RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

Blue Banana : support pour la mobilité des avatars dans les MMOGs distribués

Sergey Legtchenko

LIP6/UPMC/INRIA
4 pl. Jussieu, 75005 Paris
Sergey.Legtchenko@lip6.fr

Résumé
Les jeux massivement multijoueurs (Massively Multiplayer Online Games, ou MMOGs) se sont progressivement imposés comme une classe extrêmement populaire d'applications distribuées, avec des millions d'utilisateurs actifs. Afin d'offrir une qualité de jeu acceptable, ces applications doivent afficher de manière très réactive le monde virtuel qui entoure le joueur. Cependant, les MMOGs pair-à-pair existants n'arrivent pas à satisfaire ces contraintes, car le mouvement des joueurs provoque de nombreux échanges de données entre les nœuds du réseau logique (overlay). Les systèmes actuels n'anticipant pas cette mobilité, ces données ne sont pas délivrées à temps, provoquant des fautes transitoires au niveau applicatif. Pour résoudre ce problème, nous proposons Blue Banana, un mécanisme d'anticipation du mouvement des joueurs couplé à un module de préchargement de pairs permettant d'adapter en *avance* l'overlay aux besoins du MMOG. Notre évaluation s'appuie sur des traces dérivées de Second Life, un MMOG populaire. Elle montre que notre mécanisme permet de diminuer de 20% le nombre de fautes transitoires avec un surcoût réseau de seulement 2%.

Mots-clés : Pair à Pair, Tolérance aux fautes, MMOG, Mobilité de données, Adaptabilité

---

1. Introduction

Au fil des ans, les MMOGs sont progressivement devenus l'une des classes d'applications émérgentes les plus populaires. Les MMOGs offrent à leurs utilisateurs un environnement virtuel (ou EV) dans lequel ces utilisateurs, représentés par leurs *avatars* peuvent se déplacer et interagir librement [26]. Ces EV hébergent des millions de participants actifs sur toute la planète et génèrent des revenus substantiels [2]. De telles applications doivent donc posséder d'excellentes propriétés de passage à l'échelle afin de supporter le nombre colossal de joueurs. Elles doivent en outre satisfaire des contraintes proches du temps réel afin d'offrir une expérience de jeu satisfaisante.
Les MMOGs populaires existants possèdent tous une architecture client-serveur [30, 31]. Cela induit nécessairement de mauvaises capacités de passage à l'échelle pour l'application, ainsi qu'un coût financier élevé pour le producteur du MMOG [2, 14]. Afin de surmonter ces limitations, une nouvelle génération de MMOGs reposant sur des overlays pair-à-pair a vu le jour [3, 4, 9, 10, 13]. Dans ces MMOGs, la charge applicative est équitablement répartie entre tous les pairs de l'overlay. Chaque nœud stocke donc une *connaissance locale* de l'EV. Afin de pouvoir correctement afficher le monde virtuel autour de l'avatar, le nœud doit acquerir un ensemble de blocs de données qui décrivent l'espace de l'EV dans lequel se trouve l'avatar. On

appelle cet ensemble de blocs *aire de jeu* de l'avatar. Pour afficher son aire de jeu, un nœud doit trouver les pairs qui stockent les blocs nécessaires. On appelle ces pairs les *aînés* de l'aire de jeu.
L'une des principales difficultés auxquelles doit faire face un MMOG distribué est la construction et la mise à jour de l'aire de jeu d'un avatar lorsqu'il se déplace. En effet, l'aire de jeu d'un avatar en déplacement change constamment, et le nœud responsable de cet avatar doit rapidement récupérer les données de la nouvelle aire de jeu depuis ses aînés. Le mouvement virtuel d'un avatar déclenche donc un échange de données dans l'overlay. En outre, plus le déplacement de l'avatar est rapide, plus l'intervalle de temps disponible pour récupérer les données est restreint. Si un nœud est incapable de récupérer toutes les données sur son aire de jeu en un temps raisonnable (i.e, sans dégrader l'expérience de jeu), on dit que le nœud provoque une *faute transitoire* [1]. Le délai limite est en général de quelques centaines de millisecondes au plus [5].
Les overlays pour MMOG existants essayent de résoudre le problème en s'adaptant au fur et à mesure en réaction à la mobilité virtuelle. Lorsqu'un avatar se déplace dans l'EV, le nœud qui en est responsable modifie son voisinage dans l'overlay afin de pouvoir retrouver toutes les données nécessaires à son aire de jeu en un nombre de *hops* restreint [3, 6, 13]. Cependant, ces overlays ne font que *réagir* au mouvement : les nœuds ne modifient leur voisinage qu'après que le déplacement ait commencé. Cela laisse très peu de temps au nœud pour localiser les aînés de la nouvelle aire de jeu et récupérer les données nécessaires. Si le mouvement est trop rapide, ou si la quantité de données est trop importante, l'overlay est incapable de satisfaire la demande de l'application, provoquant des fautes transitoires. De plus, le problème peut survenir même pour des nœuds immobiles : si un avatar en déplacement pénètre trop rapidement dans l'aire de jeu de l'avatar immobile, celui-ci sera pendant un certain temps incapable de voir le nouvel arrivant.
Pour résoudre ce problème, nous proposons un nouveau mécanisme appelé Blue Banana qui *anticipe* le mouvement et, lorsque l'avatar se déplace, recherche les aînés des aires de jeu situées sur sa trajectoire. Concrètement, l'algorithme analyse le mouvement de l'avatar et, lorsque le déplacement est jugé suffisamment prévisible, le nœud précharge les aînés des aires de jeu situées en aval du mouvement de l'avatar en tenant compte de la vitesse du déplacement. Notre algorithme permet de diminuer le nombres de fautes transitoires du nœud en mouvement : le chargement des aires de jeu commence plus tôt, permettant leur rendu correct lorsque l'avatar y pénètre effectivement.
Le principal enjeu dans le design de notre algorithme est l'analyse de la mobilité virtuelle. En effet, si notre algorithme n'arrive pas a prédire de manière correcte et précise le déplacement de l'avatar, il aura tendance à précharger des données inutiles. Le problème est particulièrement important si l'avatar adopte un mouvement erratique : notre algorithme ne doit surtout pas précharger de données à chaque changement de direction. Faute de quoi, le chargement de ces données inutiles surcharge le nœud, provoquant de nouvelles fautes transitoires.
Les contributions de ce papier sont donc : 1) une analyse du mouvement des joueurs permettant de créer un modèle de mobilité et ainsi détecter les phases prévisibles du mouvement ; 2) l'implémentation de notre mécanisme de prédiction dans Solipsis, un overlay pour MMOG existant ; 3) un générateur de traces de mobilité réalistes utilisé pour évaluer Blue Banana ; et 4) une évaluation complète de Blue Banana avec des traces générées par notre générateur et injectées dans le simulateur Peersim [12]. Les principales leçons à tirer de ces travaux sont :

