Amélioration de la réactivité des réseaux pair à pair pour les MMOGs

Rapport Final

Xavier Joudiou

Encadré par: Sergey Legtchenko & Sébastien Monnet

26/07/10

Stage Lip6 : Amélioration de la réactivité des réseaux p2p pour les MMOGs (1)

Table des matières

Résumé

Depuis plusieurs années, un nouveau type d'architecture des systèmes est apparu. Il s'agit de l'architecture pair à pair, cette architecture est devenue populaire grâce à des applications de partage de fichiers. Nous allons nous intéresser aux jeux vidéos massivement multijoueur (MMOG pour Massively Multiplayer Online Games) qui sont de plus en plus populaires et qui font ressortir des problèmes que l'architecture pair à pair doit pouvoir corriger. Le problème du passage à l'échelle sera l'un des plus importants à résoudre pour permettre à un grand nombre de joueurs de participer simultanément. Nous verrons comment l'architecture pair à pair peut être une des solutions.
Pour remédier à cela, une solution consiste à remplacer le modèle client/serveur par un réseau logique pair à pair (overlay). Malheureusement, les protocoles pair à pair existants sont trop peu réactifs pour assurer la faible latence nécessaire à ce genre d'applications. Néanmoins, quelques travaux ont déjà été menés pour adresser ce problème. L'idée est d'adapter le voisinage de chaque pair afin que toute l'information dont il aura besoin dans l'avenir se trouve proche de lui dans le réseau. Il est alors nécessaire de correctement évaluer les futurs besoins de chaque pair, et de faire évoluer son voisinage à temps. Dans ce rapport, nous présenterons les deux principales solutions que nous avons mis en place et les résultats obtenus avec ces solutions. Nous parlerons ensuite des différentes autres pistes que nous avions trouvées pour continuer le travail existant dans Blue Banana. La principale solution mise en place est l'utilisation d'un cache pour les mouvements erratiques des avatars.

Remerciements
Je remercies mes deux responsables de stage Sébastien Monnet et Sergey Legtchenko, mes sœurs pour avoir relu mes rapports et pour y avoir enlevé le plus de fautes possibles.

1 Introduction

Le début de ce rapport va permettre d'observer les différents travaux déjà réalisés sur les applications pair à pair, les jeux vidéos massivement multijoueur étant le principal type d'application qui va nous intéresser. Ces applications impliquent que différentes propriétés soient vérifiées si l'on veut qu'elles soient utilisées par un grand nombre de personnes en même temps et sur une longue durée. L'architecture pair à pair peut répondre efficacement à certaines des différentes propriétés nécessaires au bon fonctionnement des applications mais elle pose un problème au niveau de la latence. Le but du stage a été d'améliorer les liens du réseau pair à pair pour que l'utilisateur puisse avoir dans son voisinage les données qui lui seront nécessaires aux itérations suivantes, pour cela il faut anticiper au mieux les mouvements du joueur.

Tout d'abord nous expliquerons pourquoi l'approche pair à pair est celle qui paraît la plus adaptée pour répondre aux différentes problématiques qu'induisent ces applications, nous en profiterons pour rappeler rapidement les caractéristiques des différentes architectures (cf 2, page 6). Ensuite, les mécanismes permettant une meilleure mise en place de l'architecture pair à pair seront expliqués, nous rentrerons un peu plus dans les détails pour Solipsis car Blue Banana l'utilise comme base (cf 4, page 13). Après avoir parlé des différents mécanisme existants qui ne prennent pas en compte la mobilité, une étude des traces des avatars dans les environnements virtuels sera alors introduite (cf 5, page 15). Cette étude des traces permettra de mieux expliquer les différentes solutions d'amélioration. Après avoir parlé des différents mécanismes déjà existants, le travail Blue Banana qui a permis de mettre en place une première amélioration sera expliqué (cf 6, page 17).

Ensuite, les deux principales améliorations, qui ont été mises en place durant ce stage, seront présentées. La première solution consiste à intégrer un cache dans les nœuds du réseau, son fonctionnement et ses performances seront expliqués. L'autre solution, implémentés durant le stage, est une amélioration une amélioration du rapatriement des données qui a été mis en place dans Blue Banana. Outre l'explication des résultats de chacune des solutions, les différentes pistes explorées, qui se sont révélées infructueuses, seront présentées. Enfin les autres pistes d'amélioration, qui ont été envisagées, seront expliquées.

2 Vers des solutions distribuées pour les MMOG

Nous souhaitons étudier les différentes raisons qui nous feraient passer d'une architecture Client/Serveur à une architecture pair à pair. Pour cela nous mettrons en évidence les différences des deux solutions, les avantages et les inconvénients d'une approche pair à pair. Il nous a été possible d'observer qu'il est devenu important, pour les éditeurs de MMOGs, d'étudier les différentes architectures en amont de la réalisation du jeu pour ainsi gérer au mieux les différentes dépenses [2] (maintenance des serveurs, ajout de add-ons,etc).

2.1 Les architectures actuelles pour les MMOGs

Dans la plupart des jeux en ligne massivement multi joueur, l'architecture est de type client/serveur (voir page 8). Dans cette architecture, il y a une forte distinction entre le client, qui envoie des requêtes au serveur et attend les réponses, et le serveur qui est à l'écoute de requêtes des clients. Cette approche simplifie la sécurité et le fonctionnent global des jeux. Par exemple, pour effectuer des mises à jour sur l'état global du jeu, il suffit de le faire sur une seule machine (le serveur principal) et il n'y a pas de problème d'incohérence entre les données. De même pour l'administration et le contrôle des tricheurs, toutes les données étant regroupées sur une seule machine, le contrôle sera plus simple que dans des systèmes distribués.

Un des problèmes que peut rencontrer l'architecture pair à pair est qu'elle aura plus de difficultés pour passer à l'échelle sans mettre de gros moyens en terme de machines et donc financier. Le serveur peut devenir un goulot d'étranglement et si un trop grand nombre de joueurs se connecte, le serveur pourrait avoir des difficultés à tenir [24]. Ce problème est résolu en ayant des serveurs de très grandes capacités ou en mettant en place des clusters de serveur. Ces solutions induisent un gros investissement dès le début de la mise en service du jeu et le coût de maintenance est élevé. Il est donc difficile de mettre cela en place pour des applications *open source* ou ayant peu de moyen, c'est surtout pour celles-ci que les architectures pair à pair peuvent être intéressantes dans l'immédiat. Un autre problème est la disponibilité du système en cas de panne du serveur, si le serveur tombe en panne alors plus personne n'aura accès à l'application que ce dernier faisait fonctionner, à la différence d'une architecture répartie.
Au vue du nombre croissant de participant à ce genre de jeux vidéos massivement multi joueur, le passage à l'échelle devient un sujet très important et c'est pour cela que les recherches sur des architectures distribuées sont de plus en plus importantes (voir Schéma 1).

Année	Nombre de publications *	Nombre de publications **
2002	0	30
2003	0	109
2004	2	235
2005	9	489
2006	9	745
2007	21	1159
2008	19	1464
2009	24	1516

Statistiques récupérées en effectuant des recherches sur le site http://portal.acm.org/ :
* pour la recherche sur: P2P MMOGs
** pour la recherche sur: P2P overlays

FIGURE 1 – Tableau montrant l'évolution des recherches dans le domaine du P2P (MMOGs et Overlay)

2.2 Le pair à pair, la solution ?

Comme il est dit précédemment, l'augmentation croissante des recherches sur le sujet atteste du fait que des limites ressortent des solutions existantes. Le problème du passage à l'échelle est sûrement le plus important, et il est l'une des raisons principales de toutes ces recherches. Les architectures pair à pair ne font plus ressortir d'entité serveur et client comme nous le connaissions dans l'architecture client/serveur. Chaque nœud sera client et serveur en fonction du temps et en fonction de ses besoins (voir page 8). Les systèmes pair à pair peuvent avoir une multitude d'utilisations, que ce soit dans le partage de fichier [30, 33, 32], la communication [35], les jeux vidéos [37], le calcul scientifique [34, 39, 28], le militaire [31], etc.

L'architecture pair à pair est faite telle qu'il n'y a pas de goulot d'étranglement,car nous passons d'un système où tout passait par un point unique à un système qui comporte un grand nombre d'entités qui peuvent toutes avoir le même rôle. L'utilisation de l'architecture pair à pair peut entrainer un grand nombre de communications. Elle nécessite des synchronisations des entités, de la gestion des ressources partagées et d'autres problèmes liés à la répartition des données. Il faut donc trouver des solutions à tous ces problèmes éventuels. Le pair à pair peut donc être plus adapté, dans certains cas, à des applications massivement multi joueur mais il faudra pouvoir garantir les mêmes propriétés que les systèmes client/serveur.

Les systèmes pair à pair sont plus difficiles à surveiller, les phénomènes de tricherie

sont donc plus difficiles à détecter, de même pour la sécurité. Nous avons pu voir qu'il existe trois types de tricherie : par Confidentialité, c'est à dire obtenir des informations non autorisées sur d'autres utilisateurs ; par Intégrité, si il y des modifications du monde, des lois physiques ou les lois du jeu non autorisées ; par Disponibilité, c'est le fait de provoquer des ralentissements ou des arrêts de partie du jeu (référence vers Challenges in P2P gaming). Il faut que le système soit aussi fiable sur le long terme et qu'il soit tolérant aux connexions et déconnexions (churn).

Les jeux vidéos sont des applications distribuées qui ne sont pas dites "critiques" (temps réel "mou"), le fait que le jeu ralentisse légèrement de manière très ponctuelle n'est pas gênant. Certaines propriétés des applications distribuées peuvent être retardées ou sautées ponctuellement. Les jeux vidéos ont des avantages qui font qu'il sera moins ardu de réaliser une distribution [4] :

- Les jeux vidéos tolèrent une consistence faible pour les différents états de l'application.
- Il peut être possible de prédire les écritures et les lectures grâce à l'ensemble des règles définies dans le jeu. Le but du stage sera l'amélioration de la prise en compte de la mobilité dans les MMOGs, dont Blue Banana [19] sera un point de départ possible.

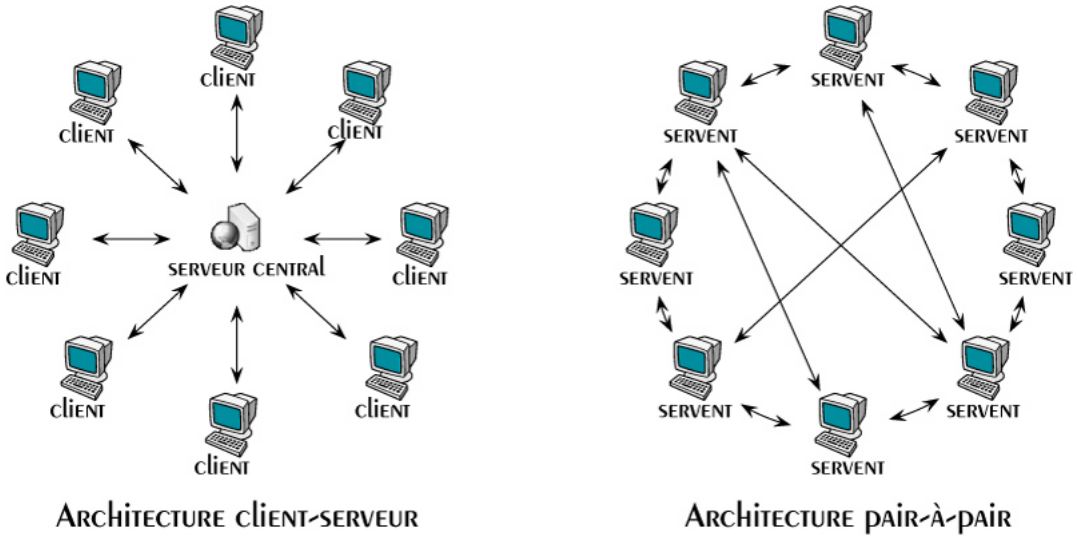

Figure 2 – Schéma des architectures pair à pair et client/serveur

3 Systèmes P2P exploitables par les MMOGs distribués

Des systèmes pair à pair sont déjà exploitables par les MMOGs distribuées, un tour d'horizon de ceux-ci va permettre de mettre en avant ce qu'ils apportent. Les concepts d'aire d'intérêt, de découpage de la carte et d'overlay vont être passés en revue.

3.1 Area Of Interest

Dans plusieurs articles, la notion de *Area Of Interest* [3, 4, 5] va apparaître ce qui montre l'utilité de ce concept. Nous pouvons déjà retrouver ce principe sous le nom de *Local Awareness* dans Sollipsis, l'idée principale de ce mécanisme est qu'une entité n'a pas besoin de connaître l'ensemble du monde virtuel à chaque instant. Alors nous allons mettre en place une zone dans laquelle l'entité sera tenue informée des différentes modifications qui seront faites sur les objets et les autres entités qui se trouvent dans la zone (voir schéma 3). Si une entité modifie sa représentation virtuelle, seulement les entités qui sont dans sa zone seront directement informées.

