# Lot 4.2

# Technologie de modélisation

***Probabilités***

# Manipulation de backoff dans CSMA/CA

| | |
|---|---|
| **Description :** | Le protocole Wi-Fi régit les connexions sans fil de plusieurs utilisateurs à un canal. Ce protocole demande aux utilisateurs de patienter un certain temps en cas de collision. Cependant des utilisateurs peuvent tenter de réduire ce temps d'attente, pénalisant ainsi les autres utilisateurs. Pour contrer ce phénomène de triche, un protocole nommé DOMINO a été proposé. Il s'agit de prouver l'intérêt de DOMINO en utilisant PRISM puis APMC. |
| **Auteur(s) :** | L. Fribourg, C. Picaronny, S. Pinot |
| **Référence :** | Averroes / Lot 4.2 / Fourniture 6 / V1.0 |
| **Date :** | mars 2006 |
| **Statut :** | validé |
| **Version :** | 1.0 |

**Réseau National des Technologies Logicielles**
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

# Historique

| | | |
|---|---|---|
| 18 novembre 2005 | V 1.0 | brouillon |
| mars 2006 | V 1.0 | mise au format averroes |

# Table des matières

**1 Introduction** **3**

**2 Description de CSMA/CA** **3**

**3 Description de DOMINO** **3**
3.1 Le tricheur . . . 3
3.2 DOMINO . . . 4

**4 Modélisation par le langage de PRISM** **4**
4.1 Les modules . . . 5
4.2 Modélisation de DOMINO . . . 7
4.3 Modélisation de la pénalisation . . . 7

**5 Résultats avec PRISM** **8**
5.1 Propriétés vérifiées par PRISM . . . 8
5.2 Gain du tricheur . . . 8
5.3 Potentiel de DOMINO à détecter les tricheurs . . . 8

**6 Utilisation d'APMC** **9**
6.1 APMC . . . 9
6.2 Amélioration du modèle . . . 9

**7 Résultats avec APMC** **10**
7.1 Détection par DOMINO . . . 10
7.2 Pénalisation . . . 10

**8 Appendice** **11**
8.1 Résultats avec PRISM . . . 11
8.2 Résultats avec APMC . . . 16

# 1 Introduction

Le protocole Wi-Fi (techniquement 802.11) régit les accès sans fil de plusieurs utilisateurs à un canal. Par exemple, les Hot Spot sont des bornes Wi-Fi auxquelles des clients peuvent se connecter pour accéder à internet. La couche MAC (Media Access Control) de ce protocole est le protocole CSMA/CA. Ce protocole est comparable à CSMA/CD mais prend en compte les contraintes d'une connexion sans fil. Des utilisateurs tentant de communiquer en même temps se brouillent mutuellement et le protocole leur dicte alors d'attendre un certain temps aléatoire avant de retenter une transmission, ce temps étant choisi dans un intervalle grandissant en fonction du nombre de collisions.

Cependant, les composants où est implémenté le protocole sont suffisamment réglables par un utilisateur averti pour lui permettre de manipuler ce protocole. On peut ainsi passer outre ce temps d'attente ou le diminuer à sa guise. On aura alors un débit supérieur aux autres utilisateurs, rendant le système non équitable.

Pour contrer ces tricheurs, un protocole a été proposé par Maxim Raya, Jean-Pierre Hubaux et Imad Aad dans [10]. Il s'appelle DOMINO. Une observation statistique d'une simulation de DOMINO a été faite par Kandaraj PIAMRAT [9] dans son mémoire de Master.

Le but de ce rapport est donc de prouver certaines propriétés de ce type de triche et de DOMINO. Pour ce faire, nous utiliserons dans un premier temps PRISM, puis APMC.

# 2 Description de CSMA/CA

Plusieurs protocoles peuvent être qualifiés de CSMA (Carrier Sense Multiple Access). C'est le cas de CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) et CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Ces protocoles ont un point commun : un utilisateur peut observer le canal et ainsi savoir s'il est occupé ou libre. Quand un utilisateur désire envoyer un message, si le canal est occupé, il tentera plus tard son envoi. Sinon, il peut utiliser le canal. Cependant, dans ces protocoles, il peut y avoir des collisions si des utilisateurs observent le canal libre en même temps.

Dans le cas d'utilisateurs se connectant à un canal physique (comme un câble), on peut utiliser la détection de collisions (CSMA/CD). Ce système demande que tout utilisateur puisse détecter qu'il y a eu une collision. Mais ceci n'est pas réalisable dans le cas d'un réseau sans fil car deux utilisateurs peuvent s'ignorer tout en accédant au canal. C'est pourquoi la méthode d'évitement de collision est choisie : un utilisateur ayant observé le canal libre attend qu'il soit libre pendant un certain temps DIFS. S'il l'est, il peut envoyer un message indiquant qu'il est prêt à transmettre. Si le canal est libre, il envoie un accusé de réception à l'utilisateur lui indiquant qu'il peut envoyer son message. À la fin de son envoi, il attend une réponse du canal lui disant que son message a été bien reçu. Sinon, il conclut qu'il y a eu une collision et retente l'envoi après un temps d'attente aléatoire, appelé de temps de *backoff*, qui augmente avec le nombre de collisions. Ainsi il y a plus de chance d'éviter les collisions, au prix de temps d'attente supplémentaires.

Les détails du protocoles peuvent être trouvés dans [2].

# 3 Description de DOMINO

Le protocole DOMINO (system for Detection Of greedy behavior in the MAC layer of IEEE 802.11 public NetwOrks) créé par Maxim Raya, Jean-Pierre Hubaux et Imad Aad dans [10] a pour but de détecter les éventuels tricheurs dans le protocole CSMA/CA. On va d'abord préciser ce qu'on entend par tricheur.

## 3.1 Le tricheur

Plusieurs types de triche sont possibles dans CSMA/CA. Mais nous nous focalisons sur un type particulier de triche : la manipulation de backoff. Celle-ci consiste à diminuer son temps de backoff, ce qui permet de prendre la main plus vite que les autres et de perdre moins de temps en attente. Quand le non tricheur choisit son backoff dans l'intervalle $[0, Max(bc)]$, ($Max(bc)$ dépendant de $bc$ le nombre de collisions), le tricheur le choisit dans un intervalle plus petit $[0, Max(bc)\frac{100-n}{100}]$, $n$ étant défini comme son pourcentage de triche.

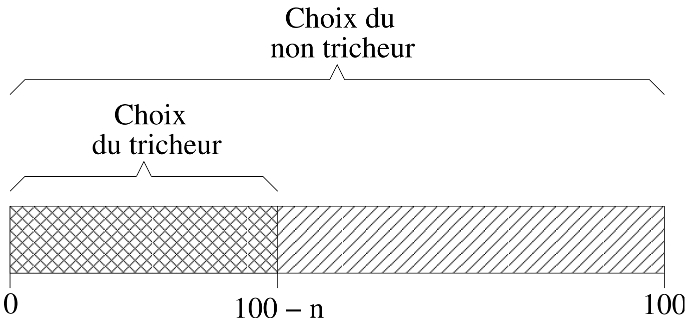


FIG. 1 – backoff du tricheur

De ce fait, la détection du tricheur ne peut être que probabiliste car un non tricheur peut, avec une certaine probabilité, effectuer les mêmes choix de backoff qu'un tricheur. La détection sera donc basée sur une étude statistique.

