

Ensuite, l'algorithme réduit le nombre de stratégies en gardant que les stratégies non redondantes.

L'étape suivante consiste à extraire l'ensemble C_p composé par les types de paquets suivants : i) les paquets rejetés par toutes les stratégies (*deny* par tout), ii) les paquets acceptés par toutes les stratégies (*accept* par tout), iii) les paquets acceptés par certaines stratégies et ayant une décision ϵ pour d'autres, iv) les paquets rejetés par certaines stratégies et ayant une décision ϵ pour d'autres, et v) les paquets dont la décision est ϵ par tout (i.e. les paquets qui sont rejetés par le réseau public). Par conséquent, l'ensemble C_p regroupe l'ensemble de paquets dont la décision prise par le réseau privé est la même quelle que soit la stratégie adoptée. Ensuite, l'algorithme analyse les stratégies relatives à ce groupe de paquets en extrayant les règles nécessaires à la satisfaction de la propriété du réseau.

La dernière étape consiste à déterminer la configuration optimale (à moindre coût) des pare-feux du réseau privé. Cette configuration est déterminée en écartant toutes les règles dont le contrôle s'exerce sur des paquets visés par la politique de sécurité et rejetés par le réseau public (ceux ayant une décision ϵ prise par toutes les stratégies). L'ensemble de règles ainsi défini représente les règles nécessaires et suffisantes pour le renforcement de la politique de sécurité souhaitée.

La Figure 4.8 présente une illustration de toute la procédure d'extraction des règles. Les paquets 1, 2, 3, et 4 constituent le domaine de la politique de sécurité que nous voulons renforcer. Ces paquets admettent une décision uniforme par toutes les stratégies (i.e. un paquet appartenant à ce groupe ne peut pas être accepté par certaines stratégies et rejeté par d'autres).

Le paquet 5 fait partie du domaine de la politique de sécurité. Cependant, ce paquet est rejeté par le réseau public en empruntant l'importe quel chemin sur le réseau vers sa destination.

		Toutes les stratégies gagnantes possibles					
		S1	S2	S3	S4	S5	S6
Tous les paquets traités	Paquet1	deny	deny	deny	deny	deny	ε
	Paquet2	deny	deny	deny	deny	ε	ε
	Paquet3	deny	deny	ε	deny	deny	ε
	Paquet4	deny	ε	deny	deny	deny	ε
	Paquet5	ε	ε	ε	ε	ε	ε
	Paquet6	ε	deny	accept	deny	deny	accept
	Paquet7	deny	ε	accept	deny	deny	ε






	Dans domaine de la politique de sécurité
	Hors du domaine de la politique de sécurité
	Stratégie la plus optimale
	Stratégie la plus restrictive
	Paquet obsolète du domaine de la politique de sécurité

Figure 4.8 Illustration de la procédure d'extraction des règles.

Les deux paquets 6 et 7 sont tous les deux hors du domaine de la politique de sécurité parce qu'ils sont acceptés par certaines stratégies et rejetés par d'autres. Ceci signifie que ces paquets peuvent être indifféremment acceptés ou rejetés sans entraver l'objectif de sécurité.

La détermination des règles de configuration de pare-feux privés est alors basée sur l'ensemble de paquets (en vert) visés par la politique de sécurité. La stratégie S_4 est la plus restrictive puisqu'elle applique un *deny* sur tous les paquets. Par contre, la stratégie S_6 est la plus optimale puisqu'elle n'exige aucune configuration supplémentaire des pare-feux privés. Il suffit d'analyser les chemins empruntés par les paquets en adoptant cette stratégie et d'en extraire de l'information pertinente afin de mettre à jour les tables de routage dans le réseau.

Après le processus de renforcement et la configuration des pare-feux, le relancement du processus de vérification, en utilisant le modèle établi dans le chapitre 3, permet de vérifier la consistance de la configuration des pare-feux privés ainsi établie.

4.5 Etude de performance du modèle de renforcement

Dans cette section, nous discutons la performance de notre approche en termes du temps de calcul et de la quantité de mémoire consommée.

Nous rappelons que notre modèle est composé de n automates où n est le nombre de nœuds dans le réseau. Chaque automate a un seul état et quatre arcs : i) le premier arc est contrôlable et permet d'envoyer un paquet de son nœud courant à un autre nœud voisin choisi aléatoirement (et éventuellement en fonction des informations disponibles dans les tables de routage), ii) le deuxième arc est incontrôlable et permet de recevoir les paquets envoyés vers un nœud destination public, iii) le troisième arc est contrôlable et permet de recevoir les paquets envoyés vers un nœud destination privé. Le pare-feu responsable de cette destination sera configuré en ajoutant dans son *ACL* les règles nécessaires permettant de forcer le réseau à satisfaire sa politique de sécurité, et iv) le quatrième arc détermine le prochain paquet à traiter. La politique de sécurité du réseau est modélisée par l'automate Propriété.

L'étude expérimentale de la performance de notre approche a été réalisée afin d'analyser les limites de l'approche par rapport à la taille des réseaux qu'elle peut traiter. Nous avons déterminé le temps de calcul et la quantité de mémoire nécessaire pour vérifier la propriété suivante :

Control : A <> Propriété.win

où Propriété définit la politique de sécurité du réseau. Le temps de calcul et la mémoire consommée ont été calculés en fonction du nombre de nœuds et la connectivité du réseau donné. Les mesures ont été effectuées en utilisant l'outil UPPAAL TIGA 0.17 installée sur une machine fedora19 (Core i5 RAM 8GO). Nous avons utilisé, en conjonction avec UPPAAL

TIGA, l'utilitaire memtime 1.3 qui permet de déterminer le temps de vérification et le taux d'utilisation de mémoire.