Dans Mercury [5], le concept d'Area Of Interest va pouvoir être utile avec le mécanisme de publish-subscribe qui est mis en place. Ce mécanisme de publish-subscribe permet l'abonnement et le désabonnement aux mises à jour d'un objet, et il permet donc d'envoyer un objet vers les nœuds qui sont abonnés.

Colyseus [4] est un travail postérieur à Mercury, les principes sont les mêmes avec l'utilisation de *range-queriable* DHT ou de DHTs pour stocker les informations. Les *range-queriable* DHTs vont s'organiser en un overlay circulaire où chaque nœud adjacent est responsable d'une suite continue de clés. Grâce à cela, il sera possible de prendre les coordonnées X comme clé et ainsi les performances seront bien meilleures qu'avec des DHTs normales (aléatoire). Les différents résultats réalisés montrent bien que les *range-queriable* DHTs utilisent moins de bande passante que les DHTs normales.

Dans Donnybrook [3], le concept d'Area Of Interest fait référence au travail réalisé dans Colyseus. Comme chaque joueur envoie ses mises à jour à chaque joueur qui se trouve dans sa zone, il faut une limitation du nombre de joueurs dans une même zone. Donnybrook introduit la notion de *player's interest set*, un joueur va pouvoir se concentrer sur un nombre fixe de joueurs (cinq dans l'article), à la différence du nombre d'objet dans l'AOI qui peut varier. Un mécanisme d'abonnement sera mis en place pour s'échanger les informations entre les joueurs. Un mécanisme d'*intérêt estimé* est mis en place pour définir les joueurs qui feront partis de son *interest set*. Plusieurs critères sont pris en compte pour choisir les joueurs qui seront sélectionnés. Trois principales propriétés sont prises en compte : la proximité spatiale entre les joueurs, les objectifs des joueurs et les interactions récentes que les joueurs peuvent avoir eu entre les deux joueurs.

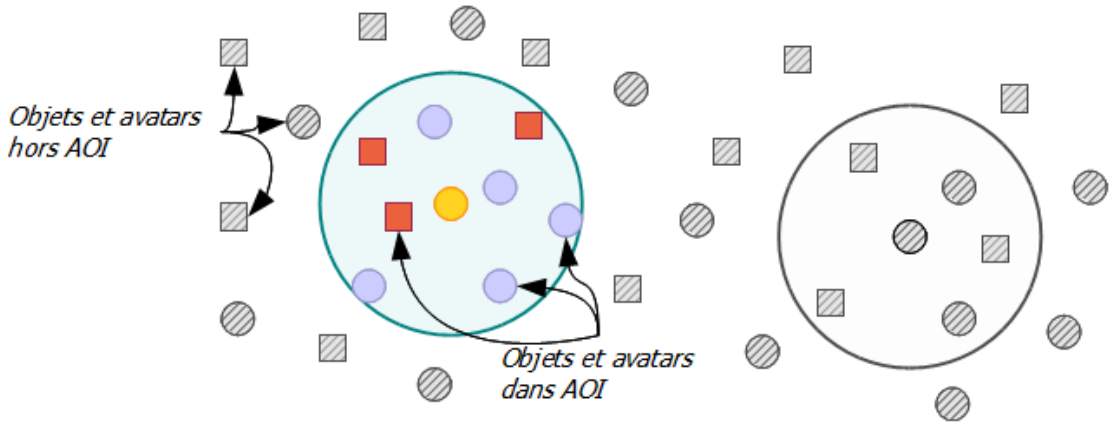

FIGURE 3 – Principe de l'*Area Of Interest*

3.2 Découpage de la carte

Pour plusieurs raisons, dont l'équilibrage de charge, le monde a été découpé en plusieurs zones. Différentes techniques pour le découpage ont été étudiées, l'une des plus courantes est d'utiliser le découpage grâce au découpage de Voronoi [12], c'est de cette technique que la triangulation de Delaunay s'inspire. Les découpages suivant ces principes sont répandus car un découpage aléatoire ne prendrait pas en compte la densité des objets dans le monde qui peut être très variable entre les régions. Le principe est de découper la monde en zone en fonction de la distance entre les différents objets qui se trouvent dans l'environnement. Dans une région, on aura un objet qui sera entouré de ses voisins. La frontière entre la région de deux voisins se trouve au milieu d'une droite séparant les deux voisins. Comme il est possible de voir sur le schéma 4, les zones ont des formes et des tailles différentes. Nous avons ainsi des zones qui comprennent un seul nœud. La différence entre la triangulation de Delaunay et le découpage de Voronoi est que pour la première, les arrêtes seront les droites entre les sommets si ils sont voisins et dans le deuxième cas, les arrêtes seront formées grâce aux médiatrices des arrêtes de la triangulation de Delaunay.

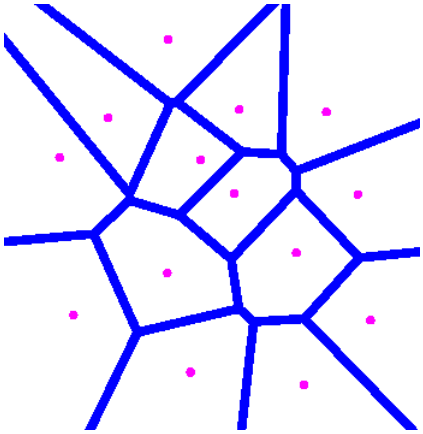

FIGURE 4 – Principe du découpage de Voronoi

3.3 Overlays

Plusieurs travaux sur la modification de l'overlay ont été fait [1, 23, 11], le but principal de ces travaux est de mettre en place un overlay qui puisse facilement s'adapter aux besoins des nœuds. Un overlay est un réseau informatique formant un graphe au sein duquel le voisinage de chaque nœud est déterminé selon un critère logique. Dans l'architecture pair à pair, il n'y a pas de connaissance globale, il n'y a qu'une connaissance locale (voisinage). Le but est de faire des liens entre les nœuds qui sont intelligents, nous verrons des overlays avec différents degrés de prise en compte de l'application.

3.3.1 Semantic overlay

Le but principal de cet overlay sémantique [1] est de créer des liens entre des nœuds qui s'intéressent aux mêmes documents, car la recherche de fichiers est devenue très importante au moment de l'arrivée des logiciels de partage de fichiers tel que Napster, Gnutella, KaZaA, etc. L'approche consiste à constituer des groupes sémantiques avec les documents et de construire un overlay au dessus du réseau pour chaque groupe. Le problème est donc d'identifier et de constituer des groupes efficacement. Trois stratégies ont été mises en place :

- *LRU :* Stratégie basée sur les éléments les plus récemment utilisés
- *History :* On maintient les liens sémantiques vers la même classe de préférence, cette technique oblige le maintien d'un nœud *counter*. C'est une technique lourde en nombre de messages et en stockage, de plus il peut y avoir des problèmes avec les *counters*.
- *Popularity :* Création de liens entre les nœuds du même type, introduction de deux paramètres : Numrep (nombre de réponse positive pour obtenir le document) et Lastreply (date de la dernière réponse)

Les résultats montrent que Popularity est l'algorithme le plus adapté, cette stratégie a un bon compromis entre efficacité et complexité de mise en place.

3.3.2 Application-Malleable Overlay

Il y a plusieurs types d'overlay, certains ne prennent pas en compte l'application, d'autres le font seulement au démarrage, d'autres essayent de réagir aux évènements de l'application. MOve [23] souhaite mettre en place un overlay malléable, c'est à dire que les communications de l'application distribué vont influencer la structure de l'overlay. Les deux buts principaux sont d'optimiser les performances de l'overlay et de garder les propriétés de tolérance aux fautes et de passage à l'échelle. Les auteurs mettent en place deux types de liens :

- *Lien non-applicatif :* Ils maintiennent l'overlay global proche d'un graphe aléatoire avec un faible degré de clustering (regroupement).
- *Lien applicatif :* Ils permettent de créer un graphe fortement connecté, pour regrouper les nœuds. Chaque nœud d'un groupe crée aléatoirement des liens applicatifs vers des membres du groupe. Cela va permettre une rapide propagation des états lors des updates et des messages applicatifs multicast.

Cette solution offre différentes possibilités d'ajout de nœud, de détection de défaillance, de remplacement de lien, etc. *group*-based applications influencent l'overlay sous-jacent en remplaçant les liens inter nodes par des liens entres les applications. L'algorithme maintient une bonne connectivité.

3.3.3 Voronoi-based Overlay Network

VON veut exploiter la localité des intérêts des utilisateurs pour maintenir la topologie pair à pair avec un faible *overhead*. VON utilise un principe d'AOI dynamique, il utilise le découpage de Voronoi et chaque nœud est représenté par un site dans le diagramme. VON introduit trois procédures principales :

- *Join Procedure :* Le textitjoining node contacte le server pour avoir un ID unique, puis envoie une requête avec ses coordonnées à tous les nœuds existants. Création de la liste des voisins, mis à jour chez les voisins, etc.
- *Move Procedure :* Lorsque un nœud bouge ses coordonnées sont mises à jour chez ses voisins (gestion si nœud est à la limite de la région, si il y a des nouveaux voisins, etc).
- *Leave Procedure :* Le nœud se déconnecte (qu'importe sa raison et la manière) et ses voisins vont se mettre à jour.

VON permet de gérer un grand nombre d'utilisateurs et de garder un topologie consistante. Il y a beaucoup de messages introduits par l'AOI et si la vitesse des utilisateurs est trop importante, on a des problèmes pour notifier les nouveaux voisins.

4 Un système P2P pour MMOG : Solipsis

Solipsis est le "socle" du travail Blue Banana [16], il est donc nécessaire de présenter les points importants pour la compréhension du travail Blue Banana. Solipsis introduit des propriétés qu'il va être important de comprendre et qui serviront lors de l'analyse des résultats.

4.1 Introduction sur Solipsis

Pour commencer, le fonctionnement global et les fonctionnalités importantes de Solipsis vont être présentés. Solipsis est fait pour accepter un nombre illimité d'utilisateurs et pour maintenir une cohérence suffisante. Il peut être accessible par n'importe quel ordinateur, il peut fonctionner sur des ordinateurs peu puissants et avec des connections internet faibles (56Kbs) ou sans fil. Une fois les nœuds connectés, ils peuvent échanger des données telles que de la vidéo, du son, le mouvement d'avatar ou toutes choses affectant la représentation du monde virtuel.

4.2 Propriétés de Solipsis

Le monde de Solipsis est un tore à deux dimensions, chaque entité détermine sa position dans le monde et elle en est responsable. Chaque utilisateur collecte les informations lui permettant de reconstituer son environnement virtuel local. Les connections entre les nœuds sont bidirectionnelles.

4.2.1 Propriétés

Solipsis doit permettre aux utilisateurs de se déplacer à travers le monde, il faut pour cela que les deux propriétés locales suivantes soient respectées.

- *Local Awareness :*
  Une entité doit être connectée avec tous ses plus proches voisins. Il est important de noter qu'une entité a la possibilité de connaître des entités en dehors de son environnement virtuel local. En revanche, cette propriété impose que l'entité située à l'intérieur fasse partie des voisins de l'entité.
- *Global Connectivity :*
  Une entité doit connaître toutes les entités se trouvant dans son "champs de vision". Elle doit pouvoir détecter l'arrivée ou le départ d'une entité de son "champs de vision". Cette propriété est basée sur la Géométrie Informatique, elle assure qu'une entité ne "tourne pas le dos" à une partie du monde. L'enveloppe convexe des entités est le plus petit polygone convexe formé par cet ensemble. Un mécanisme pour éviter qu'une partie du graphe soit isolée a aussi été mis en place.

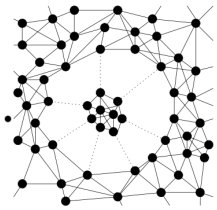

FIGURE 5 – Un composant connexe isolé et les connexions qui évitent la création d'une île

4.2.2 Maintien des propriétés

Solipsis a aussi des mécanismes de collaboration pour maintenir les propriétés précédentes, nous allons voir ces deux propriétés :

- *Spontaneous Collaboration for Local Awareness :*
  Pour vérifier la propriété *Local Awareness*, il faut qu'une entité puisse connaître tous ses voisins à chaque instant. Pour faciliter cette connaissance du voisinage, et comme il y a un grand nombre de mouvements dans le monde virtuel, un système de collaboration entre les nœuds a été mis en place. Une entité pourra alors demander régulièrement à ses voisins s'ils détectent une nouvelle entité mais cela implique un grand nombre de messages inutiles et une perte temporelle de consistence. Pour éviter ces problèmes, une entité a va prévenir une entité b si une entité c est en train de se diriger vers la zone de l'entité b.
- *Recursive Query-Response for Global Connectivity :*
  Il faut prévoir une mécanisme pour le maintien d'une entité dans l'enveloppe convexe. Quand une entité e détecte deux entités consécutives, elle se lance immédiatement à la recherche d'une ou plusieurs entités dans le même secteur en envoyant des messages dans la zone.

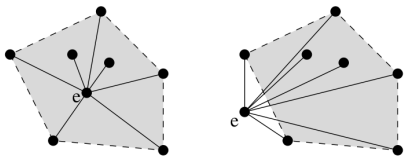

FIGURE 6 – Deux différentes enveloppes convexes des voisins de e. A gauche, e respecte la règle de *Global Connectivity* et à droite non.

