# Vers un système d'arbitrage décentralisé pour les jeux en ligne

Maxime Véron, Olivier Marin, Sébastien Monnet, Zahia Guessoum

**ComPAS'2013** : RenPar'21 / SympA'15/ CFSE'9
Grenoble, France, du 16 au 18 janvier 2013

# Vers un système d'arbitrage décentralisé pour les jeux en lignes

Maxime Véron, Olivier Marin, Sébastien Monnet, Zahia Guessoum

Laboratoire d'Informatique de Paris 6 / CNRS
Université Pierre et Marie Curie
Paris, France
Email : prénom.nom@lip6.fr

**Résumé**
L'arbitrage des jeux massivement multi-joueurs (MMOGs) repose actuellement sur des architectures centralisées, qui facilitent la détection de la triche mais empêchent les MMOGs de passer à l'échelle. La centralisation limite la taille du monde virtuel ainsi que le nombre de joueurs qui y évoluent. Nous montrons dans cet article qu'il est possible de concevoir un arbitrage pair à pair hautement efficace, même à grande échelle, aussi bien en terme de performance que de prévention de la triche.
Nos simulations montrent que notre solution permet de gérer au-delà de 30000 nœuds tout en détectant plus de 99,987% des tentatives de triche sur des dizaines de millions de requêtes d'arbitrage.

**Mots-clés :** jeux massivement multi-joueurs ; arbitrage décentralisé ; système de réputation

## 1. Introduction

Les jeux massivement multi-joueurs en ligne (MMOGs) visent à rassembler un nombre infini de joueurs dans un même univers virtuel. Pourtant dans tous les MMOGs existants, aucun ne passe à l'échelle correctement. Par tradition, ils reposent sur des architectures client/serveur (C/S) qui imposent une limite sur le nombre de joueurs (avatars) et sur les ressources qui peuvent coexister dans n'importe quel monde virtuel [9]. Une des raisons principales pour cette limitation réside dans la croyance qu'une décentralisation inhibe la protection contre la triche.
La détection de la triche est un élément clé du succès d'un jeu. Un jeu où les tricheurs arrivent à systématiquement prendre le dessus sur les joueurs suivant les règles va rapidement devenir impopulaire auprès de sa communauté. Pour cette raison, il est important pour les fournisseurs de jeux d'intégrer des protections contre les tricheurs dans leurs logiciels.
Les architectures C/S fournissent un bon contrôle des calculs sur le serveur, mais en pratique sont loin d'être à l'épreuve de la triche. Une version récente d'un MMOG populaire, nommé Diablo 3, fut victime d'un piratage du jeu qui a causé son arrêt pendant un jour entier [1]. Étant donné que les approches centralisées ne sont parfaites ni pour la sécurité ni pour le passage à l'échelle, il y a de la place pour des améliorations. La triche et les problèmes qu'elle soulève pour les approches centralisées et centralisées seront évoquées dans la section 2.
Ce papier présente une architecture d'arbitrage des jeux passant à l'échelle qui surveille le contenu du jeu à la fois efficacement et sûrement. Notre approche utilise une surcouche pair à pair (P2P) pour déléguer l'arbitrage du jeu aux nœuds joueurs du réseau. Il se base sur un système de réputation pour juger l'honnêteté des nœuds et ensuite écarter les arbitres corrompus et joueurs malicieux, pendant qu'un processus indépendant du jeu teste les nœuds avec de fausses requêtes de jeu pour accélérer la construction de la réputation.
Notre contribution principale est une approche décentralisée d'arbitrage des jeux vidéo (voir section 3) qui gère facilement plus de 30000 nœuds et permet de détecter plus de 99,9% des tentatives de triche, même dans les conditions les plus difficiles (voir section 4).

## 2. L'attrait des architectures décentralisées pour le monde du jeu vidéo

Le passage à l'échelle est une issue cruciale dans les jeux vidéos en ligne. Comme mentionné plus tôt, la génération actuelle des MMOGs souffre d'une limite de la taille des univers virtuels. Dans le cadre des jeux de rôles, la tendance est à se *grouper* pour améliorer les chances de succès contre les autres joueurs ou le jeu lui-même. Le fait de ne pas pouvoir se réunir dans un même serveur pour faute de place est une cause fréquente de frustration des joueurs. Cette limitation a aussi un impact sur la complexité et sur la taille des univers virtuels. Les joueurs favorisent souvent les jeux offrant le plus large panel d'objets/personnages avec lesquels ils peuvent interagir.

