Towards a scalable refereeing system for online gaming

Maxime Véron, Olivier Marin, Sébastien Monnet, Zahia Guessoum
Laboratoire d'Informatique de Paris 6 / CNRS
Université Pierre et Marie Curie
Paris, France
Email: firstname.lastname@lip6.fr

***Abstract*—Refereeing for Massively Multiplayer Online Games (MMOGs) currently relies on centralized architectures, which facilitates cheat prevention but also prohibits MMOGs from scaling properly. Centralization limits the size of the virtual world as well as the number of players that evolve in it. The present paper shows that it is possible to design a peer to peer refereeing system that remains highly efficient, even on a large scale, both in terms of performance and in terms of cheat prevention. Simulations show that such a system scales easily to more than 30,000 nodes while leaving less than 0.013% occurrences of cheating undetected on a mean total of 24,819,649 refereeing queries.**

***Keywords*-massively multiplayer online gaming; distributed refereeing; reputation system; scalability**

I. Introduction

Massively Multi-player Online Games (MMOGs) aim at gathering an infinite number of players within the same virtual universe. Yet among all of the existing MMOGs, none scales well. By tradition, they rely on centralized client/server (C/S) architectures which impose a limit on the maximum number of players (avatars) and resources that can coexist in any given virtual world [1]. One of the main reasons for such a limitation is the common belief that full decentralization inhibits the prevention of cheating.

Cheat prevention is a key element for the success of a game. A game where cheaters can systematically outplay opponents who follow the rules will quickly become unpopular among the community of players. For this reason, it is important for online game providers to integrate protection against cheaters into their software.

C/S architectures provide good control over the computation on the server side, but in practice they are far from cheat-proof. A recent version of a popular MMOG, namely *Diablo 3*, fell victim to an in-game hack that caused the game's shutdown for an entire day [2]. Not so long ago, someone found a security breach in *World of Warcraft* [3] and proceeded to disrupt the game by executing admin commands. Given that centralized approaches are neither flawless security-wise nor really scalable, obviously there is room for improvement. Cheating and the issues it raises for both centralized and decentralized approaches are discussed further in Section II.

This paper presents a scalable game refereeing architecture that monitors the game both efficiently and securely. Our contribution uses a peer to peer (P2P) overlay to delegate game refereeing to the player nodes. It relies on a reputation system to assess node honesty and then discards corrupt referees and malicious players, whilst independent processes in the game challenge nodes with fake game requests so as to accelerate the reputation-building process.

This paper is organized as follows. Section II makes a case for decentralized architectures as building blocks for game software, and then details our system model and failure model. Section V depicts our main contribution: a decentralized approach for game refereeing that scales easily above 30,000 nodes and allows detecting more than 99.9% of all cheating attempts, even in extremely adverse situations. Section VI outlines the characteristics of the reputation system our solution requires in order to achieve such performance, and details the simple reputation system we designed following this outline. Finally section VII gives a preliminary performance evaluation obtained by simulating our solution.

II. The appeal of decentralized architectures for online games

Scalability is a crucial issue for online games. As mentioned earlier, the current generation of MMOGs suffers from a limit on the size of the virtual universe. In the case of online role-playing games, the tendency is to team up in *parties* to improve the odds against other players and the game itself. Being unable to gather a party of avatars in the same virtual world because there are not enough slots left on the server is a fairly frequent cause for frustration among players. The limitation on the scale also has consequences on the size and complexity of a virtual world. Players usually favor games that offer the largest scope of items/characters they can interact with.

The lack of scalability in the current trend of online games is mainly due to their C/S architecture [1]. Although the server may be a virtual entity composed of multiple physical nodes, its capacity remains the main limitation in terms of data storage, network load, and computation load. To compensate for this, game providers create multiple universes, either partitions of a global universe (like *Second Life* islands) or parallel universes (like *World of Warcraft* realms). Each universe gets hosted on its own separate cluster of servers. At great cost, this does increase the overall number of concurrent players, but it still does not allow in-game interactions between two players evolving in separate universes. To give some idea of

the magnitude of this problem, unofficial sources estimate the maximum number of avatars per *World of Warcraft* server to be around 15,000 at the launch of the game, 60000 nowadays with the evolution of the computing power[4]; the total number of *World of Warcraft* players reaches beyond 11,000,000. In this number of possible players on a server, only a few part will be logged in at the same time, creating an average of 1,500 concurrent players on a server dimensioned for 60,000 accounts[5]. Since the number of virtual resources with which players can interact is also limited by the capacity of the servers, creating multiple universes does not allow to enhance the richness and complexity of the virtual world either.

