
Algèbre des Processus pour l'Analyse des Performances des Nœuds Actifs

L. Kloul — A. Mokhtari

PRiSM

Université de Versailles

45, Av. des Etats-Unis

78035 Versailles Cedex, France

{kle,amok}@prism.uvsq.fr

RÉSUMÉ. Nous étudions l'impact de l'introduction de la gestion active dans un réseau sur les paquets traditionnels traversant les nœuds actifs le constituant. Dans le cadre de l'infrastructure ARFANet, deux configurations de l'architecture d'un nœud actif sont considérées. Dans chaque cas, nous comparons les performances du nœud actif à celles d'un nœud traditionnel en utilisant le formalisme d'algèbre des processus PEPA. Nous nous intéressons, en particuliers, à des mesures de performance telles que les taux de perte des paquets standards et la latence du nœud pour ce même type de paquets.

ABSTRACT. We investigate the impact of introducing active networking on traditional packets flowing through the active nodes of a network. In the context of the ARFANet infrastructure, we consider two configurations of the active node architecture. For each configuration, we compare the performance of an active node and of a traditional one, using the process algebra formalism PEPA. We are mainly interested in performance measures such as loss rates of standard packets and the node latency for this kind of packets.

MOTS-CLÉS : Réseaux actifs, Règles actives, Algèbre des processus, PEPA, Taux de pertes, Latence.

KEYWORDS: Active networks, Actives Rules, Process Algebra, PEPA, Loss rates, Latency.

1. Introduction

Dans un réseau actif, les traitements à effectuer dans les différents nœuds peuvent être adaptés aux besoins de l'utilisateur et/ou de l'application elle-même. Les fonctionnalités spécifiques à un utilisateur ou à une application sont intégrées aux nœuds du réseau (commutateurs programmables) ou dans les paquets les traversant (des capsules), sous forme de méthodes ou de petits programmes. Dans la première approche, l'utilisateur envoie d'abord ses paquets de programmes dans certains nœuds du réseau. Lorsqu'un paquet de données de l'utilisateur arrive aux nœuds spécifiés, son en-tête est évalué et le programme approprié est exécuté sur le contenu du paquet. Dans la deuxième approche, chaque paquet ou capsule se compose d'un petit programme qui est transmis dans le flux de données (in-band) et exécuté dans chaque nœud traversé par le paquet.

Bien que ces approches semblent différentes, le concept de réseau actif utilisé derrière est fondamentalement le même. Dans les deux approches, le contenu des paquets est extrait et expédié à un environnement pour être exécuté immédiatement (approche des capsules) ou au bon moment (approche commutateurs programmables). Ceci suggère que les mécanismes et les primitives impliqués dans ces opérations sont non seulement indépendants de l'approche utilisée mais également de l'application elle-même.

Le besoin de développer un modèle de programmation commun comprenant des modèles communs de codage des programmes réseau, des primitives intégrées disponibles dans chaque nœud et l'attribution des ressources du nœud a montré la nécessité d'une architecture commune capable d'accepter différents langages de paquets et d'environnements d'exécution. L'idée d'une architecture basée principalement sur trois couches a été suggérée dans [CAL 98] et depuis utilisée dans différents travaux [HAR 99][MER 00][TUL 01]. Ces couches sont la couche des *applications actives* qui comporte les services d'applications pour les données utilisateurs et la gestion du réseau, la couche des *environnements d'exécution* qui interprète les paquets actifs et exécute les applications actives, et finalement le *système d'exploitation* du nœud qui gère plusieurs types d'environnements d'exécution et l'attribution des ressources du nœud à ces environnements.

Dans cet article, nous nous intéressons aux performances des nœuds actifs dans le cadre de l'infrastructure ARFANet dont l'architecture à couches [BOU 02][HIL 04] est similaire à celle développée dans [CAL 98]. Cette infrastructure est basée sur la notion de règles actives communément utilisées dans le domaine des bases de données actives. Dans un réseau actif, le traitement à effectuer dans les nœuds peut être adapté aux besoins de l'utilisateur et/ou de l'application elle-même. Les règles actives fournissent une sémantique explicite qui facilite le raisonnement sur le comportement des applications et le contrôle de leur exécution. Elles nous permettent de décrire un système ou le comportement d'une application avec une granularité très fine qui peut facilement évoluer en même temps que l'application. Dans ce contexte, une application peut être décrite comme un ensemble de règles actives, où chaque règle est définie

sous la forme d'un tuple Évènement-Condition-Action (ECA). L'exécution de l'application consiste alors à détecter les évènements, évaluer les conditions et lancer les actions correspondantes.

En utilisant le formalisme de l'algèbre des processus PEPA, nous comparons les performances d'un nœud "passif" ou standard dont la seule fonctionnalité est de retransmettre les paquets le traversant avec les performances d'un nœud actif ARFANet.

PEPA (Performance Evaluation Process Algebra) est un formalisme, développé par Hillston en 1994[HIL 94], qui étend l'algèbre des processus classiques en associant, à chaque action, une variable aléatoire, représentant la durée. Ces variables aléatoires sont supposées être exponentiellement distribuées, ce qui établit clairement une relation entre le modèle de l'algèbre des processus et un processus de Markov en temps continu. Via ce processus de Markov, des mesures de performances peuvent être extraites à partir du modèle.

L'un des attraits de ce formalisme est sa claire structure compositionnelle qui permet de construire un modèle à partir d'éléments ou composantes qui reflètent la composition du système à modéliser. Ceci permet, d'une part, une meilleure appréhension de la complexité des modèles à construire et d'autre part, une meilleure exploitation des propriétés structurelles lors de la résolution du modèle.

Par ailleurs, PEPA est une approche formelle qui comprend un *calculus*, basé sur des relations d'équivalence formellement définies, pour la manipulation et l'analyse des modèles. Celui-ci comprend une technique de simplification des modèles qui exploite l'équivalence des comportements pour partitionner l'espace d'états, c'est la technique d'*agrégation* [HIL 94]. Pour répondre au problème de l'explosion de l'espace d'états, une seconde solution a été développée. Elle consiste à faire un mapping d'un modèle PEPA vers une représentation compacte du générateur associé à la chaîne de Markov sous-jacente [HIL 02]. Cette technique est basée sur l'utilisation de l'algèbre de Kronecker. Ces deux techniques ont été intégrées à l'outil PEPA Workbench [GIL 94], offrant ainsi à l'utilisateur la possibilité de choisir la technique optimale pour son modèle et même la possibilité de combiner de manière appropriée ces techniques.

Bien que l'on retrouve l'aspect compositionnel ainsi que les techniques d'agrégation et/ou de représentation tensorielles dans d'autres formalismes tels que les réseaux d'automates stochastiques (RAS) de Plateau [PLA 85][BEN 03], les réseaux de Petri stochastiques superposés (Superposed GSPN) de Donatelli [DON 94] ou les réseaux d'activités stochastiques (Stochastic Activity Networks) de Sanders [SAN 91], PEPA a en plus le mérite d'être très simple à utiliser et, comme nous le verrons plus loin, il permet d'avoir une description intuitive du système à modéliser grâce à son petit, mais néanmoins riche, ensemble d'opérateurs.

De plus, plusieurs autres outils utilisent PEPA comme formalisme de description. Ainsi, il est l'un des langages en entrée de, entre autres :

- l'outil de modélisation *Möbius* développé par le groupe PERFORM [CLA 01], qui grâce à sa technique de sauvegarde sur disque permet de repousser les limites de

la modélisation en termes d'espace d'états.

- *PRISM model checker* [KWI 02] qui permet l'analyse des systèmes stochastiques et probabilistes en offrant à l'utilisateur la possibilité de vérifier une propriété logique pour un modèle donné. Plusieurs types de logiques peuvent être considérés dont CSL [AZI 96] pour les chaînes de Markov à temps continu.

Dans ce travail, nous considérons deux configurations de l'architecture du nœud actif et dans chaque cas, nous comparons ses performances à celle d'un nœud standard. De plus, les résultats des deux modèles analytiques sont comparés aux mesures effectuées sur la plateforme ARFANet. Nous sommes principalement intéressés par des mesures de performances telles que les taux de perte des paquets standards, le débit et la latence du nœud pour ce type de paquets.

Le papier est structuré comme suit. Dans la section 2 nous présentons le formalisme PEPA. La section 3 est consacrée à la présentation de l'infrastructure ARFANet et de la première configuration considérée. Dans la section 4, nous décrivons le modèle PEPA développé pour cette configuration. Les critères de performance qui nous intéressent sont définis dans la section 5. Les résultats numériques sont présentés et discutés dans la section 6. Dans un souci d'améliorer les résultats obtenus, dans la section 7 une seconde configuration est considérée, son modèle PEPA correspondant présenté et ses performances évaluées. Enfin, nos conclusions sont présentées dans la section 8.

2. Le formalisme PEPA

2.1. La syntaxe de PEPA

Un modèle PEPA est décrit comme l'interaction d'un ensemble de *composantes*. Chaque composante peut exécuter un ensemble d'actions : une action $a \in \mathcal{A}ct$ est décrite comme une paire (α, r) , où $\alpha \in \mathcal{A}$ est le *type* de l'action et $r \in \mathbb{R}^+$ est le paramètre de la distribution exponentielle négative régissant sa durée.

PEPA dispose d'un ensemble réduit, mais suffisamment puissant, d'opérateurs pour modéliser un comportement complexe à partir de comportements simples. Ces opérateurs sont ceux de l'algèbre des processus classiques : le préfixe, le choix, la composition parallèle et l'abstraction. Dans ce qui suit, nous expliquons brièvement chacun de ces opérateurs. La sémantique opérationnelle formelle de PEPA est décrite dans l'Annexe A.

Préfixe : Une composante peut avoir un comportement purement séquentiel, entreprenant à plusieurs reprises une action après une autre, retournant éventuellement par la suite au début de son comportement. Dans ce cas, l'opérateur préfixe, noté “.”, est utilisé pour indiquer la première action à effectuer. Par exemple, $(\alpha, r).p$ effectuera une activité de type α avec une durée moyenne de $1/r$, puis se comportera comme la composante P . Dans certains cas, le taux d'une action est en dehors du contrôle d'une composante. Celle-ci joue alors un rôle passif et le taux de l'activité pour cette com-

posante est dénoté par le symbole \top (appelé “ top ”). Une telle action est effectuée en coopération avec une autre composante qui elle contrôle le taux.