5 Traces des utilisateurs dans les MMOGs

Après avoir exposées des solutions qui ne prennent pas en compte la mobilité des joueurs, nous allons étudier celle-ci pour introduire la solution proposé par le travail Blue Banana. Pour étudier la mobilité dans les MMOGs,différentes techniques de collecte de traces ont été mises en place, nous présenterons celles-ci et expliquerons pourquoi ce travail de collecte est important pour améliorer les performances des solutions pair à pair pour les MMOGs.

5.1 Les différentes techniques de récupération de trace

5.1.1 Les objectifs et les techniques de collecte de trace

Dans la littérature, il y a plusieurs études des traces des utilisateurs dans des différents environnements virtuels [26, 15]. La plupart vont récupérer les traces des avatars sur des jeux vidéos tel que World Of Warcraft [38] et Second Life [36]. Nous avons pu voir qu'il peut y avoir des différences entre des MMOGs. Par exemple dans Second Life l'environnement est beaucoup plus interactif (possibilités plus étendues de modification de l'environnement) que dans World Of Warcraft, ce qui peut affecter des différences de résultats des solutions en fonction du jeu [21, 20].

Ces travaux vont permettre de bien comprendre les différents comportements des joueurs, ils nous permettront de détecter les différents comportements en fonction des zones plus ou moins peuplées. Grâce à ces travaux, il sera possible de faire ressortir des modèles montrant le comportement des avatars dans le monde virtuel. Ces modèles permettent de mettre en place une distribution spatiale des avatars, qui sera plus cohérente que dans les recherches anciennes où les distributions étaient uniformes [17]. Différentes mesures vont apparaître comme : le nombre de joueurs, le nombre d'arrivées et de départs, le temps moyen d'une session, la distribution des joueurs, etc. L'étude des traces des joueurs permet aussi de détecter les tricheurs qui utilisent des bots.

Certaines techniques utilisent un bot qui est introduit dans le jeu et qui va récupérer des informations sur les autres joueurs. Le bot va récupérer des informations à intervalle régulier, nous pourrons ainsi analyser les déplacements et vérifier les modèles. Par exemple dans [21], une librairie open source *libsecondlife* a été développée pour mettre en place le bot dans Second Life. Pour vérifier que les données récupérées sont cohérentes, une technique de positionnement de sept bots dans une région ressort. Quatre bots statiques sont placés à chaque coin de la région, un autre statique va se placer au centre et deux autres vont bouger suivant un schéma défini. Nous pouvons alors enregistrer les positions des avatars se trouvant dans la région et nous aurons sept fois les informations sur les positions. Nous comparerons alors les données pour voir si des déviations existent entre les valeurs et si elles sont importantes.

5.1.2 Limitations de la collecte des traces

Des limitations existent pour la collecte des traces. Une des premières est que les mondes virtuels sont très souvent découpés en région ou en île or les bots que nous introduisons ne peuvent pas traverser les régions et ils ne peuvent pas, par exemple, suivre des avatars. Dans [21], la possibilité de différencier un avatar qui va dans une autre région et un qui quitte le jeu n'est pas possible. Nous ne pouvons pas savoir ce que va faire l'avatar en dehors de la zone. Il y aussi un problème avec la détection des avatars qui se trouvent sur des objets et il n'y pas de prise en compte de la coordonnée z. Une autre limitation de la collecte des traces est qu'elle se fait en très grande majorité de façon manuelle, il est donc très long et fastidieux de récupérer un nombre de traces suffisant pour effectuer des bonnes analyses. La collecte de trace peut aussi avoir des problèmes de passage à l'échelle car les systèmes de collecte existants travaillent à une petite échelle. Par exemple dans Second Life, le monde étant découpé en îles indépendantes, l'addition des traces collectées sur chaque île pour en faire une étude globale, n'est pas cohérente.

5.2 Observations des traces

La collecte de toutes ces traces a permis de faire beaucoup d'observations sur les déplacements des avatars dans les mondes virtuels. Cela a aussi permis de faire valider ou invalider des modèles qui avaient été mis en place [21].

5.2.1 Hotspots

Une des observations qui est ressortie de ces études est l'existence de différentes zones dans le monde, une autre observation est que les mouvements des avatars étaient très différents en fonction de la zone où ils se trouvent. Deux types de zone peuvent se dégager : des zones très peuplées avec des avatars ayant des mouvements très aléatoires et lents, et des zones qui sont entre les zones peuplées, où les avatars se déplacent rapidement et suivant souvent une trajectoire rectiligne. Des phénomènes de déplacement en groupe ont aussi pu être repérés [22].

5.2.2 Waypoints

Comme nous pouvons le voir dans le schéma 7, il y a des "routes" entre les différents points de regroupement. Ces routes vont nous permettre de mieux anticiper les déplacements des avatars entre les *Hotspots*.

L'étude des traces a aussi permis de détecter les habitudes des joueurs, comme le fait qu'ils jouent à certaines heures plutôt que d'autres et avec des durées de connexion différentes en fonction de l'heure de la journée.

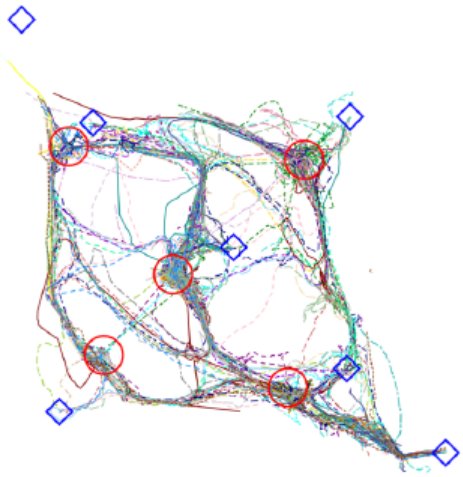

FIGURE 7 – Battle 980 movement paths

6 Prise en compte de la mobilité : Blue Banana

Il nous a été possible de voir, dans les chapitres précédents, des mécanismes qui permettent de faire évoluer le système en réaction à des événements. Solipsis ne pourrait difficilement fonctionner dans un monde avec des traces réalistes. En prenant en compte les études des traces des joueurs, Blue Banana a permis de mettre en place un mécanisme d'anticipation des mouvements des avatars pour mieux s'adapter.

Blue Banana présente une solution aux différents problèmes que peut rencontrer une architecture pair à pair dans les MMOGs. La mobilité des avatars implique de nombreux échanges de données à travers le réseau pair à pair. Comme les overlays de l'état de l'art n'anticipent pas cette mobilité, les données nécessaires ne seront pas chargées à temps, ce qui conduit à des défaillances transitoires au niveau applicatif. Blue Banana a été réalisée pour résoudre ce problème, il modélise et prédit les mouvements des avatars ce qui permet à l'overlay de s'adapter par anticipation aux besoins du jeux.

6.1 Solutions introduites

Blue Banana est implémenté au dessus de Solipsis qu'il a été possible d'étudier dans le chapitre 4. Plusieurs observations ont été faites et différentes optimisations en sont ressorties. Il nous a été possible d'observer plusieurs types de zone (dense ou non, cf. 5) et que les mécanismes d'adaptation sont trop tardifs pour être mis en place dans la réalité (le chargement des données sera trop lent).

6.1.1 Les états de l'avatar

Une des premières innovations qui a été introduite est la distinction de plusieurs états d'un avatar. Comme il a été possible de voir dans le chapitre sur la collecte de traces, un

avatar se comporte différemment en fonction des zones du monde. Trois états ont donc été introduits :

- **H**(alted) : l'avatar est immobile.
- **T**(ravelling) : l'avatar se déplace rapidement sur la carte et il a une trajectoire droite.
- **E**(xploring) : l'avatar est en train d'explorer une zone, sa trajectoire est confuse et sa vitesse est lente.

Le changement d'état de l'avatar se fait en fonction de la vitesse de celui-ci, si la vitesse devient supérieure à une borne définie et que l'avatar est dans l'état E alors l'avatar passe en état T. Ce modèle pourrait être affiné par la suite en prenant en compte l'accélération ou l'historique des mouvements. Sur la figure 8, nous pouvons mieux distinguer les différents changements d'état. Chaque nœud va agir en fonction de cette automate, il sera initialisé à l'état **H**(alted).

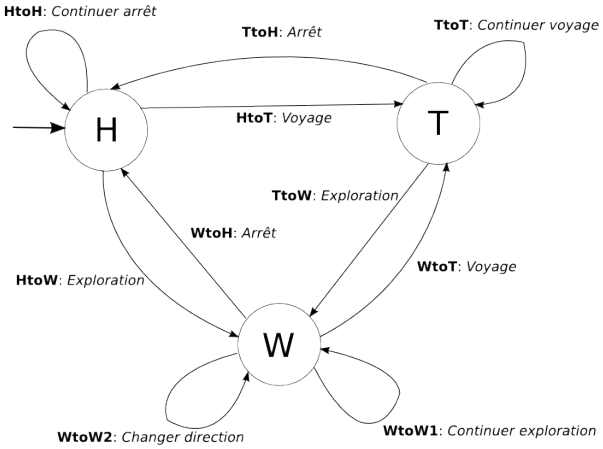

FIGURE 8 – *Automate décrivant les mouvements d'un avatar.* ***En gras*** *: le nom de la transition, en italique sa sémantique*

6.1.2 Anticipation des mouvements

Un autre mécanisme a été mis en place, il s'agit d'anticiper les mouvements d'un avatar, pour cela deux suppositions sont faites : seulement une prédiction courte est cohérente, et plus l'avatar se déplace rapidement, plus il y a de chance qu'il continue dans la même direction [19]. Comme nous pouvons voir sur la figure 9, en fonction du vecteur de mouvement de l'avatar, le nœud B, s'il est dans l'état **T**, va chercher des nœuds qui se trouvent sur la trajectoire probable de l'avatar, tant que son ensemble de voisins n'est pas plein. Le nœud B va envoyer un message aux voisins qui sont le plus près de lui par rapport au vecteur de mouvement. Un mécanisme pour évaluer si le nœud n'est pas

trop près, et donc rapatrier des données ne servirait pas car le temps des communications serait supérieur au temps du déplacement de l'avatar. Un des risques est de rapatrier des nœuds qui seront inutiles si l'avatar va changer de direction ou d'état. L'amélioration de ce point fait parti des futures pistes pour améliorer l'algorithme d'anticipation des mouvements.

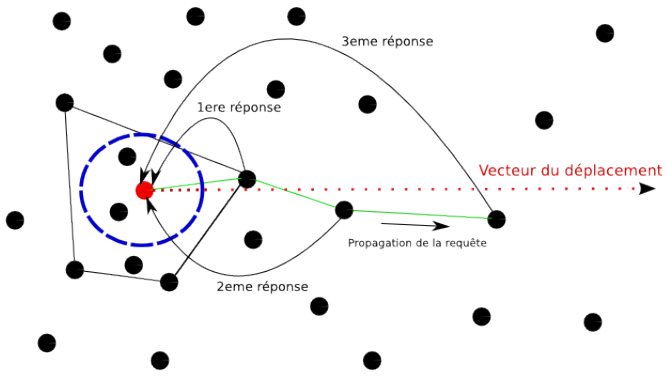

FIGURE 9 – Algorithme de propagation

6.2 Expérimentations et Résultats

Le travail a été testé sur le simulateur à évènements discrets PeerSim [14], les expérimentations ont eu pour objectif de comparer Solipsis avec et sans Blue Banana.

6.2.1 Le simulateur PeerSim et la description des expérimentations

PeerSim est un simulateur de réseau pair à pair, qui a deux modes de fonctionnement : par cycles ou par évènements. C'est une API riche et modulaire qui est codée en Java, c'est une composante du projet BISON de l'université de Bologne (Italie). Ce simulateur permet de simuler un large nombre de machines et de tester différentes configurations du réseau. Le simulateur va faire des simplifications sur les couches réseau et les contraintes physiques (latence, pannes, ...). Chaque nœud est considéré comme un module qui va échanger des messages avec les autres nœuds du système. La plupart des plateformes de simulation est basée sur le modèle à évènements discrets. Il est possible de distinguer deux entités : les nœuds et les messages. Le temps va seulement évoluer à chaque nouvel évènement sur un nœud.

Au départ de la simulation, une carte initiale des traces est introduite dans le simulateur, la carte provient d'une étude de La et Michiardi [18] dans Second Life. Ensuite, le simulateur va initialiser l'overlay de Solipsis et vérifier que les deux règles de Solipsis sont bien respectées sur chaque nœud, nous insérerons ensuite le reste des traces. Il faut aussi régler les différents paramètres du simulateur (nombre d'avatar, surface du monde, densité, accélération des avatars, vitesse de connection, etc). Plusieurs métriques sont mises en place pour évaluer les résultats :

- *Violation of Solipsis fundamental rules* : Regarde si les propriétés de *Global Connectivity* et de *Local Awareness* sont respectées.
- *Knowledge of nodes ahead of the movement* : Mesure pour les avatars qui se déplacent rapidement, le temps moyen pour qu'il connaisse un nœud qui sera sur sa trajectoire.
- *Exchanged messages count* : Mesure l'impact de Blue Banana sur le réseau, cela va compter le nombre de messages introduits par Blue Banana et Solipsis.

