Une analyse du protocole PGM avec UPPAAL[1]

Béatrice Bérard — Patricia Bouyer — Antoine Petit

LSV – CNRS UMR 8643 & ENS de Cachan,
61 avenue du Président Wilson,
F-94235 Cachan Cedex
{berard,bouyer,petit}@lsv.ens-cachan.fr

RÉSUMÉ. *Le protocole PGM (Pragmatic General Multicast) est un protocole de transport multipoint qui inclut des exigences de fiabilité. Il est conçu pour minimiser les redondances dues aux acquittements négatifs (NAK), ainsi que la surcharge du réseau par retransmission de paquets perdus. Ce protocole a été présenté à l'IETF (Internet Engineering Task Force) sous forme d'une spécification "ouverte". Ce travail est centré sur la principale propriété de fiabilité qui doit être garantie par PGM :* un récepteur reçoit tous les paquets de données transmis ou réémis après perte, ou bien il détecte les paquets perdus définitivement. *Pour vérifier cette propriété, nous proposons une modélisation (simplifiée) de PGM par un réseau d'automates temporisés. Nous examinons ensuite la validité de la propriété en utilisant le model-checker* UPPAAL. *Les résultats obtenus mettent en évidence certains paramètres, de la valeur desquels dépend la satisfaction de cette propriété.*

ABSTRACT. *Pragmatic General Multicast (PGM) is a reliable multicast protocol, designed to minimize both the probability of negative acknowledgements (NAK) implosion and the loading of the network due to retransmissions of lost packets. This protocol was presented to the Internet Engineering Task Force as an open reference specification. In this paper, we focus on the main reliability property which PGM intends to guarantee:* a receiver either receives all data packets from transmissions and repairs or is able to detect unrecoverable data packet loss. *To this purpose, we propose a modelization of (a simplified version of) PGM via a network of timed automata. Using* UPPAAL *model-checker, we then study the validity of the reliability property above, which turns out to not be always verified but to depend of the values of several parameters that we underscore.*

MOTS-CLÉS : *Vérification, model-checking, automates temporisés, protocole de communication.*

KEYWORDS: *Verification, model-checking, timed automata, communication protocols.*

1. Travail soutenu par le projet RNTL AVERROES (Analysis and VERification for the Reliability Of Embedded Systems).

1. Introduction

Depuis l'introduction des automates temporisés [ALU 90], de nombreux travaux ont été consacrés à l'étude théorique de modèles temporisés ou à leur analyse effective. La conception d'algorithmes de vérification a conduit au développement d'outils pour le model-checking temporisé, par exemple HYTECH [HEN 97], KRONOS [BOZ 98] or UPPAAL [LAR 97], qui ont été appliqués avec succès à de multiples études de cas. Dans ce travail, nous proposons la vérification de propriétés de fiabilité du protocole multipoint PGM.

Protocoles multipoint avec fiabilité. Ce type de protocole est conçu pour la distribution fiable d'information, depuis plusieurs sources vers plusieurs destinataires. Les applications cibles de cette technologie sont par exemple les diffusions video, la duplication de bases de données ou les téléchargements de logiciels. La fiabilité des protocoles point à point (comme TCP) est habituellement obtenue par l'envoi d'acquittements positifs (ACK) du récepteur vers la source. Lorsqu'on cherche à étendre cette méthode aux protocoles multipoint avec un nombre élevé de récepteurs, il en résulte une inondation du réseau par ces acquittements. Le développement des protocoles multipoint s'est donc tout d'abord attaché à l'élimination de ce type de messages, en conservant plutôt les acquittements négatifs (NAK), émis par les récepteurs en cas de perte de paquets. Cependant, au cours de phases de congestion du réseau, il se peut que de nombreux acquittements négatifs redondants soient émis, suivis de retransmissions elles-aussi redondantes.

Le protocole PGM. Ce protocole (Pragmatic General Multicast) fait partie de la deuxième génération de protocoles multipoint, conçus pour remédier aux problèmes mentionnés ci-dessus : en minimisant à la fois la probabilité d'explosion de NAKs et la charge du réseau due aux retransmissions de paquets perdus. La gestion des paquets mémorisés par la source utilise un nouveau type de paquets, appelés SPM (Source Path Message), ainsi que des temporisations sur la durée de conservation des paquets. Le protocole PGM a été développé conjointement par Cisco systems et TIBCO, et il a été présenté à l'IETF (Internet Engineering Task Force) sous forme d'une spécification "ouverte" [SPE 01]. Il est actuellement utilisé comme un prototype, généralement sur IP, et entre dans une phase de standardisation.

Contribution. Dans cet article, nous examinons la principale propriété de fiabilité que doit garantir PGM [SPE 01] : *un récepteur reçoit tous les paquets de données transmis ou réémis après perte, ou bien il détecte les paquets perdus définitivement*. La section 2 décrit les principales caractéristiques de PGM. Dans les sections 3 et 4, nous proposons un modèle simplifié du protocole sous forme d'un réseau d'automates temporisés. La méthode de vérification est exposée dans la section 5 et les conclusions dans la section 6.

2. Description du protocole PGM

Nous décrivons brièvement le protocole avec une seule source, conformément à la présentation de [SPE 01]. Nous omettrons les mécanismes concernant la charge du réseau, qui ne sont pas pertinents pour la fiabilité.