La Figure 4.9 et la Figure 4.10 décrivent la performance modèle, respectivement, en termes de temps de vérification et la mémoire consommée, en fonction de la taille du réseau (nombre de nœuds et la connectivité).

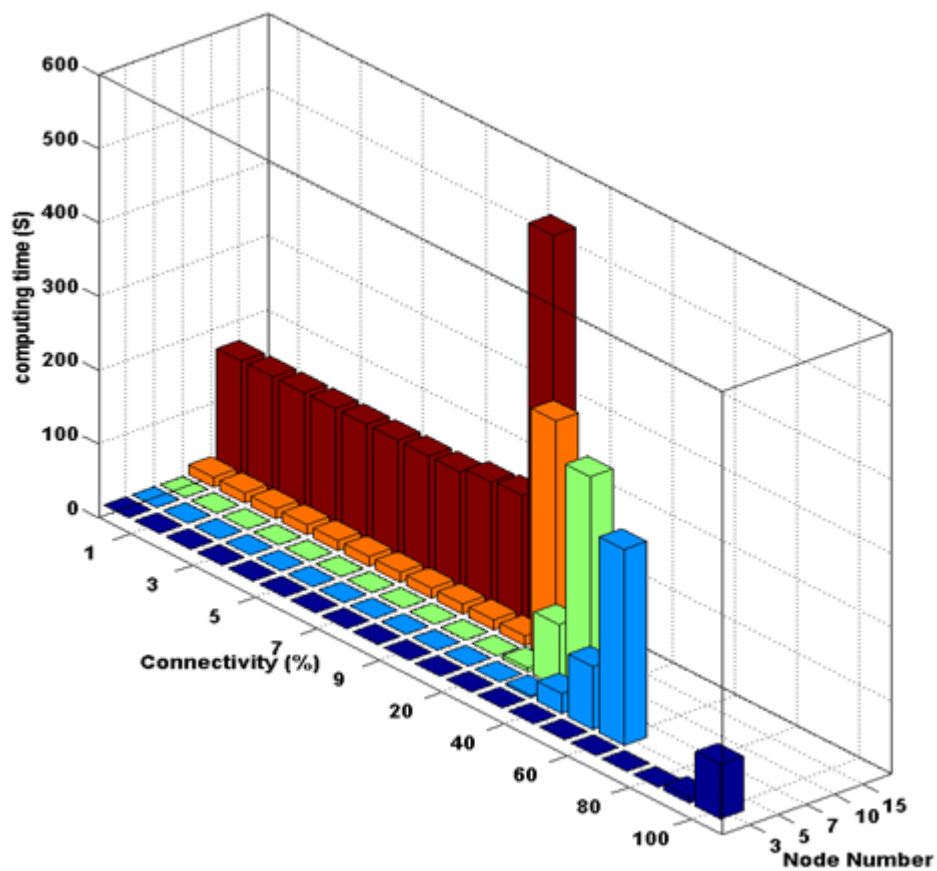


Figure 4.9 Temps de vérification de la propriété de renforcement.

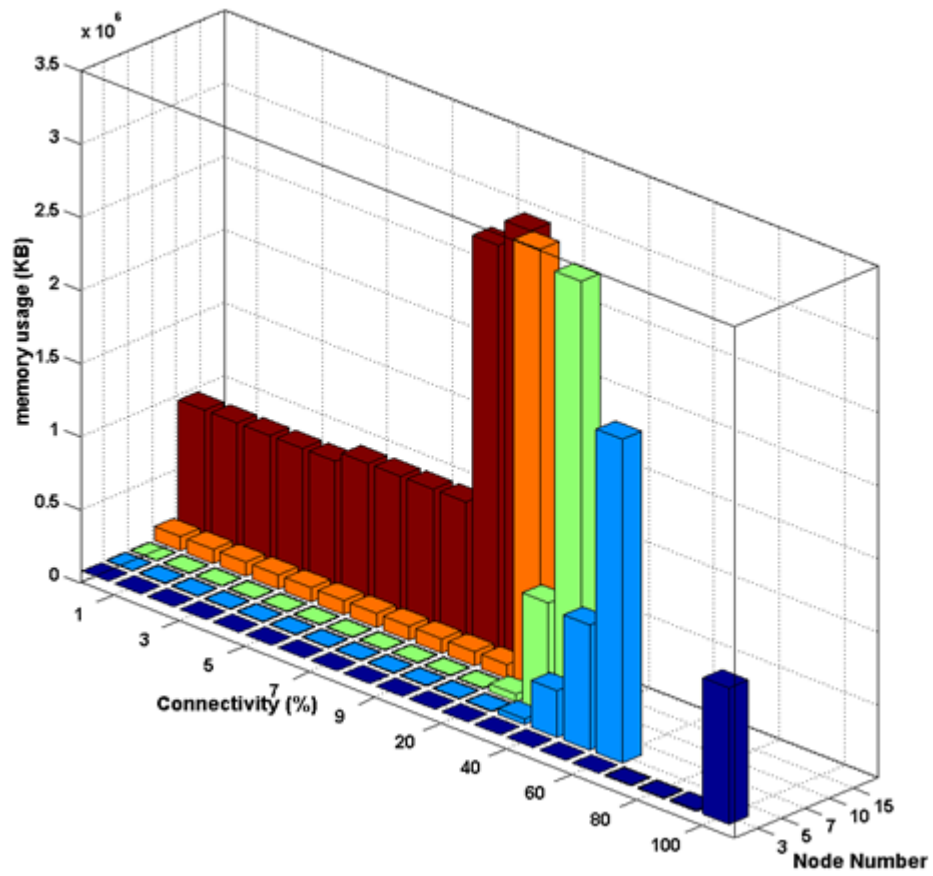


Figure 4.10 Taux d'utilisation de la mémoire pour la propriété de renforcement.