Choix : Un choix entre deux comportements possibles est représenté comme la somme des différentes possibilités, par exemple, $(\alpha, r).P + (\beta, s).Q$. La condition de compétition (*race condition*) est supposée gouverner le comportement des actions simultanément déclenchables. Ainsi cet opérateur représente un choix préemptif avec re-échantillonnage. La nature continue des distributions de probabilités garantit que les actions ne peuvent pas se produire simultanément. Quand une action a plus d’un résultat possible, elle est représentée par un choix de différentes actions, une pour chaque résultat possible. Les taux de ces actions sont choisis de manière à refléter leurs probabilités relatives.

Composition parallèle : La composition parallèle est utilisée pour représenter les cas où deux composantes du système *coopèrent* pour réaliser une certaine action. Par exemple, $P \bowtie_L Q$ se compose de deux composantes P et Q qui doivent coopérer pour réaliser les actions qui sont dans l’ensemble de coopération L . Les actions dont le type n’est pas dans cet ensemble peuvent être effectuées de manière indépendante et concurrente par les deux composantes. Les actions dans L exigent la participation simultanée des deux composantes. L’action résultante ou l’action partagée aura le même type que les deux actions de la contribution et un taux reflétant le taux de l’action de la composante la plus lente des deux. Notez que cela signifie que le taux d’une action passive deviendra le taux de l’action avec laquelle elle coopère.

Abstraction : Il est souvent commode de cacher quelques actions, les rendant privées à la composante ou aux composantes impliquées. La durée des actions est inchangée, mais leur type devient caché, apparaissant à la place comme le type inconnu τ . Les composantes ne peuvent pas synchroniser ou coopérer sur τ . L’utilisation de l’opérateur d’abstraction a deux implications. Premièrement, elle permet d’éviter qu’une composante ajoutée ultérieurement au modèle puisse interagir ou interférer avec une action privée. Deuxièmement, les actions privées ne peuvent contribuer au calcul des mesures, ce qui, par conséquent, suggère des simplifications du modèle.

En utilisant des constantes pour nommer les composantes et des définitions récursives nous pouvons décrire des composantes avec des comportements infinis sans utiliser un opérateur explicite de récursion.

Exemple : On considère un simple système composé d’un serveur et d’une file d’attente de capacité finie N . Le nombre total de clients dans le système ne peut pas dépasser N . On suppose que les arrivées et les services sont exponentiellement distribués de paramètres λ et μ , respectivement.

Le système peut être représenté comme l'interaction de deux composantes *Serveur* et *Buffer*.

$$Serveur \stackrel{\text{def}}{=} (service, \mu).Serveur$$

$$Buffer_0 \stackrel{\text{def}}{=} (arrive, \lambda).Buffer_1$$

$$Buffer_i \stackrel{\text{def}}{=} (arrive, \lambda).Buffer_{i+1} + (service, \top).Buffer_{i-1} \quad 0 < i < N$$

$$Buffer_N \stackrel{\text{def}}{=} (service, \top).Buffer_{N-1}$$

La composante du modèle *Système* est :

$$\text{Système}_0 \stackrel{\text{def}}{=} Serveur \underset{\{service\}}{\boxtimes} Buffer_0$$

Remarque : Contrairement à des formalismes de l'algèbre de processus tel que μ CRL, PEPA n'est pas un formalisme paramétré. Tous les paramètres du modèle doivent être connus au départ. Ainsi dans l'exemple ci-dessus, la valeur de N doit être fixée lors de la description du modèle. Cependant, une composante peut avoir des activités dont le taux est défini comme une fonction de l'état d'une ou de plusieurs autres composantes du modèle [HIL 02].

2.2. Le processus de Markov associé

Dans un modèle PEPA, si une composante P exécute une activité (α, r) puis se comporte comme P' , alors celle-ci est appelée *composante dérivée* de P .

A partir de n'importe quelle composante PEPA P , on peut construire de manière récursive l'ensemble des dérivées de P , noté $ds(P)$. Cet ensemble décrit l'ensemble des dérivées (comportements) possible pouvant résulter de P .

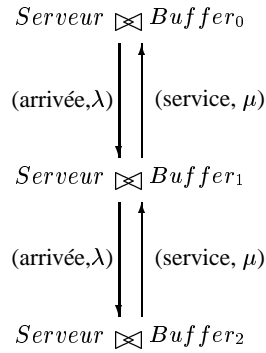
A partir de l'ensemble des dérivées de la composante du modèle $ds(C)$, on peut construire le *graphe de dérivation* qui est un multigraphe dirigé dont l'ensemble des noeuds est $ds(C)$ et dont les arcs sont les transitions possibles entre ces noeuds.

Le processus de Markov associé à un modèle PEPA est un processus de Markov à temps continu. La génération de ce processus est basée sur le graphe de dérivation du modèle. Un état est associé à chaque noeud du graphe et les transitions entre les états sont obtenues à partir de celles du graphe. Le taux de la transition entre deux états de la chaîne de Markov est la somme des taux de l'activité sur les arcs entre les deux noeuds correspondant du graphe.

Exemple : Considérons le modèle PEPA décrit précédemment (section 2.1) et dont la composante est :

$$\text{Système}_0 \stackrel{\text{def}}{=} Serveur \underset{\{service\}}{\boxtimes} Buffer_0$$

Si on suppose que $N = 2$, l'ensemble de dérivation de la composante du modèle est $ds(\mathbf{Système}_0) = \{Serveur \underset{\{service\}}{\boxtimes} Buffer_j, 0 \leq j \leq 2\}$ et le graphe de dérivation est le suivant :



La chaîne de Markov sous-jacente au modèle (figure 1) est générée à partir du graphe de dérivation. Il est clair que $j = 0, \dots, 2$ représente évidemment le nombre de clients dans le système.

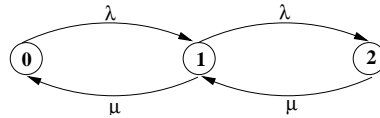


Figure 1. Chaîne de Markov

2.3. Résolution numérique

Un outil expérimental appelé PEPA Workbench [GIL 94] a été développé pour le formalisme PEPA. A partir de la description d'un modèle PEPA, cet outil fournit le comportement stationnaire du système modélisé en calculant la distribution stationnaire (vecteur de probabilités). Plusieurs techniques de résolution ont été intégrées à cet outil, parmi lesquelles on trouve les deux méthodes itératives *sor* (successive over-relaxation) et *linear biconjugate gradient*.

Par exemple, la résolution de la chaîne de Markov (figure1) associée à notre modèle PEPA consiste à résoudre l'équation suivante :

$$\Pi \times Q = 0$$

où $\Pi = (\pi_0, \pi_1, \pi_2)$ représente la distribution stationnaire que l'on veut calculer et Q est le générateur infinitesimal correspondant à la chaîne de Markov.

$$Q = \begin{pmatrix} -\lambda & \lambda & 0 \\ \mu & -(\lambda + \mu) & \lambda \\ 0 & \mu & -\mu \end{pmatrix}$$

Une fois obtenu, Π représente le facteur clé nécessaire au calcul des mesures de performance recherchées par l'utilisateur. Le calcul de ces mesures peut être effectué grâce à l'outil PEPA *State finder* qui fait partie intégrante du PEPA Workbench. Une fois le modèle résolu, PEPA *State finder* permet d'identifier les états qui satisfont certaines propriétés et à partir de ça l'utilisateur peut calculer les mesures de performance qui l'intéressent.

Dans note exemple, on peut être intéressé par le calcul du taux de perte des clients qui arrivent dans le système. Ce taux est donné par :

$$Taux_perte = \lambda \times \pi_2$$

π_2 étant la probabilité d'être dans l'état 2 de la chaîne de Markov, c'est-à-dire la probabilité que la file soit saturée.

3. ARFANet : une infrastructure basée sur les règles actives

L'idée de base d'un réseau actif consiste à fournir les mécanismes de déclenchement qui permettent à certains nœuds (nœuds actifs) de réagir aux événements externes convoyés par les paquets d'information ou produits par l'environnement du système d'exploitation du nœud. Dans ARFANet, une application peut être décrite comme un ensemble de règles actives, où chaque règle est définie sous la forme d'un tuple Événement-Condition-Action.

3.1. Composantes d'une règle active

La forme générale d'une règle active est la suivante :

On <nom d'événement> *If* <condition> *Then* <invocation de l'action>.

– *Spécification de l'événement* : Un événement est tout signal discret qui nécessite la réaction d'un nœud actif en déclenchant un service d'applications. Exemples : arrivée d'un paquet à un nœud, ACK, NACK ou les ACK-Timeout qu'un nœud peut recevoir.

– *Spécification de la condition* : Une condition peut être une formule logique ou une fonction booléenne. Les variables de la partie condition peuvent se rapporter aux données de l'instance de l'événement ou aux données persistantes dans le nœud ou les deux. Par exemple, une condition simple peut tester si un paquet a été envoyé ou non, s'il est toujours dans le cache ou non, etc...

– *Spécification de l'action* : La partie action peut être n'importe quelle action que le nœud actif peut exécuter pour la communication de données, la gestion du réseau, etc. Exemples : envoyer un paquet, envoyer un ACK ou un NACK, demander une attribution de ressource, appliquer une procédure de reconfiguration de réseau,...

Les règles sont regroupées en modules et un service d'applications est défini par un ou plusieurs modules. L'en-tête de chaque module contient un ensemble de paramètres de la sémantique qui caractérisent le comportement des règles (pour plus de détails, voir [BOU 02]). L'exécution de l'application consiste alors à détecter les événements, évaluer les conditions et enfin lancer les actions correspondantes.

3.2. Architecture d'un nœud actif

Dans ARFANet [BOU 02], chaque nœud du réseau est défini par une architecture à trois couches (Figure 2). En plus du système d'exploitation, le nœud comprend la couche *services d'application*, la couche *moniteurs actifs* et enfin *la machine virtuelle*.

– *Le moniteur actif* ou l'environnement d'exécution met en application la sémantique opérationnelle d'une classe de programmes d'applications, chacune étant définie par un ensemble de règles actives (ECA). Un nœud actif est équipé d'un ou de plusieurs moniteurs actifs, chacun étant consacré à une classe d'applications exigeant la même sémantique d'exécution. Les moniteurs actifs sont dynamiquement chargés (publiés) dans un ou plusieurs nœuds actifs.