### 3.2 DOMINO

Le protocole DOMINO est un ensemble de tests qui retourne une valeur booléenne $test_i$ pour chaque utilisateur $i$. Un test valide une suspicion de triche. Il peut prendre plusieurs formes (nous détaillerons celle que par la suite nous étudieront). Chaque $test_i$ a un compteur $triche_i$ et un paramètre $K$. L'algorithme de DOMINO est le suivant :

**if** $test_i = true$ **then**
  $triche_i := triche_i + 1$
  **if** $triche_i > K$ **then**
    La station est détectée comme étant un tricheur
    Appeler la fonction de pénalisation
  **end if**
**else if** $triche_i > 0$ **then**
  $triche_i := triche_i - 1$
**end if**

Dans le but de diminuer les détections erronées, $triche_i$ doit atteindre la valeur $K$ pour que la station soit détectée comme étant un tricheur. Pour récompenser les stations qui suivent le protocole, $triche_i$ est décrémenté chaque fois que $triche_i$ n'est pas vérifié.

La fonction de pénalisation peut être appelée quand une station est détectée comme étant un tricheur ($triche_i = K$). Elle a pour but de pénaliser le tricheur pour limiter son gain, et de permettre un accès équitable aux stations qui suivent le protocole.

Comme nous étudions un seul type de triche, nous nous restreignons au test qui nous intéresse. Ce test consiste à comparer la valeur moyenne du backoff de la station (*backoff réel*) et la valeur moyenne du backoff d'une station suivant le protocole (*backoff attendu*) multipliée par une constante $\alpha$ (on prendra $\alpha = 0.9$). Si La moyenne de la station est inférieure à la moyenne d'une station qui suit le protocole multipliée par $\alpha$, le test est vrai.

## 4 Modélisation par le langage de PRISM

PRISM (PRobabilistIc Symbolic Model checker) [5] est un model checker probabiliste permettant de vérifier des propriétés d'un processus de décision Markovien [3] entre autre. Or CSMA/CA

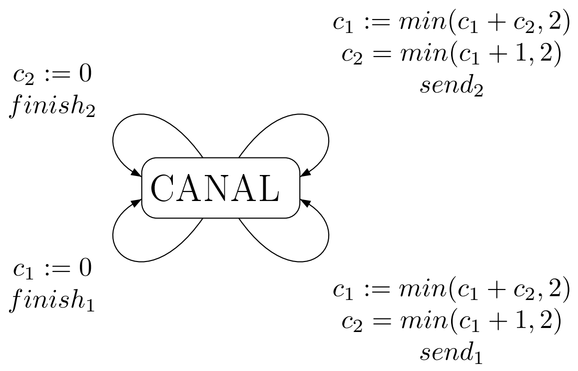


FIG. 2 – Module du canal pour 2 utilisateurs

peut être modélisé par un processus de décision Markovien. En effet, le système allie probabilisme (par le choix du backoff par exemple) et non déterminisme (par la taille des paquets envoyés par exemple). Le langage de PRISM est basé sur les *reactive modules* [1]. C'est une description du modèle à l'aide de plusieurs modules pouvant interagir et avoir des actions synchronisées (un manuel [8] est en ligne). Chaque module comporte deux parties : une déclaration des variables locales et un ensemble de commandes. Les commandes sont de cette forme :

$$[\textit{étiquette}]garde- > p_1 : update_1 + \cdots + p_n : update_n;$$

La *garde* est une formule logique utilisant toutes les variables du système. Une $update_k$ est une mise à jour des variables du module. Si la garde est satisfaite, une $update_k$ est choisie aléatoirement suivant les probabilités spécifiées par $p_k$. Les commandes ayant la même étiquette s'exécutent de façon synchronisée.

Nous utilisons la modélisation créée par les auteurs de PRISM [6][7]. Nous nous restreignons à deux utilisateurs (nommés 1 et 2) ne tentant l'envoi que d'un message. Pour CSMA/CA, le choix a été fait de considérer un module représentant l'état du canal (par qui il est occupé), un module par utilisateur et éventuellement un module pour le temps. Des constantes représentent des temps d'attente présents dans CSMA/CA. Elles sont choisies proportionnellement avec leurs valeurs réelles.

## 4.1 Les modules

Le modèle est constitué de quatre modules : un module pour chaque station (représenté par la **figure 3**), un pour le canal (représenté par la **figure 2**) et un pour le temps. Les deux stations sont similaires, donc nous ne décrirons que la station 1.

**Module canal**

Ce module représente l'état du canal. Il a deux variables $c1$ et $c2$ respectivement pour les utilisateurs 1 et 2. Les valeurs de ces variables définissent l'état du canal :
- $c1 = 0$ : rien n'est envoyé par 1.
- $c1 = 1$ : 1 envoie un message sans brouillage.
- $c1 = 2$ : 1 envoie un message mais il a été brouillé.

Deux formules ont été définies : *busy* et *free*. Elles traduisent l'état du canal à partir des variables de ce module.

**Module station1**

Ce module décrit les événements liés à 1. Voici sa liste de variable :
- $s1$ : l'état de 1.
- $x1$ : mesure des écoulements de temps.
- $slot1$ : temps de backoff.

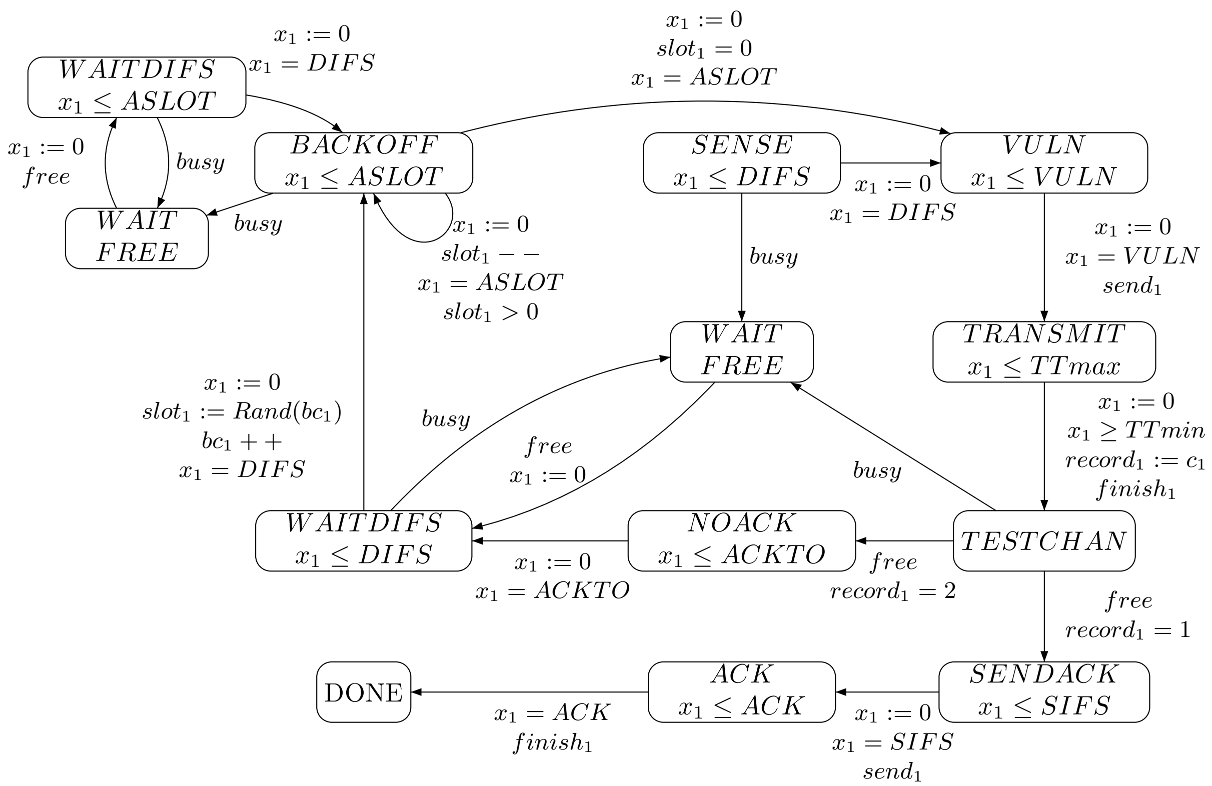


FIG. 3 – Module de l'utilisateur 1

– $bc1$ : le compteur de backoff.