Decentralized architectures usually scale far better than C/S architectures. They remove the limitations on the number of concurrent players and on the complexity of the virtual world, or at least relax these limitations significantly.

Cloud deployment is a step towards decentralization and a growing trend within the digital games industry {*ICI PLEIN DE REFS*}. Cloud gaming extends the traditional C/S cluster-based approach and allows dynamic provisioning of server resources. Additionally, it can help alleviate the computing load on the client side by running all computations on cloud servers and streaming the resulting video feed. The work presented in this paper focuses on a fully decentralized approach over P2P overlays, but it is adaptable to cloud gaming. Section III discusses the portability of our solution to clouds.

One solid argument against full decentralization is that cheat detection is very hard to enforce in a distributed context, and it may thus be less effective.

In a C/S architecture, the server side acts as a trustworthy referee as long as the game provider operates it. Let us take an example of a player who tampers with his software copy to introduce illegal movement commands. This provides an unfair advantage to the cheater as his avatar then acquires the ability to instantly reach places where it cannot be attacked. All it takes for a centralized server to set things right is to check every movement command it receives from player nodes. Such a solution will have a strong impact on the performance of the server, and especially on its ability to scale with respect to the number of concurrent avatars. It is possible to reduce the computational cost of this solution by skipping checks of the received commands. However, it entails that some cheating attempts will succeed eventually.

There are also cheating attempts that a C/S architecture is ill-fitted to address. A player about to lose can cancel a battle by launching a DDOS attack (distributed denial-of-service [6]) on the server and causing its premature shutdown. A decentralized architecture would be far more resilient against such an attack.

In a decentralized architecture, it is not easy to select which node or set of nodes can be trusted enough to handle the refereeing. For instance, let us roll back to the example where a player node sends illegal movement commands. A naïve approach could be to delegate the refereeing to the node of the cheater's opponent. Unfortunately this would introduce a breach: any malicious player node could then discard legitimate commands to their own advantage. As a matter of fact, delegating to any third party node is particularly risky: a malicious referee is even more dangerous than a malicious player.

Reaching a referee decision by consensus among several nodes boosts its reliability. Since it is both hard and costly for any player to control more than one node, the trustworthiness of a decision grows with the number of nodes involved. On the other hand, involving too many nodes in every single decision impacts heavily on performance. Even in a P2P context with no limit on the total number of nodes, waiting for several nodes to reach a decision introduces latency.

In this paper, we present a fully decentralized approach which delegates referee decisions to player nodes. It picks referees on the basis of their reliability, the latter being assessed by means of a reputation system. We also detail our experimental results, which show that a simple vote among a small number of referees increases the trustworthiness of decisions significantly without impeding the scalability of the system.

III. Cloud gaming architectures

Cloud gaming architectures may be handled in two major different ways :

- streaming games to the user device
- using the cloud as resources

Streaming games, what is called the *"cloud gaming"*, with companies such as [7] allows users to play a big collection of games in exchange of a monthly fee. *Nvidia* tried to join the cloud gaming services industry by offering his own Nvidia computing grid [8] and also proposed a similar android application to its latest handled device. The Nvidia shield can receive content directly from the home computer and later on will be able to receive from the Nvidia grid[9]. There is a strong tendency to move towards the cloud gaming as a streaming service[10].

All those services of cloud gaming relies on a highly capable gaming computer running the game and streaming its content to the user from the company datacenter. This allows tablets, smartphones and small computing power devices to handle higher quality games without efforts. As long as the user stays connected with sufficient bandwidth, he will be able to play any game he wants.

As for using the cloud as available resources, it is hard to know exactly where gaming companies store their servers. Most of the time though, when a problem or a new feature appear in a gaming platform, the term of data center appears [11]. We can then guess that nowdays, a vast majority of games still rent some rooms to store servers per region to have a total control on the machines and get the best latency possible for the users.

Cloud resources are then used to lower the stress and bandwidth requirements made by the game itself. Recently, Dota 2 developers admitted that pushing an update for their games generates then two percent of the overall network traffic of the world[12]. It is clear that a company cannot output by itself such quantity of information to each user and therefor push this update to cloud service to unload the stress. Microsoft new console XboxOne will ensure that every

XboxOne released will have the equivalent of two times its computing power in the cloud to allow complex computation in games[13] , showing the growing will to use the cloud as a matter of gaming enhancement.

It is then truly easy to integrate a cloud component to our solution. By injecting the cloud service informations as possible referees for player nodes in the game code, our solution will automatically discover the cloud service.