Figure 2. Architecture du nœud actif

– *Le service d'application* est défini comme un ensemble de règles actives qui seront exécutées selon un moniteur actif spécifique. Comme les moniteurs actifs, les services d'applications sont dynamiquement chargés dans les nœuds actifs. Le lien entre un service d'applications et le moniteur actif correspondant est fait quand les paquets du service d'applications arrivent au nœud.

– *La machine virtuelle* est la couche entre le système d'exploitation du nœud et les moniteurs actifs. Elle est installée une fois pour toutes et fournit les fonctionnalités de bases telles que :

- routage des paquets (envoyer, recevoir, annoter les paquets),

- analyse des paquets pour différencier entre ceux correspondant aux moniteurs actifs (*m-packets*), ceux correspondant aux services d'application (*s-packet*) et ceux correspondant aux données (*d-packet*).
- amorçage des *m-packets* afin de rendre le nœud actif,
- chargement et liaison des services d'applications pour être exécutés dans le nœud actif,...
- liaison des services d'applications aux moniteurs actifs ou des données d'applications aux services d'applications.

La modélisation de l'architecture à couches du nœud actif et son mode d'exécution suppose que le nœud est vu comme un ensemble de files d'attente de capacités finies. Nous nous intéressons à la configuration du réseau de files d'attente décrite par la figure 3.

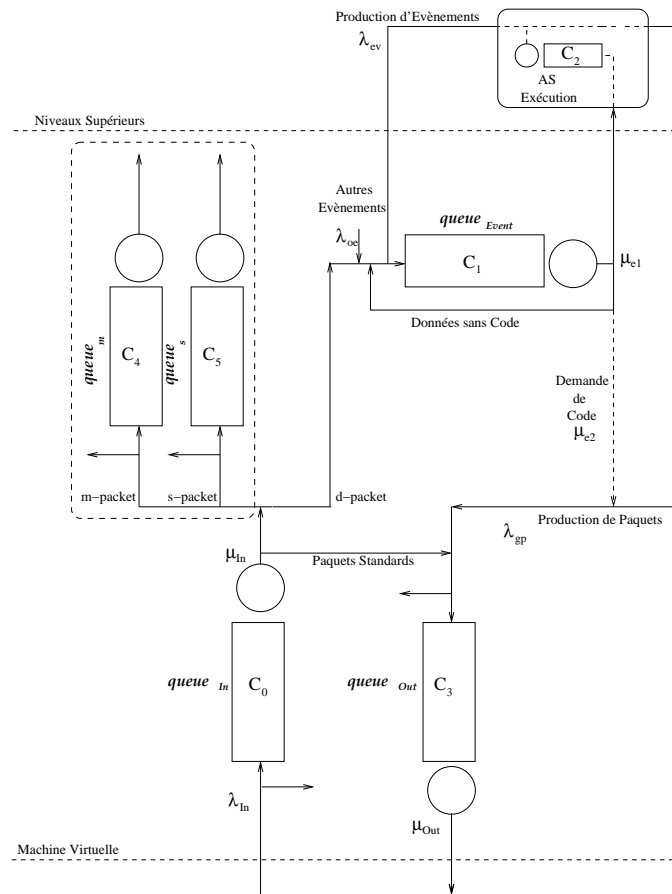


Figure 3. Réseau de files d'attente modélisant un nœud actif : Configuration 1

Dans cette configuration, les paquets arrivant au nœud sont mis dans la file d'entrée $queue_{in}$, avant d'être répartis selon leur type :

- les paquets standards (à forwarder) sont directement envoyés à la file de sortie $queue_{out}$,
- les paquets actifs de données (*d-paquets*) sont orientés vers la file des événements $queue_{event}$. Si un *d-packet* en service dans $queue_{event}$ référence un paquet de code actif (*s-packet* or *m-packet*) absent du nœud, le paquet est remis dans la file $queue_{event}$ et une requête concernant le ou les paquets absents est envoyée au réseau via la file de sortie.
- les paquets contenant le code actif des services d'applications (*s-paquets*) sont orientés vers la file $queue_s$,
- les paquets contenant le code actif des moniteurs (*m-paquets*) sont orientés vers la file $queue_m$.

En plus des *d-paquets*, $queue_{event}$ reçoit également des événements internes tels que les événements produits par le système et l'exécution des applications. A la fin de son service dans la file $queue_{event}$, le *d-paquet* est envoyé à la file d'attente de la couche application pour déclencher la règle active correspondante.

Par ailleurs, l'exécution de l'application peut générer de nouveaux paquets actifs envoyés à la file d'attente de sortie.

□

L'objectif de ce travail est l'analyse de l'impact de la présence des paquets actifs sur les performances des paquets standards. Comme ces paquets sont directement orientés de la file d'entrée vers la file de sortie, seules les couches ayant un lien direct, donc un impact, avec ces deux files sont explicitement représentées. Dans cette configuration, en plus de la machine virtuelle, la couche service d'application est représentée par la file $queue_{as}$ qui reçoit de la machine virtuelle les données nécessaires à l'exécution d'une application et qui peut produire à son tour des paquets vers la file de sortie du nœud. La couche moniteurs d'exécution n'ayant aucun lien direct avec la file de sortie, cette couche n'est pas explicitement représentée.

4. Le modèle PEPA

Comme nous nous intéressons à l'impact des nouvelles fonctionnalités des nœuds du réseau sur les paquets traditionnels ou passifs, en termes de taux de perte et de latence du nœud, le modèle PEPA développé pour cette configuration du réseau de files d'attente ne prend pas en compte les files d'attente des paquets contenant le code actif, c'est-à-dire $queue_s$ et $queue_m$. Ce modèle comprend quatre composantes : $Buffer_0^{(in)}$, $Buffer_0^{(out)}$, $Buffer_0^{(event)}$ et $Buffer_0^{(as)}$ modélisant respectivement les files $queue_{in}$, $queue_{out}$, $queue_{event}$ et $queue_{as}$. Ces composantes sont définies comme suit :

– **Composante** $Buffer_0^{(in)}$: Les arrivées du réseau à la file d'attente d'entrée du nœud sont modélisées en utilisant le type d'action in . Comme nous faisons une distinction entre les différents types de paquets, le service d'un d -paquet, s -paquet, m -paquet et d'un paquet passif est modélisé en utilisant respectivement le type d'action $service_d$, $service_s$, $service_m$ et $service_p$. La probabilité de servir un paquet de type $x = d, s, m, p$ est notée q_x . Si C_0 est la capacité de la file d'entrée, alors son comportement complet est comme suit :

$$\begin{aligned}
Buffer_0^{(in)} &\stackrel{def}{=} (in, \lambda_{in}).Buffer_1^{(in)} \\
Buffer_1^{(in)} &\stackrel{def}{=} (in, \lambda_{in}).Buffer_2^{(in)} + (service_d, q_d \times \mu_{in}).Buffer_0^{(in)} \\
&\quad + (service_p, q_p \times \mu_{in}).Buffer_0^{(in)} + (service_s, q_s \times \mu_{in}).Buffer_0^{(in)} \\
&\quad + (service_m, q_m \times \mu_{in}).Buffer_0^{(in)} \\
&\vdots \\
Buffer_{C_0}^{(in)} &\stackrel{def}{=} (in, \lambda_{in}).Buffer_{C_0}^{(in)} + (service_d, q_d \times \mu_{in}).Buffer_{C_0-1}^{(in)} \\
&\quad + (service_p, q_p \times \mu_{in}).Buffer_{C_0-1}^{(in)} + (service_s, q_s \times \mu_{in}).Buffer_{C_0-1}^{(in)} \\
&\quad + (service_m, q_m \times \mu_{in}).Buffer_{C_0-1}^{(in)}
\end{aligned}$$

– **Component** $Buffer_0^{(event)}$: L'arrivée d'un événement interne à la file d'attente $queue_{event}$ est modélisée en utilisant l'action in_{oe} . Les autres arrivées sont les événements résultants de l'exécution des services d'application, modélisés avec l'action $generate_{ev}$, et les d -paquets après leur service dans la file d'entrée ($service_d$). Le service au niveau de la file $queue_{event}$ est modélisé par deux types d'actions, $service_{e2}$ qui modélise le service d'un d -packet qui référence un paquet de code absent et $service_{e1}$ qui modélise le service d'un d -packet pour lequel tout se passe normalement avec une probabilité notée q .

$$\begin{aligned}
Buffer_0^{(event)} &\stackrel{def}{=} (service_d, \top).Buffer_1^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_1^{(event)} \\
&\quad + (generate_{ev}, \top).Buffer_1^{(event)} \\
Buffer_1^{(event)} &\stackrel{def}{=} (service_d, \top).Buffer_2^{(event)} + (service_{e1}, q \times \mu_{e1}).Buffer_0^{(event)} \\
&\quad + (generate_{ev}, \top).Buffer_2^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_2^{(event)} \\
&\quad + (service_{e2}, (1 - q) \times \mu_{e2}).(generate_{req}, \mu_{e2}).Buffer_1^{(event)} \\
&\vdots \\
Buffer_{C_1}^{(event)} &\stackrel{def}{=} (service_d, \top).Buffer_{C_1}^{(event)} + (service_{e1}, q \times \mu_{e1}).Buffer_{C_1-1}^{(event)} \\
&\quad + (generate_{ev}, \top).Buffer_{C_1}^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_{C_1}^{(event)} \\
&\quad + (service_{e2}, (1 - q) \times \mu_{e2}).(generate_{req}, \mu_{e2}).Buffer_{C_1}^{(event)}
\end{aligned}$$

– **Composante** $Buffer_0^{(as)}$: Pour modéliser le comportement de la file d'attente des applications en exécution, on considère trois types d'actions : $service_{e1}$, $generate_{ev}$ et $generate_p$. La première décrit l'arrivée d'un événement qui déclenche l'exécution d'un service d'application. Cette exécution peut avoir comme conséquence la génération de nouveaux événements ($generate_{ev}$) avec le taux λ_{ev} et de

états où l'activité $service_p$ échoue, c'est-à-dire, les états qui correspondent à une file de sortie pleine.

De même, la latence du nœud actif pour les paquets standards est la latence au niveau de la file d'entrée et celle de sortie. Pour calculer ce critère de performance, les deux files d'attente sont considérées comme des files $M/M/1/C$ et la formule de calcul du temps de réponse moyen correspondante est utilisée.