- Notre modèle de mobilité rend possible la prédiction du mouvement des avatars. En ajoutant un mécanisme d'anticipation dans Solipsis, le nombre de fautes transitoires est diminué

1. En fait, cette définition dépend de la quantité d'information nécessaire pour afficher une aire de jeu, ce qui est dépendant de l'application.

de 20% avec un surcoût réseau de seulement 2%. De plus, Blue Banana permet de précharger 20 fois plus de données à temps et ne dégrade pas la robustesse du protocole original : lorsque le mouvement est erratique, le nombre de fautes transitoires n'augmente pas.

• Les traces de mobilité générées à l'aide de notre modèle sont réalistes et une évaluation montre qu'elles possèdent les mêmes caractéristiques que les traces collectées dans Second Life [27]. Ces traces générées permettent d'extrapoler à une plus large échelle les propriétés observées dans Second Life et donc de tester notre protocole. La description du générateur ainsi que son évaluation sont omises par manque de place. Cependant, pour une spécification détaillée du générateur, le lecteur peut se référer à [17] et pour son évaluation complète voir [16].

Le reste du papier s'organise de la façon suivante. La section 2 offre une étude de la mobilité dans les MMOGs existants et présente notre modèle de mobilité qui permet de prédire le mouvement. La section 3 présente le design de Blue Banana. La section 4 présente le protocole d'évaluation et les résultats, puis la section 5 décrit l'état de l'art. Enfin, la section 6 conclut.

2. Caractéristiques de la mobilité et prédiction du mouvement

Les EVs sont très dynamiques car les avatars y possèdent habituellement une totale liberté de mouvement. De plus, les données décrivant les objets et les avatars ne sont pas uniformément répartis dans l'EV. Des études récentes de MMOGs populaires tels que Second Life [30] et World of Warcraft [31] ont montré que la distribution des avatars était fortement disparate [15, 24] : la plupart des avatars sont rassemblés dans quelques zones de forte densité, tandis que la majeure partie de l'EV est quasi-déserte. De plus, le mouvement des avatars possède des caractéristiques particulières : il est lent et chaotique dans les zones denses, rapide et rectiligne dans les zones désertes [18].

2.1. L'automate

Ces observations ont une conséquence sur le design de notre mécanisme d'anticipation, car il est nécessaire de distinguer un mouvement chaotique d'un mouvement rectiligne afin de pouvoir assurer une prédiction raisonnablement précise. Chaque nœud de l'overlay incorpore donc un automate qui décrit la mobilité de son avatar. Basé sur les caractéristiques observées dans les MMOGs existants, l'automate a deux états : (V)oyage, l'avatar se déplace rapidement dans l'EV avec une trajectoire rectiligne, (E)xploration, l'avatar explore une zone, sa trajectoire est chaotique et sa vitesse est réduite.

Lorsque le joueur interagit avec l'EV, son nœud analyse ses mouvements. Si il détecte une modification dans le comportement, il bascule l'automate dans l'état approprié. Le comportement d'un avatar est défini par sa *vitesse*. Si la vitesse de l'avatar est supérieure à un seuil fixé par l'application, l'automate passe dans l'état V, sinon il est dans l'état E. Il s'agit d'un modèle extrêmement simple qui permet de décrire de façon rudimentaire le comportement de l'avatar. Il peut bien sûr être raffiné : il est possible de prendre en compte l'accélération de l'avatar, ou bien d'analyser l'historique de ses mouvements. Cependant, bien qu'étant simple, ce modèle permet une prédiction suffisamment précise pour réduire le nombre de fautes transitoires.

Si l'automate est dans l'état V (l'avatar voyage), la trajectoire de l'avatar est hautement prévisible. Dans ce cas, notre algorithme tente de précharger les futurs aînés, i.e., les pairs qui stockent des données sur les futures aires de jeu de l'avatar.

À l'opposé, si l'avatar explore une zone (état E), sa trajectoire est chaotique et sa vitesse est réduite. Dans ce cas son comportement est difficile à prévoir, c'est pourquoi Blue Banana n'effectue aucun préchargement [2].

2. On peut remarquer que, la vitesse étant réduite, les mécanismes standards de l'overlay seront probablement assez réactifs pour éviter les fautes transitoires sans recourir au préchargement.

2.2. Anticipation du mouvement

Afin d'augmenter la précision de la prédiction, nous estimons que : (i) seule une prédiction à court terme est suffisamment précise, (ii) plus le mouvement de l'avatar est rapide, plus la probabilité qu'il continuera d'avancer sans changer de direction est forte. La première supposition implique que l'ensemble des futures positions possibles de l'avatar forment un *cône* dont le sommet est la position actuelle de l'avatar. En effet, plus une prédiction est faite loin dans le futur, plus elle est susceptible de diverger de la trajectoire réelle. La deuxième signifie que la précision de la prédiction augmente avec la vitesse de l'avatar : l'angle du cône de prédiction dépend de la vitesse.
Si tous les aînés d'une aire de jeu située à l'intérieur du cône sont préchargés à temps et si l'avatar reste dans le cône, le rendu de l'aire de jeu préchargée sera quasi-instantané.