Using cloud services as always on, always available referees will give our system a trusted entity we could always refer to in case of doubts. It will also allow our system to start immediately as there will be no need in the early beginning to firstly identify possible referees. When the cloud service will be filled with players making fights, our reputation system will be used to then assess the nodes reliableness and use them as available resources that will help us scale without limits. We could later on decide to shutdown the cloud service and let the peer to peer system handle itself.

Even though being then affordable for some users who can't buy a gaming computer, streaming services add latency and the bandwidth limitation of the user makes it challenging. Exploration games are easily compatible with this kind of gameplay, but multiplayer games like we aim at raise also a confidentiality issue : the user need to send his online username and password to a cloud service that might run other software concurrently. Even though isolation is supportingly ensured, it may frighten the users. Our solution will not rely on streaming the game content to users as we are aiming at precise response time and highly reactive multiplayer gaming.

Cloud gaming will produce a lot of bandwidth exchanges. A example of how important can be the bandwidth costs can be found at [14]. In a context of MMOGs where the bandwidth is an extremely critical point to broadcast views to players, a price to pay per bandwidth used could simply make the company loose at lot of money if they don't optimize perfectly their protocol.

Cloud services in their majority are supposed to be reliable and always on. But a quick check on [15] reveals that nearly half of them aren't available a hundred percent of the time, making them potential unreliable nodes of the network, just like peers in a P2P network. Therefor, unless the company have the means to turn towards an expensive cloud service provider, still needs to add some verification and consensus protocols to the game. Integrating complex computer science elements in one of the AAA type of games will only raise its development cost and potentially insert inconsistency in some players view if this new mechanism is not fully under control when the game is released.

That's why we think that our solution could benefit of the cloud platform as a possible add-on that will enhance its performance, but not as the most appropriate solution when it comes to massive gaming architectures.

Putting all the game content to the cloud will generate too much bandwidth and need complex algorithms relative to the reliability and response time regarding the cloud localization and offers.

Unloading performance to the cloud will still require to possess a centralized server handling the nodes and will not resolve the scalability issues of the central point of failure when the server receives a huge amount of concurrent players.

IV. System model and failure model

System model

We want our system to work on top of a P2P overlay designed for gaming, such as [16] and [17]. This assumption on the infrastructure induces our base hypotheses.

A GARDER POUR LA SECTION PERFS - Our simulation doesn't comprise a *world* component, which could be latter added with an existing P2P overlay for gaming.

Our model focuses on fights between two nodes. We argue that it is scalable in that it paves the way to handling very large numbers of one-on-one fights simultaneously. Although this model could be extended to small sets of nodes fighting co-operatively (arena-style), it is not suited for large cooperative battles which require zone control. Current C/S architectures do not scale well either as the number of opponents involved in the same battle increases. Our approach opts for strong protocol protection to show that a distributed approach can compete with a costly C/S solution.

Every game player in our model possesses a unique game identifier and associates it with a network node, typically the computer on which the player is running the game software. Every node runs the same game engine: the code that defines the world, rules, and protocols of the game. An avatar represents a player within the game world. Data describing the dynamic status of the avatar is called the player state; it is stored locally on the player's node and replicated on other peers to prevent its corruption. Every node also maintains both a list of the player's immediate neighbors within the virtual world of the game, and a list of neighbor nodes within the P2P overlay. In the rest of this paper, we use the term *neighborhood* to refer to both lists.

There is no limit on the total number of player nodes in the system. Player states are stored/replicated on network nodes and every player node maintains a list of geographical neighbors. The game downloading phase is considered complete: all static game content is installed on every node. All data exchanges are asynchronous. Nodes communicate by messages only; they do not share any memory.

We assume that all shared data pertaining to the game and the virtual universe can be accessed in a timely manner. Systems such as [16], [18], [17] already address this issue.

Both the player and the referee codes and protocols can be edited by third parties: our approach aims at detecting and addressing such tampering. The matchmaking system, the part of the game software that selects opponents for a battle, falls out of the scope of this paper. Leaving it on a centralized server would have little or no effect on the overall performance, and we assume it to be trustworthy.

Failure model

Our distributed approach aims at offering a secure gaming protocol, so as to detect every corrupted game content that passes through the network and impacts other players. Our

goal is to ensure the lowest cheat rate even when possible failures occur.

The extended classification made by Jeff Yan and Brian Randell[19] gives a complete overview of the types of cheat attempts that can happen in a game. Among these types, current C/S architectures do not detect client-side game cheats such as automatic repetitive inputs (also called bots). Nor do they fully resolve other malicious behaviors, such as gold farming , phishing, client side tools; all of which involve optimized legal actions within the game. We do not address these either in the context of this paper. Yet, since our solution allows free provisioning of computing power and memory, action/command logs and retroactive checks for patterns associated with in-game misbehavior come at virtually no cost.