Le manque de scalabilité dans les jeux en ligne actuels est principalement dû à leurs architectures C/S [9]. Bien que le serveur puisse être une entité virtuelle composée de plusieurs nœuds physiques, sa capacité reste la limitation principale en termes de stockages de données, charge réseau et puissance de calcul. Pour compenser cela, les fournisseurs de jeux créent de multiples univers, qu'ils soient des découpes d'un univers global(comme les îles de *Second Life*) ou bien des univers parallèles (comme les royaumes de *World of Warcraft*). Pour un fort surcoût, cela accroît le nombre maximum global de joueurs, mais ne permet toujours pas les interactions entre joueurs d'univers différents. Pour donner quelques idées de l'ampleur de ce problème, des sources non officielles estiment le nombre maximum d'avatars par serveur de *World of Warcraft* aux alentours de 15000 ; le nombre total de joueurs de *World of Warcraft* dépasse les onze millions. De plus, la création de multiple univers ne permet pas d'augmenter la richesse de ces derniers.

La plupart des architectures décentralisées gèrent mieux la charge que les architectures C/S. Une solution décentralisée pourrait permettre d'enlever les limites du nombre de joueurs simultanés et de complexifier le monde virtuel, ou du moins relâcher les limites de manière signifiante.

Dans une architecture C/S, le serveur agit comme un arbitre de confiance tant que le fournisseur du jeu le gère. Prenons l'exemple d'un joueur qui modifie sa copie du logiciel afin d'introduire des commandes illégales de mouvement. Cela offre un avantage non-équitable au tricheur, pouvant alors déplacer instantanément son avatar à des endroits non atteignables où il ne peut être attaqué. Tout ce dont à besoin un serveur centralisé pour gérer ce problème est de vérifier chaque mouvement reçu des joueurs. Une telle solution a un impact fort sur la performance du serveur, tout particulièrement sur sa capacité à passer à l'échelle en fonction du nombre de joueurs simultanés connectés. Il est possible de réduire le cou de calcul de cette solution en sautant une partie des vérifications des commandes reçues. Cependant, cela entraîne que des cas de triches vont potentiellement réussir.

Il y a encore des tentatives de triche qu'une architecture C/S ne peut pas résoudre. Un joueur en train de perdre peut annuler une partie en lançant une attaque DDoS (distributed denial-of-service [11]) sur le serveur, causant son arrêt prématuré. Une architecture décentralisée serait bien plus résistante vis-à-vis d'une telle attaque.

Dans une architecture décentralisée, il n'est pas facile de choisir quel nœud ou ensemble de nœuds peut être suffisamment correct pour juger des combats. Si nous reprenons l'exemple précédemment cité où un joueur envoi des commandes incorrectes. Une approche naïve serait de laisser l'arbitrage au nœud adversaire au tricheur. Malheureusement cela introduirait une faille : tous les nœuds malicieux pourraient alors refuser des commandes correctes lorsqu'elles les désavantages. Dans les faits déléguer l'arbitrage à n'importe quel nœud du réseau est particulièrement risqué : un arbitre malicieux est bien plus dangereux qu'un joueur malicieux.

Prendre une décision d'arbitrage par consensus auprès de nombreux nœuds augmente sa fiabilité. Comme il est à la fois difficile et coûteux pour un joueur de contrôler plus d'un nœud, la confiance que l'on peut avoir en une décision croit avec le nombre de nœuds impliqués. D'un autre côté, une implication d'un trop grand nombre de nœuds pour chaque décision impacterait grandement les performances. Même dans un contexte P2P sans limite concernant le nombre de nœuds, une attente d'un trop grand nombre de nœuds induit de la latence.

Dans ce papier, nous présentons une approche décentralisée qui délègue les décisions d'arbitrage aux nœuds joueurs. Elle choisi les arbitre en fonction de leur fiabilité, fiabilité fournie par le biais d'un système de réputation. Nous avons également des résultats expérimentaux, qui montrent qu'un simple vote parmi un petit nombre d'arbitres augmente la fiabilité des décisions de manière significative sans pour autant entraver la scalabilité du système.