Dans ce qui suit, nous présentons les résultats numériques obtenus pour ces critères de performance après la résolution du modèle à l'aide de l'outil PEPA Workbench.

6. Résultats numériques

Pour calculer les critères de performance que nous avons définis ci-dessus, nous devons définir les paramètres d'entrée des deux modèles. Considérons d'abord l'arrivée externe des paquets au nœud. Sachant que ces arrivées se composent de paquets actifs et de paquets standards, nous supposons dans nos expériences que $\lambda_{in} = \lambda_{in,a} + \lambda_{in,s}$, où $\lambda_{in,a}$ est le taux d'arrivée des paquets actifs et $\lambda_{in,s}$ le taux d'arrivée des paquets standards. Comme nous nous intéressons aux performances du nœud en ce qui concerne les paquets traditionnels, nous définissons $\lambda_{in,s} = K \times \lambda_{in}$ où K est la proportion de paquets standards qui varie entre 0.2 et 0.8. On notera $\overline{K} = 1 - K$ la proportion de paquets actifs arrivant au nœud. Indépendamment de la valeur de K , les paquets actifs sont divisés entre les différents types de paquets actifs selon les proportions définies dans Table 1. Le code actif, représenté par les m -paquets et les s -paquets, n'intervient que dans l'exécution des règles, pas dans leur gestion ou leur lancement. La persistance du code actif dans le cache et la modularité des règles actives permettent une intense réutilisabilité, induisant peu de circulation du code dans le réseau en comparaison avec les données actives. D'où la répartition des paquets actifs définie dans la table 1.

Par ailleurs, en supposant que la vitesse moyenne d'un routeur est de $10Mbits/s$ et que la taille moyenne des paquets est de 250 octets, on obtient un taux de service au niveau des files du nœud de $5000\text{ paquets}/s$, sauf pour la file d'attente $queue_{as}$ de la couche application dont le serveur est supposé un peu plus rapide avec un taux de $7000\text{ paquets}/s$.

Dans nos expériences initiales, nous supposons que toutes les files d'attente ont la même capacité $C_i = 5$, $i = 0, \dots, 3$. C'est la taille minimale considérée dans cette étude; afin d'étudier l'influence de la taille des buffers sur le délai et les pertes des paquets, d'autres tailles seront considérées dans la section 6.2. Pour limiter le délai, il est recommandé d'avoir des tailles aussi petites que possibles, quelques douzaines de paquets [MOR 99] selon le débit du système. Faire varier la valeur de cette taille à partir de 5 pour évaluer son impact sur les mesures de performance qui nous intéressent rentre tout à fait dans ce contexte [MOR 99]. Pour cette capacité des buffers, la chaîne de Markov résultante a 26136 états et 214632 transitions.

Les paramètres du modèle sont récapitulés dans la table 1.

Répartition des paquets actifs	Taux d'arrivée (p/ms)	Taux de service (p/ms)
d -packet : 75%	$\lambda_{ev} = 1.0$	$\mu_{in} = 5.0$
m -packet : 12.5%	$\lambda_{oe} = 1.0$	$\mu_{out} = 5.0$
s -packet : 12.5%	$\lambda_{gp} = 1.0$	$\mu_e = 5.0; \mu_{as} = 7.0$

Tableau 1. Valeurs des paramètres

6.1. Les mesures de performance

On suppose que parmi les d -paquets qui arrivent au nœud 50% ($q = 0.5$) de ces paquets référencent un code non présent dans le nœud. Les résultats obtenus pour ce modèle sont décrits dans les figures 4-6. Toutes les courbes sont tracées en fonction de la charge de la file d'entrée ; nous diminuons le temps des inter-arrivées des paquets ($1/\lambda_{in}$), augmentant ainsi la charge du nœud.

1. Le débit : Les premiers résultats pour ce modèle concernent le débit du nœud actif pour les paquets passifs et l'impact de la proportion de paquets actifs dans le nœud sur ce débit.

– La figure 4(a) montre que le débit du nœud pour les paquets passifs augmente proportionnellement au pourcentage K de paquets passifs dans le nœud, ainsi qu'au taux d'arrivée λ_{in} à la file d'entrée. On note également qu'à partir d'une certaine charge, ce débit a tendance à se stabiliser et ce pour toutes les valeurs de K .

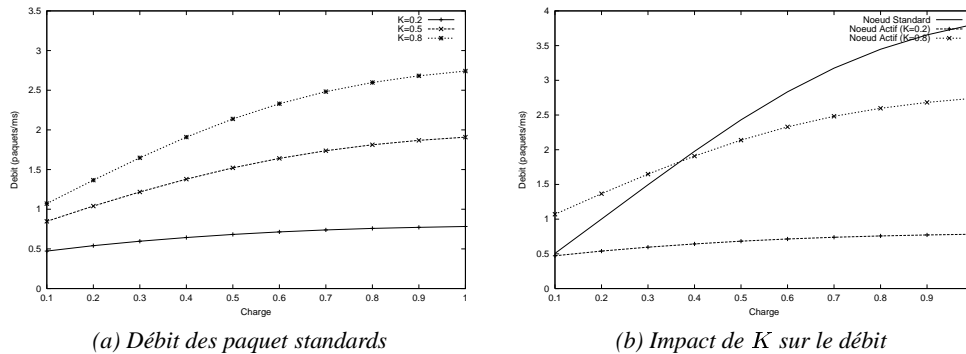


Figure 4. Le débit

– Pour avoir une idée de l'impact des nouvelles fonctionnalités du nœud sur le débit pour les paquets passifs, nous comparons le débit d'un nœud standard ou passif avec celui du nœud actif pour $K = 0.2$ et $K = 0.8$ (figure 4(b)). On note que pour

de très faibles charges, le débit du nœud passif et celui du nœud actif ne sont pas très différents, en particulier, pour $K = 0.8$. Mais plus la charge devient importante, plus ces deux débits diffèrent. Ce phénomène est dû au fait qu'une augmentation de la charge pour un nœud standard signifie l'arrivée de plus de paquets qui sont tous passifs. Alors que pour un nœud actif, cela signifie plus d'arrivées des deux types de paquets actifs et passifs. Cette différence est plus évidente lorsque la proportion de paquets passifs est très petite, c'est-à-dire lorsque $K = 0.2$.

Par ailleurs, la figure 4(b) montre que quand la charge est faible, la courbe du nœud standard est bornée par les deux autres courbes.

2. Les pertes : L'objectif de cette partie est de montrer l'impact de la présence de paquets actifs sur les pertes des paquets standards.

– La figure 5(a) montre que quelque soit la proportion de paquets actifs dans le nœud, le taux de perte des paquets passifs croît proportionnellement à la charge du nœud. De plus, plus K est important, plus les pertes sont importantes. Cependant, on note que lorsque la charge est faible (0.1 – 0.5) les pertes sont similaires pour les différentes valeurs de K . Par ailleurs, on peut noter que dans tous les cas le taux de perte augmente de manière exponentielle. Bien que nous ne le montrons pas dans ce papier, cela est dû à la file d'entrée. En effet, contrairement à la file de sortie où le taux de perte tend à devenir constant à partir d'une certaine charge, les taux de perte dans la file d'entrée sont exponentiels. Cette différence est due au fait que les paquets actifs arrivant à la file d'entrée restent dans le nœud et ne sont pas envoyés directement à la file de sortie.

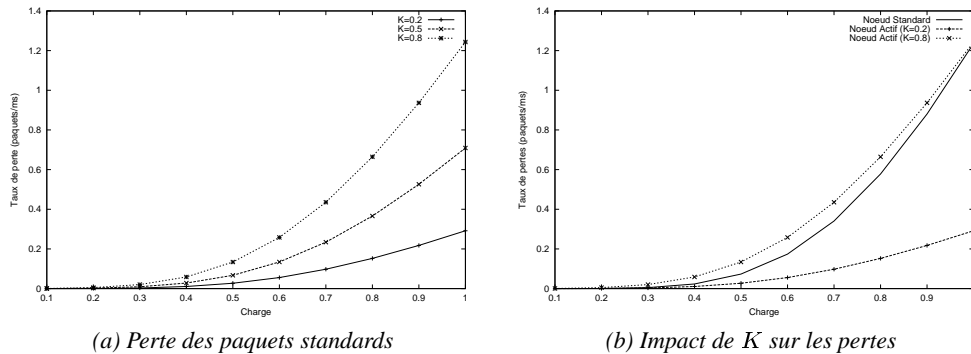


Figure 5. Les taux de perte

– Les résultats décrits dans la figure 5(b) sont particulièrement intéressants. Dans cette figure, nous comparons le taux de perte des paquets standards dans un nœud actif avec celui obtenu pour le nœud standard. La figure montre que les taux de perte pour le nœud standard sont bornés par les taux de perte quand $K = 0.2$ et ceux quand $K = 0.8$. La première peut être considérée comme la borne inférieure et la seconde

comme la borne supérieure. Ce résultat suggère que ces pertes peuvent être contrôlées pour ne pas dépasser une certaine limite en choisissant la bonne proportion de paquets actifs à admettre dans le nœud.

3. La latence : Les derniers résultats concernent la latence des paquets passifs dans un nœud actif.

– Dans la figure 6(a), nous comparons la latence dans un nœud actif pour différentes valeurs de K . Cette figure montre que plus la proportion de paquets passifs augmente, plus la latence augmente. Comme pour les pertes, le calcul de la latence considère les deux files $queue_{in}$ et $queue_{out}$. Contrairement à la file d'entrée, la file de sortie ne reçoit des paquets qui arrivent au nœud que les paquets à retransmettre (standards). Aussi, si on diminue la proportion de paquets actifs arrivant au nœud, augmentant ainsi celle des paquets standards qui y arrivent, la latence au niveau de la file de sortie augmente. Alors que cela ne change rien à la latence au niveau de la file d'entrée puisque le taux total d'arrivées reste inchangé. Cependant, la différence entre les résultats obtenus pour les différentes valeurs de K n'est pas très significative.

– Dans la figure 6(b), nous comparons la latence d'un paquet passif dans un nœud actif pour $K = 0.2$ et $K = 0.8$ avec la latence dans un nœud standard. Bien que cette figure suggère que la latence dans un nœud actif est toujours supérieure pour $K > 0.2$ à forte charge et pour $K \geq 0.2$ à faible charge, la différence avec un nœud standard n'est pas très importante.