We want our distributed approach to at least match the degree of protection guaranteed by traditional C/S architectures. For this purpose our solution addresses all of the following failures :

- delays or modifications of the communication protocol
- modifications of stored data
- denial of service attacks
- collusions relative to the number of referees per fight

Regarding collusions, we detect a relative ratio corresponding to the number of correct referees we pick per fight. Our protocol can detect collusions between C nodes as it involves $2 * C - 2$ correct referees per fight.

In order to scale, our decentralized architecture requires trustworthy nodes to act as referees. It is risky to confer referee statuses indiscriminately, as it provides an easy way for dishonest nodes to turn the game to their advantage. Hence our solution relies on the availability of a distributed reputation system [20] in order to identify the most trustworthy nodes. Any such system [21], [22] would work, as long as it collects feedback about node behaviors and computes values that describe these behaviors.

V. Design of a decentralized refereeing system

The main goal of our approach is to provide efficient means for cheat detection in distributed gaming systems. In this paper, we focus on player versus player fights.

Our cheat detection mechanism relies on the integration of a large scale monitoring scheme. Every player node participates to the monitoring of game interactions between its neighbor nodes. As such, every node takes on the role of game *referee*. To avoid collusions among players/referees in order to gain an unfair advantage, no player should have the ability to select a referee. For this purpose, our architecture relies on a reputation system to discriminate honest nodes from dishonest ones. This allows picking referees among the more trustworthy nodes, as detailed in subsection V-B.

Once selected throughout the P2P overlay, referees act as servers/super-peers for the other nodes. Any node in the network may become a referee, as long as it keeps carrying out honest transactions. Resources that were required from the centralized servers in the C/S model are now fully distributed in the peer to peer network.

A. A generic protocol for refereeing in-game battles

Figure 1 depicts our decentralized refereeing approach. An initialization phase allows the reputation system to gain sufficient knowledge for a first estimation of node behaviors. Afterwards, any node can decide to initiate a battle with any other node, and asks one or more referees to help arbitrate the outcome. A battle between two opponents comprises one or more fights until a victory occurs. During a fight, every referee monitors the game commands sent by both opponents, and then decides the outcome of the fight based on the game data and on the correctness of the commands.

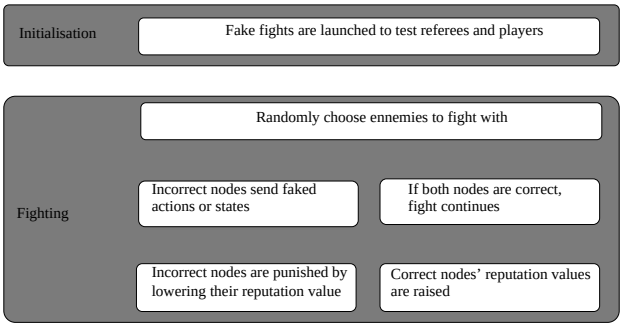

Fig. 1. Summary of our decentralized refereeing protocol

A classic issue for applications supported by reputation systems is the startup. In a situation where no transactions have yet been carried out, it is impossible to identify trustworthy nodes. The same problem holds for assessing the reliability of new nodes that join the application. We solve this issue by hiding fake requests into the flow of legitimate refereeing requests. During the game startup, all requests are fake until the reputation system estimates it can deliver reliable reputation values. Subsection VI-C details this mechanism.

When the reputation system is ready, players can initiate "real" battles under the supervision of one or more referees.

Once a battle has been initiated, the opponent nodes can create events they send to referees. Those events are based on the current player states. There are two types of events : *actions* describing inputs and *states* containing player states.

In our model, a cheater is a player node which either (a) makes up an event which does not match its state according to the game engine, or (b) delays the emission of an event.

Upon receiving an event from a player, a referee checks if the event is valid before transferring it to the player's opponent. Events are thus exchanged between two opponents under the supervision of one or more referees,until the battle ends. Three possible conditions lead to the end of a battle : one of the player health goes to 0, one or two of the players are cheating or both players agrees to stop fighting to call it a draw. If a cheater attempts to declare itself a winner illegally, both the referees and the opponent will detect the attempt.

B. Referee selection

In order to optimize cheat detection, our system picks referees among the trustworthy nodes that belong to the neighborhood of the player nodes. As described in Section VI,

trustworthy nodes are those whose reputation is above a fixed threshold T. A node A will only consider another node B as a potential referee if the reputation value associated with B as a referee increases above T. Self-refereeing is prohibited for obvious reasons: therefore player nodes are excluded from the referee selection for battles they are involved in.