6.2.2 Les résultats

Les résultats les plus intéressants montrent que Blue Banana diminue les transitions en échec de 55% à 20%, augmentent la connaissance des prochains nœuds de 270% et cela en créant un overhead de seulement 2%. Les résultats montrent que le mécanisme d'anticipation introduit par Blue Banana aide l'overlay de Solipsis à s'adapter à temps et à réduire significativement le nombre de violation des règles de Solipsis (de 55% ou 80% à 20%).

7 Introduction des Solutions

Le problème a résoudre, durant le stage, est d'améliorer la réactivité des réseaux pair à pair pour les MMOGs. La faible réactivité des protocoles pair à pair existants ne permet d'assurer la latence suffisante pour ce genre d'application. Blue Banana permet l'amélioration de la réactivité en rapatriant des données qui nous seront nécessaires dans un futur proche. Le travail effectué dans Blue Banana (cf 6, page 17) s'intéresse à une partie du jeu bien définie, qui est les déplacements entre les zones denses. Durant le stage, il a donc fallu chercher des solutions qui permettraient d'améliorer le travail existant. Plusieurs pistes de solution ont été trouvées, deux de ces pistes ont été implémentées et testées. Les deux solutions implémentées, durant le stage, sont la mise en place d'un cache pour les zones denses de l'environnement, et la modification du rapatriement des données réalisé dans Blue Banana, pour les déplacements entre les zones denses.

Solution	Partie Améliorée
Introduction d'un cache	Dans les zones denses
Amélioration du rapatriement	Entre les zones denses

Table 1 – Tableau récapitulatif des solutions ajoutées durant le stage

Dans la première solution, l'intérêt s'est porté sur une partie du jeu qui n'était pas étudiée dans Blue Banana. Le comportement des avatars dans les zones denses restait le même que dans Solipsis, nous avons donc décidé d'améliorer le fonctionnent dans ces zones denses. A l'intérieur de celles-ci, les joueurs se déplacent de façon désordonnée et bougent la plupart du temps dans un secteur restreint. L'utilisation d'un cache est alors apparu comme une solution d'amélioration évidente. Le principe du cache est très simple, un nœud oubliait des voisins dont il pourrait avoir besoin dans un futur plus ou moins proche (en fonction de la mobilité dans le jeu), la mise en place du cache va permettre de garder un nombre N de nœuds dans l'éventualité où le nœud revienne dans une zone où il est déjà venu (voir Schéma 10). Nous expliquerons les différentes solutions que nous avons testé pour le cache, et pourquoi celles que nous avons retenu fonctionnaient mieux.

La deuxième solution a consisté à modifier le rapatriement des données réalisées dans Blue Banana, pour que celui-ci se fasse de manière plus fine. Les modifications que nous avons introduites doivent permettre d'économiser des messages et d'améliorer la cohérence de la topologie. Différentes techniques et algorithmes ont été testés, les plus importantes seront décrites et expliquées.

Les deux solutions mises en place permettent d'améliorer le travail qui avait déjà été effectué dans Blue Banana. L'intérêt de nos solutions est qu'elles permettent d'améliorer une partie de l'environnement qui n'avait pas était traité (cache pour les zones denses), et d'améliorer un travail existant.

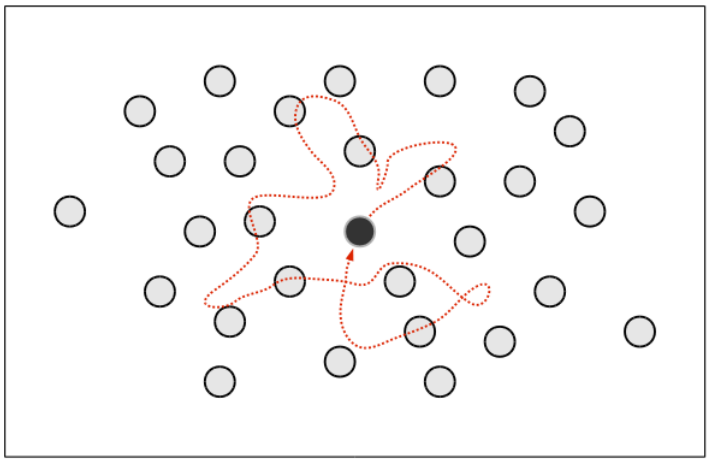

FIGURE 10 – Exemple d'une trajectoire d'un joueur dans une zone dense

7.1 Le modèle de mobilité

Un modèle de mobilité a été mis en place dans Blue Banana, nous allons le présenter rapidement car il sera utilisé pour les différents tests que nous avons réalisés pour les solutions mises en place. Pour cela, un degré de mobilité à un instant donné, a été introduit. Il s'agit du nombre d'avatars dans l'état **T** à cet instant sur le nombre total d'avatars. Le modèle de mobilité implémenté dans Blue Banana se base sur l'automate (cf partie 6.1.1). Toutes les 100ms, PeerSim décide si un avatar doit ou non changer d'état, il réalise ceci grâce aux différentes probabilités présentées dans le tableau ci-dessus. A chaque fois qu'un déplacement est décidé, une destination est choisie. L'avatar commence alors son déplacement, chaque mouvement est rectiligne mais non uniforme (car accélération). L'accélération est nulle lorsque la vitesse maximale est atteinte, cette vitesse maximale est choisie en fonction de la zone où se trouve l'avatar.

L'automate étant mis à jour toutes les 100ms, même un petit changement dans les probabilités, peut entrainer des gros changements sur la mobilité. La transition WtoT a été choisie, dans Blue Banana, pour faire varier la mobilité. Cette transition permet de déclencher le passage d'un avatar d'un état d'exploration désordonné à un état de déplacement rapide. Sur la figure 11, nous pouvons voir les différentes valeurs affectées à la transition et les degrés de mobilité correspondants.

Transition	Probabilité	État courant	Description
HtoH	0.85	Repos	Rester à l'arrêt
HtoT	0.0004	Repos	Commencer à voyager
HtoW	0.1496	Repos	Commencer à explorer
WtoH	0.01-x	Exploration	S'arrêter
WtoT	x	Exploration	Commencer à voyager
WtoW1	0.8	Exploration	Continuer à explorer dans la même direction
WtoW2	0.19	Exploration	Changer de direction d'exploration
TtoH	0.0002	Voyage	S'arrêter
TtoW	0.0005	Voyage	Commencer à explorer
TtoT	0.9993	Voyage	Continuer à voyager

TABLE 2 – *Tableau descriptif de l'automate de déplacement avec les probabilités associées à chaque transition. Les probabilités sont évaluées toutes les 100ms. x varie entre 0.0002 et 0.005 (cf. figure 11)*

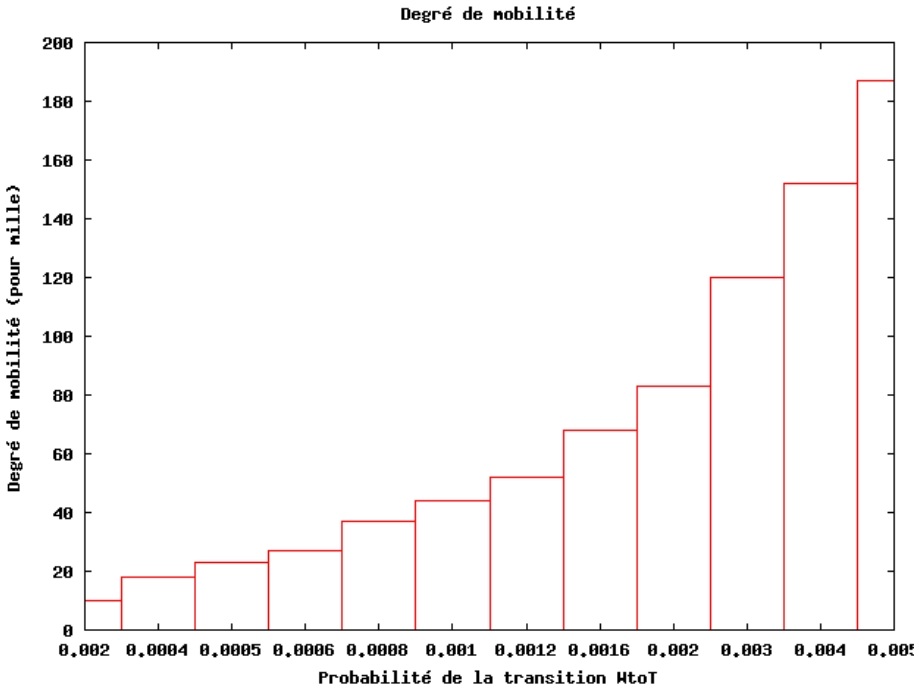

FIGURE 11 – *Influence du changement de probabilité sur la transition WtoT de l'automate sur le degré de mobilité du Métavers*

8 La mise en place d'un cache pour les zones peuplées

8.1 Introduction

Comme nous avons pu le voir dans la partie 7, la mise en place d'un cache pour les zones peuplées permettrait d'améliorer la réactivité dans l'état (**W**). L'avantage de cette solution, même si l'issue n'était pas certaine, était de s'intéresser à une partie qui n'avait pas encore été étudiée dans Blue Banana. Nous avons donc inséré un cache pour chaque nœud de l'environnement, celui-ci va fonctionner dans la continuité de liste des voisins d'un nœud. Nous expliquerons comment nous avons inséré le cache dans le code existant, les différentes stratégies que nous avons pu mettre en place et les différents paramètres qui vont influencer le fonctionnent du cache. Nous avons aussi permis l'utilisation du cache pour aider ses voisins quand ceux-ci cherchent des nœuds.

8.2 Explications de la mise en place du cache

8.2.1 Le fonctionnement global et la mise en place du cache dans Blue Banana

Le fonctionnement global du cache consiste à garder en mémoire un certains nombre de nœuds qui faisait parti de la liste des voisins. Ainsi comme les mouvements de l'avatar sont désordonnés, il est possible qu'il retourne vers des nœuds qu'il vient de quitter.

Sur la figure 12, nous pouvons voir les principales étapes du fonctionnement du cache. Au départ le cache et la liste des voisins sont remplis de différents nœuds. A l'étape 2, le nœud courant (rouge) se déplace et trouve des nouveaux voisins, ceux-ci sont insérés à sa liste des voisins. Deux nœuds sont alors déplacés vers le cache, ce qui pousse deux nœuds hors du cache. A la dernière étape, le nœud courant revient vers une zone qu'il connait (nous détaillerons après le mécanisme de recherche dans le cache) et il trouve deux nœuds dans son cache qui pourraient lui servir pour reconstruire son voisinage. Deux nœuds du cache sont alors insérés dans la liste des voisins du nœud courant, ce qui envoie deux nœuds de cette liste vers le cache. Dans cette exemple, plusieurs nœuds peuvent se déplacer en même temps, ce qui est le cas dans une seule des solutions de cache mise en place.

Cette solution permet d'économiser des messages de découverte des voisins (SEARCH) dans le cas de changements de direction fréquents. Il doit aussi nous permettre d'économiser des messages de connexions et de déconnexion. Nous verrons dans la partie 8.3 les différents gains de la mise en place du cache, mais aussi les limites de celui-ci. Le cache est donc le prolongement de la liste des voisins pour nœud, mais contrairement à celui-ci , il n'est pas à jour, car un nœud est ajouté dans le cache avec les dernières informations que nous avions quand il était dans la liste des voisins. Nous avons donc mis en place un système de mise à jour du cache, pour avoir des informations les plus exactes possibles sur chaque nœud. Mais ce système va coûter cher en nombre de messages, et cela est encore plus vrai plus le cache est grand. Un mécanisme de datation des éléments du cache permet de savoir depuis quand date les informations de chacun. Ce mécanisme nous permet de contacter un nœud pour savoir s'il se trouve toujours à peu près au