La station commence dans l'état $SENSE$. Si la garde $free$ est vérifiée durant DIFS, la station entre dans l'état $VULN$ pour commencer l'envoi de son message. Si le canal est occupé, la station entre dans la procédure de backoff. Le temps d'envoi est choisi de manière non déterministe dans $[TTmin, TTmax]$. L'étiquette $send_1$ synchronise l'évolution des modules de la station et du canal de manière à donner à la variable $c1$ la valeur correspondante (2 si l'envoi sera brouillé, 1 sinon). Après l'envoi, la valeur de $c1$ est enregistrée par $record1$ et la station entre dans l'état $TESTCHAN$. Si la garde $busy$ est vérifiée, la station entre dans la procédure de backoff. Sinon, la station attend durant ACK. Si $record_1 = 1$, la station recevra ACK et aura envoyé son message avec succès. Si $record_1 = 2$, la station ne recevra pas ACK et entrera dans la procédure de backoff.

Dans la procédure de backoff, la station attend que le canal soit libre durant DIFS. Puis, elle choisit aléatoirement la valeur de $slot1$. Si le canal reste libre durant ASLOT, $slot1$ est décrémenté. Sinon, la station attend que le canal soit libre, puis attend durant DIFS et reprend la procédure. Quand $slot1 = 0$, la station entre dans l'état $VULN$ et tente l'envoi de son message.

**Module temps**

Il représente l'écoulement du temps réel par la variable $t$ et permet de vérifier des propriétés dépendant donc du temps. Toutes les actions faisant augmenter le temps sont étiquetées par une même étiquette *time*. Alors, si $t$ n'est pas égal au temps maximal, $t$ est incrémenté. Ainsi, les actions non étiquetées par *time* ne font pas augmenter $t$ (comme les actions du calcul du temps de backoff par exemple). Cette variable est limité par un temps maximum appelé *deadline*. Quand $t = deadline$, le système se fige (le système est dans un état absorbant).

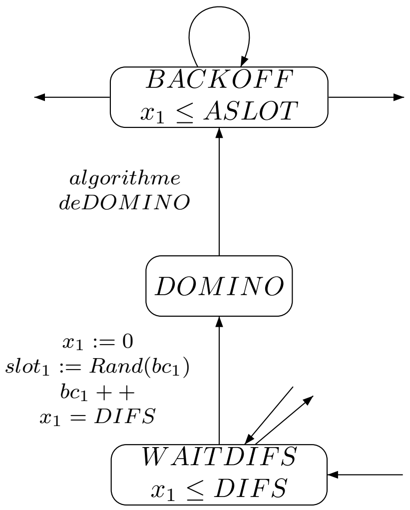


FIG. 4 – Modification pour introduire $DOMINO$

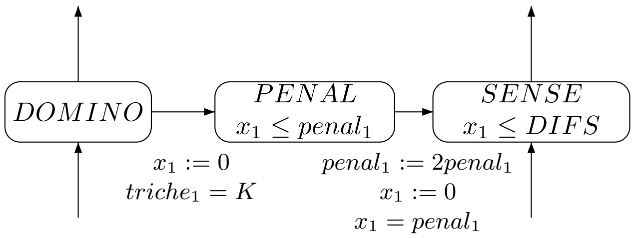


FIG. 5 – Modification pour introduire la fonction de pénalisation

## 4.2 Modélisation de DOMINO

Pour modéliser DOMINO, on introduit un nouvel état $DOMINO$. Une fois le backoff choisi, la station entre dans cet état. Les valeurs moyennes du backoff réel et attendu sont mises à jour. Puis l'algorithme de DOMINO s'effectue pour modifier la valeur de $triche_1$. Enfin, l'utilisateur passe à la procédure de backoff.

Les modifications nécessaires à l'introduction de DOMINO sont illustrées dans la **figure 4**.

## 4.3 Modélisation de la pénalisation

Nous proposons la fonction de pénalisation suivante : détecté comme tricheur, un utilisateur doit attendre un temps $penal_i$. À chaque nouvelle pénalisation, $penal_i$ est doublé. On espère ainsi qu'un tricheur sera rapidement pénalisé et qu'un non tricheur, en cas de détection erronée, sera peu pénalisé. Le modèle est alors modifié pour introduire cette fonction de pénalisation. Quand on invoque la fonction de pénalisation (à partir de l'état $DOMINO$), on se met dans un état d'attente $PENAL$. On y reste un temps $penal_1$, puis on retourne dans $SENSE$. À chaque pénalisation, le temps $penal_1$ est doublé. Les modifications du modèle sont illustrées dans la **figure 5**.

# 5 Résultats avec PRISM

Les résultats obtenus par PRISM portent sur le gain du tricheur et la détection de DOMINO.

## 5.1 Propriétés vérifiées par PRISM

PRISM permet de vérifier des propriétés du système décrites par une formule de PCTL. Mais il permet aussi de calculer des probabilités d'événements. Le système étant non déterministe, la probabilité d'un événement n'est pas uniquement définie. C'est donc les probabilités maximales ou minimales qui seront calculées.

À titre d'exemple, la probabilité minimale pour que 1 ait envoyé son message avec succès au temps $T$ va être calculée en soumettant à PRISM la formule :

Pmin= ? [ true U (s1=12 & t<=T) ]

## 5.2 Gain du tricheur

Nous voulons observer l'effet de la manipulation de backoff sur la capacité d'envoi du tricheur. Pour cela, nous avons calculé la probabilité minimum pour chacun d'avoir envoyé son message en un temps au plus $T$.

Cependant, la modélisation doit être adaptée pour observer le gain du tricheur. En effet, pour des petites valeurs du nombre de collisions, le temps de backoff est petit. Ainsi le tricheur qui diminue ce temps ne modifie pas son comportement de manière sensible quand il y a peu de collisions. Or il est peu probable d'avoir, pour ce modèle à deux utilisateurs, un nombre élevé de collisions. Ceci se traduit par des résultats très peu sensibles au pourcentage de triche. On peut observer ce phénomène en comparant les **figure 6** et **10**, qui sont très proches. Pour résoudre ce problème, nous avons modifié le modèle en se plaçant dans la situation où il y a déjà eu des collisions. On peut observer l'influence des collisions initiales en comparant les **figure 9** et **13**.

Nous avons ainsi calculé les probabilités d'atteindre l'état final en fonction du temps pour les valeurs de triche de 50% et 75% et pour le cas où il y a déjà eu 0, 1, 2 et 3 collisions. On calcule ces probabilités pour le tricheur, pour le non tricheur, et pour le cas de deux utilisateurs non tricheurs. Les graphes sont fournis dans l'appendice (**figures 6** à **13**).

Les résultats montrent le gain du tricheur. En effet, à un instant donné, la probabilité pour que le tricheur ait envoyé son message est plus grande que celle du non tricheur. De plus, le calcul de la probabilité d'envoi du message dans le cas de deux utilisateurs non tricheurs permet de voir que le tricheur a toujours une probabilité d'envoi plus forte que la probabilité d'envoi en l'absence de triche. De même, le non tricheur a toujours une probabilité plus faible d'envoyer que la probabilité d'envoi en l'absence de triche.

D'autre part, l'augmentation de la triche a pour conséquence une augmentation du gain du tricheur. En effet, le tricheur à 75% a une plus forte probabilité d'envoi que le tricheur à 50%. De plus, le non tricheur en face du tricheur à 75% a une probabilité plus faible d'envoi que le non tricheur en face du tricheur à 50%. On peut, pour cela, comparer les **figures 6**, **7**, **8** et **9** avec respectivement les **figures 10**, **11**, **12** et **13**.