Les résultats expérimentaux montrent que le modèle est limité par le problème d'explosion combinatoire. En effet, les entrelacements entre les différents nœuds sont couplés avec le choix de règles à chaque envoi d'un paquet vers un nœud privé. Ceci a multiplié le nombre d'états à explorer. Par conséquent, notre modèle peut uniquement renforcer la sécurité sur des réseaux de petites tailles. Par exemple, Il peut renforcer la sécurité des réseaux composés de 3 nœuds fortement connectés (100% de connectivité) en 50 s avec un taux d'utilisation de mémoire voisin de 0.5 GB. Par ailleurs, le modèle peut traiter des réseaux à 5 nœuds mais avec un taux de connectivité moins élevé (uniquement 60%).

Au maximum, notre modèle peut traiter des réseaux à 15 nœuds avec un taux de connectivité de 10% en 500 s et en consommant 2.5 GB de mémoire. Ces valeurs sont relativement élevées par rapport à la taille des réseaux traités. Ceci montre que le problème d'explosion combinatoire limite la capacité de l'approche à gérer des grands réseaux avec un taux de connectivité relativement élevé. Dans la section suivante, nous présentons une nouvelle abstraction de notre approche faisant face à ce problème majeur.

4.6 Comment atténuer le problème d'explosion combinatoire

L'étude de performance, décrite dans la section précédente, montre que notre approche se heurte au problème d'explosion combinatoire. Pour contourner ce problème, nous avons adopté les mêmes abstractions prévues pour le modèle de vérification qualitatif basé sur l'évolution des nœuds présenté dans le chapitre 3. Par exemple, les valeurs correspondantes aux différents paramètres du modèle (i.g. le nombre de nœuds clients, le nombre de protocoles et le nombre de ports) ont été limités. La liste statique comportant tous les types de paquets non redondants a été gardée. Une structure de données contenant les informations de parcours liées à chaque paquet, a été de même utilisée. Nous avons, également, mis en œuvre un automate permettant de définir la topologie d'un réseau d'une façon statique pour éviter de traiter en concurrence plusieurs exemplaires d'un même réseau.

Cependant, toutes ces mesures et ces abstractions ne sont plus suffisantes pour notre modèle de renforcement. En effet, le choix de règles à chaque réception d'un paquet par un nœud privé, fait multiplier le nombre d'états à explorer. Par conséquent, nous avons proposé une autre modèle basé sur l'évolution des paquets.

Contrairement à la première spécification, ce modèle abstrait l'évolution des nœuds en se focalisant sur l'évolution de l'état des paquets d'un nœud à un autre.

Dans cette nouvelle abstraction, nous spécifions un réseau avec un seul automate représenté par la Figure 4.11.

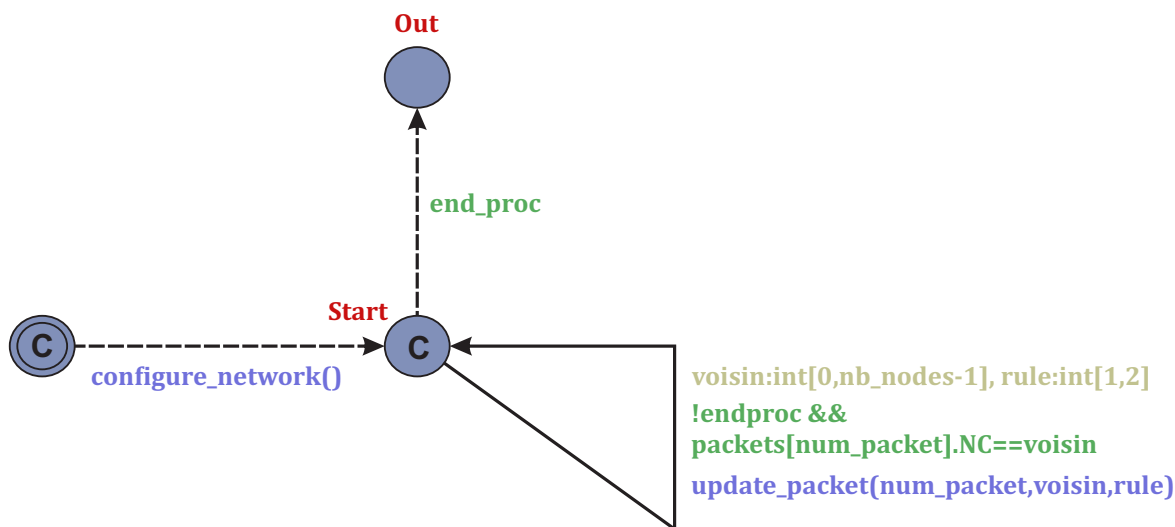


Figure 4.11 Automate "Réseau" pour le modèle de renforcement

Cet automate joue le rôle d'un contrôleur permettant, ainsi, de renforcer la politique de sécurité envisagée sur le réseau. La procédure *configure_network()* permet de spécifier la topologie du réseau dont nous voulons renforcer la sécurité et génère une liste exhaustive contenant tous les types de paquets non redondants. Cette procédure permet également de spécifier les listes de contrôle d'accès des pare-feux publics.

La prochaine destination du paquet est choisie aléatoirement en utilisant l'option de sélection. Le contrôleur choisit la règle à appliquer au paquet au niveau du réseau privé de façon à satisfaire la politique de sécurité générale du réseau. Ensuite, la procédure *update_packet()* est appelée pour mettre à jour le statut du paquet. Si la destination choisie est publique alors le statut du paquet sera mis à jour en fonction de la décision des règles du pare-feu responsable. Si, par contre, la destination choisie est privée alors le statut du paquet en question sera mis à jour en fonction de la décision prise par le contrôleur *accept* ou *deny*.