3. Implémentation de Blue Banana au dessus de Solipsis

Nous avons implementé notre algorithme de préchargement au dessus de Solipsis [13] car son overlay dédié aux MMOGs évolue en fonction du mouvement virtuel. Cependant, comme la plupart des overlays actuels, Solipsis n'anticipe pas le mouvement des avatars : il se contente de récupérer des voisins dans toutes les directions indépendamment du mouvement et n'arrive donc pas à construire les aires de jeu lorsque la mobilité augmente.

3.1. Vue d'ensemble de Solipsis

Solipsis est un overlay dédié aux MMOGs. Chaque nœud de l'overlay est responsable d'un seul avatar. Dans Solipsis, les aînés d'une aire de jeu sont les nœuds dont les avatars se trouvent dans le proche voisinage de celle-ci. L'information sur une aire de jeu est donc dispersée parmi les nœuds dont les avatars se trouvent à proximité de cette aire de jeu dans l'EV. Solipsis se charge de maintenir un ensemble de voisins directs pour chaque nœud. Le routage des messages étant fait de proche en proche, la latence augmente proportionnellement à la distance à parcourir dans l'overlay. Afin de réduire cette latence, Solipsis essaye donc de maintenir pour chaque nœud les aînés de l'aire de jeu courante au sein de son voisinage direct dans l'overlay. C'est à dire que si deux avatars A et B sont voisins dans l'EV, l'overlay s'adapte pour que B soit en un temps fini dans le voisinage direct de A et vice versa. Pour assurer cette propriété, Solipsis suit deux règles fondamentales :

1. *Règle de la connaissance locale.* Un avatar a possède une aire de jeu circulaire ω_a centrée sur sa position. Si un autre avatar b est à l'intérieur de ω_a, a et b doivent être voisins dans l'overlay. La taille de ω_a varie afin que le nombre de voisins de a se situe dans un intervalle donné.
2. *Règle de l'enveloppe convexe.* Soit N_e l'ensemble des voisins de e dans l'overlay. L'avatar de e doit se situer à l'intérieur de l'enveloppe convexe de l'ensemble formé par les avatars de N_e. Informellement, cette propriété assure qu'un avatar ne causera pas de partition de la topologie de Solipsis en "tournant le dos" à une portion de l'EV.

Pour faire respecter ces propriétés, Solipsis implémente un mécanisme appelé *collaboration spontanée*. À chaque instant, grâce à des mises à jour périodiques, un nœud connait les coordonnées et les zones de connaissance locale de tous ses voisins directs. Lorsqu'il détecte que l'un de ses voisins entre dans la zone de connaissance d'un autre, il envoie un message aux deux voisins afin de les informer de cette proximité. Nos simulations ont montré que ce mécanisme était très efficace : grâce à lui la plupart des avatars en mouvement sont rapidement détectés. La règle de l'enveloppe convexe assure qu'un nœud est entouré par ses voisins, augmentant l'efficacité de la collaboration spontanée.
Pour résumer, lorsque la règle de la connaissance locale est enfreinte pour un nœud n, un

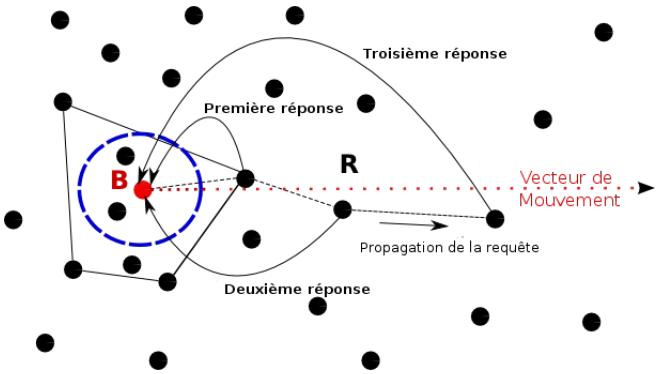

FIGURE 1 – Algorithme de propagation : la requête est transmise aux nœuds situés en aval du mouvement.

avatar est arrivé dans l'aire de jeu de n, mais n n'est pas encore connu de n, provoquant de ce fait une faute transitoire. Lorsque la règle de l'enveloppe convexe est enfreinte pour n, celui-ci n'est plus entouré par son voisinage, et est donc susceptible de ne plus être prévenu de l'arrivée d'un avatar dans son aire de jeu, provoquant également des fautes transitoires.
Lorsqu'il se déplace, un avatar enfreint constamment les règles fondamentales, car la collaboration spontanée n'est pas toujours capable de réagir à temps. Pour cette raison, un mécanisme plus efficace est nécessaire.

3.2. Implémentation d'un mécanisme d'anticipation

Bien qu'il soit implémenté dans Solipsis, notre mécanisme d'anticipation est générique. Il peut être implémenté au dessus de n'importe quel overlay qui a la propriété de modifier sa topologie en réaction au mouvement des avatars. La fonction principale de Blue Banana est de fournir sur chaque nœud un tampon contenant des pairs préchargés. Ces pairs sont récupérés en aval du mouvement de l'avatar. Lorsque l'avatar approche dans son mouvement un nœud préchargé, celui-ci est ajouté par Blue Banana à l'ensemble des voisins géré par Solipsis. Ainsi, au fur et à mesure du mouvement, les pairs préchargés sont instantanément mis à la disposition de Solipsis, minimisant les fautes transitoires.