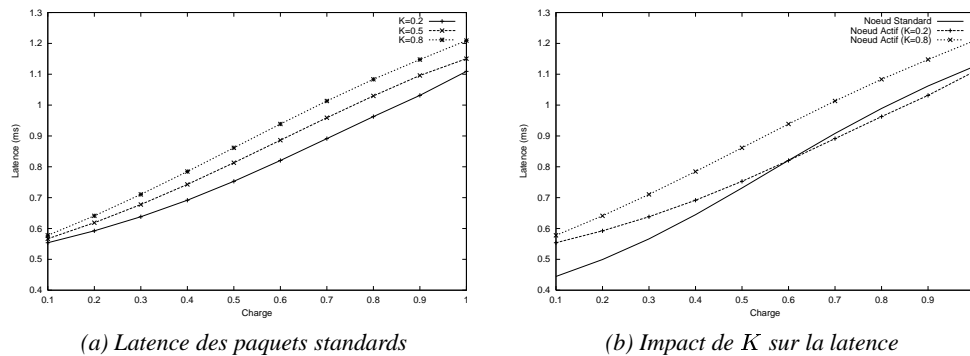


Figure 6. La latence

Pour voir si les résultats obtenus pour les nœuds actifs avec une configuration de ce type se confirment, nous étudions ci-dessous l'impact de la capacité des files d'attente sur les pertes et la latence.

6.2. Impact de la capacité des buffers

Dans cette partie, nous étudions l'impact de l'augmentation de la taille des files d'attente sur le taux de perte des paquets standards et la latence d'un nœud actif pour ce type de paquets. Pour cela, nous considérons deux cas, $C_i = 10$ et $C_i = 15$, $i = 0, \dots, 3$. Comme toutes les files ont la même capacité, nous notons C la taille de ces files. Les résultats obtenus sont présentés dans les figures 7-9.

1. Les pertes : la figure 7 montre l'impact de la taille des buffers sur la perte des paquets standards pour $K = 0.2$ et $K = 0.8$.

– Dans la figure 7(a), nous comparons le taux de perte des paquets standards pour les trois tailles des buffers $C = 5, 10, 15$ et pour $K = 0.8$. Cette figure montre que les taux de perte croissent exponentiellement en fonction de la charge, et cela quelque soit la taille des buffers. De plus, ces taux diminuent logiquement à mesure que la capacité des buffers augmente. Par ailleurs, on note que la différence entre les taux de perte quand $C = 10$ et $C = 15$ est négligeable, en particulier pour une charge entre 0.1 et 0.6. Cette différence est toujours plus petite que la différence entre les pertes quand $C = 10$ et $C = 5$. Cela suggère que si nous continuons à augmenter la taille des buffers, le taux de perte peut continuer à diminuer. Cependant, il semble probable que nous atteindrons une capacité des buffers à laquelle ce taux ne diminuera plus de manière aussi significative.

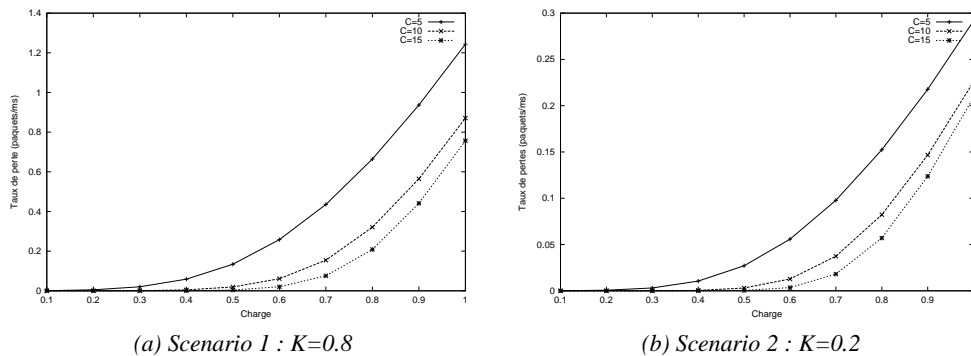


Figure 7. Les taux de pertes

– La figure 7(b) montre les résultats obtenus pour $K = 0.2$, c'est-à-dire lorsque le trafic des paquets standards est très perturbé. Comme dans la figure 7(a), nous pouvons voir que le taux de perte est inversement proportionnelle à la taille buffers. Cependant, nous pouvons voir aussi que les taux de perte sont très petits comparés à ceux obtenus pour $K = 0.8$. Ceci est dû aux pertes enregistrées dans la file de sortie. Comme tous les paquets standards sont dirigés vers la file d'attente de sortie, quand $K = 0.8$ il

est plus que probable que cette file d'attente soit pleine et plus de paquets standards soient perdus.

2. La latence : La figure 8 montre l'impact de la taille des buffers sur la latence. Comme précédemment, nous considérons deux scénarios : $K = 0.2$ et $K = 0.8$.

– Dans la figure 8(a), nous comparons la latence obtenue pour les différentes tailles de buffers. La proportion des paquets actifs est $\bar{K} = 0.2$, c'est-à-dire que le trafic des paquets standards est légèrement perturbé. Cette figure montre que pour de faibles charges, la taille des buffers n'a pas un grand impact sur la latence, mais dès que la charge devient plus importante la latence augmente proportionnellement à la taille des buffers.

– La figure 8(b) présente les résultats obtenus quand la proportion des paquets actifs est $\bar{K} = 0.8$, c'est-à-dire quand le trafic des paquets actifs est beaucoup plus important que le trafic des paquets standards. Elle montre que la capacité des buffers semble avoir un impact plus modéré que lorsque $\bar{K} = 0.2$. En comparant les deux scénarios, on remarque que plus la taille des buffers augmente, plus la différence entre les résultats des deux scénarios est importante. Cela est dû au fait que les paquets actifs arrivant au nœud vont devoir rester dans le nœud pour un traitement ultérieur, alors que les paquets passifs sont tout de suite acheminés vers la file de sortie. Lorsque $K = 0.8$, 80% du trafic est orienté vers la file de sortie, ce qui explique une plus grande latence, comparé au cas où uniquement 20% ($K = 0.2$) du trafic est immédiatement acheminé vers cette même file. De plus, plus la taille des buffers augmente, plus ce phénomène s'accroît.

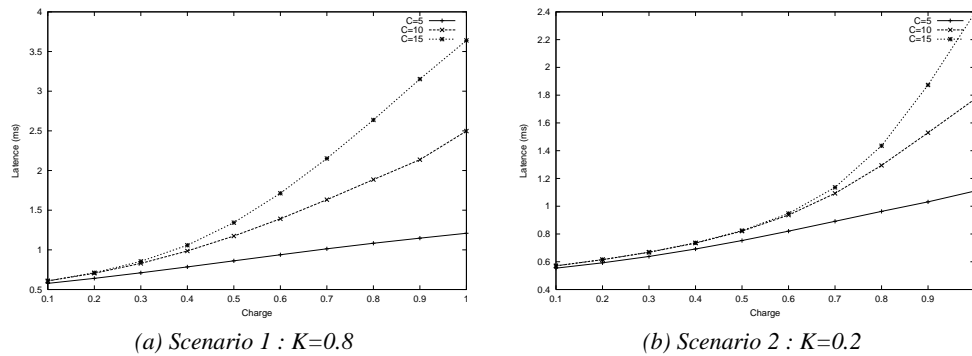
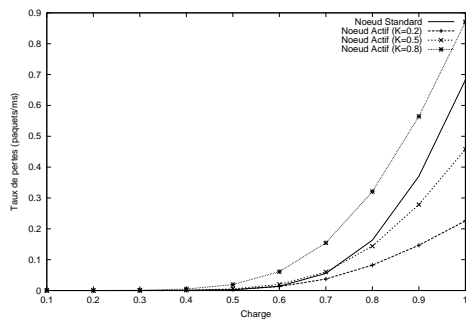


Figure 8. La latence

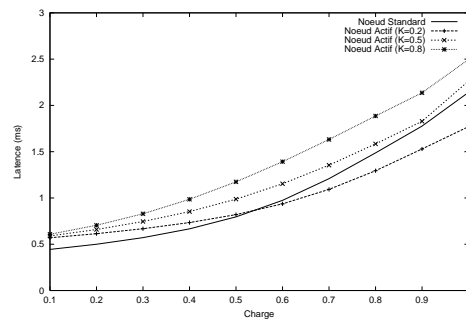
3. Actif versus passif : Dans ce qui suit, nous comparons, d'une part, le taux de perte dans un nœud standard avec le taux de perte dans un nœud actif, et d'autre part, la latence d'un paquet standard dans les deux types de nœuds. Ces comparaisons sont

faites pour une capacité des buffers $C = 10$ et pour différentes valeurs de K : 0.2, 0.5, 0.8.

– Les résultats obtenus dans la figure 9(a) pour les pertes confirment les résultats décrits dans la figure 5(b) pour $C = 5$. A savoir que le taux de perte dans un nœud standard est borné par le taux de perte dans un nœud actif. De nouveau ceci suggère que la perte de paquets passifs dans un nœud actif dépend de la proportion des paquets actifs arrivant au nœud. Et ces pertes peuvent être contrôlées pour ne pas dépasser une certaine limite si nous choisissons la bonne proportion de paquets actifs à admettre dans le nœud. Nous pouvons voir que $K = 0.8$ peut être utilisé comme une borne supérieure comme le montre la figure 5(b). Cependant la figure 9(a) montre qu'il est préférable d'utiliser $K = 0.5$ plutôt que $K = 0.2$ comme borne inférieure.



(a) Impact de \bar{K} sur les taux de pertes : $C = 10$



(b) Impact de \bar{K} sur la latence : $C = 10$

Figure 9. Nœud standard versus Nœud actif

– Comme pour le cas $C = 5$ (Figure 6), les nouvelles fonctionnalités du nœud pénalisent très peu les paquets standards et ce quelque soit la valeur de K . On peut noter que pour de fortes charges, la latence dans le nœud standard est bornée par la latence quand $K = 0.2$ (borne inférieure) et $K = 0.5$ (borne supérieure).

6.3. ARFAnet versus PEPA

Dans cette partie nous comparons les résultats obtenus avec le modèle PEPA de *Configuration 1* et les mesures effectuées sur la plateforme ARFANet.