**Modèle système et modèle de fautes**
Notre système adopte une surcouche P2P comme [6] ou [4]. Ces surcouches structures induisent nos hypothèses de base. Il n'y a pas de limite au nombre total de nœuds joueurs. Les états des joueurs sont stockés/répliqués dans les nœuds du réseau et chaque nœud joueur maintiens une liste de ses voisins géographiques. La phase de téléchargement du jeu est considérée complète : toutes les données statiques du jeu sont installées sur chaque nœud. Tous les échanges de données sont asynchrones. Les nœuds ne communiquent que par échange de messages ; ils ne partagent pas de mémoire.
Le code des joueurs et des arbitres peuvent être édités par des tierces personnes : notre approche vise à détecter et adresser ce genre de tentatives. Le système de *matchmaking*, la partie logicielle du jeu qui choisi les adversaire pour le combats sors de la visée scientifique de ce papier. La laisser sur un serveur centralisé n'aurait que peu d'impact sur les performances et nous assumons cette partie comme fiable.
Nous voulons que notre approche distribuée puisse au moins atteindre la même protection que le traditionnel C/S. Pour ce faire notre solution répond à toutes ces potentielles fautes : délai ou modification du protocole de communication, modification des données stockées, attaques DDoS, collusion de deux nœuds. Notre but est d'assurer un taux de triche faible même quand ces fautes apparaissent.
Certaines attitudes malicieuses, comme le *gold farming* et le *phishing*, n'utilisent pas d'actions illégales dans le jeu. Elles ne sont pas encore totalement détectées ni résolues par les architectures C/S actuelles. Nous ne les traiterons pas dans ce papier.
Afin de pouvoir passer à l'échelle, notre architecture décentralisée a besoin de nœuds fiables pour agir comme arbitre. Il est risqué de confier un statut d'arbitre indistinctement, étant donné que cela fourni une méthode facile pour un nœud malhonnête de tourner le jeu a son avantage. C'est pourquoi notre solution repose sur une disponibilité d'un système de réputation distribué pour identifier les nœuds les plus dignes de confiance. Un des systèmes décrits dans [7] pourrait fonctionner, tant qu'il collecte des avis à propos des comportements des nœuds et qu'il calcule des valeurs qui vont décrire ces comportements. Nous avons conduits notre approche avec notre propre système de réputation inclut, mais sa description dépasse le cadre de ce papier.

## 3. Conception d'un arbitrage décentralisé

Le but principal de notre approche est de fournir suffisamment de moyen pour détecter la triche dans les jeux vidéos décentralisés. Dans ce papier, nous nous concentrons dans les combats joueurs contre joueurs. Le mécanisme de détection de la triche est basé sur l'utilisation de pairs particuliers appelés *arbitres*. Pour éviter les collusions, un joueur ne doit pas avoir la possibilité de choisir un arbitre précis. Dans ce but, notre approche se base sur un système de réputation pour séparer les nœuds honnêtes des nœuds malicieux. cela permet de choisir les arbitres parmi les nœuds les plus fiables.
Nous déterminons qu'un noeud agit correctement en fixant une limite T sur la valeur de réputation. La valeur de T est fortement dépendante de la conception du jeu ; elle doit être placée à un compromis entre détection de la triche et efficacité. Un nœud ne va en considérer un autre comme correct que si sa valeur de réputation dépasse T. Après avoir été sélectionnés dans le réseau P2P, les arbitres vont agir comme des serveurs/super-pairs pour les autres nœuds. N'importe quel nœud du réseau peut devenir un arbitre, tant qu'il continue de traiter les transactions correctement. Les ressources qui étaient requises par les serveurs dans le modèle C/S sont maintenant entièrement distribuées dans le réseau pair à pair.

### 3.1. Un protocole générique pour arbitrer les combats en jeu

Chaque joueur possède un identifiant unique qui est associé à son nœud réseau, typiquement l'ordinateur sur lequel le joueur est en train d'exécuter le contenu logiciel. Chaque noeud lance le même moteur de jeu : le code qui défini le monde, ses règles, et ses protocoles de jeu. Un avatar représente un joueur dans le monde virtuel. Les données décrivant le statut dynamique de l'avatar est appelé l'état du joueur ; il est stocké localement sur le noeud du joueur et répliqué sur d'autres pairs pour protéger de sa corruption. Chaque noeud maintient aussi une liste des voisins dans le monde virtuel ainsi qu'une liste de ses voisins dans le réseau P2P.
La figure 1 dépeint notre approche décentralisée d'arbitrage. Une phase d'initialisation permet au système de réputation d'obtenir suffisamment de connaissances à propos des comportements de chaque nœud. Ensuite, chaque nœud peut décider d'initier un combat avec n'importe quel autre nœud et de-

mande à un ou plusieurs arbitres de juger l'issue du combat. Un combat entre deux adversaires engendre plusieurs escarmouches jusqu'à ce qu'une victoire émerge. Pendant un combat, chaque arbitre juge les commandes des joueurs envoyées, et ensuite décide de la suite du combat en fonction des données du jeu et de la validité des commandes reçues.