Description de l'algorithme. Le mécanisme se décompose en deux parties. La première partie analyse les mouvements de l'avatar afin de détecter un changement de comportement. Lorsque l'automate passe dans l'état V [3] et que le tampon de préchargement n'est pas plein, la deuxième partie du mécanisme est déclenchée. Une requête de préchargement est envoyée à son voisin qui se trouve le plus près de son *vecteur de mouvement* (cf. Figure 1). Le message contient la description du cône de préchargement, la vitesse courante, ainsi qu'un TTL.
Sur réception d'une requête de préchargement envoyée par B sur un nœud R, R estime tout d'abord la position courante de B en utilisant une estimation de la latence réseau, la position initiale et la vitesse de l'avatar (ligne 1 de l'algorithme 1). Puis, l'algorithme 1 vérifie si B n'est pas trop près de R (ligne 3). En effet, si B dépasse R pendant l'échange de messages, R se situera en amont de B lorsque la réponse sera reçue par B, et la réponse sera inutile. Puis, si R est situé suffisamment loin (lignes 3 à 12), R analyse son voisinage et sélectionne les nœuds situés à l'intérieur du cône de préchargement de B (ligne 4) et les envoie à B (lignes 10 à 12). Si la taille de cet ensemble de candidats dépasse le TTL, seuls les premiers TTL candidats sont sélectionnés (ligne 5 à 6), et si R ne possède pas de voisins interessants, R n'envoie pas de réponse à B (ligne 9). Tant que le TTL n'a pas expiré, R réémet la requête à son voisin le plus proche du vecteur de mouvement de B (lignes 13 à 13). À la fin de l'échange, si aucun message

3. i.e., la vitesse limite est dépassée.

Algorithm 1: Remplissage du tampon de préchargement.

```
Result: un ensemble de nœuds préchargés.
positionEmetteur = estimerPositionCouranteEmetteur (msg);
ttl = msg.TTL ();
if (positionEmetteur, maPosition) ≥ distMin then
    nœudsProcheTrajectoire = choisirLePlusProche (voisinage, msg);
    taille = nœudsProcheTrajectoire.récupererTaille ()−1;
    if taille > ttl then
        taille = ttl ;
        nœudsProcheTrajectoire = nœudsProcheTrajectoire [1 .. taille ];
    if taille > 0 then
        ttl = ttl − taille + 1;
        réponse.ajouterEnsemble (nœudsProcheTrajectoire);
        envoyer (réponse, msg.emetteur ());
if ttl > 0 then
    msg.fixerTTL (ttl −1);
    envoyer (msg, trouverProchainNœudDansTrajectoire (msg));
```

n'a été perdu et si les messages arrivent à temps, B récupère TTL pairs situés à l'intérieur de son cône de préchargement.

Surcoût réseau. Blue Banana n'est pas lié avec le protocole de maintenance de Solipsis : les voisins préchargés sont d'abord placés dans un tampon de préchargement, et ne sont donc pas directement incorporés dans le voisinage géré par Solipsis. Ainsi, aucune ressource réseau n'est utilisée pour maintenir l'état des liens préchargés comme cela est fait pour ceux gérés par Solipsis. Il serait possible de périodiquement maintenir l'état des liens préchargés, mais nous avons préféré ne pas gaspiller de ressources pour des liens qui peuvent s'averer inutiles. La comparaison de ces deux approches est laissée à des travaux futurs.

Par conséquent, une fois placés dans le tampon, la position de l'avatar du nœud préchargé n'est pas mise à jour, et il est possible que le lien devienne obsolète. Blue Banana supprime automatiquement les nœuds préchargés inutilement (i) lorsqu'ils ont été dépassés par l'avatar en mouvement, (ii) lorsque l'avatar change de direction ou (iii) lorsque son automate change d'état.

Afin de compenser le faible surcoût réseau, les nœuds pourvus d'un module Blue Banana tirent profit de la bonne prévisibilité du mouvement des avatars dans les zones désertes. Dans Solipsis, un nœud propage périodiquement les coordonnées de son avatar à tous ses voisins, afin qu'ils puissent mettre à jour leur vue de l'EV. Avec Blue Banana, la période de ces mises à jour est doublée lorsque l'automate est dans l'état *V*. Pour pallier au manque de mises à jour, les voisins de l'avatar en mouvement prédisent sa position en utilisant sa dernière position et sa dernière vitesse connues. Cette technique est une forme simple de *navigation à l'estime* et peut aisément être améliorée avec des procédés bien connus dans le domaine du jeu en ligne [7, 22, 23].

4. Évaluation

Cette section présente une évaluation détaillée de Blue Banana. L'évaluation compare Solipsis *avec* et *sans* Blue Banana, afin de mesurer les performances du mécanisme d'anticipation. Blue Banana ainsi que Solipsis ont été implémentés dans le simulateur à événements discrets PeerSim [12]. Nous avons choisi PeerSim car il est largement utilisé pour tester des applications distribuées [1, 8, 11].

4.1. Mode opératoire

Pour obtenir une évaluation plus réaliste, nous avons utilisé des traces collectées dans Second Life [15]. La taille de ces traces est d'une centaine d'avatars simultanément présents dans l'EV. Il nous a semblé pertinent d'augmenter l'échelle des traces pour tester Blue Banana. Nous effectuons donc l'évaluation par injection de traces dérivées de Second Life créées par notre générateur. Ainsi, l'EV simulé respecte les caractéristiques de mobilité et de distribution observées dans Second Life [17]. À l'initialisation, les avatars sont aléatoirement placés dans l'EV. Lorsque l'overlay de Solipsis a été correctement initialisé, le reste de la trace est injectée dans le simulateur.

Les paramètres de la simulation sont : 1) 1000 avatars, 2) Une surface correspondant à 9 îles de Second Life, 3) 3 agglomérations de forte densité, 4) Densité des agglomérations : 9549 avatars par kilomètre carré, ce qui est, par exemple, comparable à la densité de population moyenne à New York, 5) L'accélération des avatars est fixée à 5 $m.s^{-2}$, 6) Les nœuds ont une connexion avec 10 Mbit en flux descendant et 1 Mbit en flux montant, 7) La latence réseau de chaque lien est aléatoirement fixée entre 80 et 120 ms. Un rapide calcul indique qu'avec l'accélération adoptée, la vitesse maximale des avatars *entre* les agglomérations peut dépasser la vitesse d'un TGV. Cette vitesse n'est pas exagérée, car tous les MMOGs fournissent à leurs participants un moyen de transport rapide [4]. De plus, l'accélération est prise en compte, ce qui signifie que les avatars n'atteignent pas instantanément la vitesse maximale. En pratique, cette vitesse est atteinte seulement dans cas ou un avatar se déplace d'un coin de la carte jusqu'au coin opposé. La vitesse réelle de la plupart des avatars est bien plus faible : la figure 2 montre que 99% de tous les déplacement se font sur une longueur inférieure à 40m, ce qui signifie que la vitesse ne dépasse pas 20 $m.s^{-1}$ dans ces cas.