We designed our referee selection mechanism to reduce the control any single node can have over fight outcomes. In other words, a single player node will have a low probability of managing to impose referee nodes. To achieve this goal, two player nodes involved in a battle must agree on the selection of referees. The player that initiates the battle first searches for available referees and books them for a fight. The player will then suggest this list of referees to its opponent. The opponent then double checks that those referees are trustworthy. A node that makes a lot of incorrect selections will quickly go down in the node reputation values, thus leading to its detection and possible exclusion from the system.

Even though sending constant fake referees list could disturb users game experience, the system will reduce the reputation value of the attackers in less than a minute, become resistant to this attack, and will automatically absorb it. We scale here way better than a centralized server who sometimes even needs a full shutdown of the platform.

Since reputation values are dynamic, it is possible for a referee to lose its trustworthiness in the middle of a battle. In this case, the battle is canceled: all refereeing requests associated with the corrupt referee are tagged as fakes and will only be used as information for the reputation system.

C. Cheat detection

Every fight constitutes an event which one of the opponents may try to corrupt to his advantage. Our refereeing system verifies events as follows.

Any player node can create events based on the game engine. In order to change the state of its avatar, a player node must issue a refereeing request containing an event description.

Upon receiving two descriptions associated with the same event –one for each opponent–, a referee will use the game engine to verify:

- the initial state and its consistency with the replicas stored in the P2P overlay;
- every action, to assess whether it is coherent with the player state;
- the new state of every opponent, to ensure that the actions got applied;
- victory announcements if any, to prevent the most obvious cheating attempts.

A battle triggers a loop of verifications on the referee (Figure 2); one verification per fight, until a victory occurs or until the battle gets cancelled.

Referees for a same battle send their decisions back to the player nodes directly; they don't communicate to reach a consensus. This does not introduce a breach in our security, as we can detect incorrect players while they try to cheat on decisions. Our architecture systematically detects cheating attempts, whether they come from a malicious player or from a malicious referee. If one of several referees sends a wrong decision to the players, the players will detect the inconsistency. If an incorrect player decides to take into account the wrong decision, correct referees will detect an incorrect player state at the next iteration. Finally, if a wrong decision turns an incorrect node into a victory, and if the incorrect node omits to send a message to claim its victory, both its opponent and the referees will eventually consider it as malicious.

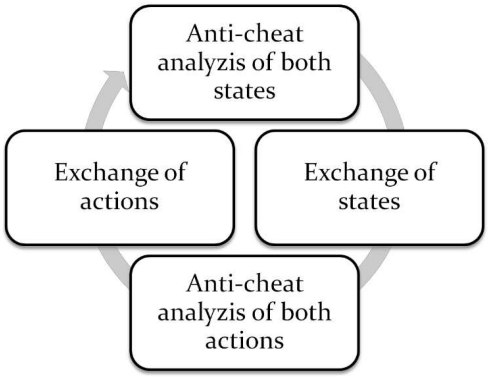

Fig. 2. Referee automaton used to analyze a full fight.

D. Multiplying referees to improve cheat detection

One referee isn't enough to ensure that our approach is sufficiently cheat-proof. A malicious node can temporarily send correct responses to gain a good reputation, and then issue corrupt decisions if it manages to acquire a legitimate referee status. Associating more than one referee with the same battle counters such behaviors.

Our solution enables players to select N referees for the same battle, with N an odd number. Once they have agreed on the referee selection, the players submit their requests concurrently to every referee. A player that receives $\frac{N}{2}+1$ identical replies may consider the result as trustworthy.

This approach has important advantages. It strengthens the trustworthiness of the arbitration, since the probability of picking $\frac{N}{2}$ malicious referees at the same time is considerably lower than that of selecting a single malicious referee. At the same time, it improves the detection of both malicious players and malicious referees and helps preventing collusions between a player and a referee. Collusion is a costly strategy, and the cost grows exponentially with the number of nodes involved. This is even truer with our approach since:

- a player alone cannot influence the selection of the referees,
- colluders must first work to obtain good reputations before starting to cheat,
- and a node can never know whether it is handling a legitimate or a fake refereeing request.

We analyzed the impact of the number of referees on the efficiency of the cheat detection and on the overhead. The results of this analysis, along with other results, are presented in section VII.