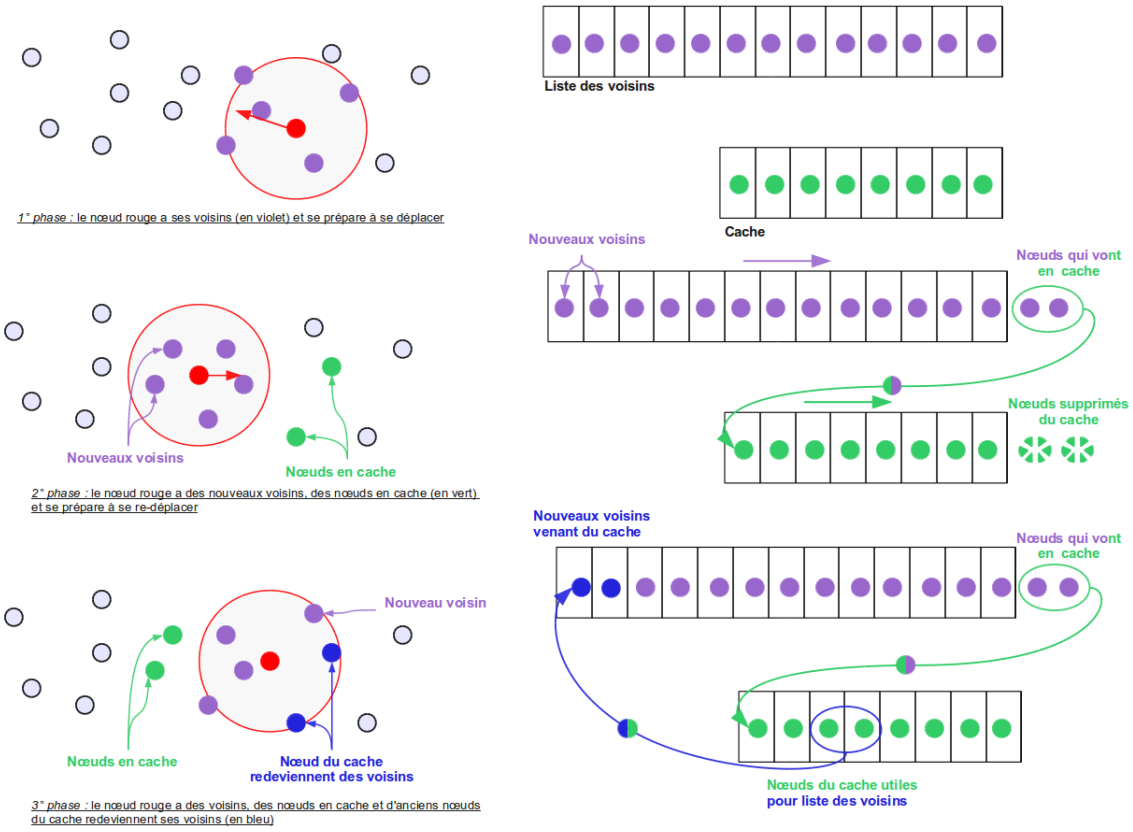

FIGURE 12 – Exemples du fonctionnement global du cache

même endroit, et ainsi l'ajouter ou non à notre voisinage. Ces paramètres sont en option et nous verrons dans la partie 8.3 quelles sont les meilleures combinaisons d'options et si elles sont toutes utiles ou non.

8.2.2 Les différentes versions du cache

Deux versions, pour le fonctionnent du cache, ont été testées durant la phase de codage. Nous parlons ici de la gestion des données dans le cache et non de la recherche dans celui-ci qui se fera juste après. Au départ le cache était géré selon le principe *First In First Out*, la gestion du cache ne tenait pas compte des mises à jours. Ces mises à jour peuvent avoir lieu lors d'une requête en échec vers un nœud du cache.Le nœud ayant trop bougé est alors mis à jour dans le cache. Une gestion du cache en fonction de la localité a aussi été mise en place, ce qui permet de faire sortir du cache les nœuds les plus éloignés de la position actuelle du nœud.

Deux implémentations ont aussi été testées pour la recherche dans le cache, l'une renvoie un résultat et l'autre renvoie plusieurs nœud à ajouter. Les tests montrent que la deuxième implémentation donne de meilleurs résultats (voir chapitre 8.3).

8.2.3 La modification du code existant pour insérer la recherche dans le cache

Avant de regarder les algorithmes de recherche dans le cache, nous allons vous expliquer comment ils sont appelés et quelles sont les modifications introduites par rapport au code original. Lorsqu'un nœud rentre dans la fonction *solipsisRecoverTopology* si le nœud est dans l'état **Wandering** alors on passe dans la fonction *MaintainTopology*, sinon on effectue le traitement normal. Nous pouvons voir ci-dessous une partie du code de la fonction MaintainCache. Pour commencer, nous testons l'état de l'entité appelante, ensuite en fonction de la stratégie, nous faisons un traitement particulier. Nous nous concentrerons sur la stratégie de base qui ajoute un seul nœud. La fonction de recherche nous renvoie donc un résultat. S'il n'est pas nul et que sa date de mise à jour n'est pas trop ancienne, nous enlevons le nœud du cache, ajoutons le nœud à la liste des voisins et renvoyons 1 à la fonction appelante pour lui signifier que le traitement à été fait. Ensuite en fonction de la valeur de l'option *contact_node*, nous contactons ou non le nœud renvoyé par la fonction de recherche. Nous faisons ceci pour savoir s'il a beaucoup bougé depuis le dernier moment où nous l'avons vu. Nous retournons 0 pour signifier à la fonction appelante qu'aucun nœud n'a été ajouté et qu'elle peut faire le traitement de base.

Partie du code de la fonction MaintainCache

```
1  public int maintainCacheTopology() {
     ...
     if ( this.mainVirtualEntity.getStateMachine().getState() == WANDERING ) {
       switch (this.strategieCache) {
       case SolipsisProtocol.FIFO:
6        neighbor = cache.searchCacheNeighborKnoledgeRay(...,this.knowledgeRay);
       if (neighbor != null){
         if ( neighbor.getTime() + time_limite > CommonState.getIntTime()){
           cache.RmCache(neighbor);
           addLocalView(neighbor);
11         return 1;
         }
         if (contact_node == 1){
           ...
           cache_request = new CacheRequest(neigh,source);
16         cache_test = new Message(Message.CACHE_UPD, this.getPeersimNodeId(),
             this.mainVirtualEntity.getId(), neighbor.getId(), cache_request);
           neighbor.setQuality(NeighborProxy.CACHED);
           this.send(cache_test,neighbor);
           return 0;
21       }else{
           return 0;
         }
       }else{
         return 0;
26     }
       case SolipsisProtocol.FIFOMULT:
         ...
     }
   }
```

8.2.4 Les algorithmes de recherche dans le cache

Nous allons expliquer comment nous recherchons les données dans le cache, et nous détaillerons les différentes pistes que nous avons testé dans un ordre chronologique. Tout d'abord nous avons sélectionner les nœuds en terme de distance. L'idée était de récupérer les plus proches de notre nouvelle position. Cette solution était assez simple à mettre en place, mais nous pensions que les résultats seraient meilleurs si nous prenions en compte la capacité d'un nœud à aider à refaire l'enveloppe connexe du nœud courant. Nous avons alors implémenté une solution qui rendait un résultat positif si un nœud dans le cache permettait de reconstruire l'enveloppe du nœud (voir schéma 13). Cette solution a même été agrémentée d'un test si le nœud faisait avancer positivement l'enveloppe connexe mais ne la reconstruisait pas immédiatement.

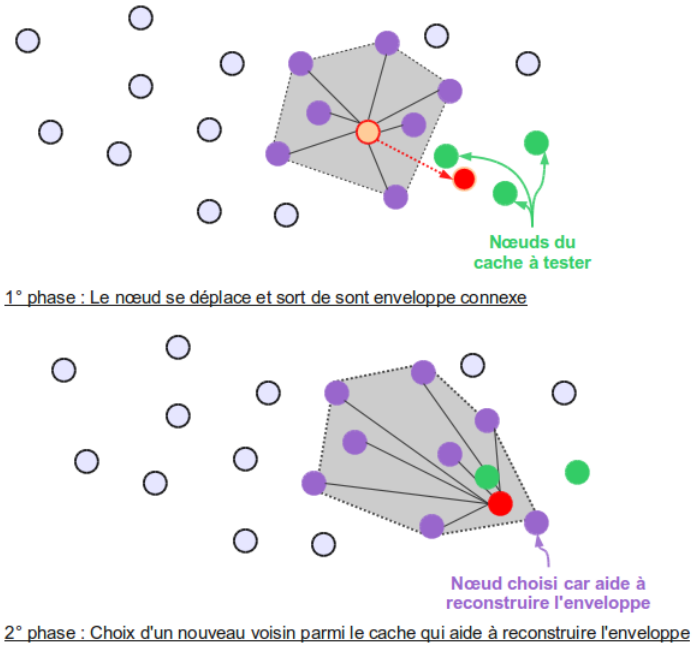

FIGURE 13 – Schéma montrant la solution de recherche dans le cache aidant à reconstruire l'enveloppe

Le problème de cette solution est qu'elle privilégiait l'enveloppe connexe à la propriété de *Local Awareness*. Car nous pouvons nous retrouver dans la situation, comme dans la figure 13, où un nœud ne fait pas parti de la liste des voisins alors qu'il se trouve dans l'enveloppe connexe. Cette solution nous donnait donc des résultats, pour les propriétés de Solipsis, qui étaient moins bons que la version sans le cache.

Nous sommes alors revenus à une solution ressemblant à la première solution que nous avions mise en place. Chaque nœud comporte une zone de connaissance, nous avons donc décidé de nous servir de cette dernière pour réaliser les conditions dans la fonction de recherche. La fonction de recherche regarde si un nœud du cache est dans

la zone de connaissance du nœud courant. Si plusieurs correspondent un système pour choisir aléatoirement est mis en place, comme dans la recherche de voisin original. Une autre fonction qui recherche en prenant en compte la région géométrique a aussi été mise en place, les résultats sont équivalents à la version avec la zone de connaissance.

8.2.5 La mise en place de l'aide aux voisins grâce au cache

Le cache d'un nœud va donc lui servir pour essayer de connaitre son environnement plus rapidement, mais il pourrait aussi aider les voisins du nœud. Dans cette optique, l'aide aux voisins a été mise en place. Ceci permet au cache d'avoir une autre utilisation, ce qui pourrait permettre d'économiser quelques autres messages.

Lorsqu'un nœud cherche des voisins, il envoie un message SEARCH (voir schéma 14). Il suffit donc d'insérer une recherche dans le cache au bon endroit dans la fonction de traitement de ce message. Nous insérons donc, dans la fonction *processSearchMsg*, une fonction de recherche dans le cache. Le nœud va tout d'abord regarder dans sa liste des voisins et si aucun ne correspond, nous regardons dans le cache si un nœud peut correspondre à la requête. La recherche dans le cache se fait, de façon similaire à une recherche dans le cache pour un nœud local, en utilisant la zone de connaissance que l'on aura préalablement transmis dans le message.

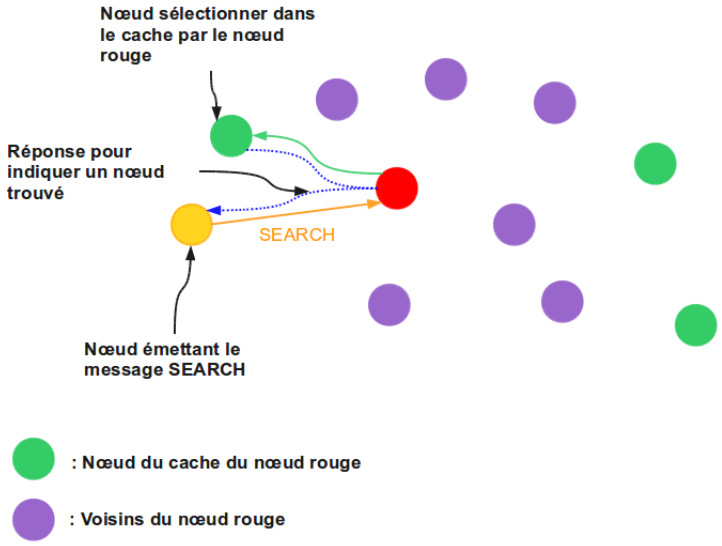

FIGURE 14 – Schéma montrant l'aide du cache pour le traitement d'un message SEARCH

8.3 Résultats et observations sur le cache

Nous allons présenter les différents résultats sur le cache. Nous comparerons chaque version avec une version de base où l'on ne trouve ni cache ni prefetch, et avec une version avec le prefetch implémenté dans Blue Banana. Les principales métriques pour comparer

les différents résultats sont la cohérence de la topologie et le nombre de messages qui sont exprimés en fonction de la mobilité des avatars. Le calcul de la cohérence de la topologie consiste à mesurer, à chaque instant, le nombre de nœud qui sont dans la zone de connaissance d'un autre nœud mais qui ne fait pas parti des voisins de ce dernier.

8.3.1 Résultats avec une première configuration du cache

Tout d'abord, nous allons regarder le nombre de cache Hit et Miss pour le cache normal. Il s'agit de voir le taux de réussite du cache et donc si ce mécanisme est souvent utlisé. Dans ces premiers résultats, le cache est configuré de tel sorte qu'il n'utilise pas l'aide aux voisins, qu'il ne contacte pas un nœud du cache s'il est trop vieux. Cette configuration ne va récupérer que les nœuds du cache qui sont très proches et qui ont été ajoutés au cache récemment (voir tableau ci dessous). La taille du cache correspond au nombre de nœud qu'il peut contenir, la limite de distance correspond à la distance à partir de laquelle nous ne récupèrerons pas le nœud. La limite de temps est la différence maximum qu'il doit y avoir entre le temps courant et la date de rafraichissement du nœud dans le cache (généralement l'ajout de celui-ci dans le cache).