La Figure 4.12 et la Figure 4.13 montrent la performance du deuxième modèle, respectivement, en termes du temps de vérification et utilisation de la mémoire.

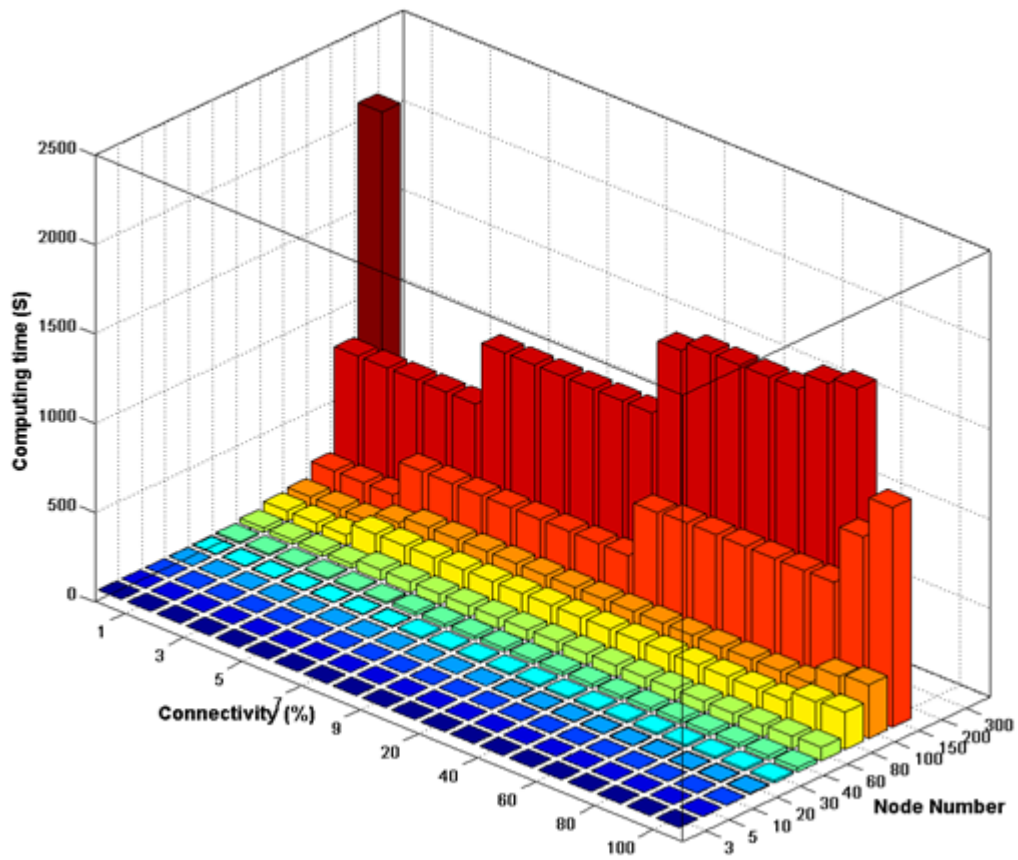


Figure 4.12 Temps de vérification pour le modèle de renforcement réduit.

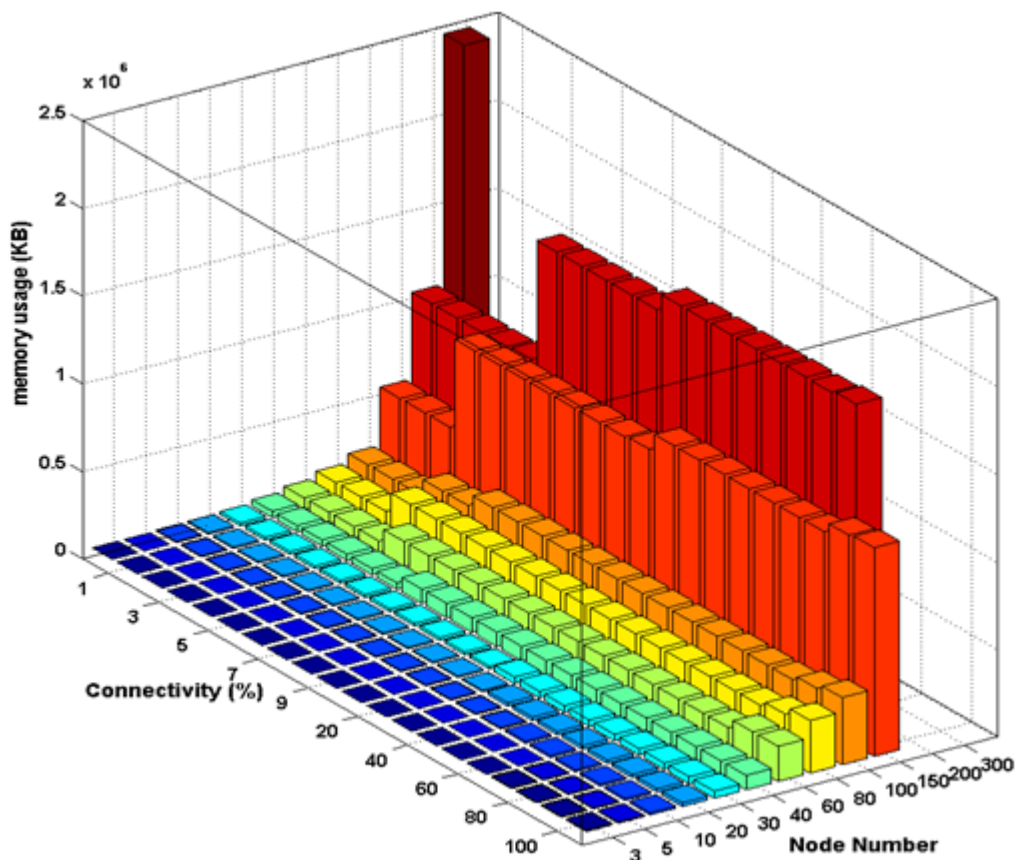


Figure 4.13 Taux d'utilisation de la mémoire pour le modèle de renforcement réduit.