## 5.3 Potentiel de DOMINO à détecter les tricheurs

Pour modéliser DOMINO, nous avons ajouté aux stations un compteur de triche borné au niveau du backoff. Cependant nous avons du nous limiter dans sa modélisation pour des raisons pratiques. En effet, les modèles sont déjà de taille importante et ajouter un compteur de triche ne fait qu'augmenter la taille du modèle, déjà rapidement soumise à une explosion combinatoire. Nous limitons ce compteur de triche sera limité à deux.

Nous avons calculé la probabilité maximale pour que le compteur de triche atteigne 1 et 2 dans les cas de triche à 25% et 50% (les cas 50% et les suivants sont très similaire au niveau de DOMINO), pour le tricheur et le non tricheur et dans les cas de 0, 1, 2 et 3 collisions au départ.

Cependant pour des petites valeurs de collisions initiales, les probabilités obtenues sont très faible car le backoff est trop faible pour que DOMINO puisse vraiment différencier le tricheur du non tricheur. Les tableaux sont fournis dans l'appendice.

Les résultats montrent que DOMINO détecte plus le tricheur que le non tricheur. De plus, plus la triche est importante, plus la détection de DOMINO est efficace.

# 6 Utilisation d'APMC

Comme nous l'avons dit, nous avons du nous restreindre sur le modèle (le nombre de messages envoyés, le nombre d'utilisateurs et les paramètres de DOMINO) car la taille du modèle était trop importante. C'est pourquoi nous avons utilisé le model checker APMC, ce qui permet de mieux prendre en compte la taille du système. On peut trouver plus de précision sur APMC dans [4].

## 6.1 APMC

APMC (Approximate Probabilistic Model Checker) est un model checker probabiliste . Il permet de vérifier des propriétés sur une DTMC. APMC utilise une méthode de Monte-Carlo efficace pour approximer la probabilité d'une propriété monotone exprimée en LTL.

Le non déterminisme du modèle ne peut donc plus être pris en compte et est alors remplacé par des choix probabilistes uniformes.

Pour calculer $Prob(F|longueur = k)$ la probabilité d'une formule $F$ de LTL sur les chemins de longueur $k$, on doit vérifier cette formule sur tout les chemins de longueur $k$. Cependant, le nombre exponentiel de chemins rend les calculs rapidement impossibles. Une autre méthode consiste alors à ne calculer qu'une valeur approchée de cette probabilité. C'est ce que fait APMC de manière efficace.

APMC prend en entrée le code d'un modèle (le même code que celui de PRISM), une formule $F$ de LTL, la longueur $k$ des chemins et la taille $N$ de l'échantillon et renvoie le tirage d'une variable aléatoire $X_k(F)$ qui est une approximation de $Prob(F|longueur = k)$. Dans le cas où $F$ est monotone, on peut approcher $Prob(F)$, la probabilité pour que $F$ soit satisfaite sur des chemins quelconques, en calculant $X_k(F)$ pour un $k$ grand.

Le fonctionnement d'APMC est le suivant : APMC construit un programme `C` qui simule le modèle, représentant chaque variable et effectuant les transitions selon le code. Puis il exécute $k$ transitions de ce programme un nombre $N$ de fois en comptant le nombre d'exécutions ayant vérifiés la formule $F$. Enfin APMC renvoie le nombre d'exécutions vérifiant la formule divisée par $N$.

De cette manière, on peut prouver, avec une borne de Chernoff [4], qu'APMC renvoie bien le tirage d'une variable aléatoire qui approche $Prob(F|longueur = k)$ dans le sens suivant :

$$Prob[|X_k(F) - Prob(F|longueur = k)| \leq \epsilon] \geq 1 - \delta$$

Avec $N = \frac{4log\left(\frac{2}{\delta}\right)}{\epsilon^2}$.

Ainsi, APMC a de nombreux avantages :
- Pas de construction d'un modèle, juste un programme `C` de taille proche de celle du code.
- Complexité faible en fonction du modèle et des paramètres de précision.
- Possibilité de déployer APMC sur tout un réseau.
- Même langage que PRISM.

## 6.2 Amélioration du modèle

Le but de l'utilisation d'APMC étant d'obtenir des résultats pour un modèle plus proche de la réalité, nous avons apporter au modèle les modifications suivantes :
- Les utilisateurs ont une infinité de messages en attente.
- De 2 à 5 utilisateurs : L'intérêt du rajout d'utilisateurs est d'obtenir plus de collisions afin que DOMINO ait plus de données pour calculer une moyenne.

– Meilleure modélisation de DOMINO : Le compteur de triche n'est plus limité et on calcule réellement la moyenne des backoffs.

# 7 Résultats avec APMC

Tout d'abord, nous avons validé le passage à un modèle probabiliste en nous assurant que les probabilités issues de ce modèle étaient bien comprises entre les probabilités minimales et maximales du modèle non déterministe.

Puis, nous avons pu calculer des probabilités sur un modèle plus proche de la réalité.

## 7.1 Détection par DOMINO

Nous avons calculé la probabilité pour que le compteur de triche atteigne une certaine valeur, dans le cas de 25%, 50% et 75% de triche et pour 2, 3, 4 et 5 utilisateurs. Ces résultats sont exposés respectivement **figures 14**, **15**, **16** et **17**. On en tire les conclusions suivants :

Pour toutes les valeurs du compteur de triche, la probabilité du tricheur est supérieure à celle du non tricheur, ce qui montre la capacité de DOMINO à détecter les tricheurs. Bien sûr, les deux probabilités baissent quand la valeur du compteur augmente, mais on constate que la probabilité du non tricheur baisse plus vite que celle du tricheur, ce qui permet de trouver des valeurs de $K$ (la valeur à partir de laquelle la pénalisation commence) assurant une bonne probabilité de détection de la triche et une faible détection erronée du non tricheur.

La probabilité pour que le compteur du tricheur atteigne une certaine valeur augmente avec le pourcentage de triche. Ce qui montre que DOMINO détecte plus précisément ceux qui trichent fortement.

La détection erronée des non tricheurs (le faux positif) semble indépendante du comportement du tricheur.

## 7.2 Pénalisation

Pour observer les conséquences de la fonction de pénalisation, nous avons calculé les probabilités pour qu'un utilisateur envoie au moins $k$ messages. On calcule ces probabilités sans pénalisation puis avec pénalisation et ce pour le tricheur et un non tricheur. Les graphes correspondants sont disponibles en appendice pour des triches de 25%, 50% et 75% respectivement aux **figures 18**, **19** et **20**. Les paramètres sont 4 utilisateurs, $K = 3$ et une pénalisation initiale de 256 unité de temps.

Les résultats nous permettent de comparer la capacité d'envoi du tricheur avant et après pénalisation, ce qui permet de savoir si la fonction de pénalisation contraint bien le tricheur. Puis l'on compare la probabilité d'envoi du tricheur et celle du non tricheur après pénalisation, ce qui permet de savoir si le tricheur est maintenant défavorisé par rapport au non tricheur. Enfin la probabilité d'envoi du non tricheur après pénalisation est supérieure à la probabilité d'envoi avant. Donc le non tricheur n'est pas touché par la pénalisation et retrouve un accès correct au canal suite à la pénalisation du tricheur.

Les résultats constatés nous permettent de dire que notre fonction de pénalisation répond bien aux exigences énoncées.

# 8 Appendice

## 8.1 Résultats avec PRISM

Voici les graphes des probabilités minimum pour avoir atteint l'état final au temps $k$ pour des triches de 50% et 75% et pour le cas où il y a déjà eu 0, 1, 2 et 3 collisions.