Paramètre	Valeur
Taille du cache	25
Limite de distance	1500
Limite de temps	1500
Contact Nœud	Faux
Mise à jour du cache	Faux
Aide aux voisins	Vrai

Table 3 – Tableau montrant les valeurs utilisées pour la configuration n°1

Nous pouvons voir que plus la mobilité augmente plus le nombre de cache Hit augmente et le nombre de cache Miss diminue (voir figure 15). Cela est dû au fait que plus la mobilité augmente plus le cache aura des entrées récentes, car les nœuds vont changer de direction plus souvent et plus rapidement. Notre politique de cache ne prenant en compte que des nœuds récemment ajoutés au cache, les résultats en terme de cache Hit sont bien meilleurs. Nous pouvons aussi noter qu'il ya de moins en moins d'accès au cache car la somme des Hit et des Miss diminue, cela est du au fait que de plus en plus de nœuds sont dans un état **T**.

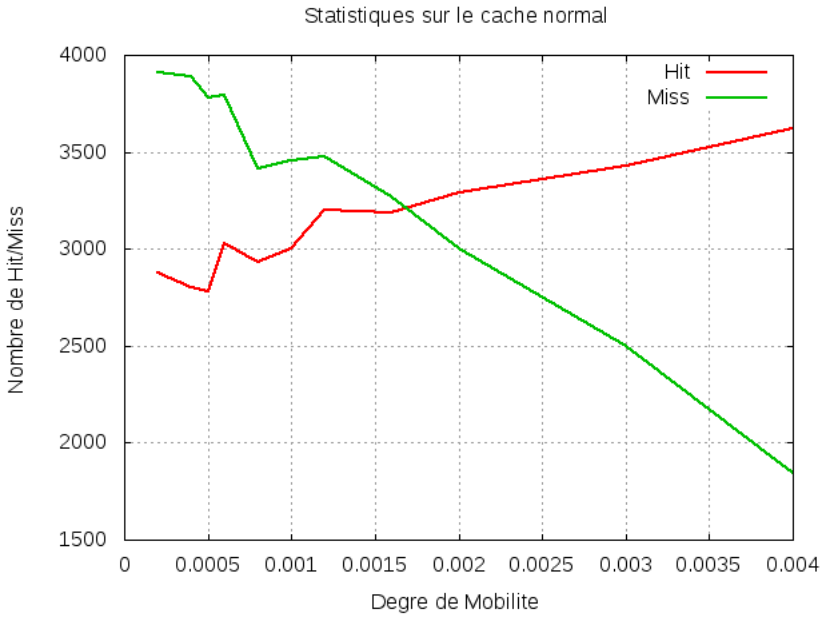

FIGURE 15 – Schéma montrant les caches Hit et caches Miss pour le cache normal

Ensuite nous pouvons observer le nombre de messages en fonction du degré de mobilité (voir figure 16). Les deux versions du cache vont nous permettre d'économiser des messages car lors d'un cache Hit nous ne faisons pas le traitement de base qui provoquait l'envoie d'un message. Le traitement du cache se fait sans coût en terme de message. Si l'on fait le traitement de base après le passage dans le cache, un gain en nombre de message sera tout de même réalisé. Ce gain résulte du fait que le nœud connaîtra mieux son environnement et la prochaine fois qu'il devra entrer dans la fonction *MaintainCache* sera retardée. Ces observations sont les mêmes pour le cache à réponse simple et le cache à réponse multiple.

Nous allons maintenant observer la cohérence de la topologie avec les différentes versions du cache. PROBLEME AVEC CACHE SIMPLE. La version du cache fonctionnant avec une recherche renvoyant plusieurs nœuds donne des résultats meilleurs que la version de base (sans cache et sans prefetch). A FINIR

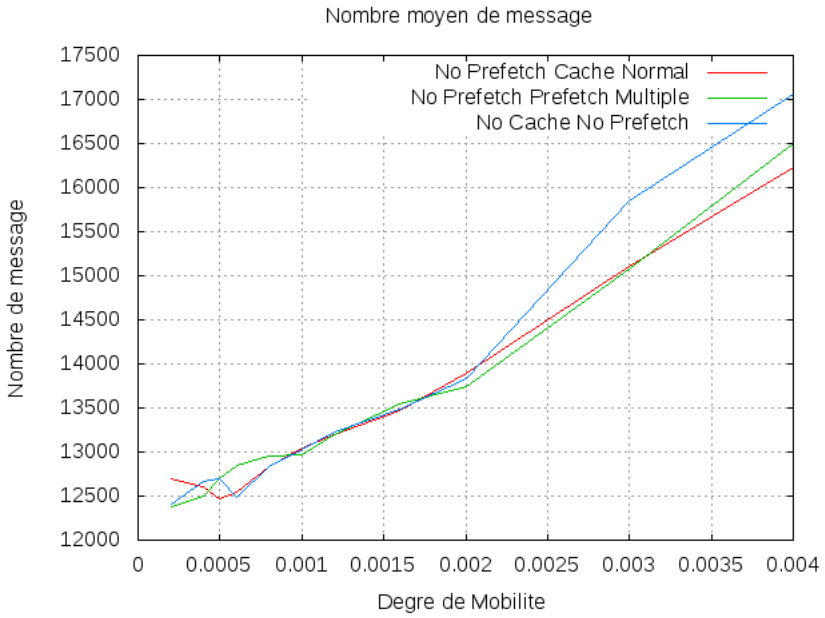

FIGURE 16 – Schéma montrant le nombre de messages

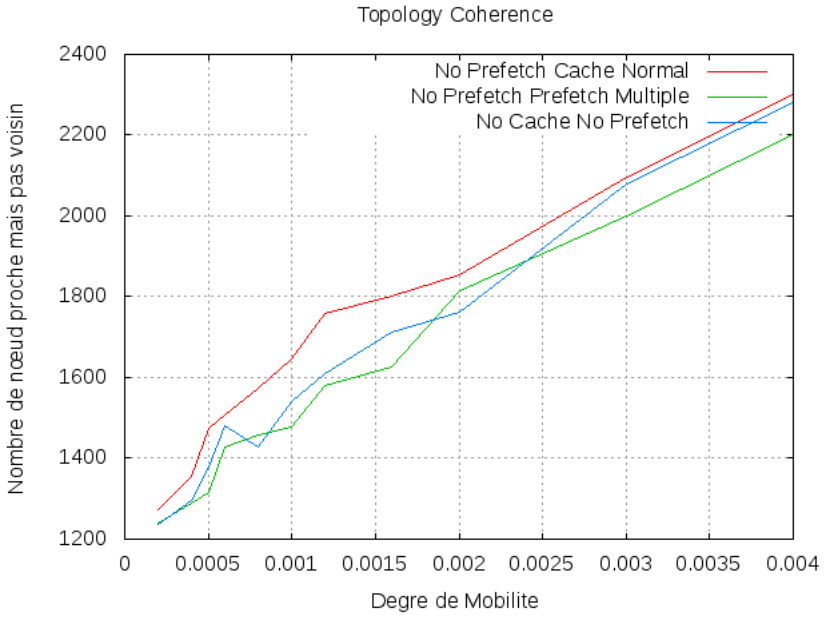

FIGURE 17 – Schéma montrant la cohérence de la topologie

8.3.2 Résultats avec une seconde configuration du cache

A FINIR ? ?

Paramètre	Valeur
Taille du cache	25
Limite de distance	3500
Limite de temps	3000
Contact Nœud	Vrai
Mise à jour du cache	Faux
Aide aux voisins	Vrai

TABLE 4 – Tableau montrant les valeurs utilisées pour la configuration n°3

8.3.3 Résultats avec une troisième configuration du cache

Paramètre	Valeur
Taille du cache	25
Limite de distance	3500
Limite de temps	3000
Contact Nœud	Vrai
Mise à jour du cache	Vrai
Aide aux voisins	Vrai
Fréquence de mise à jour	5000

TABLE 5 – Tableau montrant les valeurs utilisées pour la configuration n°3

A FINIR ? ?

8.4 Conclusion et perspectives du cache

La mise en place du cache permet d'économiser des messages et cela est d'autant plus vrai que la mobilité est grande. Un des problèmes dans les résultats du cache est qu'une version simple fait perdre de la cohérence de topologie, et ce même si l'on effectue aussi le traitement de base. WHY ? RE TESTER et MODIFIER CODE ?

9 Amélioration du prefetch de Blue Banana

L'objectif est de rapatrier plus finement les données que dans la solution introduite dans Blue Banana, et surtout d'essayer de ne pas rapatrier des données inutiles. Pour le moment, nous regardons juste les nœuds qui sont dans notre champs de vision (un cône) et qui ne sont ni trop loin, ni trop près. Mais il serait intéressant d'avoir plus d'informations sur les nœuds avant de les rapatrier, comme leur direction ou leur vitesse par exemple. Nous avons donc regarder quelques critères qui pourraient permettre de rapatrier les données de façon plus fine.

9.1 Les changements introduits sur la version de Blue Banana

Actuellement un nœud, qui est dans l'état **T**(ravelling), va chercher des nœuds qui se trouvent sur la trajectoire probable de l'avatar, tant que son ensemble de voisins n'est pas plein. Ce mécanisme va donc rapatrier des données qui sont à bonne distance (pas trop près à cause des temps de communication). Un des risques difficilement évitable est de rapatrier des nœuds qui sont inutiles si l'avatar, dont nous rapatrions les données, change de direction ou d'état. Un autre problème est que l'on peut rapatrier des avatars qui sont dans le cône mais qui s'en écartent ou des avatars qui arrivent à grande vitesse vers notre avatar (voir figure 18). Le mécanisme existant n'observe pas les différentes propriétés des nœuds (vitesse, direction, état, etc). Des nœuds sûrement superflus vont être rapatriés : modifier le prefetch pourrait nous permettre de faire ce traitement plus finement.

Nous pouvons voir un exemple des modifications qu'entraineraient le nouveau prefetch, dans le schéma de la figure 18. Les nœuds verts représentent les nœuds qui seront rapatriés, les nœuds rouges ceux qui sont dans la zone de rapatriement mais qui ne le seront pas. Sur la figure, nous pouvons voir que nous gardons les nœuds qui sont stables, qui bougent peu ou qui bougent dans le même sens que le nœud courant (nœud gris foncé). Les nœud qui sont rouges ont des directions inverses au nœud courant et des vitesses élevées, ils ne seront donc pas rapatriés.

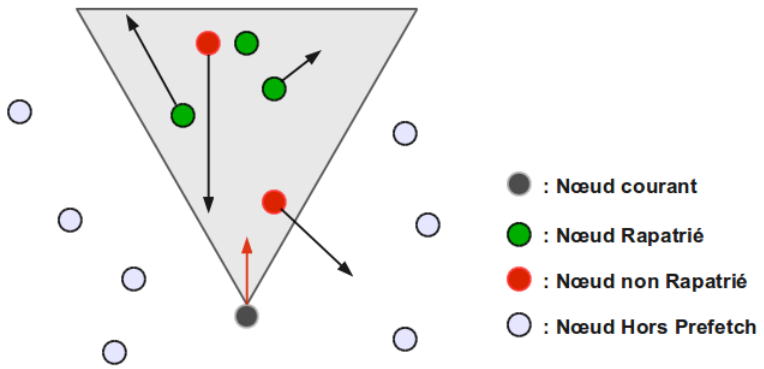

FIGURE 18 – Exemple de gain possible pour le prefetching

Pour réaliser cette sélection, il faut donc, en plus de la distance avec le nœud, tester différentes autres propriétés. Dans les messages de prefetch actuels, se trouve le vecteur du nœud faisant la requête de prefetch. Nous pourrons, avec celui-ci, regarder la direction et la longueur de ce nœud. Nous pourrons ainsi comparer ces critères chez le nœud faisant la requête et le nœud qui a reçu le message. Nous avions commencé par filtrer les nœuds qui étaient dans l'état **T**(ravelling), en faisant la somme du vecteur de prefetch et de celui du nœud courant ainsi si la norme est plus grande que la norme du prefetch, nous sélectionnons ce nœud. Nous faisons alors le traitement correspondant si la somme était plus grande que la norme du vecteur de prefetch. Cette solution laisse passer des problèmes que nous souhaitions éviter et va supprimer le rapatriement de nœud qui aurait pu être intéressant. Ainsi un nœud qui avance de façon très lente vers le nœud de prefetch ne serait pas pris car la norme serait plus petite. Ce cas peut être réglé par la mise en place d'une longueur de battement.

$$\textit{Somme des longueurs} \geq \textit{Norme du vecteur de prefetch} +/- \Delta$$

Les problèmes d'un nœud qui arrive à grande vitesse vers le nœud de prefetch ou d'un nœud qui a une direction qui s'écarte du nœud de prefetch, restent toujours présents. Nous avons donc cherché à regarder en plus des normes, les directions des différents nœuds et de les comparer.

En utilisant les directions, une des idées a été de rapatrier de préférence les nœuds qui vont dans la même direction (vois figure 19). Si les nœuds vont vers les cotés, par rapport au vecteur de prefetch, il faut que la longueur du vecteur soit alors pas trop grande pour qu'il soit rapatrié. De même pour les nœuds qui vont dans la direction opposée au vecteur de prefetch.

Les ajouts ont été faits dans le module de mise en place du prefetch de Blue Banana. Des fonctions de comparaison d'angle et de norme y ont été ajoutées, et nous les avons insérées dans la fonction *processPrefetchMsg* (voir code ci-dessous). Après avoir fait les traitements de base, comme vérifier que le nœud soit assez loin, nous testons pour savoir si le prefetch amélioré est activé.