La source émet des paquets ordonnés de données : ODATA (Original Data), avec une certaine fréquence, dans une fenêtre de transmission. Les éléments du réseau sont organisés dans une arborescence et les paquets sont transmis le long d'un chemin. Si un récepteur détecte un paquet manquant, il émet répétitivement un acquittement négatif (NAK) vers le dernier élément du chemin. Celui-ci retransmet le NAK vers la source en suivant le chemin dans l'ordre inverse, et diffuse une confirmation (NCF). Lorsque le récepteur reçoit ce NCF, il arrête l'émission des NAK. Cette opération est répétée à chaque niveau du chemin de remontée vers la source. Les données manquantes doivent alors être renvoyées (RDATA pour Repair Data) par la source ou par un élément fixé (DLR pour Designated Local Repairer). Comme la fiabilité du protocole dépend fortement des NAKs et des NCFs, l'acheminement de ces messages est réalisé au niveau de la couche réseau. De plus, des messages SPM (Source Path Messages) sont émis par la source avec une certaine fréquence, en alternance avec les ODATA, de façon à :

1) maintenir des informations de routage, pour garantir que les NAKs remontent par le chemin inverse de celui qui a servi pour les ODATA et

2) fournir des informations sur les capacités de retransmission aux récepteurs.

Une architecture possible d'un réseau pour PGM est représentée sur la figure 1.

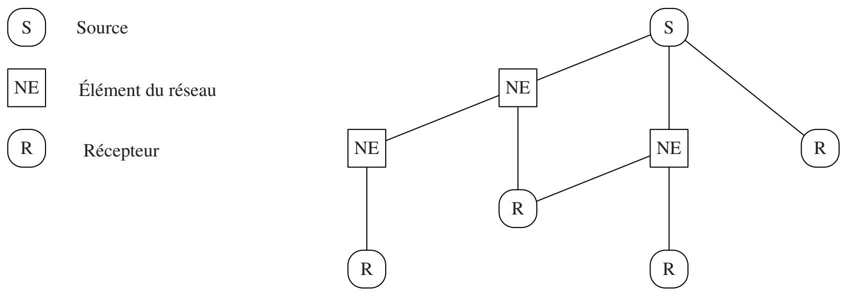


Figure 1. *Une architecture de réseau*

Fenêtre de transmission. Plus précisément, en supposant que les paquets sont de taille constante, on peut utiliser une numérotation circulaire des paquets ODATA de 0 à $2^{32} - 1$. La gestion des tampons de la source se fait au travers de la *fenêtre de transmission* :

– son extrémité gauche, txw_trail, est le numéro du plus ancien paquet disponible en réémission,

– son extrémité droite, txw_lead, est le numéro du dernier paquet envoyé.

La fenêtre est considérée comme vide si txw_trail = txw_lead + 1 et sa taille est bornée par $2^{16} - 1$. Les extrémités de la fenêtre sont incluses dans les messages SPM mais les stratégies de décalage de la fenêtre ne sont pas précisées. Chaque récepteur maintient également une fenêtre de réception, d'extrémités rxw_trail et rxw_lead, qui évolue en fonction des informations reçues de la source, soit par les paquets de données, soit par les SPMs.

Fréquences et priorités. La source envoie prioritairement les messages NCFs en attente, puis les SPMs en attente, et n'envoie des ODATA ou des RDATA qu'en l'absence de NCFs ou de SPMs. L'ordre entre ODATA et RDATA dépend des applications, et de la réception de NAKs. On distingue deux catégories de SPM : en présence de O/RDATA, des SPMs "d'ambiance" sont émis par la source avec une fréquence régulière. En l'absence de paquets de données, des SPMs "d'impulsion" sont émis avec une fréquence décroissante, pour maintenir à jour les fenêtres de réception et favoriser la détection des pertes.

3. Modélisation de PGM par des automates temporisés

Nous présentons maintenant la construction du modèle, en utilisant les caractéristiques d'UPPAAL [BEN 98].

3.1. *Modéliser avec* UPPAAL

Une présentation complète de l'outil UPPAAL peut être trouvée dans [LAR 97, BEN 98, AMN 01, SCH 01]. Nous en donnons rapidement les principales caractéristiques, qui seront illustrées dans la section 4.

Pour UPPAAL, un modèle consiste en un ensemble d'automates temporisés, qui communiquent par une synchronisation binaire, utilisant des canaux et une syntaxe du type émission/réception. Ainsi, sur le canal c, un émetteur envoie le signal c ! et un récepteur se synchronise avec lui par le signal complémentaire c ?. Les automates temporisés d'UPPAAL sont une variante des modèles originaux de [ALU 94]. Ce sont des structures finies manipulant deux types de variables : des horloges, qui évoluent de manière synchrone avec le temps, et des variables entières discrètes bornées. Un état de l'automate peut comporter une condition sur les horloges, appelée *invariant*, qui doit être satisfaite pendant toute la durée passée dans cet état. Une *transition* de l'automate est étiquetée par :

– une *garde*, qui exprime une condition sur les valeurs des variables. Cette condition doit être satisfaite pour franchir la transition,

– une *synchronisation* de la forme c ! ou c ?,

– une remise à zéro de certaines horloges et une mise à jour de certaines variables entières.

Chacun de ces trois éléments est optionnel. L'absence de synchronisation indique une action interne de l'automate.