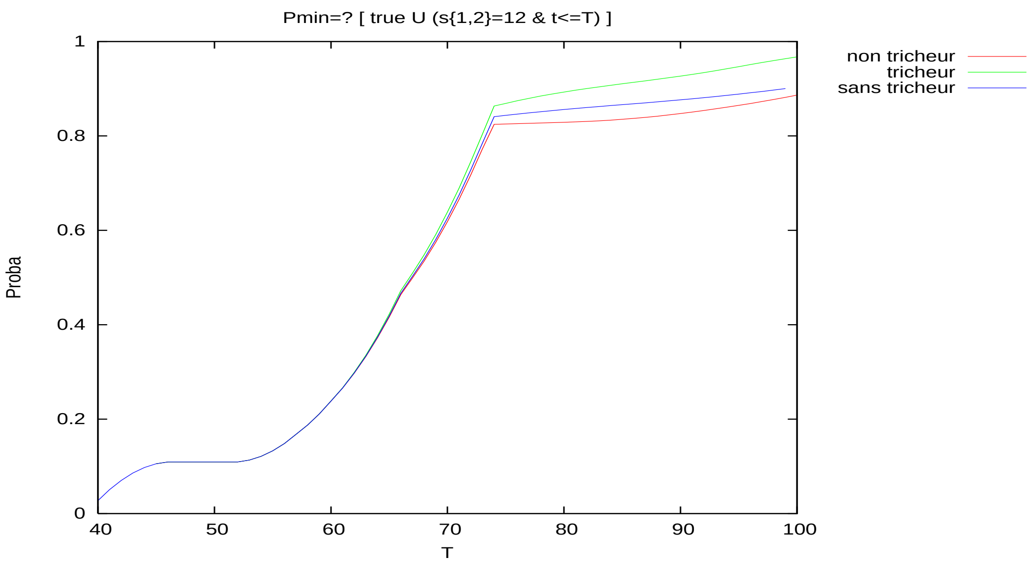


FIG. 6 – Triche à 50%, 0 collision

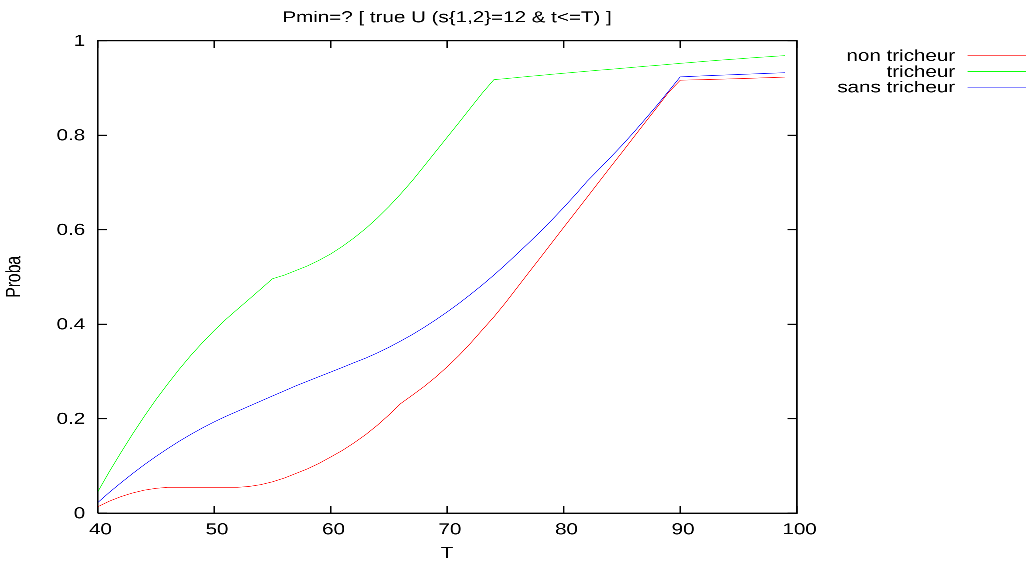


FIG. 7 – Triche à 50%, 1 collision

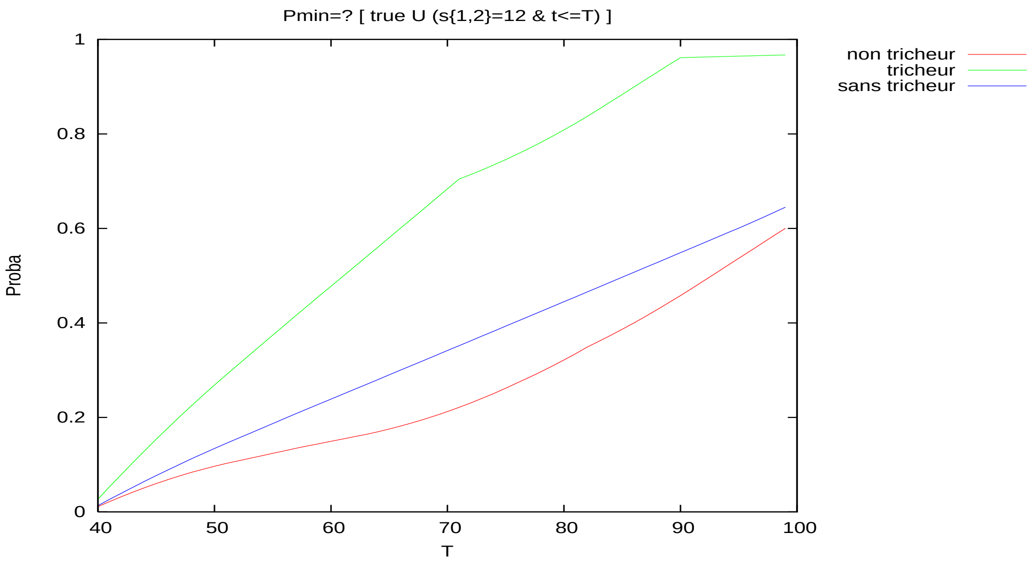


FIG. 8 – Triche à 50%, 2 collisions

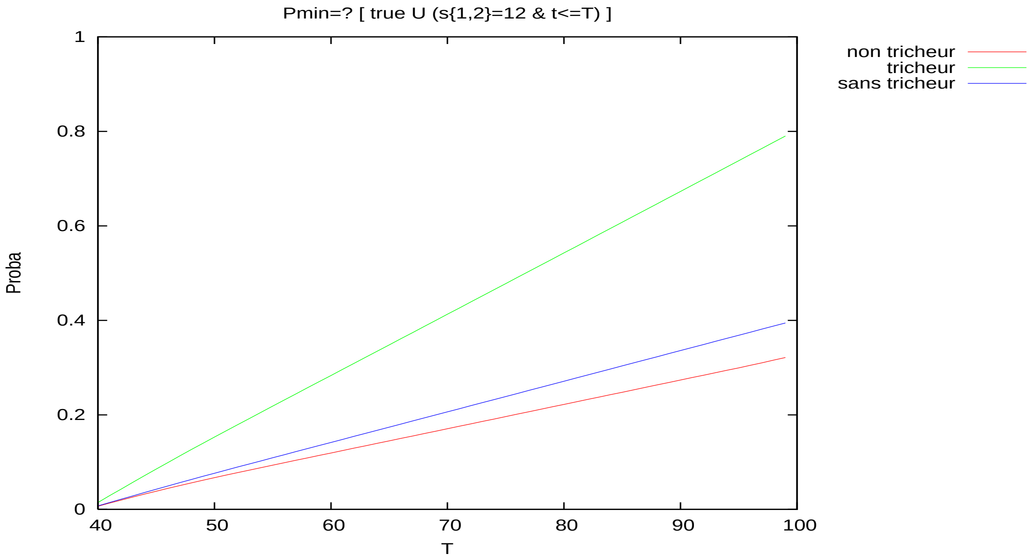


FIG. 9 – Triche à 50%, 3 collisions

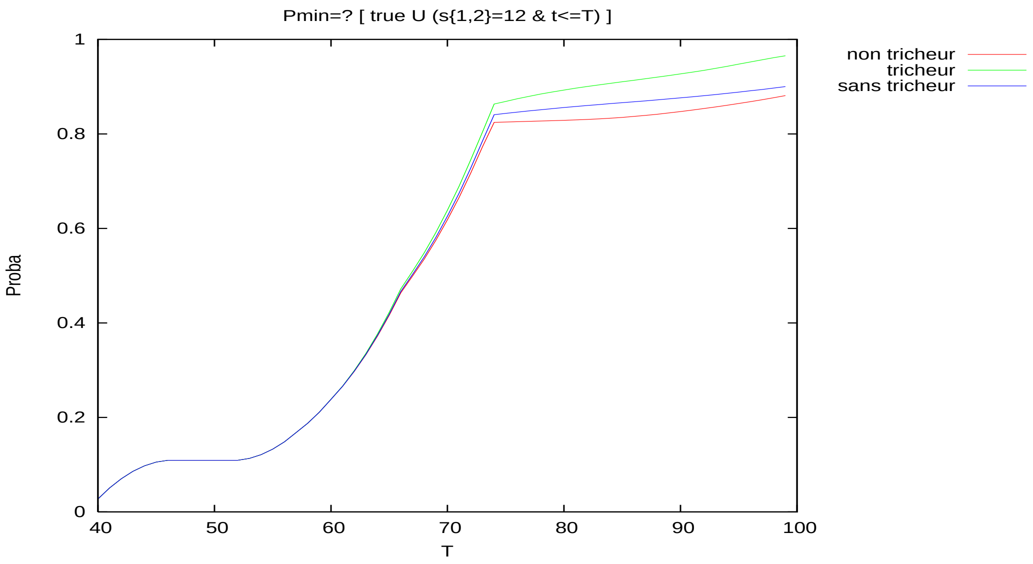


Fig. 10 – Triche à 75%, 0 collision