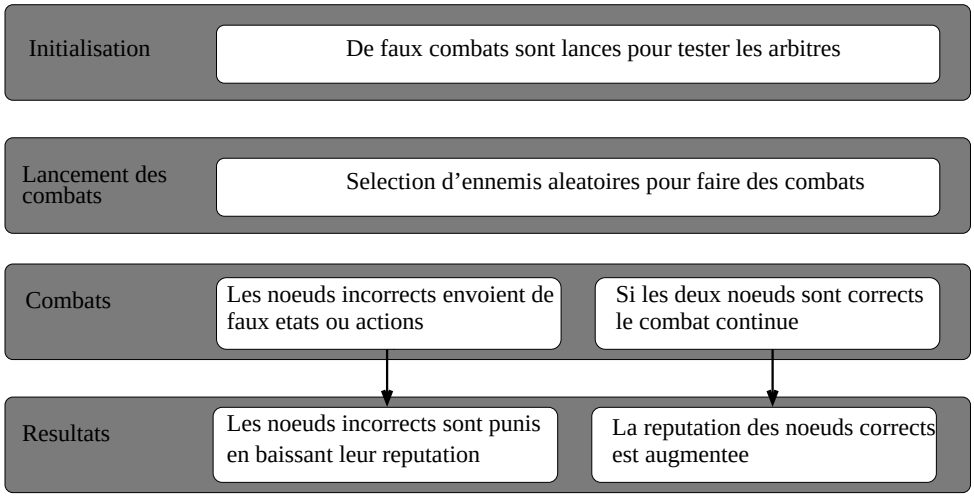


FIG. 1 – Résumé de notre protocole décentralisé d'arbitrage

Un problème classique des applications utilisant un système de réputation est le lancement. Comme il n'y a eu aucune transaction pour le moment, il est impossible de déterminer les nœuds fiables. Le même problème apparaît pour juger la fiabilité des nœuds rejoignant l'application.
Nous résolvons ce problème en cachant de fausses requêtes dans le flux des demandes normales d'arbitrage. Sur la réception d'une requête d'arbitrage, un nœud de peut pas déterminer si elle est fausse ou légitime ; il va donc devoir traiter chaque requête comme légitime. Cela accélère le rassemblement d'information de réputation sur les nœuds, et aussi décourage de tricher : la falsification étant déjà risquée, pour vouloir la tenter pour n'obtenir aucun bénéfice ?
Au lancement du jeu, toutes les requetes sont fausses jusqu'à ce que le système de réputation puisse délivrer des valeur de réputation fiables. Quand le système de réputation est pret, les joueurs peuvent lancer les "vrais" combats sous la supervision d'un ou plusieurs arbitres.
Quand un combat a été initialisé, les nœuds s'opposant peuvent créer des événements qu'ils enverront aux arbitres. Ces événements sont de deux types : les *actions* décrivant des actions dans le jeu et les *états* contenant les états des joueurs.
Dans notre modèle, un tricheur est un joueur qui essaye de soit (a) faire un événement qui ne peut pas correspondre à son état selon le moteur du jeu ou (b) retarder l'émission d'un événement.
A la réception d'un événement d'un joueur, un arbitre vérifie si l'événement est valide avant de le transférer à son adversaire. Les événements sont alors échangés entre les deux adversaires uniquement sous la supervision d'un ou plusieurs arbitres, jusqu'à qu'un événement autorisant la fin d'un combat et déclare un vainqueur. Si un tricheur tente de se déclarer vainqueur de manière illégale, les arbitres et le joueur ennemi détecteront sa tentative.
Afin d'optimiser la détection de la triche, notre système choisi les arbitres parmi les plus fiables qui appartiennent au voisinage du joueur en tant que nœud. Rappelons que des nœuds fiables sont ceux dont la valeur de réputation a dépassé un seuil fixé. Les nœuds impliqués dans un combat sont exclus de la sélection d'arbitre, étant donné que l'auto arbitrage est prohibé pour des raisons évidentes.
Comme les valeurs de réputation sont dynamiques, il est possible pour un arbitre de perdre sa fiabilité dans le milieu d'un combat. Dans ce cas, le combat est annulé : toutes les requêtes d'arbitrages seront marquées comme fausses et ne seront utilisées que comme informations pour notre système de réputa-

tion.

### 3.2. Détection de la triche

Chaque combat constitue un événement dans lequel chaque nœud peut essayer de corrompre le combat a son avantage. Notre système d'arbitrage vérifie les événements comme décrit par la suite.
Chaque joueur peut créer des événement basés sur le moteur du jeu. De façon à changer l'état de son avatar, un joueur doit demander une demande d'arbitrage contenant la description de l'événement.
A la réception de deux descriptions associées au même événement –un pour chaque combattant–, un arbitre va utiliser le moteur du jeu pour vérifier :
– l'état initial et sa consistance avec les répliques stockées dans le réseau P2P ;
– chaque action, pour évaluer si elles sont cohérentes avec l'état du joueur ;
– le nouvel état de chaque combattant, pour assurer que toutes les actions ont été appliquées ;
– des annonces de victoires, pour protéger des tentatives de triche les plus simples.
Un combat déclenche une boucle de vérifications chez l'arbitre (voir figure 2) ; une vérification par combat jusqu'à ce qu'une victoire apparaisse, ou que le combat soit annulé.