Ces deux figures montrent que notre nouveau modèle basé sur l'évolution des paquets peut traiter des réseaux composés de 100 nœuds fortement connectés (i.e. taux de connectivité à 100%) en seulement 100 s avec un taux d'utilisation de mémoire d'environ de 300 MB.

De plus, notre modèle peut également traiter des réseaux de 150 nœuds fortement connectés (i.e. taux de connectivité à 100%), en 700 s en consommant environ 0.7 GB de mémoire.

Notre nouvelle approche peut aussi renforcer la sécurité des réseaux de 200 nœuds avec un taux de connectivité assez élevé (80%). Elle peut aller jusqu'aux 300 nœuds mais avec un taux de connectivité très faible.

En comparaison avec le premier modèle, nous constatons une nette diminution dans le taux de consommation de mémoire, ainsi que dans le temps de calcul nécessaire à la vérification des réseaux de petites tailles. En effet, nous remarquons que notre modèle permet de traiter des réseaux composés de 30 nœuds fortement connectés dans un délai assez réduit (quelques secondes) avec un taux d'utilisation de mémoire très faible.

L'abstraction de la communication entre les différents automates via les canaux de synchronisation en réduisant le nombre d'automates dans le modèle en un seul automate, a

permis de diminuer le nombre d'entrelacements de manière significative au cours du processus de vérification. De ce fait, l'espace d'états a été énormément réduit et le problème de l'explosion combinatoire a été grandement atténué.

4.7 Conclusion

Dans ce chapitre, nous proposons un modèle abstrait décrivant un réseau composé d'un ensemble de nœuds contrôlés par des pare-feux. Ce modèle permet de générer une configuration optimale des pare-feux privés satisfaisant la politique de sécurité envisagée. Notre solution formelle s'appuie sur la technique de synthèse de contrôleur.

Dans la première partie de ce chapitre, nous avons présenté notre modèle de renforcement. Ensuite, nous avons discuté la méthodologie d'extraction des règles en expliquant comment peut-on utiliser le modèle afin de déterminer les règles nécessaires et suffisantes à une configuration optimale du réseau.

Afin de discuter les limites de notre approche, nous avons élaboré une étude expérimentale de performance. Cette étude a montré que notre approche se heurte au problème d'explosion combinatoire en permettant uniquement de renforcer la sécurité des réseaux de petites tailles.

Pour remédier à ce problème majeur, nous avons proposé une nouvelle abstraction basée sur l'évolution des paquets. L'étude expérimentale de la performance de la nouvelle abstraction montre que cette dernière est beaucoup plus résistante au problème d'explosion combinatoire. En effet, elle permet de renforcer la sécurité sur des réseaux composés de 200 nœuds fortement connectés avec un temps de calcul et un taux de mémoire consommée raisonnables.

CONCLUSION ET RECOMMANDATIONS

Dans le présent mémoire, nous nous sommes intéressés à l'utilisation des méthodes formelles comme cadre solide d'aide à la configuration et à la validation des réseaux informatiques. Un réseau informatique est un ensemble d'équipements reliés entre eux pour offrir des services très diversifiés tels que la communication entre les différents utilisateurs du réseau et le partage des ressources.

L'un des défis majeurs des réseaux informatiques est d'assurer la sécurité des données. Plusieurs dispositifs, tels que les IDS et les pare-feux, ont été mis en œuvre pour contrôler les échanges d'information et détecter les intrusions ou menaces susceptibles d'entraver la sécurité des données.

Parmi les mécanismes de sécurité les plus largement déployés, ce mémoire s'est intéressé aux pare-feux et plus précisément à la conception d'approches formelles d'aide à la configuration de pare-feux. Dans cette perspective, trois approches formelles sont proposées :

La première approche permet d'analyser le comportement qualitatif d'un réseau. Elle s'appuie sur la technique de model checking de l'outil UPPAAL, pour vérifier formellement la consistance d'un réseau à pare-feux de bout en bout (end-to-end security behavior) vis-à-vis d'un objectif de sécurité du réseau. Elle permet aussi de détecter les incohérences de configuration des pare-feux distribués, notamment, l'incohérence de croisement de chemins. Pour atténuer le problème d'explosion combinatoire inhérent au model checking, nous avons proposé deux modèles spécifiant le comportement d'un réseau. Afin de valider l'impact de ces abstractions sur les performances du processus de vérification, nous avons élaboré une étude expérimentale qui montre en fonction de la taille et de la topologie du réseau, le temps et l'espace mémoire nécessaires à la vérification des propriétés. Les résultats expérimentaux montrent un gain significatif en temps et en utilisation mémoire. En effet, la deuxième abstraction permettent de vérifier des réseaux de plus grandes tailles que la première. Elle permet de vérifier des réseaux de 300 nœuds fortement connectée et voire des réseaux de 1800 nœuds avec un faible taux de connectivité. Cette contribution a été acceptée pour présentation à la 19ième édition de "IEEE Symposium On Computers and Communications" qui aura lieu du 23 au 26 juin 2014 à Madeira, Portugal (voir Moussa *et al.*, 2014).