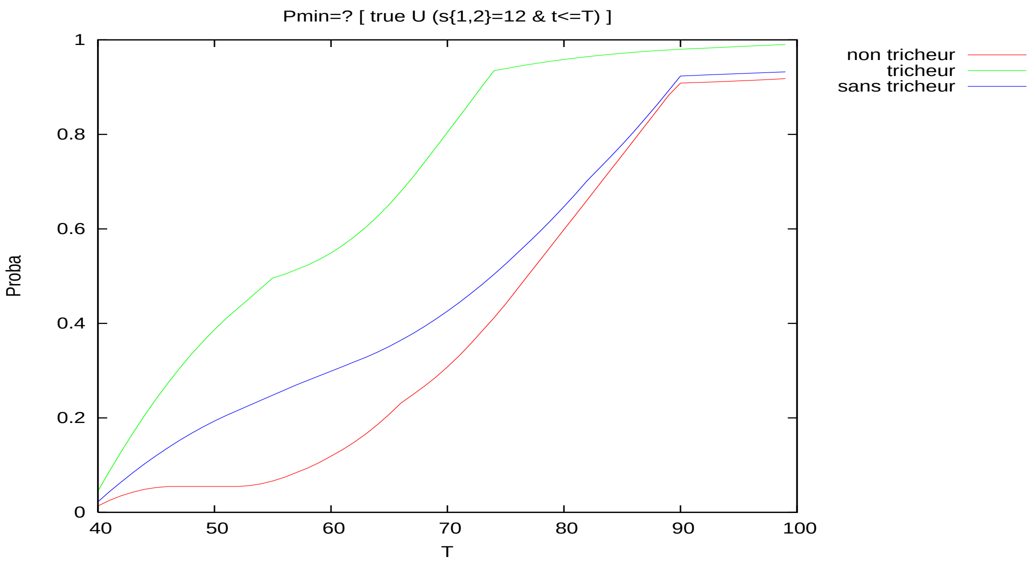


Fig. 11 – Triche à 75%, 1 collision

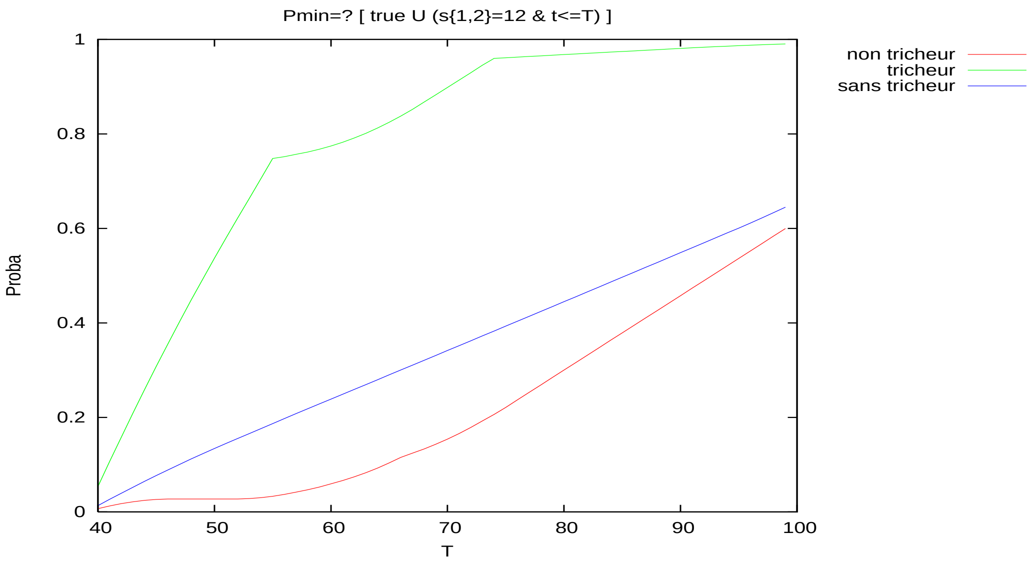


FIG. 12 – Triche à 75%, 2 collisions

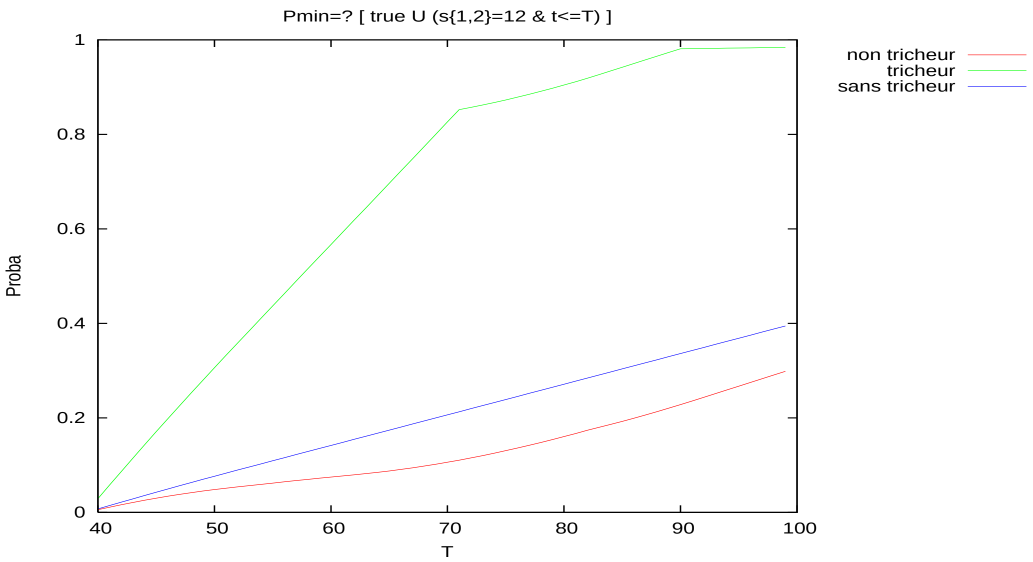


FIG. 13 – Triche à 75%, 3 collisions

Voici les tableaux des valeurs de la probabilité maximale pour que le compteur de triche ait la valeur 1 et 2. On prend en compte les triches de 25% et 50% et on débute avec 2 et 3 collisions.