Plus formellement, une *configuration* est une paire (l, v) où l (pour *location*) est un vecteur (*location*) indiquant l'état de chacun des automates et v est une valuation des variables, c'est-à-dire une application associant à chaque horloge une valeur réelle et à chaque variable discrète une valeur entière. Une exécution est une suite de configuration qui part d'un état initial de chacun des automates avec toutes les valeurs des variables à zéro. La sémantique de ce modèle contient trois types de changements de configurations (ou mouvements) :

– **Délai :** le temps peut progresser dans les états courants d'une durée d, à condition que l'invariant de chacun de ces états reste satisfait. Les valeurs des horloges augmentent de d et les valeurs des variables discrètes ne changent pas.

– **Synchronisation :** Si deux actions complémentaires de deux composants sont possibles, et que les gardes associées aux transitions sont satisfaites, ces deux composants se synchronisent. Les états correspondants sont modifiés, et les valeurs des horloges et des variables discrètes sont modifiées selon les indications de remises à zéro et de mises à jour.

– **Action interne :** Si une action interne (sans synchronisation) d'un composant est possible (sa garde est satisfaite), cette action peut être exécutée indépendamment des autres composants : l'état et les variables du composant sont modifiés comme dans le cas précédent.

Remarquons que les canaux d'UPPAAL permettent la synchronisation mais pas la communication de données. Pour simuler la transmission d'information sur un canal c, on définit deux variables globales info_envoi et info_reception. L'émetteur possède une transition d'étiquette c ! tandis que le récepteur possède une transition d'étiquette c ? avec la mise à jour info_reception := info_envoi. Ce mécanisme permet donc de modéliser la transmission de la valeur de la variable info_envoi sur le canal c.

Une autre caractéristique utile pour garantir des séquences atomiques de mouvements est la notion d'états *instantannés* ("committed" pour UPPAAL), qui seront étiquetés par c dans les figures. Dans un tel état, aucun délai n'est permis, ce qui impose un mouvement immédiat du composant concerné. Ainsi, deux transitions liées par un état instantanné seront exécutées sans délai intermédiaire.

3.2. *Un modèle simplifié*

Le modèle que nous construisons est une abstraction du protocole, qui omet les aspects non pertinents pour la propriété à vérifier. Cette étape est indispensable pour réduire la taille du modèle, et obtenir ainsi des performances acceptables en termes de consommation mémoire et de temps d'exécution. Cependant, le modèle doit conserver les informations nécessaires à l'apparition d'erreurs potentielles du protocole. Le travail réalisé montre que même avec de fortes réductions, des faiblesses inhérentes au protocole sont détectées.

Nouc considérons une topologie fixe pour le réseau, qui correspond exactement à l'arborescence de routage. Ceci permet de s'affranchir des informations nécessaires à la construction des chemins. De plus, comme nous l'avons mentionné dans la section 2, des mécanismes spécifiques assurent la fiabilité de transmission des NAKs. Nous pouvons donc supposer que ces messages ne sont pas perdus : un seul NAK est envoyé par un récepteur directement à la source et les NCFs sont omis. Nous supposons aussi que seule la source peut réémettre des paquets et qu'il n'y a pas d'autre DLR. Enfin, nous prenons un taux constant pour l'émission des SPMs, y compris en l'absence de données, et une taille constante des paquets. En combinant cette hypothèse avec le fait que nous ignorons le contenu des paquets, nous pouvons représenter les données par leur numéro. De façon similaire, un NAK consiste simplement en le numéro de la donnée manquante. La figure 2 montre un schéma minimal du réseau qui résume les simplifications proposées.

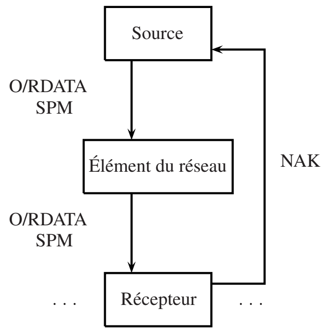


Figure 2. *Schéma simplifié*

Le modèle complet du protocole est obtenu par composition des différents composants. Une source ou un élément de réseau (appelé "transmission") sont représentés chacun par un automate temporisé tandis qu'un récepteur comporte deux automates qui interagissent de manière asynchrone. Le premier reçoit les données et les SPMs et le second détecte les pertes et envoie les NAKs. Nous obtenons ainsi un ensemble de $2 + 2n$ automates temporisés, pour n récepteurs.

La structure du modèle est présentée dans la figure 3. Les canaux sont représentés par des arcs joignant un émetteur et le récepteur associé, les variables globales utilisées pour les transferts de données entre parenthèses. Les principales variables discrètes sont indiquées par le style `machine à écrire`, avec des crochets [] pour les tableaux. Les variables d'indices ou les variables de moindre importance sont omises. Les horloges sont en *italique*.

Les communications fonctionnent de la manière suivante. La source communique avec un élément du réseau en envoyant les O/RDATA et les SPMs *via* les canaux "src_ne_data" et "src_ne_spm" respectivement. Sur le premier canal ("src_ne_data"), la variable “src_sqn” indique le numéro de la donnée émise. Sur le second canal, les variables “src_trail” and “src_lead” correspondent aux extrémités de la fenêtre de transmission. À son tour, un élément de réseau peut transmettre les mêmes informations, avec des canaux et des variables similaires. L’envoi de NAKs d’un récepteur vers la source suit le même schéma.