Nous rapatrions alors les nœuds si :

- L'angle du vecteur du nœud courant est proche de l'angle du vecteur de prefetch (environ 45° de chaque côté).
- La somme des normes est supérieure à celle du vecteur de prefetch multipliée par un et demi.
- Le nœud courant n'est pas dans l'état **T**(ravelling).
- L'angle du vecteur du nœud courant n'est pas proche de l'angle du vecteur du nœud de prefetch mais la norme du nœud courant est inférieure à celle du nœud de prefecth.

Si le prefetch amélioré n'est pas activé, nous exécutons le traitement initial proposé par Blue Banana. Avec notre amélioration, nous devons donc émettre moins de message et améliorer la cohérence de la topologie si la sélection est bien faite.

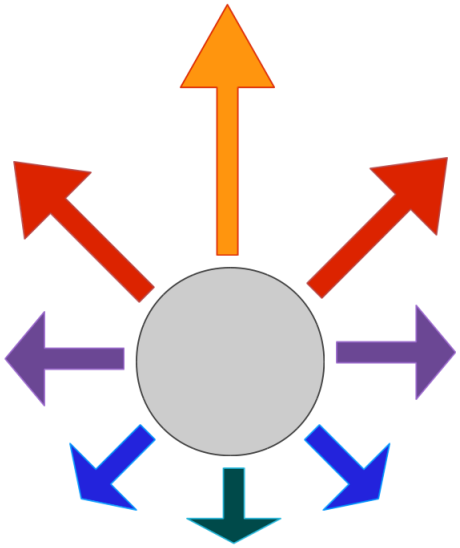

FIGURE 19 – Préférence de prefetching

Partie du code de la fonction processPrefetchMsg

```
public void processPrefetchMsg(Message msg) {
    ...
    if (!this.protocol.hasNeighbor(prefetch.getSource().getId())) {
        this.sendPrefetchConnectMessage(prefetch.getSource());
    }
    if (ttl > 0) {
      if (this.isFarEnough(prefetch)) {
        destinations = this.choosePrefetchDestinations(prefetch);
        size = destinations.size();
        size = (size > ttl)?ttl:size;
        msg.setTtl(ttl - size);
        if (size > 0) {
          this.protocol.send(msg, this.proxies.get(destinations.get(0)));
          for (int i = 1; i < size; i++) {
            if (this.protocol.getPrefetch_ameliore() == 1){
              if (isGoodPrefetch(...) || isGoodDirection(...) ||
              isMaybeGoodPrefetch(...) || this...getState() != TRAVELLING ){

                propagateMsg = this.protocol.createFoundMsg(...);
                this.protocol.send(propagateMsg, ...);
              }
            }else{

              propagateMsg = this.protocol.createFoundMsg(...);
              this.protocol.send(propagateMsg, ...);
            }
          }
    ...
}
```

9.2 Les résultats et les observations sur le prefetch amélioré

Nous allons présenter les différents résultats obtenus avec le prefetch nouvelle version. Nous comparerons notre version à une de base où l'on ne trouve ni cache ni prefetch, et avec une version avec le prefetch implémenté dans Blue Banana. Les principales métriques pour comparer les différents résultats sont la cohérence de la topologie et le nombre de message qui sont exprimés en fonction de la mobilité des avatars.

Le changement que nous avons implémenté, dans le prefetch, nous permet d'économiser des messages comme il est possible de voir sur la figure 20. Cela est dû au fait que nous examinons les nœuds avant de les rapatrier. En ne rapatriant pas tous les messages, nous économisons donc des messages de réponses au nœud qui a demandé le prefetch. De plus comme le rapatriement se fait de façon plus fine, nous rapatrions moins de nœuds inutiles ce qui permet de faire moins de requêtes de recherche de voisins par la suite.

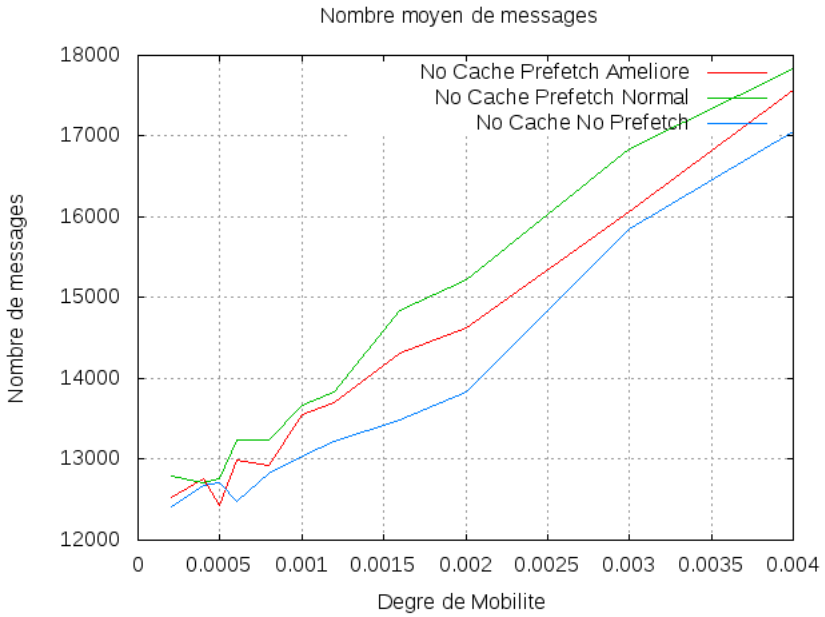

FIGURE 20 – Schéma montrant le nombre de messages

La solution que nous avons mis en place permet d'avoir une meilleure cohérence de la topologie. Certains nœuds, qui était pris et qui se révélaient être inutiles, ne sont plus rapatriés. Le nombre de nœuds qui est dans la zone de connaissance d'une autre nœuds mais pas dans sa liste des voisins, augmentent logiquement avec le degré de mobilité. Ce nombre augmente car les mouvements sont en hausse et il y a donc de plus en plus de changements dans la liste des voisins à faire, et les différents délais des messages ou des mécanismes sont limitant pour conserver une bonne topologie. La modification du prefetch améliore aussi sensiblement la règle de l'enveloppe connexe de Solipsis.

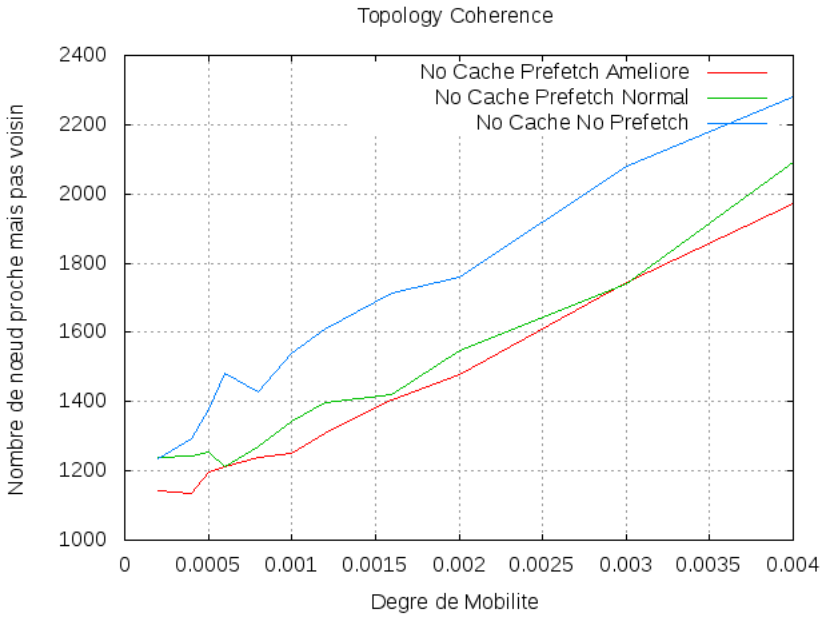

FIGURE 21 – Schéma montrant la cohérence de la topologie

9.3 Conclusion et perspectives

Le gain sur la cohérence de la topologie de notre solution, est minime mais étant donné qu'il économise en plus des messages, le résultat est donc positif. Le gain sur la topologie pourrait sûrement être meilleur si nous regardions encore d'autres paramètres. Mieux regarder les directions, en fonction de la distance des nœuds avec le nœud qui souhaite rapatrier, pourrait peut être permettre de mieux choisir les nœuds à rapatrier. Une autre amélioration pourrait être de chercher parmi des nœuds qui peuvent être hors du cône, mais dans un périmètre proche pour ne pas rajouter trop de nœuds à contacter sinon trop de messages seraient alors émis. Si ce mécanisme permet de d'augmenter l'efficacité du prefetch, il peut aussi être possible de regarder avec un angle plus grand de façon périodique.

10 Les autres améliorations possibles

Une partie du stage a été de rechercher comment il serait possible de faire évoluer le travail réalisé dans Blue Banana. Les différentes pistes, que nous avons trouvé, vont donc être présentées. La première consiste à se servir des déplacements en groupe des avatars. Une présentation de ces mouvements de groupes permettra de mieux comprendre en quoi la piste d'amélioration peut être intéressante. Un mécanisme de connaissance des routes entre les Hotspots pourrait aussi être intéressant. Car actuellement les avatars se déplacent de façon aléatoire entre les Hotspots, alors que dans les jeux vidéos les joueurs utilisent souvent les mêmes routes pour se déplacer. Cette piste permettrait de rendre les simulations et les solutions plus adaptées aux comportements des joueurs dans les MMOGs.

10.1 Les déplacements en groupes

Les activités en groupe des avatars dans les MMOGs sont une part très importante de l'expérience que le joueur de MMOGs recherche [7, 6]. Un tour d'horizon rapide des comportements sociaux des joueurs permettra de comprendre l'intérêt de cette solution. Des études ont démontré que des mouvements groupés existaient dans World of Warcraft [22].La coopération et le complémentarité des joueurs d'un même groupe permettent la réussite plus facile des missions. Les groupes (guildes) sont donc de plus en plus importants, et le fonctionnement fait penser au fonctionnement d'une famille de la mafia [13].

10.1.1 Étude des habitudes des joueurs de MMOG

Plusieurs études [10, 27, 6, 25] des comportements des joueurs mettent en avant l'importance des différentes sortes d'interactions sociales. Pour 41% des joueurs, l'interaction sociale est l'aspect favori des MMOGS [10]. Même si certaines études [9, 8] expliquent que les joueurs préfèrent jouer seul, elle sont toutes d'accord pour dire que les interaction sociales sont courantes, et sont un facteur important des MMOGs.

Les liens sociaux entre les joueurs ressemblent à des interactions dans le monde *réel*, avec des comportements d'adhésion que l'on peut comparer aux mariages, avec des échanges commerciaux et même des *services d'église*. Des normes sociales sont aussi présentes dans le jeu en plus des règles mises en place par l'éditeur (jargons, abréviations, émoticones, etc). Dans [25], l'auteur a réalisé une étude sur les habitudes des joueurs de EverQuest. Il ressort de cette étude que les personnes interrogées prennent beaucoup de plaisir à avoir des relations sociales dans le jeu (3° action la plus appréciée). Lorsque les joueurs jouent en groupe, ils le font la majorité du temps avec des gens qu'il connaissent déjà (moins de 10% avec des inconnus). Plus de 50% des sondés sont très contents d'être dans leur guilde.

Dans [9], les auteurs se sont intéressés aux dynamiques sociales dans les MMOG, et particulièrement dans le jeu World Of Warcraft [38]. Une des premières observations est que les joueurs vont jouer différemment en fonction de leur niveau. Il est possible de

remarquer que le temps passé en groupe évolue en fonction du niveau du joueur (voir figure 22). Le pourcentage de temps passé en groupe évolue en même temps que le niveau du joueur.

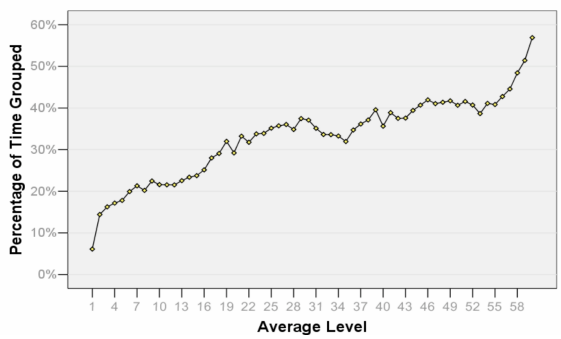

FIGURE 22 – Temps moyen passé en groupe, par niveau

World Of Warcraft encourage les joueurs à former des groupes en utilisant deux mécanismes. Premièrement, pour qu'une complémentarité entre les habilitées des joueurs se crée. Deuxièmement, beaucoup de quêtes dans le jeu sont difficiles à réaliser tout seul. De même qu'il y a des différences de temps passé en groupe en fonction du niveau, des différences se dégagent entre le temps passé en groupe en fonction de chaque espèce se distingue aussi. Dans World Of Warcraft, 66% des avatars appartiennent à une guilde et ce chiffre atteint 90% si l'on tient compte des joueurs ayant au moins le niveau 43. Les joueurs appartenant à une guilde joue en moyenne plus souvent qu'un joueur sans guilde.