La deuxième approche vise à étudier le comportement quantitatif d'un réseau (évaluation quantitative des performances du réseau). Elle est basée sur le model checking statistique et est implémentée en utilisant l'outil SMC UPPAAL. Nous avons proposé un modèle formel qui permet d'intégrer des paramètres de QoS (i.e. le délai d'acheminement des paquets de bout en bout, délai d'attente et le taux de perte). Par exemple, nous avons exprimé des propriétés

qui vérifient la permissivité et la restrictivité des réseaux et calculent le taux de perte de paquets dans un réseau.

La troisième approche permet d'automatiser la configuration des pare-feux en renforçant des politiques de sécurité sur le réseau. Elle s'appuie sur la technique de synthèse de contrôleur implémentée dans l'outil UPPAAL-TIGA. Elle consiste à représenter un réseau par un ensemble d'automates de jeu où le choix de règles de pare-feux, à appliquer aux paquets en transit, peut être forcé de manière à satisfaire un objectif de sécurité global du réseau. UPPAAL-TIGA permet de synthétiser les stratégies sûres permettant de renforcer la politique de sécurité du réseau. Afin de faire face au problème d'explosion combinatoire, nous avons également proposé deux abstractions et étudié leurs impacts sur les performances en temps et en espace mémoire. Cette étude a montré que ce problème a été grandement atténué avec la deuxième abstraction qui permet de vérifier des réseaux composés de jusqu'à 200 nœuds fortement connectés avec un temps de calcul et un taux d'utilisation de mémoire raisonnables.

Comme perspective, nous proposons d'améliorer les modèles proposés en prenant en considération des fonctions de routage permettant de guider l'acheminement des paquets dans le réseau. Une telle mesure permet d'automatiser la mise en œuvre des stratégies de renforcement qui sont générées lors de l'application de la troisième approche. Un autre aspect, qu'il serait intéressant de considérer, est la combinaison de nos approches avec celles qui traitent la consistance des règles intra-pare-feux. Nous proposons également d'appliquer nos approches sur des études de cas réels afin de ressortir l'aspect pratique de notre travail.

LISTE DES RÉFÉRENCES

- ABBES, T., BOUHOULA, A. et RUSINOWITCH, M. (2008). An inference system for detecting firewall filtering rules anomalies. *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 2122–2128.
- ABRIAL, J.-R. (2010). *Modeling in Event-B : system and software engineering*. Cambridge University Press.
- AL-SHAER, E. S. et HAMED, H. H. (2004). Modeling and management of firewall policies. *Network and Service Management, IEEE Transactions on*, 1, 2–10.
- ALTISEN, K. (2001). *Application de la synthèse de contrôleur à l’ordonnancement de systèmes temps-réel*. Thèse de doctorat, Grenoble, INPG.
- ALTISEN, K., BOUYER, P., CACHAT, T., CASSEZ, F., GARDEY, G. ET AL. (2005). Introduction au contrôle des systèmes temps-réel. *Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR’05)*. 367–380.
- ALTURKI, M. et MESEGUER, J. (2011). Pvesta : A parallel statistical model checking and quantitative analysis tool. *Algebra and Coalgebra in Computer Science*, Springer. 386–392.
- ALUR, R. (1999). Timed automata. *Computer Aided Verification*. Springer, 8–22.
- ALUR, R., COURCOUBETIS, C. et DILL, D. (1993). Model-checking in dense real-time. *Information and computation*, 104, 2–34.
- ARNOLD, A. (1990). Systèmes de transitions finis et sémantique des processus communicants. *TSI. Technique et science informatiques*, 9, 193–216.
- BAIER, C., KATOEN, J.-P. ET AL. (2008). *Principles of model checking*, vol. 26202649. MIT press Cambridge.
- BALLARINI, P., DJAFRI, H., DUFLOT, M., HADDAD, S. et PEKERGIN, N. (2011). Cosmos : a statistical model checker for the hybrid automata stochastic logic. *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*. IEEE, 143–144.
- BARTAL, Y., MAYER, A., NISSIM, K. et WOOL, A. (1999). Firmato : A novel firewall management toolkit. *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 17–31.

- BASAGIANNIS, S., PETRIDOU, S., ALEXIOU, N., PAPADIMITRIOU, G. et KATSAROS, P. (2011). Quantitative analysis of a certified e-mail protocol in mobile environments : A probabilistic model checking approach. *Computers & Security*, 30, 257–272.
- BATT, G., PAGE, M., CANTONE, I., GOESSLER, G., MONTEIRO, P. et DE JONG, H. (2010). Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. *Bioinformatics*, 26, i603–i610.
- BEHRMANN, G., COUGNARD, A., DAVID, A., FLEURY, E., LARSEN, K. G. et LIME, D. (2007). Uppaal-tiga : Time for playing games! *Computer Aided Verification*. Springer, 121–125.
- BEHRMANN, G., DAVID, A. et LARSEN, K. G. (2004). A tutorial on uppaal. *Formal methods for the design of real-time systems*, Springer. 200–236.
- BEHRMANN, G., HUNE, T. et VAANDRAGER, F. (2000). Distributing timed model checking—how the search order matters. *Computer aided verification*. Springer, 216–231.
- BEHRMANN, G., LARSEN, K. G. et RASMUSSEN, J. I. (2005). Priced timed automata : Algorithms and applications. *Formal Methods for Components and Objects*. Springer, 162–182.
- BEN YOUSSEF, N., BOUHOULA, A. et JACQUEMARD, F. (2009). Automatic verification of conformance of firewall configurations to security policies. *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*. IEEE, 526–531.
- BENGTSSON, J., LARSEN, K., LARSSON, F., PETTERSSON, P. et YI, W. (1996). *UPPAAL—a tool suite for automatic verification of real-time systems*. Springer.
- BÉRARD, B., BIDOIT, M., FINKEL, A., LAROUSSINIE, F., PETIT, A., PETRUCCI, L. et SCHNOEBELEN, P. (2010). *Systems and software verification : model-checking techniques and tools*. Springer Publishing Company, Incorporated.
- BIERMANN, E., CLOETE, E. et VENTER, L. M. (2001). A comparison of intrusion detection systems. *Computers & Security*, 20, 676–683.
- BILLAMI, H. et BENDAHMANE, R. (2014). *Etude d'un réseau optique ADM 10Gbit/s*. Thèse de doctorat.
- BRYAND, R. E. (2013). Binary decision digrams and beyond.