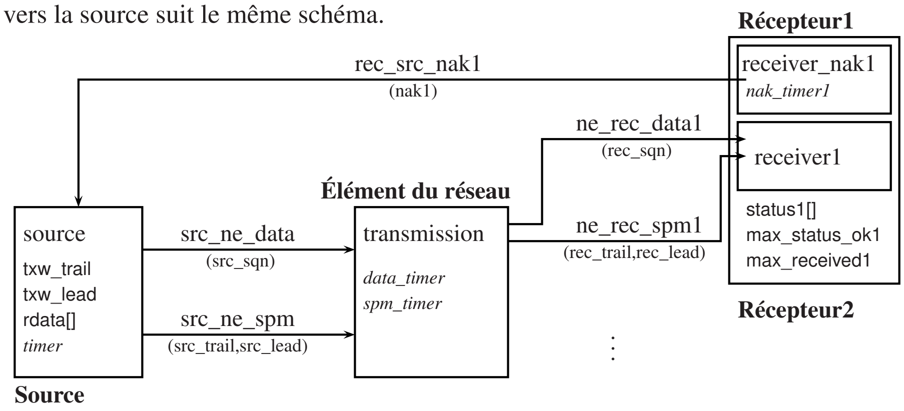


Figure 3. *Structure du modèle*

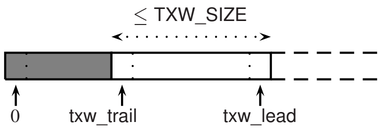
4. Composants du modèle

Nous décrivons maintenant les composants et, pour simplifier, nous supposons que le modèle comporte un seul élément de réseau et deux récepteurs.

4.1. *La source*

Elle réalise trois opérations : l’envoi des SPMs, qui contiennent les informations sur la fenêtre de transmission, l’envoi des O/RDATA, avec une mise à jour de la fenêtre et la réception des NAKs, avec mise à jour de l’ensemble des données RDATA en attente.

La fenêtre de transmission est modélisée par les deux variables entières txw_trail et txw_lead, qui donnent ses extrémités respectives : le numéro du plus ancien paquet disponible pour rémission et celui du plus récent paquet émis. La taille maximale de la fenêtre correspond au paramètre TXW_SIZE, de sorte que la fenêtre est pleine lorsque txw_lead - txw_trail = TXW_SIZE. Dans ce cas, toutes les RDATA en attente sont envoyées et la fenêtre est décalée : txw_trail est incrémentée de la valeur du paramètre TXW_ADVANCE. Une configuration de la fenêtre de transmission est représentée ci-dessous :



L'ensemble des RDATA en attente est représenté par le tableau rdata. L'automate de la figure 4 décrit les opérations de la source.

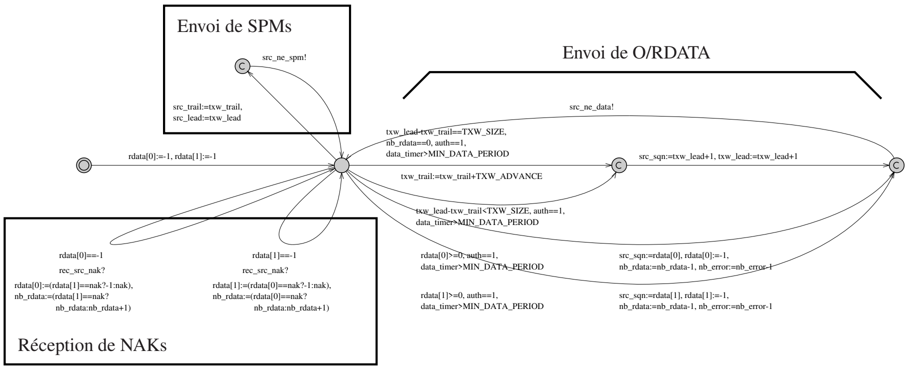


Figure 4. *Modèle de la source*

Lors de la réception d'un NAK, la source conserve son numéro dans le tableau rdata. l'envoi de SPMs est réalisé périodiquement, sans modification des structures de données de la source. L'émission de données est la partie la plus compliquée : si la fenêtre n'est pas pleine (*i.e.* txw_lead - txw_trail < TXW_SIZE), alors la source envoie des ODATA vers le réseau (la périodicité d'émission est paramétrée par l'entier MIN_DATA_PERIOD et testée par l'horloge timer) ; si la fenêtre est pleine (*i.e.* txw_lead - txw_trail = TXW_SIZE), alors la source n'envoie plus de ODATA, mais les RDATA en attente. Lorsque ces émissions sont terminées, la fenêtre est décalée avant de nouveaux envois de ODATA. L'émission de RDATA impose une mise à jour du tableau rdata (partie droite de l'automate).

Remarque. La figure ci-dessus illusitre les principales caractéristiques des automates temporisés présentés informellement dans la section 3.1. La transition en bas à gauche est étiquetée par une garde sur la variable rdata[0] (cette variable doit valoir -1), une synchronisation "rec_src_nak ?" et une mise à jour des deux variables discrètes rdata[0] and nb_rdata. Cette transition est franchie dès que la condition est vraie et qu'un autre composant peut se synchroniser avec l'action complémentaire rec_src_nak !. Dans la partie "Envoi de SPMs", on trouve un état étiqueté par "Committed" : en effet, les deux mises à jour consécutives doivent être atomiques. Dans la partie "Réception de NAKs", on trouve un raccourci syntaxique d'UPPAAL : "value :=(cond ?value1 :value2)" signifie que si cond est vraie, alors value prend la valeur value1, sinon elle prend la valeur value2.