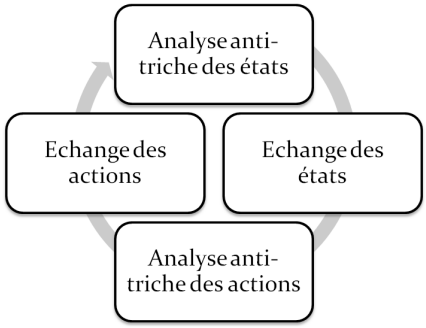


FIG. 2 – Automate des arbitres utilisé pour un combat

Les arbitres d'un même combat envoient leurs décisions aux joueurs directement ; ils ne communiquent pas pour atteindre un consensus. Cela n'introduit pas de brèche dans notre sécurité, vu que nous pouvons détecter les nœuds qui tentent de tricher sur les décisions. Notre architecture détecte systématiquement les tentatives de tricherie, qu'elles viennent d'un joueur ou d'un arbitre. Si l'un des arbitres envoi une mauvaise décision aux joueurs, les joueurs vont détecter son inconsistance. Si un joueur incorrect décide de prendre la mauvaise décision, les arbitres corrects vont détecter l'état de ce joueur incorrect à la prochaine itération. Enfin, si une mauvaise décision donne un nœud incorrect comme vainqueur, et si le nœud incorrect oubli d'envoyer un message annonçant sa victoire, son adversaire et les arbitres vont le considérer éventuellement comme malicieux.

### 3.3. Multiplier les arbitres pour améliorer la détection de la triche

Un arbitre n'est pas suffisant pour assurer que l'approche est suffisamment résistante à la triche. Un nœud malicieux peut temporairement envoyer des réponses correctes, et ensuite émettre des décisions corrompues si il arrive à acquérir un statut d'arbitre fiable. Une association de plusieurs arbitres dans un seul combat contre ce genre de comportements.
Le premier avantage de cette approche est qu'elle améliore la détection des arbitres malicieux. La probabilité de prendre N arbitres malicieux au même moment est considérablement plus basse que celle d'en prendre un seul.
Le second avantage est que cela nous aide a empêcher les collusions entre un joueur et un arbitre. La collusion est une stratégie coûteuse, et le coût croit exponentiellement avec le nombre de nœuds impliqués. cela est encore plus vrai dans notre approche considérant que :
– un joueur ne peut pas influencer la sélection des arbitres,

– les nœuds voulant faire une collusion doivent en premier temps obtenir de bonnes réputations avant de commencer à tricher,
– un nœud arbitre ne peut jamais savoir quand il traite une requête d'un combat ou d'un faux combat.

Nous avons analysé l'impact du nombre d'arbitre sur l'efficacité de la détection de la triche et sur le surcoût. Les résultats de cette analyse, parmi d'autres résultats, sont présentés en section 4.

## 4. Évaluation des performances

Dans ce papier nous visons à offrir un système d'arbitrage alternatif à l'architecture du C/S dans le contexte des MMOGs. Le surcoût généré par notre solution doit être gardé bas afin de permettre la scalabilité et pour offrir une expérience d'utilisateur qui est indiscernable de celle des jeux en C/S.

Afin de permettre la comparaison de notre approche avec le client/serveur, nous définissons un serveur comme un ordinateur capable de gérer l'ensemble des nœuds du réseau. De cette façon nous pouvons additionner l'ensemble des charges des nœuds obtenues dans notre simulation, comme si nous avions à les connecter à un seul serveur.

### 4.1. Configuration et paramètres de la simulation

Nos simulations sont basées sur le moteur de simulation à événement discrets de peersim [10]. Pour tester nos critères de performance nous avons lancé plus de 40 simulations avec un temps simulé de 24 heures chacune.

Nos métriques sont :
– le pourcentage de triche non détecté,
– la consommation de bande passante,
– la quantité de latence ajoutée.

**La taille de notre réseau** est mise à la plus grande valeur que notre simulateur pouvait gérer sur nos machines de tests : 30000 nœuds. Cette valeur est supérieure à l'habituelle limite du modèle client/serveur qui empêche les compagnies de jeux de créer des univers avec plus de 15000 joueurs dans une qualité de service acceptable. Nous avons également lancé des courtes simulations jusqu'à 60000 nœuds qui ont montré le même comportement que celles à 30000. Notre approche semble se comporter de manière presque indépendante du nombre de nœuds dans le réseau. Nous avons toutefois lancé ces simulations à 30000 à cause du temps et de l'espace mémoire nécessaire à l'accomplissement de ces dernières.

**La latence** est choisie de manière aléatoire dans un nombre aléatoire entre 10 et 40ms pour chaque message. Cet ensemble de valeur est de nos jours plutôt commun quand nous nous connectons à un serveur. Le surcoût en termes de latence de notre solution sera alors calculé en notant le délai que nos messages impliquent sur les messages de jeu envoyés à un serveur.