Pour pouvoir faire cette comparaison, nous avons dû calculer le taux de service réel de la plateforme et adapter les paramètres du modèle PEPA à ce nouveau taux de service (26 paquets/s). Nos tests ont été faits sur une machine Linux avec un processeur "AMD Athlon(tm) XP 1500+" de 1339 MHz, 256 Ko de mémoire cache et une mémoire vive 256Mo.

La figure 10 montre les résultats obtenus pour le débit et le taux de perte pour une taille des files $C=10$ et pour les deux proportions de paquets passifs $K = 0.2$ et $K = 0.8$. Les limites matérielles et logicielles nous ont contraint à limiter la charge de la file d'entrée entre 0.3 et 1.3.

– La figure 10(a) représente le débit du noeud pour les paquets standards obtenu avec le modèle PEPA, d'une part, et mesuré sur la plateforme ARFANet, d'autre part. Cette figure montre que les deux débits sont très proches et ceux quelque soit le degré de perturbation du trafic des paquets standards (K). Lorsque $K = 0.2$, le débit du modèle PEPA et celui d'ARFANet ont le même comportement, le premier étant toujours légèrement supérieur au second. Par contre lorsque le trafic des paquets standard est très peu perturbé ($K = 0.8$), le débit du modèle PEPA est supérieur uniquement dans la première moitié de la charge. Dans la seconde moitié la tendance s'inverse.

– Pour les pertes, on voit, dans la figure 10(b), que lorsque $K = 0.2$ les pertes du modèle PEPA sont très similaires à celle de l'implémentation, et ceci quelque soit la charge en entrée du noeud. Par contre lorsque $K = 0.8$, les deux taux de pertes sont initialement très proches ($charge < 0.7$). Plus la charge augmente, plus l'écart entre les deux taux de perte augmente, le taux de pertes du modèle PEPA étant plus important. A noter que, dans un souci de clarté de la première partie de la figure (0.3 – 0.7), on se limite dans cette figure à une charge maximale égale à 1.0.

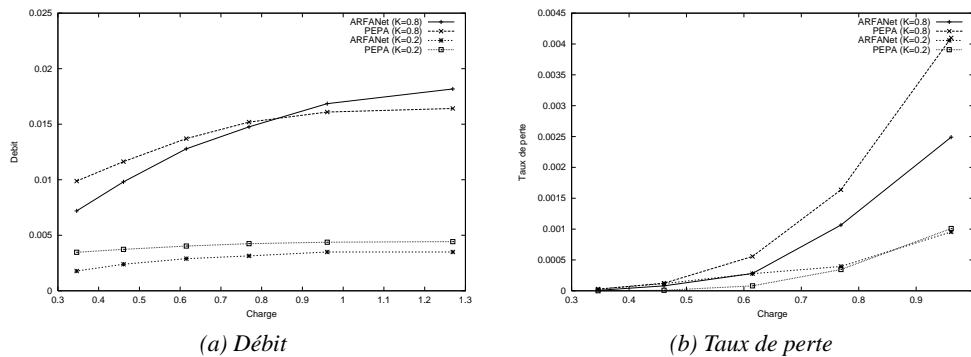


Figure 10. ARFANet versus PEPA

□

Bilan : Si l'étude analytique nous donne plus de contrôle au niveau du choix des paramètres, surtout du taux de service, dans le cas de l'implémentation, celui-ci nous est imposé par la configuration matérielle qui ne correspond pas forcément à la réalité.

Toute fois, de manière globale, les résultats des tests confirment ceux de la modélisation avec PEPA, et par conséquent confortent les conclusions de l'étude analytique sur l'influence de l'introduction des paquets actifs sur les paquets passifs.

□

Dans la configuration de la figure 3, tous les *d-packets* sont acheminés vers la file des événements (*queue_{event}*). Lorsqu'un de ces paquets est servi dans la file, s'il référence un code non présent dans le noeud, une requête de code actif est générée et envoyée vers la file de sortie, et le *d-packet* est remis dans la file. Cela peut avoir pour conséquence la génération de plusieurs requêtes de code par *d-packet* et de ce fait la présence d'un plus grand nombre de paquets de requêtes dans la file de sortie. Ceci peut être la cause de la perte de certains paquets standards et la cause d'une plus grande latence dans la file de sortie.

Dans ce qui suit, nous proposons de modifier partiellement notre configuration pour tenir compte de cet aspect. Pour distinguer cette nouvelle configuration de la précédente, on notera la première *Configuration 1* et la nouvelle *Configuration 2*.

7. Configuration 2

Dans cette configuration (Figure 11), une file supplémentaire *queue_{dw}* est introduite. Elle permet la sauvegarde des *d-paquets* qui référencent un code actif *s-paquet* ou *m-paquet* non présent dans le noeud. L'absence des paquets de code référencés par certains *d-paquets* n'est plus détectée au niveau de la file des événements, mais plutôt au niveau de la file d'entrée. Dans ce cas, le paquet est orienté vers la file *queue_{dw}* et une requête des paquets absents est envoyée au réseau via la file de sortie *queue_{out}*. Les autres files de cette nouvelle configuration sont similaires à celles dans *Configuration 1*.

7.1. Le modèle PEPA

Le modèle PEPA correspondant à la seconde configuration comprend, en plus des composantes du modèle précédent, la composante $Buffer_0^{(dw)}$ qui modélise la file *queue_{dw}*. Les composantes $Buffer_0^{(out)}$ et $Buffer_0^{(as)}$ sont identiques à celles du modèle précédent. Cependant la présence de la file *queue_{dw}* dans cette configuration apporte des changements au niveau des autres composantes, c'est-à-dire $Buffer_0^{(in)}$ et $Buffer_0^{(event)}$.

– **Composante $Buffer_0^{(in)}$** : Nous utilisons *service_{dw}* pour modéliser le service d'un *d-paquet* qui référence un paquet de code inexistant. A cette activité nous asso-

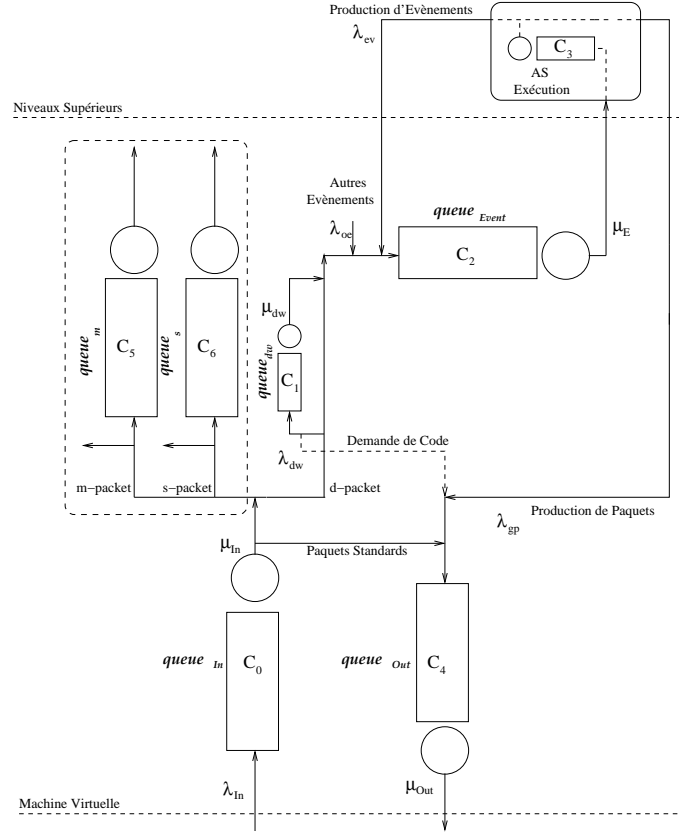


Figure 11. Réseau de files d'attente modélisant un nœud actif : Configuration 2

cions la probabilité q_{dw} . La génération d'une requête pour un paquet de code absent est modélisée avec l'action $generate_{req}$.

$$\begin{aligned}
 Buffer_0^{(in)} &\stackrel{def}{=} (in, \lambda_{in}).Buffer_1^{(in)} \\
 Buffer_1^{(in)} &\stackrel{def}{=} (in, \lambda_{in}).Buffer_2^{(in)} + (service_d, q_d \times \mu_{in}).Buffer_0^{(in)} \\
 &\quad + (service_p, q_p \times \mu_{in}).Buffer_0^{(in)} + (service_s, q_s \times \mu_{in}).Buffer_0^{(in)} \\
 &\quad + (service_{dw}, q_{dw} \times \mu_{in}).(generate_{req}, \mu_{rq}).Buffer_0^{(in)} \\
 &\quad + (service_m, q_m \times \mu_{in}).Buffer_0^{(in)} \\
 &\quad \vdots \\
 Buffer_{C_0}^{(in)} &\stackrel{def}{=} (in, \lambda_{in}).Buffer_{C_0}^{(in)} + (service_d, q_d \times \mu_{in}).Buffer_{C_0-1}^{(in)} \\
 &\quad + (service_p, q_p \times \mu_{in}).Buffer_{C_0-1}^{(in)} + (service_s, q_s \times \mu_{in}).Buffer_{C_0-1}^{(in)} \\
 &\quad + (service_{dw}, q_{dw} \times \mu_{in}).(generate_{req}, \mu_{rq}).Buffer_{C_0-1}^{(in)} \\
 &\quad + (service_m, q_m \times \mu_{in}).Buffer_{C_0-1}^{(in)}
 \end{aligned}$$

– **Composante** $Buffer_0^{(event)}$: Comme tous les paquets dans la file sont du même type, à savoir des d -packets dont le code actif correspondant est présent dans le noeud, le service dans $queue_{event}$ est modélisé par une seule action $service_e$. Soit C_1 la capacité de cette file, son comportement est comme suit :

$$\begin{aligned}
Buffer_0^{(event)} &\stackrel{def}{=} (service_d, \top).Buffer_1^{(event)} + (service_{ew}, \top).Buffer_1^{(event)} \\
&\quad + (generate_{ev}, \top).Buffer_1^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_1^{(event)} \\
Buffer_1^{(event)} &\stackrel{def}{=} (service_d, \top).Buffer_2^{(event)} + (service_{ew}, \top).Buffer_2^{(event)} \\
&\quad + (generate_{ev}, \top).Buffer_2^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_2^{(event)} \\
&\quad + (service_e, \mu_e).Buffer_0^{(event)} \\
&\vdots \\
Buffer_{C_1}^{(event)} &\stackrel{def}{=} (service_d, \top).Buffer_{C_1}^{(event)} + (service_{ew}, \top).Buffer_{C_1}^{(event)} \\
&\quad + (generate_{ev}, \top).Buffer_{C_1}^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_{C_1}^{(event)} \\
&\quad + (service_e, \mu_e).Buffer_{C_1-1}^{(event)}
\end{aligned}$$