Collusion is only avoided when a majority of the referees given for a fight are correct. This implies that the referees

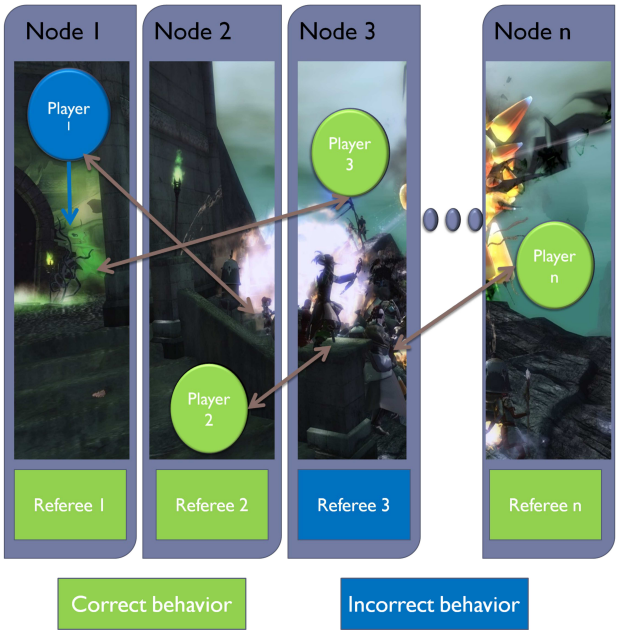

Fig. 3. N-referee configuration

aren't part of the collusion . Therefor it allows us to detect when there is less or equal than $\frac{N}{2}+1$ colluding nodes, N being the number of referees, by simply checking most popular decision of referees. As an example, with 3 referees we can only detect collusions of two node, be they player nodes or referee. Our refereeing system ensures that every possible combination of colluding nodes will result in a detection of this behavior. No possible configuration will allow to undergo an illegal action as long as we have a majority of correct referees.

VI. Improvement of the refereeing system through reputation

In order to scale, our decentralized architecture requires nodes to act as referees. It is risky to confer the referee status to every node indiscriminately, as it provides an easy way for dishonest nodes to turn the game to their advantage. Therefore our approach requires a way to pick out nodes that are reliable enough to act as referees.

Our solution integrates a distributed reputation system to identify the most trustworthy nodes. A reputation system [20] aims to collect and compute feedback about node behaviors. Feedback is subjective and obtained from past interactions between nodes, yet gathering feedback about all the interactions associated with one node produces a relatively precise opinion about its behavior in the network. In our case, this allows detecting players that cheat and to avoid potentially malicious referees.

There are two levels of trustworthiness for every node: as a player within the game (*player reputation*), and as a referee for the game (*referee reputation*). We dissociate both levels entirely as a node can play the game honestly while spreading incorrect information about other nodes, and vice versa. We use the former to decrease overheads (see Subsection VI-D), and the latter to establish a list of reliable referees before submitting requests for arbitration.

A. Assessment of the reputation

Every node stores a local estimation of the *player reputation* and of the *referee reputation* associated with every node in its neighborhood. Given our failure model, it makes no sense to fully trust a remote node, and therefore a reputation value can never equal the maximum value to translate this behavior in our simulation. In our system, a reputation value belongs to $[0, 1000[$. Value 0 represents a node which cannot be trusted, whereas the reputation value of a very trustworthy node approaches the unreachable 1000. Initially, when assessing a node that has no known history as a player (respectively as a referee), its *player reputation* (resp. *referee reputation*) value is set to 0.

There are two types of direct interaction between nodes that lead to a reputation assessment. A refereeing request causes the referee node to assess the *player reputation* of the requester, and the player node to assess the *referee reputation* of the requestee. The referee selection process triggers a mutual *player reputation* assessment between nodes.

Every time a node interacts with another node, both nodes assess the outcome of the interaction and update their local value for the reputation of their counterpart. A valid outcome increases this value ($reward$), while an incorrect outcome will decrease it ($punishment$). In our case, punishments must always have a greater impact on the reputation value than rewards. This prevents occasional cheaters from working their way to a good reputation value, which in turn confers an advantageous position for avoiding cheat detection or for acquiring referee status.

Evaluating a reputation through direct interactions only is a bad idea. Firstly it means that, in order to consolidate its reputation assessment, every single node must carry out multiple transactions with all the others: it is costly both in terms of time and resources. Secondly a malicious node may act honestly with a restricted set of nodes in order to escape banishment from the game if it gets detected by other nodes.

In our solution, nodes exchange their reputation assessments to help build a common view of their neighborhood in terms of player/referee trustworthiness. They exchange this reputation values in a sporadic manner. For reputation values that will imply a diminution it is sent as soon as possible as those information are resulted from an incorrect behavior. In the opposite way, if it is an increase the message will be emitted in the end of fight. We delay this message not only to save bandwidth, but also to make sure we are not raising the reputation of a node that still have the possibility to cheat during this fight.