**La consommation de la bande passante** est obtenue par chaque message envoyé dans notre simulation. Il est facile de calculer combien de messages sont envoyés, et combien de messages avec un contenu particulier sont envoyés. Nous pouvons alors mettre en relation le contenu des messages avec des réels usages mémoire comme des entiers, ou des flottants pour calculer des tailles de messages réalistes.

**La charge CPU :** nous donnons un *"point de charge CPU"* à chaque création d'action/état (utilisation CPU liée au jeu) et à chaque test d'action/état (utilisation CPU liée à notre approche). Notez que cette méthode surestime notre surcoût CPU. Dans une réelle implémentation le cout des tests est bien inférieur à celui d'autres opérations dépendantes en CPU (comme les calculs d'ombres).

**Taux de triche toléré :** .Comme il est dur de trouver des taux de triche qui passe non détectée dans les jeux, nous avons décidé que ce nombre devait être le plus bas possible, tout en conservant la capacité à passer à l'échelle et le fiable coût réseau.

### 4.2. Ratio de nœuds incorrects

Il est important de comprendre qu'une personne utilisant le service peut vouloir tricher en tant que joueur, mais ne veut pas forcément déranger les autres joueurs. Il est aussi possible qu'à l'opposé cette personne ne veuille qu'endommager le système tout en ne jouant pas du tout. C'est pour cela que nos deux ratios ne sont pas reliés et impliquent que la partie joueur ou arbitre d'un nœud puissent être incorrectes de manière indépendante.

Même si nous trouvons dans la littérature que 5% est une valeur forte quand nous considérons les nœuds

incorrects dans les réseaux pairs à pairs, nous avons fixé ce nombre de joueurs qui vont potentiellement tricher à 30%. Notre nombre d'arbitre incorrects est lui fixé à 10%.
Nous avons mis seulement 10% d'arbitres incorrects dans notre réseau afin de laisser plus de possibilités de combats. S'il n'y a pas assez de bons arbitres dans le réseau nous ne pouvons pas laisser assez de combat se dérouler, et la triche ne va pas apparaître vu que les combats ne peuvent être faits. Nous avons donc besoin d'une valeur plus basse d'arbitre tricheurs.
Si nous bougeons le nombre de nœuds incorrect entre 0% et 45%, cela ne change pas le nombre de triche non détectée, *qui resta en dessous de* $0,013\%$ dans nos 20 simulations. Nous avons donc gardé 10% d'arbitres incorrects, qui reste une valeur forte. Nos simulations montrent que le nombre de combats possibles décroit quand le nombre d'arbitre incorrect croit, ce qui signifie que les arbitres incorrects sont bien détectés et évités.
Aussi longtemps que le nombre de noeuds corrects reste suffisamment élevé pour faire des combats ensemble, la solution marche. Mais, avec plus de 45% d'arbitres incorrect et 30% de joueurs incorrects, le système trouve difficilement un ensemble d'arbitres et de deux joueurs corrects pour commencer et finir un combat.

### 4.3. Résultats

**Surcoût CPU :** comme nous avons expliqué notre surcoût CPU est relié au nombre d'arbitre que nous utilisons. Il est apparu que, même avec notre optimisation des tests, l'approche utilisant 5 arbitres présente un surcoût trop important pour être utilisée. Si les jeux de nos jours utilisent en moyenne deux cœurs CPU, nous pouvons considérer qu'un processeur quad-core peut supporter un surcoût maximum de 100%.
Nous calculons les valeurs de surcoût CPU comparées avec la valeur de surcoût CPU qui serait nécessaire pour ne laisser que 1% de triche non détectée. En faisant ainsi, nous obtenons que la solution la plus efficace en terme d'utilisation CPU est l'approche avec 3 arbitres. Elle offre un rapport de $0,0128\%$ d'utilisation CPU pour 1% de triche. Les deux autres solutions sont quand à elles moins performantes : $32,6625\%$ pour la mono-arbitre et $2,10\%$ pour la cinq-arbitres.
Cela montre à quel point notre approche "plusieurs arbitres" est efficace en terme d'utilisation CPU comparée à celle mono-arbitre.
**Surcoût latence :** En premier, nous avons besoin de voir si notre protocole n'ajoute pas trop de latence sur le jeu lui-même. Nous avons donc relevé les surcoût latence de chaque message. Nos résultats montrent que nous ajoutons une latence comparable à celle d'un échange de message entre les nœuds de notre réseau : 40ms. Cela est causé par la communication en deux phases entre les pairs et les arbitres. Un message n'a plus besoin d'aller au serveur, mais à besoin de passer à travers un pair, dont la latence est plus grande que celle d'un serveur centralisé. Même si nous considérons un réseau lent de 100ms, nous arriverons à 200ms, ce qui reste acceptable.
**Surcoût bande passante :** Nous avons mesuré que la bande passante de notre solution n'utilise en moyenne que 4ko/s quand elle est stabilisée, comparé aux 40Mo/s qui auraient été nécessaires au serveur (vu dans les figures 4.
Le pic de débit en architecture C/S monte à 67Mo/s ce qui est, en réalité, impossible à gérer pour un seul ordinateur. Dans un jeu actuel, le serveur serait tombé et les joueurs auraient été déconnectés. Il est commun de voir ce comportement dans les jeux actuels.
Notre approche est basée sur un système de réputation. Son classement permet de trouer les nœuds de confiance. Le cout d'une telle solution reste faible : nous avons analysé combien de bande passante notre mécanisme de réputation consommait sur les 4ko/s. Il est apparu que, comparé à la taille des messages de jeu, nous avons utilisé seulement $1,5\%$ de toutes les données envoyées. (comme montré en figure6).