4.2. Métriques d'évaluation

Pour évaluer Blue Banana, on fait varier le *degré de mobilité* de l'EV, qu'on définit comme la proportion des avatars qui se trouvent en mouvement rapide. Plus précisément, il s'agit du nombre de nœuds dont l'automate est dans l'état V. Ce sont précisément ces nœuds qui ont les plus fortes contraintes temporelles pour le chargement de leurs aires de jeu. Par conséquent, plus le degré de mobilité est élevé, plus les contraintes sur l'overlay sont grandes, entrainant plus de fautes transitoires. Le modèle de mobilité dérivé de Second Life a un degré de mobilité d'environ 5,5% (ce qui signifie que le nombre moyen d'avatars qui sont simultanément à l'état V est d'environ 5,5% à tout moment de la simulation). Nous faisons donc varier le degré de mobilité entre 0.5% et 11% pendant l'évaluation. Chaque résultat presenté constitue une moyenne faite sur dix itérations aléatoires.

Les métriques suivantes sont utilisées pour évaluer la résistance à la mobilité virtuelle :

- *Violation des règles fondamentales de Solipsis.* Le non respect de la règle de connectivité globale ou de la règle de l'enveloppe convexe provoque des fautes transitoires (voir la description de Solipsis dans la section 3.1).
- *Connaissance des pairs en aval du mouvement.* Cette métrique permet de mesurer, pour les avatars en déplacement rapide, le temps moyen pendant lequel ils ont connaissance d'un nœud situé en aval du mouvement. On mesure également le *nombre* de nœuds connus en aval du mouvement. Cette métrique est importante car elle donne une indication sur la quantité d'information qu'il est possible de récupérer à propos d'une aire de jeu avant de l'avoir atteinte.
- *Nombre de messages échangés.* Cette métrique évalue l'impact réseau de Blue Banana. On ne mesure que le nombre de messages, car les messages de maintenance de Solipsis et les messages de Blue Banana ont une taille réduite : ils contiennent uniquement des métadonnées sur les

4. Par exemple, les joueurs de Second Life peuvent voler.

nœuds (identifiant Solipsis, coordonnées virtuelles, adresse IP).

L'évaluation de Blue Banana selon la deuxième métrique ne prend en compte que le sous-ensemble des avatars dont l'automate est dans l'état V.
En effet, Blue Banana envoie des requêtes de préchargement uniquement dans l'état V. Or, la proportion de nœuds dans l'état V est extrêmement réduit : il ne s'agit que de 11% pour le degré de mobilité le plus élevé. La majeure partie des avatars se déplace peu, et connait donc les aînés de son aire de jeu depuis très longtemps. Si l'ensemble des avatars avaient été considérés pour cette métrique, les valeurs des nœuds en mouvement auraient été noyés dans la masse, faussant les résultats.

4.3. Analyse des résultats

La figure 2 présente la comparaison de Blue Banana (lignes continues) avec Solipsis (pointillés) selon les trois métriques.
Le principal résultat de la comparaison est que pour le degré de mobilité de 5,5% (le degré observé dans les vraies traces), Blue Banana (i) réduit le nombre de fautes transitoires de 20%, (ii) augmente le temps moyen de connaissance des nœuds en aval du mouvement de 270% et (iii) a un surcoût réseau de seulement 2%. Ces résultats sont détaillés dans le reste de la section.

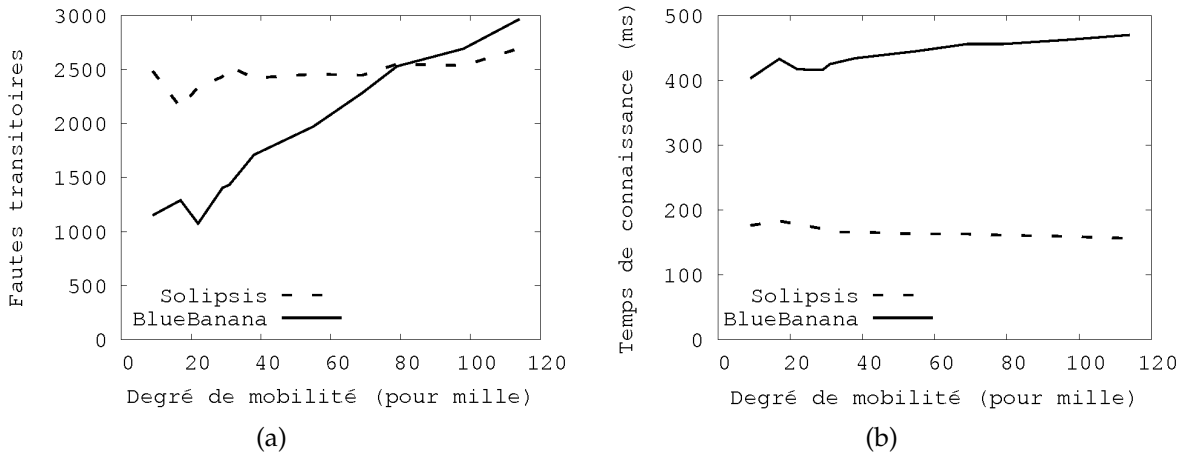

FIGURE 2 – (a) Nombre moyen de fautes transitoires par seconde, (b) Temps moyen de connaissance des pairs en avant du mouvement.