25% de triche

| | | triche=1 | triche=2 | taille du modèle |
|---|---|---|---|---|
| 2 collisions | tricheur | 0.032 | 5.1E-4 | 9 043 363 |
| | non tricheur | 0.024 | 3.8E-4 | 8 475 880 |
| 3 collisions | tricheur | 0.670 | 0.011 | 9 025 166 |
| | non tricheur | 0.505 | 0.008 | 8 457 683 |

50% de triche

| | | triche=1 | triche=2 | taille du modèle |
|---|---|---|---|---|
| 2 collisions | tricheur | 0.048 | 1.15E-3 | 6 352 531 |
| | non tricheur | 0.024 | 0.57E-3 | 5 544 505 |
| 3 collisions | tricheur | 1.0 | 0.024 | 6 339 401 |
| | non tricheur | 0.5 | 0.012 | 5 531 375 |

## 8.2 Résultats avec APMC

Voici les graphes des probabilités pour que le compteur de triche aille jusqu'à $k$ pour des triches de 25%, 50% et 75% et pour 2 à 5 utilisateurs.

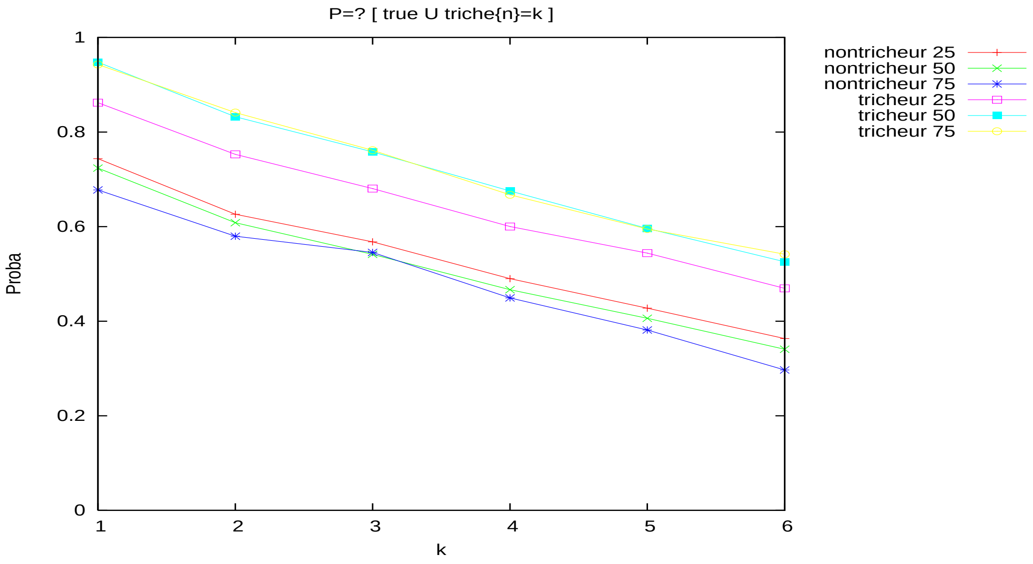


FIG. 14 – 2 utilisateurs

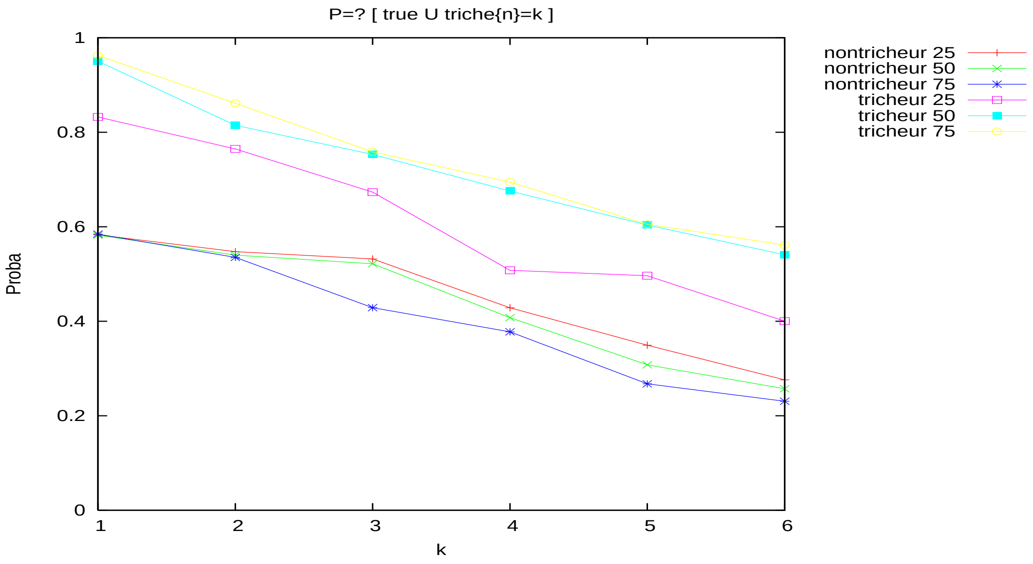


FIG. 15 – 3 utilisateurs

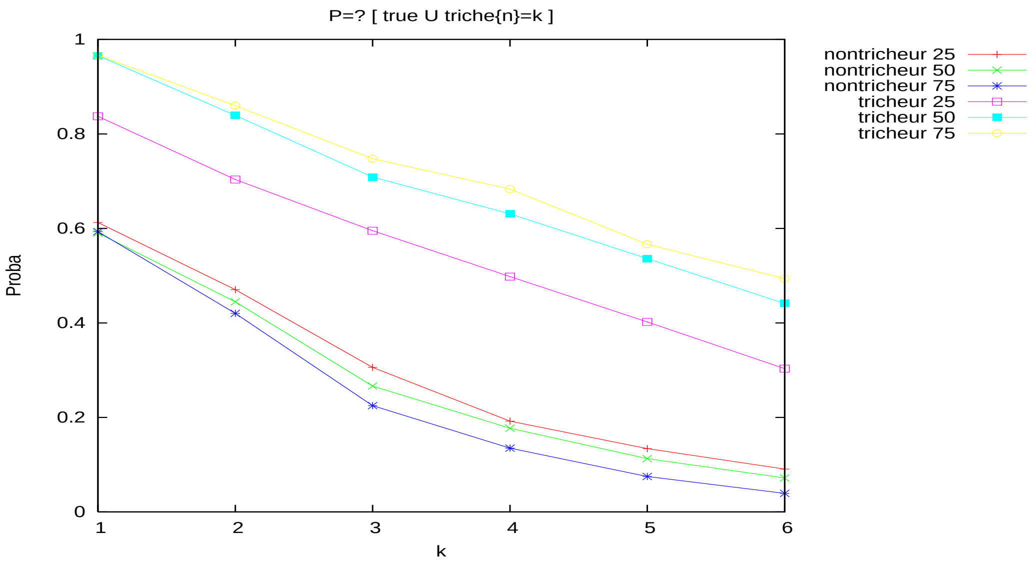


FIG. 16 – 4 utilisateurs

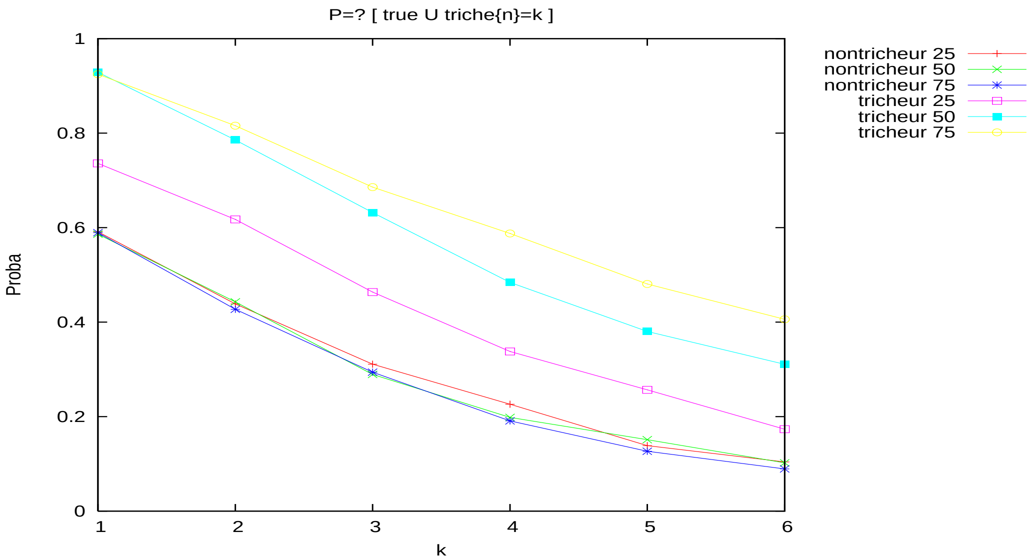


FIG. 17 – 5 utilisateurs

Voici les graphes des probabilités pour que le nombre de messages envoyés soit au moins de $k$ pour des triches de 25%, 50% et 75% et pour 4 utilisateurs.