Dans la figure 23, il est possible de voir à quoi ressemble les liens entre les différents joueurs d'une même guilde de 41 membres. Tout d'abord 17 membres de la guilde n'ont jamais été observés dans la même zone qu'un autre membre. Un noyau central se distingue, il est composé de 8 joueurs qui jouent souvent ensemble, 3 autres joueurs forment un trio central où les liens épais montrent qu'ils passent beaucoup de temps ensemble. Les autres joueurs jouent avec 2 (ou moins) membres de la guilde.

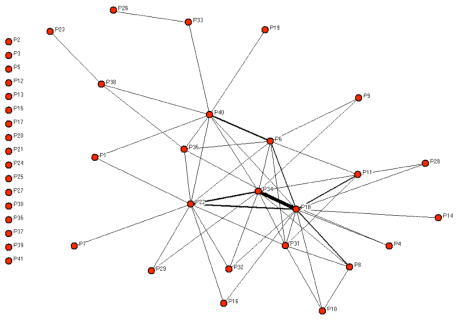

FIGURE 23 – Co-location network dans une guilde de taille moyenne

10.1.2 Conclusion sur l'étude des mouvements

L'aspect communautaires des MMOGs est un des points les plus importants pour les joueurs. De plus, la majorité des études expliquent que la plupart des joueurs jouent et bougent en groupe pendant un temps non négligeable. Ces mouvements de groupe pourraient nous permettre de faire évoluer le module de rapatriement des données, en formant des groupes où seulement certaines entités feraient les différents traitements, la liste des voisins d'une entité comprendrait plusieurs élément stable.

Cette solution engendrerait de refaire le système de mobilité pour simuler des mouvements de groupe. Ces études sur les mouvements de groupe sont réalisées sur certains jeux [38, 29], mais dans d'autres jeux ces mécanisme de groupe peuvent être moins perceptibles [36]. Il faudrait aussi mettre en place un système d'équilibrage des requêtes lors de mouvements de groupe, sinon certains nœuds travailleront toujours pour les autres.

10.1.3 Solution d'utilisation des mouvements de groupes

L'utilisation des mouvements de groupe pourraient nous permettre de réorganiser le rapatriement des données, et ne plus considérer les nœuds indépendamment mais comme formant un groupe. Ces groupes devront avoir une organisation flexible et efficace. Les nœuds se trouvant en avant du groupe, selon la direction, pourraient être les seuls à rapatrier des données, ainsi des messages seraient économisés. Il faudrait ensuite transmettre les données vers les autres membres du groupe, plusieurs méthodes de diffusion peuvent être mises en place. La formation du groupe pourrait aussi permettre de mettre entre parenthèse la recherche de voisins pour certains nœuds.

La figure 24 montre un exemple de ce à quoi pourrait ressembler la solution. Les nœuds, en gris foncé, vont rapatrier les données qui peuvent être intéressantes. Les autres nœuds du groupe n'auront pas rapatrier les données, mais il faudra ensuite diffuser ces données vers le reste du groupe.

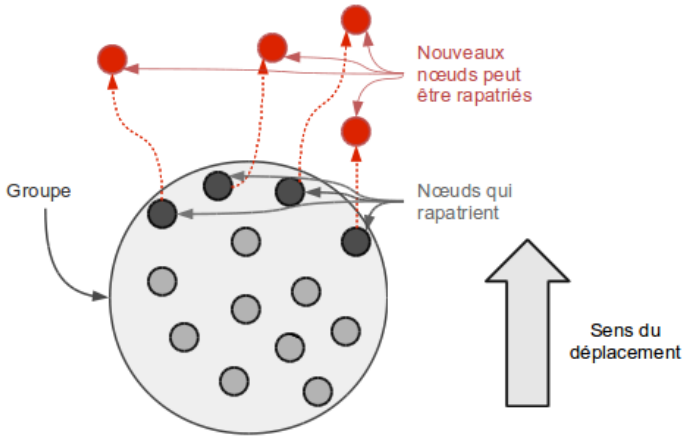

FIGURE 24 – Une piste pour les déplacements en groupe

10.2 Mécanismes de connaissance des routes entre les Hotspots

Dans cette solution, nous voudrions permettre aux avatars de suivre des routes, pour le contournement d'un obstacle par exemple. Il faudrait modifier le modèle pour ajouter des obstacles dans l'environnement. Deux solutions apparaissent rapidement pour créer ses routes. La première solution serait de définir des chemins pour contourner les obstacles, et de conserver ces chemins dans l'environnement. La deuxième solution consisterait à mettre en place un mécanisme d'apprentissage des routes par les avatars. Les avatars pourraient apprendre les routes au fur et à mesure des passages, et laisser des indications pour les avatars suivants. Cette solution permettrait de simuler un comportement réel, et ainsi d'essayer d'améliorer cette situation.

Les modifications sur le modèle ne seront peut être pas très simples à mettre en place. Il faudrait aléatoirement, comme il a été fait pour les Hotspots, définir des zones où les avatars ne pourraient pas passer ou seraient ralentis. Sur la figure 25, nous pouvons voir des trajectoires permettant d'éviter l'obstacle. Il faut définir si ces trajectoires se font par apprentissage ou si elles sont données avec l'initialisation de la carte.

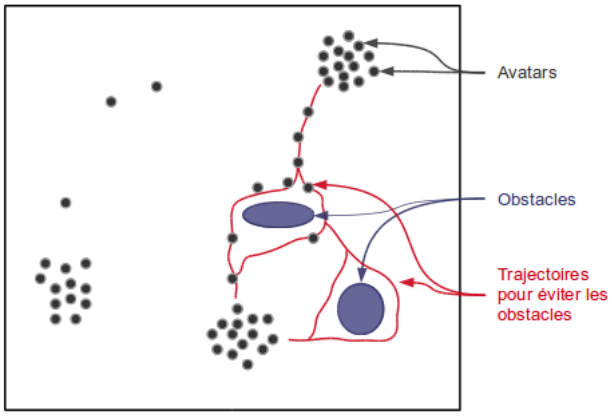

FIGURE 25 – Exemple de trajectoire d'évitement d'un obstacle

10.3 Conclusion sur les autres améliorations possibles

TODO

D'autres idées d'améliorations sont aussi apparu mais sans qu'elles ne soient étudiées précisément. La possibilité de créer des liens entre des nœuds qui sont proches dans le réseau est un des idées apparues. Ainsi deux nœuds qui sont proches pourrait échanger des données rapidement et ces données pourrait servir au nœud ultérieurement ou il pourrait les faire partager à d'autres nœuds sur le réseau virtuel. En s'inspirant du cache, les nœuds pourrait se souvenir, en fonction de la zone géographique où ils se trouvent, d'anciens nœuds déjà rencontrés et ainsi tenter de communiquer avec eux prioritairement.

11 Conclusion

Durant le stage, nous avons étudié les différentes architectures disponibles pour les MMOGs, nous avons observé les limites et les avantages que peuvent avoir les architectures client/serveur et pair à pair. Ensuite grâce à l'étude des traces, nous avons pu dégager différents points permettant l'amélioration de la prise en compte de la mobilité dans les MMOGs. Nous avons vu différents mécanismes permettant d'améliorer la réactivité dans les applications pair à pair, et plus précisément dans les MMOGs, mais la plupart fonctionnait, au mieux, en réaction aux évènements [23]. Blue Banana est un mécanisme d'anticipation des mouvements, ce qui permet de mieux faire évoluer le réseau. Nous avons développé des solutions d'amélioration, en nous appuyant sur ce qui a été fait dans Blue Banana.

Les deux solutions mises en place nous ont permis de travailler sur deux problèmes distincts. La mise en place du cache a permis d'essayer de trouver une amélioration à un endroit qui n'avait pas encore été modifié. Les modifications apportées au rapatriement des données ont améliorées la solution mise en place dans Blue Banana. TODO

Références

[1] Exploiting semantic proximity in peer-to-peer content searching. In *FTDCS '04 : Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, pages 238–243, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Reis Tiago Alves and Licinio Roque. Because players pay : The business model influence on mmog design. In Baba Akira, editor, *Situated Play : Proceedings of the 2007 Digital Games Research Association Conference*, pages 658–663, Tokyo, September 2007. The University of Tokyo.

[3] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook : enabling large-scale, high-speed, peer-to-peer games. *SIGCOMM Comput. Commun. Rev.*, 38(4) :389–400, 2008.

[4] Ashwin Bharambe, Jeffrey Pang, and 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.

[5] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury : supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4) :353–366, 2004.

[6] Vivian Hsueh-hua Chen and Henry Been-Lirn Duh. Understanding social interaction in world of warcraft. In *ACE '07 : Proceedings of the international conference on Advances in computer entertainment technology*, pages 21–24, New York, NY, USA, 2007. ACM.

[7] Vivian Hsueh-hua Chen, Henry Been-Lirn Duh, and Hong Renyi. The changing dynamic of social interaction in world of warcraft : the impacts of game feature change. In *ACE '08 : Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 356–359, New York, NY, USA, 2008. ACM.

[8] Nicolas Ducheneaut and Robert J. Moore. The social side of gaming : a study of interaction patterns in a massively multiplayer online game. In *CSCW '04 : Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 360–369, New York, NY, USA, 2004. ACM.

[9] Nicolas Ducheneaut, Nicholas Yee, Eric Nickell, and Robert J. Moore. "alone together ?" : exploring the social dynamics of massively multiplayer online games. In *CHI '06 : Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 407–416, New York, NY, USA, 2006. ACM.

[10] Chappell D. Griffiths M. D., Davies M. N. O. Breaking the stereotype : The case of online gaming. *Cyber Psychology and Behavior*, pages 81–91, 2003,6.

[11] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. Von : a scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4) :22–31, July 2006.

[12] Shun-Yun Hu and Guan-Ming Liao. Scalable peer-to-peer networked virtual environment. In *NetGames '04 : Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 129–133, New York, NY, USA, 2004. ACM.

[13] Mikael Jakobsson and T.L. Taylor. The sopranos meets everquest - social networking in massively multiplayer online games, 2003.

[14] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. http://peersim.sourceforge.net/.

[15] Zhi-Qiang Jiang, Wei-Xing Zhou, and Qun-Zhao Tan. Online-offline activities and game-playing behaviors of avatars in a massive multiplayer online role-playing game. *EPL (Europhysics Letters)*, 88(4) :48007, 2009.

[16] J. Keller and G. Simon. Solipsis.

[17] Björn Knutsson, Massively Multiplayer Games, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games, 2004.

[18] Chi-Anh La and 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.

[19] Sergey Legtchenko, Sébastien Monnet, and Gaël Thomas. Blue Banana : resilience to avatar mobility in distributed MMOGs. Research Report RR-7149, INRIA, 2009.

[20] Huiguang Liang, Ransi Nilaksha Silva, Wei Tsang Ooi, and Mehul Motani. Avatar mobility in user-created networked virtual worlds : measurements, analysis, and implications. *Multimedia Tools Appl.*, 45(1-3) :163–190, 2009.

[21] Huiguang Liang, Ian Tay, Ming Feng Neo, Wei Tsang Ooi, and Mehul Motani. Avatar mobility in networked virtual environments : Measurements, analysis, and implications. *CoRR*, abs/0807.2328, 2008.

[22] John L. Miller and Jon Crowcroft. Avatar movement in world of warcraft battlegrounds. In *Netgames 2009*. IEEE, November 2009.

[23] Sebastien Monnet, Ramses Morales, Gabriel Antoniu, and Indranil Gupta. Move : Design of an application-malleable overlay. *Reliable Distributed Systems, IEEE Symposium on*, 0 :355–364, 2006.

[24] Christoph Neumann, Nicolas Prigent, Matteo Varvello, and Kyoungwon Suh. Challenges in peer-to-peer gaming. *SIGCOMM Comput. Commun. Rev.*, 37(1) :79–82, 2007.

[25] Yee Nicholas. The norrathian scrolls : A study of everquest. *http ://www.nickyee.com/eqt/report.html*, 2001.

[26] Daniel Pittman and Chris GauthierDickey. A measurement study of virtual populations in massively multiplayer online games. In *NetGames '07 : Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 25–30, New York, NY, USA, 2007. ACM.

[27] Nick Yee. The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments. *Presence : Teleoper. Virtual Environ.*, 15(3) :309–329, 2006.

[28] Chord. http://pdos.csail.mit.edu/chord/.

[29] EverQuest. http://everquest.station.sony.com/.

[30] Gnutella. http://www.gnutella.fr/.

[31] Jxta. https://jxta.dev.java.net/.

[32] KaZaA. http://www.kazaa.com/.

[33] Napster. www.napster.com/.

[34] Pastry. http://research.microsoft.com/en-us/um/people/antr/Pastry//.

[35] Skype. www.skype.com/.

[36] Second Life. http://secondlife.com/.

[37] Star Wars Galaxie. http://www.starwarsgalaxie.com/.

[38] World of Warcraft. http://www.worldofwarcraft.com/.

[39] Xtremweb. http://www.xtremweb.net/.