4.2. *Un récepteur*

Un récepteur réalise deux tâches indépendantes : il reçoit des données (avec mise à jour de la fenêtre de réception) et envoie des NAKs lorsqu'il détecte un paquet manquant. Les deux automates qui forment le récepteur ne sont donc pas synchronisés.

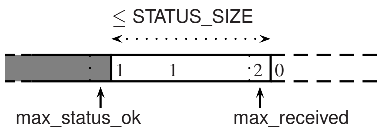
La principale structure de données du récepteur est la fenêtre de réception. Dans le protocole original, d'autres structures sont ajoutées pour prendre en compte les pertes de paquets non consécutifs. Ici, nous utilisons un seul mécanisme pour les deux opérations : un tableau circulaire status dont la taille est donnée par le paramètre STATUS_SIZE. Deux numéros sont conservés : max_received représente le numéro le plus élevé dans l'ensemble des données reçues et max_status_ok est le numéro le plus élevé de l'ensemble des données dont le statut est connu du récepteur. Pour tout paquet de numéro inférieur ou égal à max_status_ok, le récepteur sait s'il a été reçu ou s'il est perdu. Une entrée du tableau status contient :

– 0 si le récepteur n'a pas d'information sur la donnée,

– 1 si le récepteur doit envoyer un NAK pour cette donnée,

– 2 si la donnée est reçue.

La fenêtre de réception peut être décrite par le schéma ci-dessous :



Les figures 5 et 6 décrivent les deux parties du récepteur.

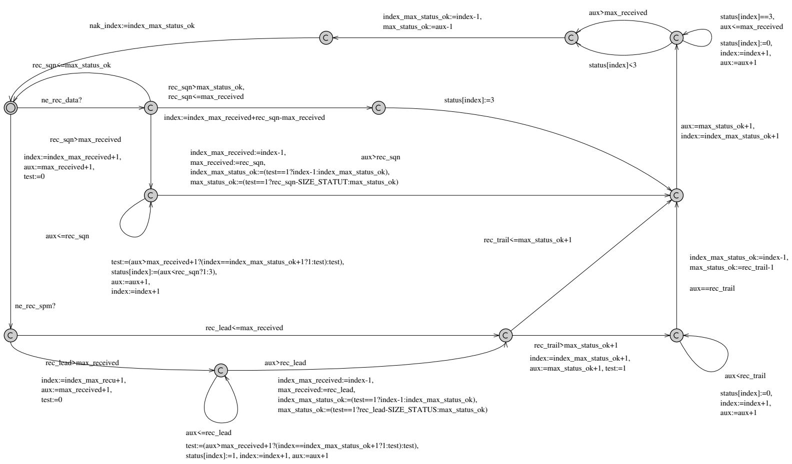


Figure 5. *Réception des données et des SPMs*

Lorsque le récepteur reçoit une donnée ou un SPM, il met à jour le tableau status de la fenêtre de réception. Là-encore, certaines transitions sont enchaînées par des états “committed”, afin de rendre atomiques les actions correspondantes. Ceci a aussi l’avantage de réduire l’ensemble des états accessibles dans le système complet, en restreignant le non-déterminisme.

Le deuxième module du récepteur (figure 6) réalise l’envoi des NAKs. Il examine le tableau status et lorsqu’une donnée est détectée comme manquante (l’entrée correspondante est 1), il envoie un NAK. Les NAKs sont émis avec une fréquence donnée par le paramètre FREQUENCE_NAK, dont la valeur est testée par l’horloge nak_timer. Lorsque l’opération est inutile, la boucle de l’état initial remet cette horloge à zéro, en attendant une nouvelle détection.

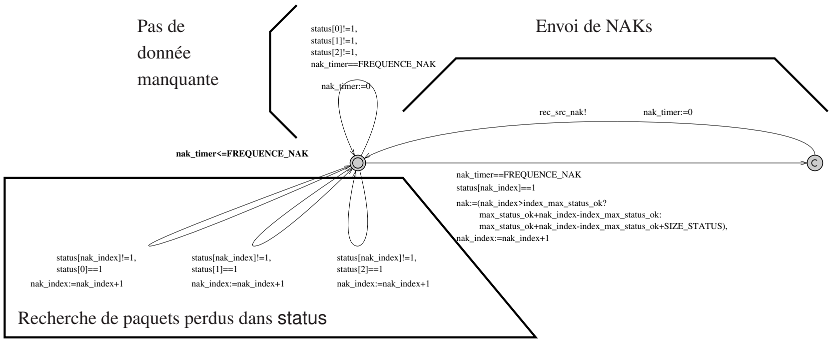


Figure 6. *Envoi des NAKs*

4.3. *Un élément de réseau*

Un élément de réseau transmet les données et les SPMs aux récepteurs. Nous l’utilisons également pour simuler les pertes éventuelles et les délais de transmission. Le modèle correspondant à deux récepteurs est représenté dans la figure 7.

La partie de l’automate située au-dessus de la transition diagonale réalise la transmission de données, la transition courbe sur la droite représentant une perte. Comme nous l’avons déjà expliqué dans la section 3.1, la synchronisation binaire d’UPPAAL ne permet pas de modéliser directement la diffusion à deux récepteurs. Nous avons là-encore utilisé un état “committed” pour effectuer les deux envois sucessifs de façon atomique. La seconde partie de l’automate transmet les SPMs de manière similaire, sans perte, en garantissant l’envoi d’un SPM après chque envoi de donnée. La transition diagonale correspond au cas de SPMs isolés, avec une fréquence donnée par la valeur du paramètre MIN_SPM_PERIOD.