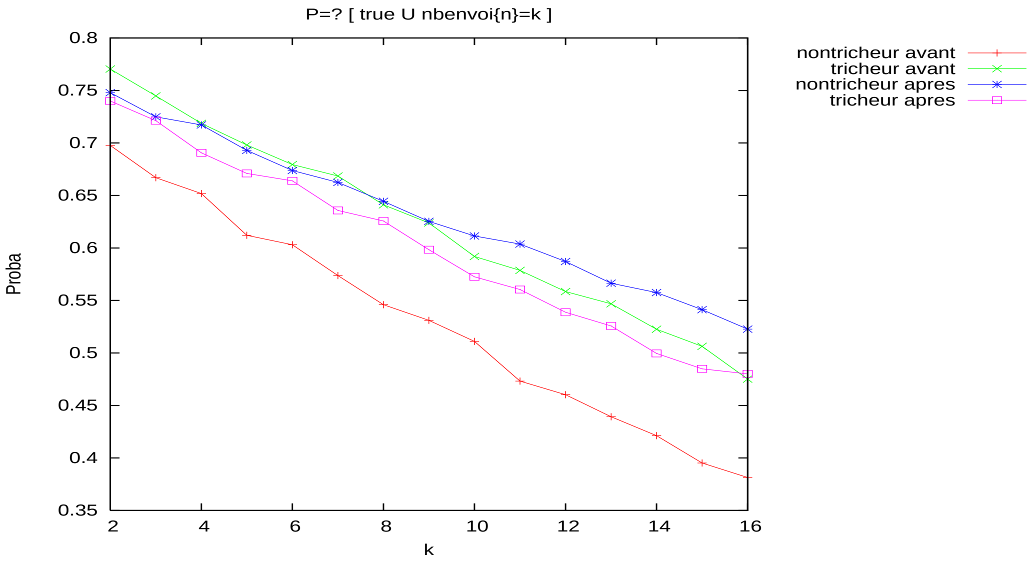


FIG. 18 – triche à 25%

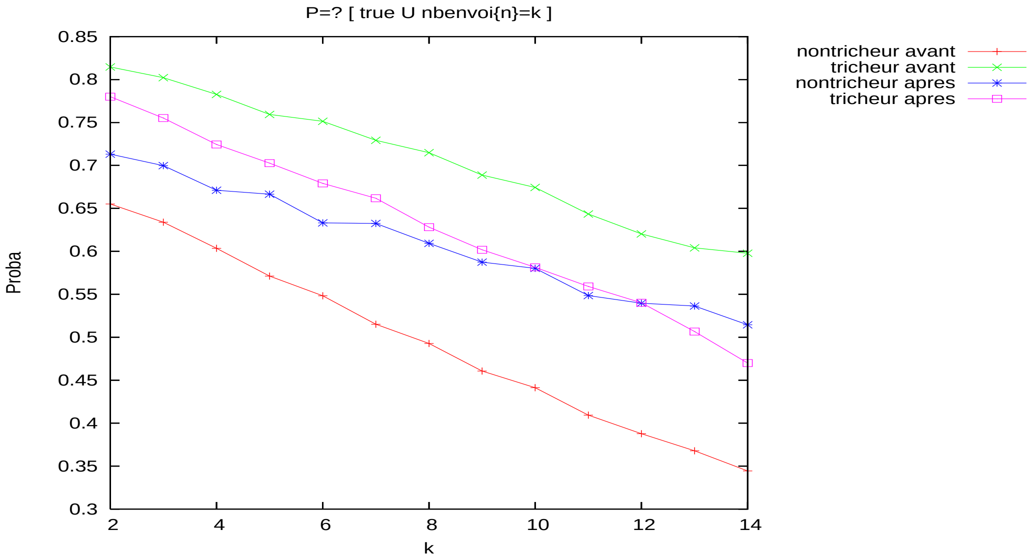


FIG. 19 – triche à 50%

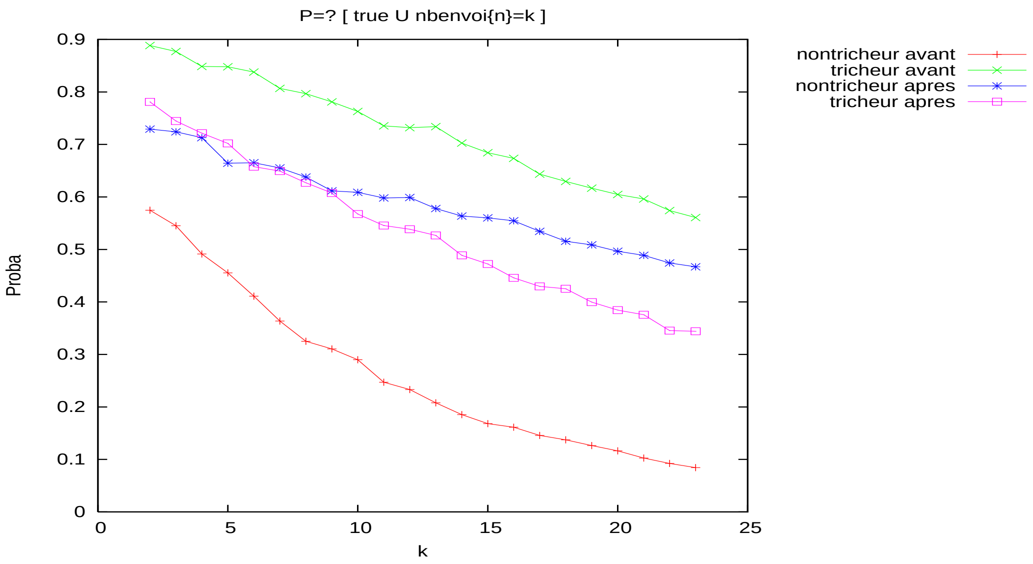


FIG. 20 – triche à 75%

## Références

[1] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formals Methods in System Design*, 15(1) :7–48, 1999.

[2] Pablo Brenner. A technical tutorial on the IEEE 802.11 protocol.

[3] C. Derman. *Finite-State Markov Decision Process.* academic Press, 1970.

[4] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peronnet. Approximate probabilistic model checking.

[5] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM : A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, 2006. To appear.

[6] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In H. Hermanns and R. Segala, editors, *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.

[7] Gethin Norman. IEEE 802.11 wireless lan. http ://www.cs.bham.ac.uk/~dxp/prism/casestudies/wlan.php.

[8] Dave Parker, Gethin Norman, and Marta Kiatkowska. *PRISM 2.1 User's Guide*, 2004.

[9] Kandaraj PIAMRAT. Manipulation de backoff dans la norme IEEE 802.11. Master's thesis, University of Texas, 2005.

[10] Maxim Raya, Jean-Pierre Hubaux, and Imad Aad. DOMINO : A system to detect greedy behavior in IEEE 802.11 hotspots.