## 5. Travaux connexes

Il y a eu de nombreuses recherches significatives sur les surcouches P2P pour les jeux vidéos. De nombreuses approches,comme Colyseus [2], BlueBanana [8], et Solipsis [4], visent à adapter les positions des nœuds dans le réseau pour les besoins des applications. Cela les rends compatibles avec des MMOGs. Colyseus est suffisamment efficace pour supporter des jeux FPS. Les MMORPGs ont des taux de rafraîchissement plus bas et ont de bien plus petites bande passante nécessaires que les jeux FPS [3].

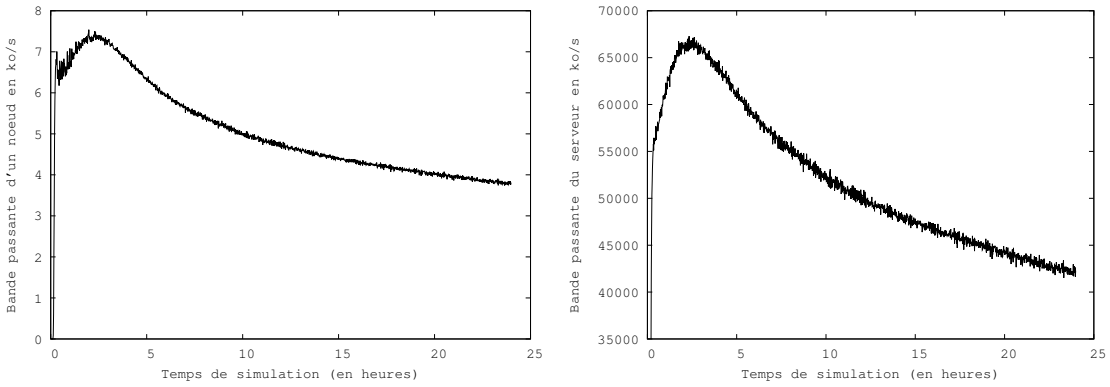


FIG. 3 – Comparaison de la bande passante utilisée entre un nœud du réseau et le serveur centralisé

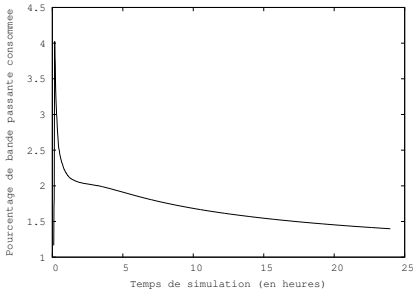


FIG. 4 – Bande passante consommée par le système de réputation

Bluebanana et Solipsis adaptent leurs réseau à la mobilité des avatars. Malheureusement aucune de ces approches n'adressent le problème de la triche.

Pour gérer cette question ouverte, [6], [12], et [5] ont proposé des solutions distribuées de détection de la triche. Dans [6] et [12] les auteurs utilisent des entités de confiance pour gérer la sécurité, et dans [5] les super-pairs sont utilisés comme proxy pour la sécurité. Ils rassemblent toutes les informations nécessaires pour créer une approche décentralisée et fiable des MMOGs, et introduisent le concept d'arbitrage décentralisé dans le réseau P2P en implantant des entités de confiance. Comme chaque action dans un jeu peut être discrétisé en de petits événements, l'analyse et l'arbitrage des actions reçues d'un jeu sont déléguées à ces entités.

Ces solutions introduisent les bases pour le jeu pair à pair décentralisé. Cependant ils ne tolèrent pas un grand nombre de nœuds incorrects. Notez que notre approche est similaire à RACS [12]. Cependant, RACS limite l'arbitrage à un unique arbitre. Il ne gère ni la coopération parmi de multiple abrites, ni la sélection de ces entités de confiance.