– **Composante** $Buffer_0^{(dw)}$: Les paquets de données dont le code actif correspondant est absent du noeud sont stockés dans la file d'attente $queue_{dw}$ ($service_{dw}$). L'action $service_{ew}$ modélise le service dans $queue_{dw}$ des d -packets dont le code est finalement arrivé au noeud.

$$\begin{aligned}
Buffer_0^{(dw)} &\stackrel{def}{=} (service_{dw}, \top).Buffer_1^{(dw)} \\
Buffer_1^{(dw)} &\stackrel{def}{=} (service_{dw}, \top).Buffer_2^{(dw)} + (service_{ew}, \mu_{dw}).Buffer_0^{(in)} \\
&\vdots \\
Buffer_{C_4}^{(dw)} &\stackrel{def}{=} (service_{dw}, \top).Buffer_{C_4}^{(dw)} + (service_{ew}, \mu_{dw}).Buffer_{C_4-1}^{(dw)}
\end{aligned}$$

Le réseau : Le comportement complet du réseau de files d'attente de cette configuration est donné par l'équation suivante :

$$System \stackrel{def}{=} \left(\left(\left(Buffer_0^{(in)} \bowtie_{\mathcal{K}} Buffer_0^{(event)} \right) \bowtie_{\mathcal{L}} Buffer_0^{(as)} \right) \bowtie_{\mathcal{M}} Buffer_0^{(out)} \right) \bowtie_{\mathcal{N}} Buffer_0^{(dw)}$$

Où les ensembles de coopération sont $\mathcal{M} = \{service_p, generate_p, generate_{req}\}$, $\mathcal{K} = \{service_d\}$, $\mathcal{L} = \{service_{e1}, generate_{ev}\}$ et $\mathcal{N} = \{service_{dw}, service_{ew}\}$.

7.2. Résultats numériques

On suppose que le taux de service dans la file d'attente $queue_{dw}$ est $\mu_{dw} = 7.0$. Comme dans le modèle précédent, on suppose que, parmi les d -packets arrivant au noeud, 50% ($q_{dw} = 0.5$) sont orientés vers la file $queue_{dw}$. De plus, on suppose que $\mu_{rq} = 2.0$. Les résultats obtenus pour ce modèle sont décrits dans les figures 12-14.

1. Le débit : La première partie de nos résultats concerne le débit du nœud actif pour les paquets standards. La figure 12 montre que ces résultats sont très similaires à ceux obtenus pour *Configuration 1*. Cependant, en comparant la figure 12(b) avec son équivalente dans le premier modèle (Figure 4(b)), on note une petite différence lorsque la charge est faible (0.1 – 0.4). Le débit est toujours inférieur à celui dans le noeud actif.

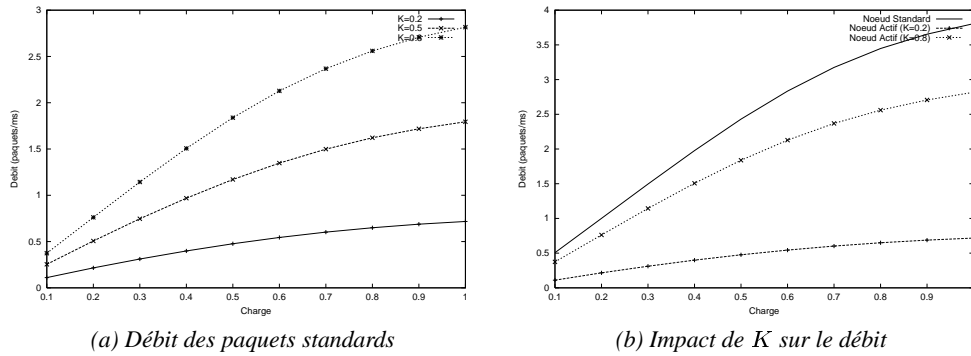


Figure 12. Le débit

2. Les pertes : La seconde partie de nos résultats concerne l’impact de la proportion de paquets actifs sur les taux de perte des paquets passifs. La figure 13 illustre les résultats obtenus.

– Bien qu’ayant le même comportement que celles dans la figure 13(a), les courbes décrites dans la figure 5(a) ne sont pas de la même grandeur. On remarque en effet que moins de paquets sont perdus dans le second modèle (*Configuration 2*). Ainsi pour $K = 0.8$, le taux de perte peut baisser de près de 30% lorsque la charge est très importante.

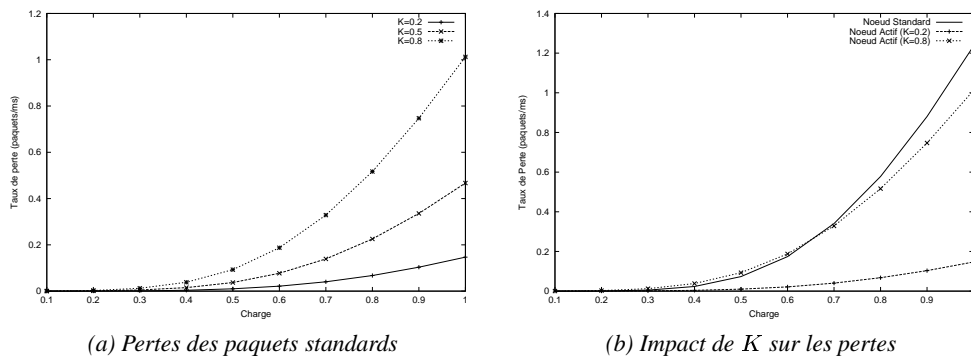


Figure 13. Les taux de perte

– Dans la figure 13(b), nous comparons les pertes dans un nœud standard avec les pertes dans un nœud actif pour $K = 0.2$ et $K = 0.8$. Lorsque la charge est faible (0.1 – 0.5) et que le trafic des paquets standards est très perturbé ($K = 0.2$), les pertes sont similaires à celles du nœud standard. Par contre, dès que la charge commence à être importante, on enregistre moins de pertes dans le nœud actif. Encore une fois, ceci est dû au fait qu’une augmentation de la charge pour un nœud actif implique une augmentation des paquets passifs et des paquets actifs, et en particulier ces derniers lorsque $K = 0.2$.

Par contre, le comportement des pertes lorsque le trafic des paquets standards est peu perturbé ($K = 0.8$) est très similaire à celui des pertes dans le nœud standard. A très forte charge, on note cependant le même phénomène que pour $K = 0.2$, mais en beaucoup moins accentué.

3. La latence : Les derniers résultats concernent la latence du nœud pour les paquets passifs.

– La figure 14(a) montre que le taux des paquets actifs dans le nœud n’a pas un grand impact sur la latence du nœud pour les paquets passifs. Par ailleurs, on note que la latence subie par les paquets standards est légèrement plus petite que dans *Configuration 1*. Comme pour les pertes, cette différence est due à l’introduction de la file $queue_{dw}$ dans le second modèle, mais contrairement aux pertes, cette différence n’est pas très significative.

– En comparant la latence du nœud passif avec celle du nœud actif (Figure 14(b)), on voit que, dans le nœud actif, les paquets passifs subissent une latence très proche de celle dans le nœud passif.

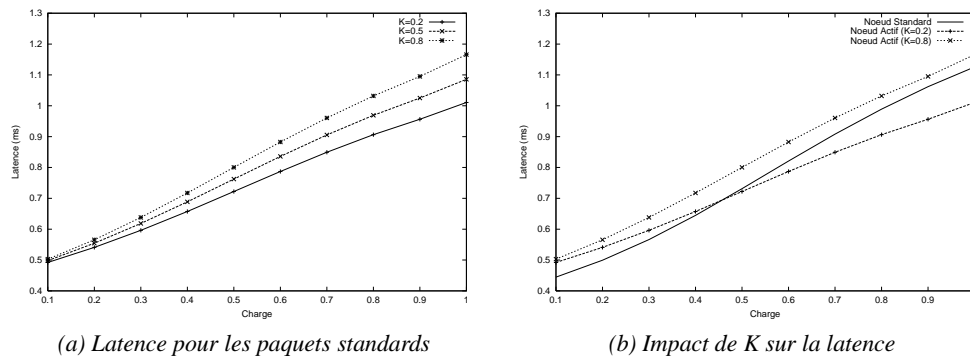


Figure 14. La latence

7.3. ARFANet versus PEPA

Dans cette partie nous comparons les résultats obtenus avec le modèle PEPA de *Configuration 2* et les mesures effectuées sur la plateforme ARFANet.

La figure 15 montre les résultats obtenus pour le débit et le taux de perte pour une taille des files $C=10$ et pour les deux proportions de paquets passifs $K = 0.2$ et $K = 0.8$.

– La figure 15(a) représente le débit du noeud pour les paquets standards obtenu avec le modèle PEPA, d’une part, et celui mesuré sur la plateforme ARFANet, d’autre part. Cette figure montre que les deux débits ont exactement le même comportement et leurs courbes sont presque confondues quelque soit la valeur de K . Dans le cas de *Configuration 1*, ceci n’est vrai qu’à moitié. Pour $K = 0.2$, les débits ont le même comportement, mais dans le cas de $K = 0.8$, leurs courbes s’intersectent.