Figure 4 gives an example of the way reputation assessments get propagated among nodes. Node $O(ther)$ starts interacting with node $R(eferee)$ to get information about a node he will soon fight. It so happens that node R is currently refereeing a battle between nodes $M(alicious)$ and $G(ood)$, and has therefore assessed their reputations. Node R piggybacks its assessments on its messages to node O.

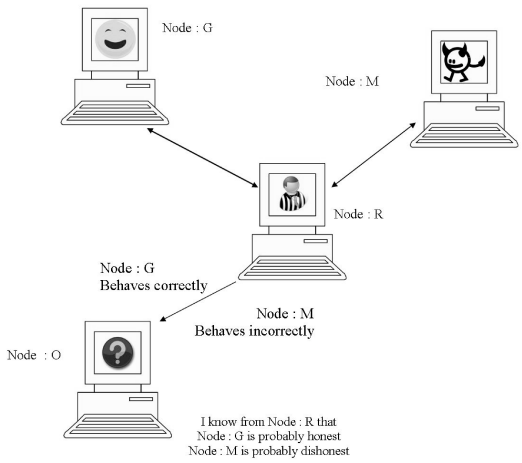

Fig. 4. Global reputation assessment in the network

The indirect reputation assessment of a node A by another node B generally relies on three types of information:

1) the current reputation value that B associates with A,
2) the evolution of the behavior of A as perceived by B,
3) and the recent opinions that B or other nodes may express about A.

Upon receiving fresh data about the behavior of a node, reputation systems such as TrustGuard [23] use a *PID (proportional-integral-derivative)* formula on these three pieces of information to compute a new reputation value. The principle of a PID formula is to carry out a weighted sum of the local reputation value at $t-1$, of the integral of the local reputation values since the system startup, and of the differential with newly received reputation values:

$$R(t+1) = \alpha * R(t) + \beta * \frac{1}{t} * \int_0^t R(t)\,dt + \gamma * \frac{d}{dt} * R(t)$$

The values for parameters α, β and γ are totally dependent on the application. A high value for α will confer a greater importance to past reputation values stored locally. This is useful in systems where close neighbors cannot be trusted. Parameter β focuses on the consistency of the behavior of a node. Systems with a high value for β prevent malicious nodes from wiping their slate clean with a few honest transactions. Finally, parameter γ reflects the transitivity of the reputation, in other words the direct impact of a new opinion on the local assessment. A high value for γ implies that the local reputation value of a node will be more sensitive to new values expressed locally or by other nodes.

Our reputation system simplifies this computational model to facilitate its implementation. We achieve this simplification by transforming the full model into an arithmetic progression. Let $I(t)$ be the value of the integral at time t, $D(t)$ the value of the differential at time t, and $a(t)$ a reputation value received from another node at time t.

Our first transformation is to reduce the integral to an iterative average:

$$I(t) = \frac{(I(t-1)+a(t))}{2}$$

This transformation adds a fading factor to the past behavior of the evaluated node. This is fine, as most reputation systems introduce a similar factor to increase the impact of the recent behavior, and therefore to be more responsive to sudden changes in the behaviour of the evaluated node.

Our second transformation reduces the derivative to a difference between two values:

$$D(t) = (a(t) - R(t))$$

Here also the transformation suits the requirements of our cheat detection: we estimate that it is more important to detect potentially malicious nodes than to identify trustworthy nodes. The new formula for $D(t)$ causes every transaction to impact strongly on the reputation value. Since the reputation value is reset to 0 periodically (see Subsection VI-B), $R(t)$ is more likely to remain low with every dishonest game action than to converge towards a high value with successive honest actions. In our case, we estimate that it is more important to detect potentially malicious nodes than to identify trustworthy nodes.

The computation for $R(t)$ in our implementation thus breaks down to:

$$R(t+1) = \alpha * R(t) + \beta * \frac{(I(t-1)+a(t))}{2} + \gamma * (a(t) - R(t))$$

B. Parameters associated with our reputation system

Several parameters associated with our reputation system allow to adapt it to the requirements of the application. Setting the parameter values is closely related to player/referee behaviors that are bound to be specific to every game. Hence the finetuning of these values requires extensive benchmarking during the test phase of the game software. Nevertheless, in this Subsection we provide pointers for the parameterization of the reputation system.