Violation des règles de Solipsis. L'évaluation selon la première métrique est présentée dans la figure 2.a. Elle montre que l'anticipation faite par Blue Banana aide la plupart du temps l'overlay à s'adapter à temps, réduisant de manière significative le nombre d'infractions aux règles fondamentales de Solipsis. Pour des degrés de mobilité inférieurs à 8%, l'utilisation de Blue Banana diminue le nombre de fautes transitoires. Notamment, pour le degré de mobilité observé dans les traces de Second Life (5.5%), le nombre de fautes transitoires est diminué de 20%. Pour des degrés de mobilité plus faibles, Blue Banana permet d'éviter près de la moitié de toutes les infractions. En revanche, lorsque la mobilité s'accroit, l'efficacité de Blue Banana décroit. Cela se produit car lorsque la mobilité augmente, la précision des prédictions de Blue Banana diminue. En effet, plus d'avatars sont susceptibles d'avoir une vitesse élevée et donc de sortir du cône de préchargement avant d'avoir été utilisés, devenant inutiles. Enfin, lorsque le degré de mobilité dépasse 8%, Blue Banana cesse d'être bénéfique à l'overlay. Dans ces EVs très dynamiques, beaucoup de pairs préchargés sont également en mouvement rapide, et seront probablement en dehors du cône lorsqu'ils seront ajoutés à l'ensemble des voisins gérés par

Solipsis. Cependant, ce degré de mobilité est de loin supérieur aux degrés observés dans les traces de Second Life. Pour résumer, la première métrique montre que Blue Banana fait baisser le nombre de fautes transitoires et suggère que la mobilité des pairs devrait être prise en compte lors du préchargement.

Connaissance des pairs en aval du mouvement. L'évaluation selon la deuxième métrique est présentée dans la figure 2.b. On peut aisément constater que pour les avatars en déplacement rapide, le temps de connaissance des pairs situés en aval du mouvement est bien plus élevé avec Blue Banana. En effet, il est 2 à 3 fois plus important qu'avec Solipsis (2.7 fois plus important pour le degré de mobilité réaliste). De plus, des mesures complémentaires montrent qu'un nœud dont l'automate Blue Banana est dans l'état V connait en moyenne 7.5 pairs en aval du mouvement. Lorsque Blue Banana est désactivé, il ne connait en moyenne qu'*un seul* nœud dans la direction du mouvement. Ces résultats permettent d'estimer la quantité moyenne d'information qu'un avatar en déplacement rapide peut espérer récupérer à temps en aval du mouvement. Pour un degré de mobilité de 5.5%, un nœud qui utilise Blue Banana pourrait télécharger environ 430 Ko d'information sur sa future aire de jeu (avec 10 Mbit en débit descendant). Sans Blue Banana, il ne pourrait télécharger dans les mêmes conditions que 20 Ko environ. L'application peut donc se permettre d'afficher *à temps* nettement plus (environ 20 fois plus) d'informations sur le monde virtuel qui entoure le joueur, améliorant clairement l'expérience de jeu.

Surcoût réseau. Le dernier résultat important de notre évaluation est le faible coût réseau de Blue Banana. La figure 3 montre que lors de son utilisation, le surcoût est d'environ cinq messages par nœud par seconde, ce qui est négligeable comparé au nombre de messages de maintenance générés par l'overlay de Solipsis. En effet, un nœud Solipsis envoie entre 130 et 230 messages de maintenance par seconde en fonction du degré de mobilité. Le surcoût de Blue Banana est par conséquent évalué à 2%. De plus, les messages de Blue Banana ont une taille faible et fixe [5]. Ce résultat très positif est du au fait que les prédictions de Blue Banana sont suffisamment précises. Dans la plupart des cas, lors de l'infraction à une règle fondamentale de Solipsis, un nœud préchargé sera ajouté à l'ensemble des voisins géré par Solipsis. Cette action rétablira la règle enfreinte, évitant l'envoi de messages de maintenance (voir section 3.2). En fait, le faible surcoût provient de tous les pairs qui ont été inutilement préchargés à cause d'une prédiction incorrecte. Enfin, l'intervalle de mise à jour des avatars en mouvement rapide est doublé (voir section 3.2), ce qui économise également des messages. Cette dernière optimisation explique le fait que pour un degré de mobilité supérieur à 8%, Blue Banana permet même de diminuer le nombre de messages : le nombre d'avatars en mouvement rapide est élevé, un nombre important de messages est donc économisé grâce à cette optimisation.

5. État de l'art

Overlays non-adaptatifs. Dans la décennie passée, un effort de recherche considérable a été mené sur les overlays pair-à-pair. Pourtant, la plupart des overlays existants ne prennent absolument pas en compte les besoins applicatifs [19, 21, 25]. C'est en partie pourquoi il est actuellement difficile de réaliser un EV distribué. Les pairs partageant la même aire de jeu n'ont aucune raison d'être proches dans l'overlay, et doivent pourtant communiquer de manière très efficace car les données qu'ils stockent sont fortement corrélées. La cause en est historique : ces overlays ciblent une application précise : le partage de fichiers en lecture seule. Généralement, ce type d'overlay s'occupe seulement de fournir un service de routage efficace entre les nœuds. Blue Banana n'est pas compatible avec ce type d'overlays.

5. voir la description de la métrique *Nombre de messages échangés*.

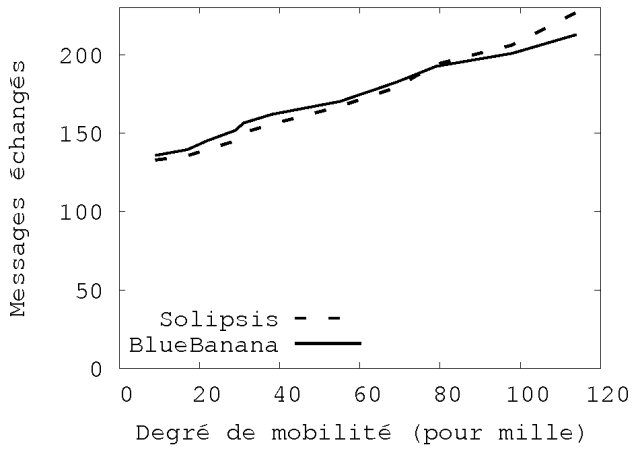

FIGURE 3 – Nombre moyen de messages envoyés par seconde sur chaque nœud.