- BULYCHEV, P., DAVID, A., LARSEN, K. G., MIKUČIONIS, M. et LEGAY, A. (2011). Distributed parametric and statistical model checking. *arXiv preprint arXiv :1111.0370*.
- BULYCHEV, P., DAVID, A., LARSEN, K. G., MIKUČIONIS, M., POULSEN, D. B., LEGAY, A. et WANG, Z. (2012). Uppaal-smc : Statistical model checking for priced timed automata. *arXiv preprint arXiv :1207.1272*.
- CAPRETTA, V., STEPIEN, B., FELTY, A. et MATWIN, S. (2007). Formal correctness of conflict detection for firewalls. *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*. ACM, 22–30.
- CASSEZ, F. (2003). Vérification qualitative. *ETR septembre*, 91–104.
- CASSEZ, F., DAVID, A., FLEURY, E., LARSEN, K. G. et LIME, D. (2005). Efficient on-the-fly algorithms for the analysis of timed games. *CONCUR 2005–Concurrency Theory*, Springer. 66–80.
- CHAURE, R. (2010). An implementation of anomaly detection mechanism for centralized and distributed firewalls. *chance*, 7.
- CHESWICK, W. R., BELLOVIN, S. M. et RUBIN, A. D. (2003). *Firewalls and Internet security : repelling the wily hacker*. Addison-Wesley Longman Publishing Co., Inc.
- CIMATTI, A., CLARKE, E., GIUNCHIGLIA, E., GIUNCHIGLIA, F., PISTORE, M., ROVERI, M., SEBASTIANI, R. et TACCHELLA, A. (2002). Nusmv 2 : An opensource tool for symbolic model checking. *Computer Aided Verification*. Springer, 359–364.
- CLARKE, E. M., GRUMBERG, O. et PELED, D. (1999). *Model checking*. MIT press.
- CLARKE, E. M., KLIEBER, W., NOVÁČEK, M. et ZULIANI, P. (2012). Model checking and the state explosion problem. *Tools for Practical Software Verification*, Springer. 1–30.
- COOK, S. A. (1971). The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 151–158.
- DAGORN, N. (2006). Détection et prévention d'intrusion : présentation et limites.
- DANG, Z. et KEMMERER, R. A. (1997). Using the astral model checker for cryptographic protocol analysis. *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*. Citeseer.

DAVID, A., LARSEN, K. G., LEGAY, A., MIKUČIONIS, M. et WANG, Z. (2011). Time for statistical model checking of real-time systems. *Computer Aided Verification*. Springer, 349–355.

DENNING, D. E. (1987). An intrusion-detection model. *Software Engineering, IEEE Transactions on*, 222–232.

DINGEL, J. et FILKORN, T. (1995). Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving. *Computer Aided Verification*. Springer, 54–69.

EL-ATAWY, A. et SAMAK, T. (2012). End-to-end verification of qos policies. *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 426–434.

ERONEN, P. et ZITTING, J. (2001). An expert system for analyzing firewall rules. *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*. 100–107.

FALL, M. N. (2010). *Sécurisation formelle et optimisée de réseaux informatiques*. Thèse de doctorat, Université Laval.

GARCIA-ALFARO, J., CUPPENS, F., CUPPENS-BOULAHIA, N., MARTINEZ, S. et CABOT, J. (2013). Management of stateful firewall misconfiguration. *Computers & Security*, 39, 64–85.

GARCIA-ALFARO, J., CUPPENS, F., CUPPENS-BOULAHIA, N. et PREDA, S. (2011). Mirage : a management tool for the analysis and deployment of network security policies. *Data Privacy Management and Autonomous Spontaneous Security*, Springer. 203–215.

GAWANMEH, A. et TAHAR, S. (2011). Modeling and verification of firewall configurations using domain restriction method. *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*. IEEE, 642–647.

GLEICHAUF, B. et TEAL, D. (1996). Netranger high-level overview version 1.1. *Wheel-Group Corp., Nov*.

GOUDA, M. G. et LIU, X.-Y. (2004). Firewall design : Consistency, completeness, and compactness. *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*. IEEE, 320–327.

GREEN, C. et ROESCH, M. (2003). The snort project : version 2.1. 0. *User Manual, www.snort.org*.