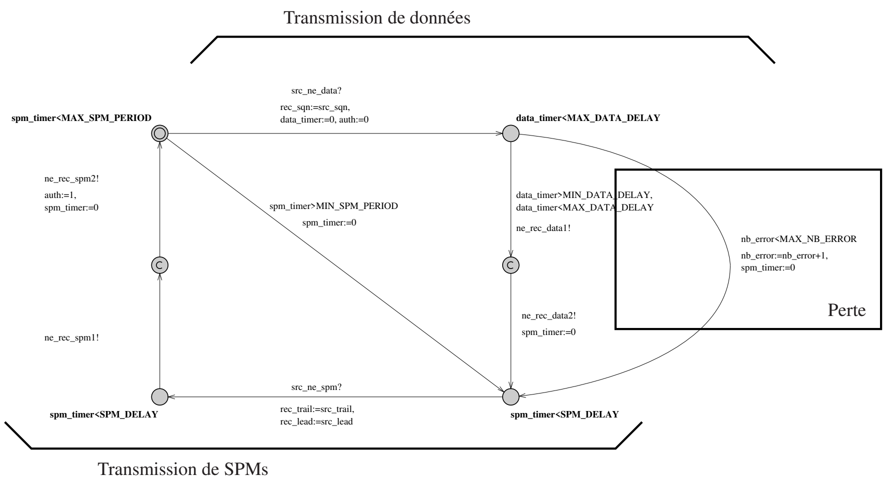


Figure 7. *Élément de réseau pour deux récepteurs*

5. Vérification de propriétés

5.1. *Les propriétés*

La première propriété à vérifier est celle qui est spécifiée dans [SPE 01] : « Pour chaque donnée, tout récepteur sait s'il l'a reçue ou s'il ne la recevra jamais », et désignée dans ce qui suit par **Info-perte**. Dans le modèle décrit précédemment, Cette propriété peut être vérifiée en utilisant un automate observateur (ou automate de test, figure 8), qui compare la fenêtre de réception avec la fenêtre de transmission.

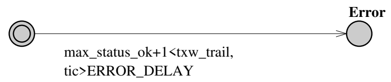


Figure 8. *Automate* obs *pour le test de* **Info-perte**

La caractéristique de l'automate de test est que la propriété **Info-perte** n'est pas vérifiée si l'état Error est accessible. En effet, la donnée d'indice max_status_ok+1 est la première dont le statut soit inconnu du récepteur. Si cette donnée n'est plus présente dans la fenêtre de transmission, ce statut restera définitivement inconnu. Comme la borne inférieure de la fenêtre de transmission est txw_trail, cette condition s'exprime par : max_status_ok+1<txw_trail, qui est la condition pour atteindre l'état Error. De plus, une horloge tic ainsi qu'un paramètre ERROR_DELAY sont ajoutés au récepteur. L'horloge est remise à zéro à chaque mise à jour de la fenêtre de réception, de façon à calculer le délai de réaction du réseau, qui est représenté par la valeur du paramètre ERROR_DELAY : c'est la durée minimale nécessaire à une

donnée pour atteindre un récepteur depuis la source. L'automate observateur ne peut donc atteindre l'état Error que si (en plus de la condition ci-dessus) un délai suffisant s'est écoulé depuis la dernière mise à jour de la fenêtre de réception.

La seconde propriété que nous examinons est l'absence de perte de données, notée **Pas-de-perte** : "toute donnée manquante sera finalement reçue". Comme la spécification [SPE 01] le laisse clairement apparaître, cette propriété est généralement fausse. Nous montrons toutefois qu'elle peut être garantie sous certaines hypothèses dans le modèle simplifié. La négation de la propriété s'exprime dirctement dans ce modèle par le fait que, pour un récepteur donné, aucun état dans lequel test vaut 1 n'est accessible. En effet, ceci se produit lors d'une mise à jour du tableau status, si la variable max_status_ok doit être incrémentée alors que la donnée manquante qui correspond n'a jamais été reçue.

5.2. *Expériences*

Nous utilisons l'outil de model-checking UPPAAL[1] pour implémenter le modèle décrit précédemment. Les paramètres de ces expériences sont présentés dans le tableau 1. Les deux premiers concernent la topologie du réseau (nombre de récepteurs et d'éléments de réseau). Les six suivants sont liés à la taille des structures de données (bornes des entiers, taille des tableaux, etc.). Enfin, sept autres paramètres sont destinés aux temporisations du trafic sur le réseau. Les valeurs des paramètres qui seront fixées sont indiquées dans le tableau. Un point d'interrogation indique un paramètre dont la valeur pourra être modifiée.

5.2.1. *Vérification de la propriété* **Info-perte**

Nous présentons les résultats obtenus dans le cas d'un seul récepteur. [2] Le modèle comporte 960 états de contrôle (dans le produit synchronisé), 5 horloges et 25 variables bornées.

Le tableau 2 résume les résultats obtenus pour la propriété **Info-perte** pour un récepteur. Les valeurs des paramètres sont :

SPM_DELAY	1
MIN_DATA_DELAY	2
MAX_DATA_DELAY	3
MIN_SPM_PERIOD	3
MAX_SPM_PERIOD	5
MIN_DATA_PERIOD	1
NAK_PERIOD	2

1. Version 3.2.4 of UPPAAL2k, voir http ://www.uppaal.com
2. Après 28 heures de calcul, la vérification sur un Sun Ultra 420R avec 4096 MB de RAM a été stoppée.