Un autre type d'approche est de mélanger les bénéfices d'un équilibrage de charge P2P avec un serveur centralisé, comme dans [5]. Une telle approche hybride a également besoin d'identifier des nœuds de confiance dans le réseau pour déléguer les taches d'arbitrage. Bien que cela aide le serveur quand ce dernier a besoin de gérer un grand nombre d'opération dépendantes du CPU, le nombre maximum de joueurs connectés simultanément reste limité, en dessous des besoins que pourrait avoir un MMOG.

Notre solution peut utiliser n'importe quel système de réputation décentralisé qui identifie les arbitres potentiels. Elle ne repose ni sur des contrôleurs de régions du réseau, ni sur des nœuds choisi de la surcouche P2P. De plus notre approche permet de déléguer chaque arbitrage à plusieurs arbitres. Comme le montre nos simulations, cela fourni une manière efficace de résister à la triche sans perdre la scalabilité.

## 6. Conclusion et travaux futurs

Les architectures pair à pair entièrement décentralisées pour les MMOGs sont en émergence. Cependant, elles font face à un problème majeur : il est difficile de contrôler le respect des règles d'un jeu dans une architecture entièrement décentralisée.
Nous proposons une solution basée sur un système de réputation. L'évaluation de notre approche montre qu'il est possible de fournir un système anti-triche dans un grand MMOG pair à pair sans dégrader le passage à l'échelle. Nous avons obtenu seulement $0,0128\%$ de triche non détectée tout en ayant une moyenne de jusqu'à 40% de nœuds incorrects. De plus, nous avons calculé qu'un serveur aurait eu au moins besoin de 40Mo/s de bande passante pour gérer la même quantité de nœud que nous avons utilisé, comparés au faible 7ko/s maximum par nœud du réseau P2P.
Nous planifions désormais d'étendre notre approche à d'autres types de triche, par exemple quand les joueurs essayent tricher contre leur environnement. Nous sommes aussi en train d'améliorer notre solution actuelle pour limiter le nombre de tests quand le système est stable, améliorant l'utilisation CPU sans dégrader la précision de détection.
Nous planifions également de concevoir une approche basée sur un système multi-agents. Les système basés sur des agents fournissent des comportement variés et de riches interactions entres les arbitres. Les arbitres incorrects pourraient être supprimés et remplacés quand ils sont détectés, excluant les nœud incorrects du réseau, ou de nouveaux comportements pourraient émerger comme détecter le vol d'un compte de joueur ou beaucoup d'autres mauvaises attitudes.

## Bibliographie

1. Blizzard admits diablo 3 item duping is why asia's server was shutdown, June 2012.
2. Bharambe (A.), Pang (J.) et Seshan (S.). – Colyseus : a distributed architecture for online multiplayer games. *In : Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*. pp. 12–12. – Berkeley, CA, USA, 2006.
3. Chen (K.-T.), Huang (P.), Huang (C.-Y.) et Lei (C.-L.). – Game traffic analysis : an mmorpg perspective. *In : Proceedings of the international workshop on Network and operating systems support for digital audio and video*. pp. 19–24. – New York, NY, USA, 2005.
4. Frey (D.), Royan (J.), Piegay (R.), Kermarrec (A.-M.), Anceaume (E.) et Le Fessant (F.). – Solipsis : A Decentralized Architecture for Virtual Environments. *In : 1st International Workshop on Massively Multiuser Virtual Environments*. – Reno, NV, États-Unis, 2008.
5. Goodman (J.) et Verbrugge (C.). – A peer auditing scheme for cheat elimination in mmogs. *In : Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. pp. 9–14. – New York, NY, USA, 2008.
6. Hampel (T.), Bopp (T.) et Hinn (R.). – A peer-to-peer architecture for massive multiplayer online games. *In : Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. – New York, NY, USA, 2006.
7. Jøsang (A.), Ismail (R.) et Boyd (C.). – A survey of trust and reputation systems for online service provision. *Decision Support Systems*, vol. 43, n2, mars 2007, pp. 618–644.
8. Legtchenko (S.), Monnet (S.) et Thomas (G.). – *Blue Banana : resilience to avatar mobility in distributed MMOGs*. – Rapport de recherche nRR-7149, INRIA, 2009.
9. Miller (J.) et Crowcroft (J.). – The near-term feasibility of P2P MMOG's. *In : Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*, pp. 1 –6.
10. Montresor (A.) et Jelasity (M.). – PeerSim : A scalable P2P simulator. *In : Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pp. 99–100. – Seattle, WA, sep 2009.
11. Specht (S. M.) et Lee (R. B.). – Distributed denial of service : taxonomies of attacks, tools and countermeasures. *In : Proc. of the Int. Workshop on Security in Parallel and Distributed Systems*, pp. 543–550.
12. Webb (S. D.), Soh (S.) et Lau (W.). – RACS : A Referee Anti-Cheat Scheme for P2P Gaming. *In : Proc. of the Int. Conference on Network and Operating System Support for Digital Audio and Video (NOSSDAV'07)*.