- GUEFFAZ, M. (2012). *ScaleSem : model checking et web sémantique*. Thèse de doctorat, Université de Bourgogne.
- GUTTMAN, J. D. (1997). Filtering postures : Local enforcement for global policies. *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE, 120–129.
- HALLE, S., NGOUPE, E. L., VILLEMAIRE, R. et CHERKAOUI, O. (2013). Distributed firewall anomaly detection through ltl model checking. *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 194–201.
- HENZINGER, T. A., HO, P.-H. et WONG-TOI, H. (1997). Hytech : A model checker for hybrid systems. *Computer aided verification*. Springer, 460–463.
- HOLZMANN, G. J. (2004). *The SPIN model checker : Primer and reference manual*, vol. 1003. Addison-Wesley Reading.
- JEFFREY, A. et SAMAK, T. (2009). Model checking firewall policy configurations. *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*. IEEE, 60–67.
- KATOEN, J.-P., ZAPREEV, I. S., HAHN, E. M., HERMANNNS, H. et JANSEN, D. N. (2011). The ins and outs of the probabilistic model checker mrmc. *Performance evaluation*, 68, 90–104.
- KHORCHANI, B., HALLÉ, S. et VILLEMAIRE, R. (2012). Firewall anomaly detection with a model checker for visibility logic. *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 466–469.
- KIM, G. H. et SPAFFORD, E. H. (1994). The design and implementation of tripwire : A file system integrity checker. *Proceedings of the 2nd ACM Conference on Computer and Communications Security*. ACM, 18–29.
- KOTENKO, I. et POLUBELOVA, O. (2011). Verification of security policy filtering rules by model checking. *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*. IEEE, vol. 2, 706–710.
- KUGLER, H., HAREL, D., PNUELI, A., LU, Y. et BONTEMPS, Y. (2005). Temporal logic for scenario-based specifications. *Tools and Algorithms for the Construction and Analysis of Systems*, Springer. 445–460.
- KWIATKOWSKA, M., NORMAN, G. et PARKER, D. (2011). Prism 4.0 : Verification of probabilistic real-time systems. *Computer Aided Verification*. Springer, 585–591.

- LARSEN, K. G., PETTERSSON, P. et YI, W. (1995). Compositional and symbolic model-checking of real-time systems. *Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE*. IEEE, 76–87.
- LIU, A. X. (2008). Formal verification of firewall policies. *Communications, 2008. ICC'08. IEEE International Conference on*. IEEE, 1494–1498.
- LIU, A. X. et GOUDA, M. G. (2009). Firewall policy queries. *Parallel and Distributed Systems, IEEE Transactions on*, 20, 766–777.
- MATOUSEK, P., RÁB, J., RYSAVY, O. et SVÉDA, M. (2008). A formal model for network-wide security analysis. *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*. IEEE, 171–181.
- MAYER, A., WOOL, A. et ZISKIND, E. (2000). Fang : A firewall analysis engine. *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 177–187.
- MÉ, L. et MICHEL, C. (2001). Intrusion detection : A bibliography. *Supélec, Rennes, France, Tech. Rep. SSIR-2001-01*.
- MECHRI, T. (2007). *Approche algébrique pour la sécurisation des réseaux informatiques*. Thèse de doctorat, Université Laval.
- MERZ, F., FALKE, S. et SINZ, C. (2012). Llbmc : Bounded model checking of c and c++ programs using a compiler ir. *Verified Software : Theories, Tools, Experiments*, Springer. 146–161.
- MOUSSA, M., OULD-SLIMANE, H., BOUCHENEB, H. et CHAMBERLAND, S. (2014). A formal framework for verifying inter-firewalls consistency. *theNineteenth IEEE Symposium on Computers and Communication (ISCC)*.
- NELSON, T., BARRATT, C., DOUGHERTY, D. J., FISLER, K. et KRISHNAMURTHI, S. (2010). The margrave tool for firewall analysis. *USENIX Large Installation System Administration Conference*.
- NORMAN, G., PARKER, D. et SPROSTON, J. (2013). Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43, 164–190.
- NOUALI-TABOUDJEMAT, M. N. (1999). Les firewalls comme solution aux problèmes de sécurité. *Revue d'Information Scientifique et Technique (Rist)*. CERIST, vol. 9, 01–12.

- PARREAUX, B. (2000). Vérification de systèmes d'événements b par model-checking pltl. *These de Doctorat, LIFC, Université de Franche-Comté*, 8.
- PUJOLLE, G. et SALVATORI, O. (2010). *Les réseaux*. Editions Eyrolles.
- RANUM, M. J. (2001). Experiences benchmarking intrusion detection systems. *NFR Security White Paper*.
- ROBERT, C. et CASELLA, G. (2011). *Méthodes de Monte-Carlo avec R*. Springer.
- ROBLÈS, B., AVILA, M., DUCULTY, F., VRIGNAT, P., KRATZ, F. *ET AL.* (2010). Evaluation de la pertinence des paramètres stochastiques sur des modèles de markov cachés.
- ROZIER, K. Y. (2011). Linear temporal logic symbolic model checking. *Computer Science Review*, 5, 163–203.
- SEN, K., VISWANATHAN, M. et AGHA, G. A. (2005). Vesta : A statistical model-checker and analyzer for probabilistic systems. *QEST*. vol. 5, 251–252.
- TOM, T. (2005). *La sécurité réseaux first step*. Pearson Education France.
- VARDI, M. Y. (1996). An automata-theoretic approach to linear temporal logic. *Logics for concurrency*, Springer. 238–266.
- VERMA, P. et PRAKASH, A. (2005). Face : A firewall analysis and configuration engine. *Applications and the Internet, 2005. Proceedings. The 2005 Symposium on*. IEEE, 74–81.
- VINCENT, A. (2001). Synthèse de contrôleurs et stratégies gagnantes dans les jeux de parité. *MSR2001*.
- WHITMAN, M. et MATTORD, H. (2011). *Principles of information security*. Cengage Learning.
- YOUNES, H. L. (2005). Verification and planning for stochastic processes with asynchronous events. Rapport technique, DTIC Document.
- YOUNG, L. J. et YOUNG, J. H. (1998). Sequential hypothesis testing. *Statistical Ecology*, Springer. 153–190.
- YOVINE, S. (1997). Kronos : A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1, 123–133.

YUAN, L., CHEN, H., MAI, J., CHUAH, C.-N., SU, Z. et MOHAPATRA, P. (2006). Fireman : A toolkit for firewall modeling and analysis. *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 15–pp.