Paramètres du protocole		
Nombre de récepteurs		1 ou 2
Nombre d'élements du réseau		1
Paramètres de taille		
Nbre max de données émises par la source	MAX_NB_DATA	10
Nombre max de RDATA en attente	MAX_NB_RDATA	2
Nombre max de données perdues	MAX_NB_ERROR	2
Taille de la fenêtre de réception	STATUS_SIZE	3
Taille de la fenêtre de transmission	TXW_SIZE	5
Décalage	TXW_ADVANCE	2
Paramètres de temps		
Délai de transmission SPM	SPM_DELAY	?
Borne inf. de la période d'envoi de SPM	MIN_SPM_PERIOD	?
Borne sup. de la période d'envoi de SPM	MAX_SPM_PERIOD	?
Borne inf. du délai de transmission données	MIN_DATA_DELAY	?
Borne sup. du délai de transmission données	MAX_DATA_DELAY	?
Période d'envoi de données	MIN_DATA_PERIOD	?
période d'envoi de NAK	NAK_PERIOD	?

Tableau 1. *Paramètres des expériences*

ERROR_DELAY	**5**	**6**	**7**	**8**	**9**	**9**	**10**	**11**	**12**
Réponse	**Vrai**					**Faux**	**Faux**		
Options	Breadth-first (Recherche en largeur du graphe) Active-clock reduction (Élimination des horloges redondantes)						Breadth-first Active-clock reduction Convex-hull approximation[3]		
Temps (en s)	193	898	911	1007	1753	550	556	552	555
Mémoire (en KB)	112464	445232	449776	480728	771280	222312	222312	222312	222312

Tableau 2. *Résultats pour* **Info-perte**

3. À chaque étape du calcul, si deux zones accessibles ont été obtenues pour un même état de contrôle, l'outil conserve la plus petite zone qui les contient. Avec cette surapproximation, un état détecté comme accessible peut ne pas l'être en réalité. Cette approximation ne peut être utilisée que pour une réponse négative.

La réponse **Vrai** signifie que la propriété E<>(obs.Error) [4] est vérifiée. Ainsi, après attente du délai ERROR_DELAY, le récepteur peut faire une erreur et penser qu'il pourra recevoir une donnée manquante, alors que celle-ci n'est plus présente dans la fenêtre de transmission. La propriété **Info-perte** n'est donc pas vérifiée. [5]

En examinant le tableau 2, on peut remarquer que la valeur 9 est la limite entre **Vrai** et **Faux**. Cette valeur peut donc être considérée comme le délai de réaction du réseau, c'est-à-dire le temps pour qu'un récepteur reçoive l'information sur la fenêtre de transmission. En conclusion, un récepteur sait toujours en un temps borné qu'il ne recevra pas une donnée perdue. Ce délai dépend de tous les paramètres de temps dans le réseau (délai de transmission des données, des SPMs, fréquence d'émission des SPMs, etc.).

5.2.2. *Vérification de la propriété* **Pas-de-perte**

Les résultats (tableau 3, dans les mêmes conditions matérielles) ont été obtenus avec deux récepteurs. Le modèle a 17280 états de contrôle, 5 horloges et 35 variables bornées, avec les valeurs suivantes des paramètres :

SPM_DELAY	1
MIN_DATA_DELAY	2
MAX_DATA_DELAY	3
MIN_SPM_PERIOD	3
MAX_SPM_PERIOD	5
NAK_PERIOD	3

MIN_DATA_PERIOD	**1**	**2**	**3**	**4**
Answer of UPPAAL	**Vrai**	**Vrai**	**Faux**	**Faux**
Options	Breadth-first Active-clock reduction		Breadth-first Active-clock reduction Convex-hull approximation	
Temps (en s)	4743	5775	918	9 39
Mémoire (en KB)	1239832	1402200	630608	597440

Tableau 3. *Résultats pour la propriété* **Pas-de-perte**

4. Dans la syntaxe d'UPPAAL, cette formule exprime le fait que l'état Error de l'automate obs est accessible

5. UPPAAL a été exécuté sur un Sun Ultra 220R, avec une CPU de 2x450 MHz et une RAM de 2048 MB, sous Solaris 8. Cependant, UPPAAL n'est pas réparti donc un seul des deux processeurs est utilisé.

La valeur **Vrai** dans le tableau s'interprète comme “le modèle vérifie la formule E<>(receiver1.test==1 or receiver2.test==1)” [6], c'est-à-dire que la propriété **Pas-de-perte** n'est pas vérifiée.

Commentaires. Le fait que la propriété **Pas-de-perte** soit fausse dans certains cas n'est pas une surprise. Elle n'est d'ailleurs pas exigée du protocole PGM qui doit plutôt garantir que le récepteur connaît le statut des données. les résultats obtenus confirment que cette propriété n'est pas vérifiée si la fréquence d'émission des données (paramètre MIN_DATA_PERIOD) est trop grande. De plus, UPPAAL fournit une trace d'exécution dans les cas d'erreur, qui correspond au scénario suivant : la source envoie une RDATA vers le réseau. Il est possible que cette donnée soit à nouveau perdue, alors que la source l'a effacée du tableau rdata en avançant la fenêtre de transmission. Le prochain NAK informant la source de cette perte arrive trop tard.