Overlays réactifs aux besoins applicatifs. Récemment, des travaux ont étudié des overlays qui modifient leur structure pour mieux satisfaire les besoins applicatifs. Par exemple, les overlays sémantiques [28] tissent des liens entre les pairs proches sémantiquement. Cela permet à ces nœuds d'être proches dans l'overlay, ce qui est bénéfique car la proximité sémantique entraine bien souvent de fortes interactions. Quelques autres systèmes récents sont capables de s'adapter aux applications [20, 29] : ils prennent en compte l'évolution des besoins applicatifs, généralement en surveillant les communications entre les pairs. Cependant, si l'application est trop dynamique, la réaction de l'overlay peut être trop tardive, menant au mieux à la perte temporaire de l'efficacité de l'overlay, et au pire à des incohérences au niveau applicatif. Ces travaux diffèrent de notre approche, car les overlays modifient leur topologie en *réaction* aux besoins applicatifs, là où notre approche tente de prédire ces besoins et adapter l'overlay par anticipation. Notre approche peut aisément être adaptée à n'importe quel overlay de ce type.

Overlays pour MMOGs distribués. Dernièrement, des travaux se sont occupés de concevoir des overlays dédiés aux MMOGs, mais aucun, à notre connaissance, ne tente de prédire les besoins applicatifs afin d'adapter sa topologie par anticipation. Les contraintes imposées par ce type d'applications sont très difficiles à respecter. En particulier, l'overlay se doit d'être extrêmement réactif afin d'assurer le service même en présence de mobilité. Varvello et al. ont implémenté une telle application sur une table de hachage distribuée (DHT) [27]. Les auteurs montrent que la réactivité de la DHT est uniquement acceptable lorsque la mobilité virtuelle est faible. Colyseus [6], une architecture décentralisée pour MMOGs à forte contraintes de latence (jeux de tir subjectif) se base également sur une DHT pour la découverte d'objets virtuels. Colyseus analyse les besoins des joueurs pour précharger des objets avant qu'il ne soient effectivement demandés. Cependant, l'overlay n'adapte pas sa propre topologie selon les besoins applicatifs. Donnybrook, la suite de Colyseus, utilise des techniques élaborées d'approximations et de navigation à l'estime afin d'alléger le coût réseau [7].
L'autre grande approche est d'utiliser des overlays adaptatifs. Dans ces systèmes, le voisinage d'un nœud de l'overlay est déterminé par le voisinage de son avatar dans l'EV. Pour chaque nœud, l'overlay tente de garder les aînés de sa zone de jeu dans le voisinage topologique du nœud. Au fur et à mesure du déplacement de l'avatar dans l'EV, le voisinage logique de son nœud évolue : l'overlay s'adapte en réaction à l'application. Grâce à cela, le voisinage logique de chaque nœud dans l'overlay finira par être en accord avec le voisinage virtuel de son avatar : chaque nœud connaitra les aînés de l'aire de jeu de son avatar. Plusieurs overlays de ce type existent à ce jour. C'est par exemple le cas de Solipsis [13] sur lequel nos expérimentations ont

été menées. Il existe également une famille d'overlays dont la topologie est basée sur le pavage de Voronoi, comme VoroNet/RayNet ou VON [3, 4, 10].
Cependant, à notre connaissance, aucun de ces overlays n'*anticipe* les besoins applicatifs. Notre module peut donc être implémenté au dessus de chacun de ces systèmes afin de les doter de capacités d'anticipation des besoins applicatifs.

6. Conclusions et perspectives

Cette étude vise à montrer les difficultés des overlays pour MMOGs à supporter la mobilité des avatars sans dégradation de service. Cela se produit car les overlays ne font que *réagir* au mouvement sans tenter de l'*anticiper*. Lorsque le déplacement est rapide, le temps d'adaptation de l'overlay est trop élevé, ce qui entraine l'apparition de nombreuses fautes transitoires au niveau applicatif. Nous proposons donc Blue Banana, un mécanisme de préchargement de pairs qui permet à l'overlay de s'adapter par anticipation. Nous montrons que Blue Banana permet d'éviter 20% de fautes transitoires sans impact significatif sur le réseau. De plus, Blue Banana permet de télécharger 20 fois plus de données utiles en cas de mobilité, autorisant un rendu plus détaillé de l'environnement virtuel. Notre approche est générique et peut être réutilisée lors de la conception de futurs overlays dédiés aux MMOGs.
Nous prévoyons d'étudier des mécanismes d'anticipation plus évolués, plus particulièrement d'explorer la possibilité de prendre en compte le mouvement relatif entre les avatars, ce qui devrait rendre Blue Banana plus efficace dans le cas de MMOGs très dynamiques.