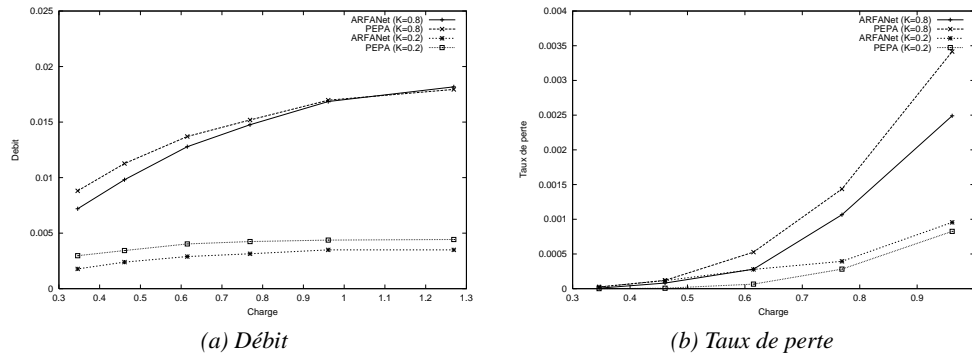


Figure 15. ARFANet versus PEPA

– Comme pour *Configuration 1*, la figure 15(b) montre que lorsque $K = 0.2$ les pertes du modèle PEPA sont très similaires à celle de l’implémentation. Et lorsque $K = 0.8$, les deux taux de pertes sont initialement très proches, mais comme pour le premier modèle, plus la charge augmente, plus l’écart entre les deux taux de perte augmente, le taux de pertes du modèle PEPA étant plus important. Cependant, contrairement à ce que nous avons vu pour *Configuration 1*, lorsque $K = 0.8$, l’écart entre les deux taux de perte est nettement plus petit, puisque les pertes enregistrées dans *Configuration 2* sont plus petites que celles obtenues pour la première configuration.

□

Bilan : Bien que le débit dans la nouvelle configuration soit toujours inférieur au débit du noeud standard, l’introduction d’une file supplémentaire permet de réduire le taux de perte des paquets standards de manière très significative. De plus, la présence

de cette file d'attente permet d'améliorer la latence dans le noeud pour les paquets passifs. Certes, cette amélioration est minime comparée à celle des pertes, mais la latence obtenue avec la seconde configuration se rapproche de celle d'un noeud standard. Cela a permis d'aboutir à une comparaison plus satisfaisante avec ARFANet.

8. Conclusion

Dans cet article, nous avons utilisé l'algèbre des processus stochastiques PEPA pour étudier les perturbations que peut subir un trafic "*normal*", qui doit co-exister avec un trafic "*actif*", dans un réseau adoptant la gestion active. Cette étude, qui est composée de deux parties, est réalisée au niveau d'un nœud du réseau actif où la gestion des paquets actifs utilise le principe des règles actives.

Dans la première partie de cette étude, une configuration de l'architecture du nœud actif a été considérée et le modèle PEPA correspondant a été utilisé pour évaluer, en termes de débit, de taux de perte et de latence, l'impact de l'introduction de paquets actifs dans un nœud sur les paquets traditionnels le traversant. Nous avons montré que la présence des paquets actifs, qui exigent un traitement plus long, a un impact négatif sur le débit du nœud pour les paquets standards.

Nous avons supposé que les paquets standards sont perdus lorsque la file d'entrée ou celle de sortie est pleine. Nous avons observé que les paquets actifs ont moins d'impact sur les pertes de la file de sortie que sur ceux de la file d'entrée, parce que les paquets actifs sont retardés dans le nœud en attente d'une activation. Aussi, le taux de perte des paquets standards est dominé par les pertes au niveau du buffer d'entrée.

Les résultats obtenus pour cette configuration suggèrent qu'il est possible de contrôler les pertes des paquets standards pour qu'elles restent similaires à celles enregistrées dans un nœud standard. De plus, la latence obtenue pour le nœud actif est assez proche de celle du nœud standard. Différentes tailles de buffers ont été utilisées et des résultats similaires ont été obtenus. De manière générale, les résultats de la configuration suggèrent que le scénario de la gestion active du réseau est faisable.

Par ailleurs, une comparaison des résultats analytiques avec ceux mesurés sur la plateforme ARFANet nous a permis de conclure que globalement ces résultats sont assez proches.

Dans un souci d'améliorer ces résultats, dans la seconde partie de cette étude, une seconde configuration de l'architecture du noeud actif a été proposée. Cette nouvelle configuration a permis d'améliorer le taux de perte de près de 30% ainsi que la latence, même si l'amélioration de celle-ci n'est pas aussi significative.

Nous avons trouvé que PEPA nous a fourni un formalisme de modélisation pratique pour réaliser cette étude. Une extension future de ce travail consiste à considérer la configuration d'un réseau de nœuds. Pour une telle étude, un formalisme comme PEPA nets serait plus approprié. PEPA nets [GIL 03] est une combinaison de PEPA et des réseaux de Petri stochastiques colorés où chaque jeton est une composante PEPA.

Ce formalisme combiné modélise de manière naturelle des applications telles que les systèmes de code mobile où les composantes PEPA sont utilisées pour représenter le code programme qui se déplace entre les réseaux hôtes (places dans PEPA nets).

9. Bibliographie

- [AZI 96] AZIZ A., SANWAL K., SINGHAL V., BRAYTON R., « Verifying continuous time Markov chains », *Computer-Aided Verification*, vol. 1102 de LNCS, Springer-Verlag, 1996, p. 169–276.
- [BEN 03] BENOIT A., BRENNER L., FERNANDES P., PLATEAU B., « Aggregation of Stochastic Automata Networks with Replicas », *Proceedings of the International Conference on the Numerical Solution of Markov Chains (NSMC'03)*, Illinois, September 2-7 2003, p. 215–234.
- [BOU 02] BOUZEGHOUB M., KLOUL L., MOKHTARI A., « A New Active Network Framework Based on Active Rules », Rapport de Recherche n° 21, 2002, PRISM.
- [CAL 98] CALVERT K., « Architectural Framework for Active Networks », Rapport de recherche, Juillet 1998, AN Architecture Working Group.
- [CLA 01] CLARK G., COURTNEY T., DALY D., DEAVOURS D., DERISAVI S., DOYLE J. M., SANDERS W. H., WEBSTER P., « The Möbius Modeling Tool », *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, Aachen, Germany, septembre 2001, p. 241–250.
- [DON 94] DONATELLI S., « Superposed Generalised Stochastic Petri Nets : Definition and Efficient Solution », SILVA M., Ed., *Proc. of 15th Int. Conf. on Application and Theory of Petri Nets*, 1994.
- [GIL 94] GILMORE S., HILLSTON J., « The PEPA Workbench : A Tool to Support a Process Algebra-based Approach to Performance Modelling », *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, vol. 794, Vienna, 1994, Lecture Notes in Computer Science, Springer-Verlag, p. 353-368.
- [GIL 03] GILMORE S., HILLSTON J., KLOUL L., RIBAUDO M., « PEPA nets : A structured performance modelling formalism », vol. 54, 2003, p. 79-104, Performance Evaluation, Elsevier Science.
- [HAR 99] HARTMAN J., PETERSON L., BAVIER A., BIGOT P., BRIDGES P., B. M., PILTZ R., PROCEBSTING T., SPATSCHECK O., « Joust : A Platform for Liquid Software », *IEEE Computer Magazine*, , 1999, p. 50-56.
- [HIL 94] HILLSTON J., *A Compositional Approach to Performance Modelling*, PhD. Thesis, The University of Edinburgh, 1994.
- [HIL 02] HILLSTON J., KLOUL L., « An Efficient Kronecker Representation for PEPA Models », *Proceedings of Process Algebra and Probabilistic Methods : Performance Modelling and Verification*, vol. 2165, Aachen, 2002, Lecture Notes in Computer Science, Springer-Verlag, p. 120-135.
- [HIL 04] HILLSTON J., KLOUL L., MOKHTARI A., « Towards a Feasible Active Networking Scenario », *Proceedings of the 3rd IEEE International Conference on Networking*, Pointre-à-Pitre, Guadeloupe, France, 1-4 March, 2004.

- [KWI 02] KWIATKOWSKA M., NORMAN G., PARKER D., « PRISM : Probabilistic Symbolic Model Checker », FIELD T., HARRISON P., BRADLEY J., HARDER U., Eds., *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, vol. 2324 de LNCS, Springer, 2002, p. 200–204, <http://www.cs.bham.ac.uk/dxp/prism/>.
- [MER 00] MERUGU S., BHATTACHARJEE S., ZEGURA E., CALVERT K., « Bowman : A node OS for Active Networks », *Proceedings of IEEE Infocom'00*, Tel Aviv, Israel, March 2000.
- [MOR 99] MORRIS R., *Scalable TCP congestion Control*, PhD. Thesis, Harvard University, Cambridge, Massachusetts, 1999.
- [PLA 85] PLATEAU B., « On the Stochastic Structure of Parallelism and Synchronisation Models for Distributed Algorithms », *Proc. ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, 1985.
- [SAN 91] SANDERS W., MEYER J., « Reduced Base Model Construction Methods for Stochastic Activity Networks », *IEEE Journal on Selected Areas in Communications*, vol. 9, n° 1, 1991, p. 25-36.
- [TUL 01] TULLMANN P., HIBLER M., LEPREAU J., « Janos : A Java-oriented OS for Active Networks », *IEEE Journal on Selected Areas of Communication*, vol. 19, n° 3, March 2001.

A. Sémantique opérationnelle structurée de PEPA

Les règles sémantiques, dans le style opérationnel structuré, sont décrites dans la figure 16 [HIL 94]. Les règles sont lues comme suit : si la ou les transitions au dessus de la ligne d'inférence peuvent être inférées, alors la transition sous la ligne peut être inférée aussi. La notation $r_\alpha(E)$ définit le taux apparent de α dans E .

<p>Préfixe</p> $\frac{}{(\alpha, r).E \xrightarrow{(\alpha, r)} E}$
<p>Coopération</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{E \bowtie_L F \xrightarrow{(\alpha, r)} E' \bowtie_L F} \quad (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha, r)} F'}{E \bowtie_L F \xrightarrow{(\alpha, r)} E \bowtie_L F'} \quad (\alpha \notin L)$ $\frac{E \xrightarrow{(\alpha, r_1)} E' \quad F \xrightarrow{(\alpha, r_2)} F'}{E \bowtie_L F \xrightarrow{(\alpha, R)} E' \bowtie_L F'} \quad (\alpha \in L) \quad \text{où } R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$
<p>Choix</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{E + F \xrightarrow{(\alpha, r)} E'} \qquad \frac{F \xrightarrow{(\alpha, r)} F'}{E + F \xrightarrow{(\alpha, r)} F'}$
<p>Abstraction</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{E/L \xrightarrow{(\alpha, r)} E'/L} \quad (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha, r)} E'}{E/L \xrightarrow{(\tau, r)} E'/L} \quad (\alpha \in L)$
<p>Constante</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{A \xrightarrow{(\alpha, r)} E'} \quad (A \stackrel{def}{=} E)$

Figure 16. La sémantique opérationnelle de PEPA