Les résultats du tableau 3 montrent également que les données ne spont pas perdues si le débit d'émission est lent. C'est une propriété forte, qui peut être assurée par certaines valeurs des paramètres.

Nous avons également testé une propriété plus forte : A[](receiver1.test==0 and (source.nb_rdata ==0 or source.live<LIVENESS)) où nb_rdata est le nombre de RDATA en attente à la source, live est une nouvelle horloge associée à la source, qui est remise à zéro à chaque envoi de donnée, et LIVENESS est un paramètre qui mesure la vivacité des émissions. Cette propriété exprime le fait que, dans toute configuration accessible, aucune détection de perte n'est réalisée (la variable test vaut 0) et, soit il n'y a pas de RDATA en attente, soit le réseau est inactif depuis moins de LIVENESS unités de temps.

Cette propriété a été testée pour différentes valeurs des deux paramètres LIVENESS et MIN_DATA_PERIOD et il s'avère qu'elle n'est jamais vérifiée. Ainsi, même pour des valeurs de MIN_DATA_PERIOD pour lesquelles il n'y a pas de perte, si le réseau est inactif (pas de donnée transmise), les RDATA peuvent être en attente indéfiniment. Dans ce cas, même si la source n'efface pas ces données, le récepteur ne reçoit pas les données manquantes.

6. Conclusion

Dans ce travail, nous proposons un modèle à base d'automates temporisés pour représenter une version simplifiée du protocole PGM. Ce modèle est construit à partir de la spécification fournie dans [SPE 01], par une composition synchronisée des différents composants : source, éléments de réseau et récepteurs. Les caractéristiques importantes des automates temporisés incluent les synchronisations avec émissions/réceptions, les contraintes d'horloges et la notion d'état “committed”.

6. Qui exprime sous UPPAAL l'existence d'un état accessible dans lequel l'une des deux variables test des récepteurs vaut 1.

Les étapes d'implémentation ont été réalisées avec le model-checker temporisé UPPAAL, et facilitées par l'interface graphique et le module de simulation présents dans cet outil. La phase de vérification a permis d'obtenir des traces d'exécution, qui fournissent des contre exemples dans les cas où les propriétés ne sont pas vérifiées.

Remarquons que même si le modèle décrit une version très simplifiée du proptocole PGM, il permet de montrer comment la propriété de fiabilité qui doit être garantie par PGM *"un récepteur ou bien reçoit tous les paquets émis ou réémis, ou bien peut détecter les pertes définitives"* est vraie lorsqu'un délai de réaction suffisant est autorisé pour le récepteur. Enfin, notons que même avec un modèle simple, nous avons été confrontés avec d'importants problèmes de mémoire. Il ne paraît donc pas raisonnable d'espérer traiter une version plus élaborée du protocole par cette méthode. En revanche, il serait intéressant de poursuivre l'étude du modèle proposé, en particulier du point de vue des valeurs des paramètres, ce qui conduirait à une meilleure compréhension de PGM.

7. Bibliographie

[ALU 90] ALUR R., DILL D., « Automata for Modeling Real-Time Systems », *Proc. 17th International Colloquium on Automata, Languages, and Programming (ICALP'90)*, vol. 443 de *Lecture Notes in Computer Science*, Springer-Verlag, 1990, p. 322–335.

[ALU 94] ALUR R., DILL D., « A Theory of Timed Automata », *Theoretical Computer Science*, vol. 126, n° 2, 1994, p. 183–235.

[AMN 01] AMNELL T., BEHRMANN G., BENGTSSON J., D'ARGENIO P. R., DAVID A., FEHNKER A., HUNE T., JEANNET B., LARSEN K. G., MÖLLER O., PETTERSSON P., WEISE C., YI W., « UPPAAL – Now, Next, and Future », *Proc. Modelling and Verification of Parallel Processes (*MOVEP*2k)*, vol. 2067 de *Lecture Notes in Computer Science*, Springer-Verlag, 2001, p. 99–124.

[BEN 98] BENGTSSON J., LARSEN K. G., LARSSON F., PETTERSSON P., YI W., WEISE C., « New Generation of UPPAAL », *Proc. International Workshop on Software Tools for Technology Transfer*, 1998.

[BOZ 98] BOZGA M., DAWS C., MALER O., OLIVERO A., TRIPAKIS S., YOVINE S., « KRONOS : a Model-Checking Tool for Real-Time Systems », *Proc. 10th International Conference on Computer Aided Verification (CAV'98)*, vol. 1427 de *Lecture Notes in Computer Science*, Springer-Verlag, 1998, p. 546–550.

[HEN 97] HENZINGER T. A., HO P.-H., WONG-TOI H., « HYTECH : A Model-Checker for Hybrid Systems », *Journal of Software Tools for Technology Transfer*, vol. 1, n° 1–2, 1997, p. 110–122, Springer-Verlag.

[LAR 97] LARSEN K. G., PETTERSSON P., YI W., « UPPAAL in a Nutshell », *Journal of Software Tools for Technology Transfer*, vol. 1, n° 1–2, 1997, p. 134-152, Springer-Verlag.

[SCH 01] SCHNOEBELEN P., BÉRARD B., BIDOIT M., LAROUSSINIE F., PETIT A., *Systems and Software Verification - Model-Checking Techniques and Tools*, Springer-Verlag, 2001.

[SPE 01] SPEAKMAN T. et al., « PGM Specification », 2001, Internet draft of the IETF (Internet Engineering Task Force), 115 pages.