Bibliographie

1. Matteo Agosti, Francesco Zanichelli, Michele Amoretti, et Gianni Conte. P2pam : a framework for peer-to-peer architectural modeling based on peersim. In Sándor Molnár, John Heath, Olivier Dalle, et Gabriel A. Wainer, editors, *SimuTools*, page 22. ICST, 2008.
2. Reis Tiago Alves et Licinio Roque. Because players pay : The business model influence on mmog design. In Baba Akira, editor, *Situated Play : Proc. of the 2007 Digital Games Research Association Conference*, pages 658–663, Tokyo, September 2007. The University of Tokyo.
3. Olivier Beaumont, Anne-Marie Kermarrec, Loris Marchal, et Etienne Riviere. Voronet : A scalable object network based on voronoi tessellations. In *21th International Parallel and Distributed Proc. Symposium (IPDPS 2007), Long Beach, USA*, pages 26–30. IEEE, March 2007.
4. Olivier Beaumont, Anne-Marie Kermarrec, et Etienne Riviere. Peer to peer multidimensional overlays : Approximating complex structures. In Eduardo Tovar, Philippas Tsigas, et Hacène Fouchal, editors, *OPODIS*, volume 4878 of *LNCS*, pages 315–328. Springer, 2007.
5. Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, et Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003. In Wu chang Feng, editor, *NETGAMES*, pages 144–151. ACM, 2004.
6. Ashwin Bharambe, Jeffrey Pang, et Srinivasan Seshan. Colyseus : a distributed architecture for online multiplayer games. In *NSDI'06 : Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 12–12, Berkeley, CA, USA, 2006. USENIX Association.
7. Ashwin R. Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, et Xinyu Zhuang. Donnybrook : enabling large-scale, high-speed, peer-to-peer games. In Victor Bahl, David Wetherall, Stefan Savage, et Ion Stoica, editors, *SIGCOMM*, pages 389–400. ACM, 2008.
8. Carmela Comito, Simon Patarin, et Domenico Talia. A semantic overlay network for p2p schema-based data integration. In Paolo Bellavista, Chi-Ming Chen, Antonio Corradi, et Mahmoud Daneshmand, editors, *ISCC*, pages 88–94. IEEE Computer Society, 2006.
9. D. Frey, J. Royan, R. Piegay, A.M. Kermarrec, E. Anceaume, et F. Le Fessant. Solipsis : A decentralized architecture for virtual environments. In *The Second International Workshop on Massively Multiuser Virtual Environments at IEEE Virtual Reality (MMVE' 09)*, Lafayette, USA, March 2008.

10. Shun-Yun Hu, Jui-Fa Chen, et Tsu-Han Chen. Von : A scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4) :22–31, Jully 2006.
11. Christian Jacob, Marcin L. Pilat, Peter J. Bentley, et Jonathan Timmis, editors. *Artificial Immune Systems : 4th International Conference, ICARIS 2005, Banff, Alberta, Canada, August 14-17, 2005*,, volume 3627 of *LNCS*. Springer, 2005.
12. Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, et Spyros Voulgaris. The Peersim simulator. http ://peersim.sourceforge.net/.
13. Joaquín Keller et Gwendal Simon. Solipsis : A massively multi-participant virtual world. In Hamid R. Arabnia et Youngsong Mun, editors, *PDPTA*, pages 262–268. CSREA Press, June 2003.
14. Sanjeev Kumar, Jatin Chhugani, Changkyu Kim, Daehyun Kim, Anthony Nguyen, Pradeep Dubey, Christian Bienia, et Youngmin Kim. Second life and the new generation of virtual worlds. *Computer*, 41(9) :46–53, 2008.
15. Chi-Anh La et Pietro Michiardi. Characterizing user mobility in Second Life. In *SIGCOMM 2008, ACM Workshop on Online Social Networks, August 18-22, 2008, Seattle, USA*, August 2008.
16. Sergey Legtchenko, Sébastien Monnet, et Gaël Thomas. Blue Banana : resilience to avatar mobility in distributed MMOGs. Research Report RR-7149, INRIA, 2009.
17. Sergey Legtchenko, Sebastien Monnet, et Gael Thomas. Blue banana : resilience to avatar mobility in distributed mmogs. *Dependable Systems and Networks, International Conference on*, 0 :171–180, 2010.
18. Huiguang Liang, Ian Tay, Ming Feng Neo, Wei Tsang Ooi, et Mehul Motani. Avatar mobility in networked virtual environments : Measurements, analysis, and implications. *CoRR*, abs/0807.2328, 2008.
19. J. Liang, R. Kumar, et K. Ross. The kazaa overlay : A measurement study. In *Proc. of the 19th IEEE Annual Computer Communications Workshop*, 2004.
20. Sébastien Monnet, Ramsés Morales, Gabriel Antoniu, et Indranil Gupta. Move : Design of an application-malleable overlay. In *Symposium on Reliable Distributed Systems 2006 (SRDS 2006)*, pages 355–364, Leeds, UK, October 2006.
21. Andy Oram. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*, chapter Gnutella, pages 94–122. O'Reilly, May 2001.
22. Jeffrey Pang, Frank Uyeda, et Jacob R. Lorch. Scaling peer-to-peer games in low-bandwidth environments. In *IPTPS '07 : Proc. of the 6th International Workshop on Peer-to-Peer Systems*, February 2007.
23. Lothar Pantel et Lars C. Wolf. On the suitability of dead reckoning schemes for games. In Lars C. Wolf, editor, *NETGAMES*, pages 79–84. ACM, 2002.
24. Daniel Pittman et Chris GauthierDickey. A measurement study of virtual populations in massively multiplayer online games. In *NetGames '07 : Proc. of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 25–30, New York, NY, USA, 2007. ACM.
25. Antony I. T. Rowstron et Peter Druschel. Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware '01)*, volume 2218 of *LNCS*, pages 329–250, Heidelberg, Germany, November 2001. Springer.
26. Daniel Thalmann, Nadia Magnenat-Thalmann, et Igor S. Pandzic. *Avatars in Networked Virtual Environments*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
27. Matteo Varvello, Christophe Diot, et Ernst W Biersack. P2P Second Life : experimental validation using Kad. In *Infocom 2009, 28th IEEE Conference on Computer Communications*, pages 19–25, Rio de Janeiro, Brazil, April 2009.
28. S. Voulgaris, A. M. Kermarrec, L. Massoulie, et M. van Steen. Exploiting semantic proximity in peer-to-peer content searching. In *10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004)*, Suzhou, China, May 2004.
29. Spyros Voulgaris, Etienne Riviere, Anne-Marie Kermarrec, et Maarten van Steen. Sub-2-sub : Self-organizing content-based publish and subscribe for dynamic and large scale collborative networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, USA, February 2006.
30. Second Life. http ://secondlife.com/.
31. World of Warcraft. http ://www.worldofwarcraft.com/.