The first obvious set of parameters α, β, γ characterizes every reputation assessment. As mentioned earlier, in the context of cheat detection we believe the system should focus on its reactivity to incorrect actions. As the main component of the formula for this purpose, γ ought to be set to a high value.

Values υ (up) and δ (down) correspond respectively to rewards and punishments. As mentioned earlier, setting δ significantly superior to υ discourages malicious behaviors and prevents the promotion of corrupt nodes to referee status.

Every local reputation value associated with neighboring nodes is reset after every ρ updates; ρ is a parameter of the game code, as it depends on the robustness of the game design. This reset strategy helps prevent dishonest nodes from earning a good reputation, and at the same time protects honest nodes from bad recommendations spread by dishonest nodes.

We determine whether a node acts honestly by enforcing a global threshold T on reputation values. This threshold has two main uses in our solution:

- it serves as the main metric for the referee selection mechanism described in Subsection V-B,
- and it allows reducing the CPU load by randomly skipping refereeing requests from reputable nodes, as described in Subsection VI-D.

Similarly to ρ, the value of T is highly dependent on the game design; it must be set to the best trade-off between

cheat detection and efficiency. If the game software designers expect a proportion of cheaters that is either extremely high or extremely low, then the threshold value must be set very high. Indeed in such cases the reputation of incorrect nodes will be close to that of correct nodes, and cheaters will be harder to detect. Therefore a high threshold value will ensure that fewer malicious nodes will slip through our cheat detection. Conversely, a system that encounters a moderate proportion of cheaters will witness the emergence of two distinct reputation value averages among nodes : one average for honest nodes and another average for dishonest ones. The threshold becomes easier to set, as its value can be chosen somewhere between both averages. In general terms, the threshold value is inversely proportional to the distance between the number of cheaters in the system and half the size of the network.

C. Fake testing and jump start

Identifying trustworthy nodes is particularly tricky in two specific situations:

1) during the game startup,
2) and when a new node joins the application.

In both cases, reliable reputation values cannot be assessed for lack of transactions allowing to study node behaviors.

We solve this issue by integrating both *fake testing* and an *initialization phase*.

Fake testing consists in hiding fake requests into the flow of legitimate refereeing requests.

The hiding process is simple : we do not produce anything different than an usual fight will produce. When two nodes searches for a referee to make their fight, if no referee is available they are going to make the same referee selection once more, but this time to test the referee. A test is launched by the nodes when the referee to test is picked. At this moment the nodes will simply launch a request to the referee as if they were starting a normal fight. During the duration of the fake fight, the testing protocol will always vary to not make it easy to recognize but follow those rules :

- send incorrect and correct actions/states
- verify the referee answers to the actions/states
- keep the ratio correct/incorrect events high enough
- never send too much incorrect actions in a row

Those four points are kept repeatedly until the referee reputation value is changed enough to take one of two possible decisions : launch a real fight with this referee or broadcast that this referee is confirmed as incorrect.

It is important to note that in this way to test referees we are totally undetectable from the referee, but we will undergo some modifications of the nodes reputation values as the referee will think players are cheating. This is the main reason why a node will never send too many incorrect actions during a test : avoiding to have the reputation value resettled to 0.

In the end of the test , both nodes will send a confirmation to the referee that they both want to cancel the fight. Being able to forfeit/surrender is a possible procedure in all games. Here we use it also to put an undetectable end to a test than the referee had no chance to identify.

Even though one could argue that there is a strong chance to be tested when we are entering the system,and then use this information to abuse the system to raise its own reputation value, it is important to remember that in our system there is never a state of total trust. The trust will fall the instant a correct node will detect it has been included in a fight with corrupted referees. Moreover the fake testing accelerates the reputation assessment among nodes, and also discourages cheating attempts: tampering is already risky, why try it for potentially no benefit if our reputation value will be destroyed later?

New nodes start with an *initialization phase* where they only send out fake requests to their neighbor nodes. A node ends its *initialization phase* as soon as it has identified enough potential referees to start a fight – the minimum number of referees required for a fight is discussed in Section VII. The *initialization phase* has a very negative impact on the *player reputation* of the node as it sends both correct and incorrect commands to test for potential referees.

This procedure of hidden testing will continue for all game long. Frequently when searching for the best referees possible in their neighbors the nodes will find busy referees, and therefor they need to test again the other referees in decreasing order of reputation value.

Note that we are not forced to then test only bad referees when we pick one of the referee in the list of untrustworthy referees. As an example, an incorrect referee might just have been targeted by someone who wanted to lower its reputation, or simply had a spike of latency during a fight, producing incorrect behavior in the point of views of the players. Tests are therefor always useful during the full game run time.

D. Reducing the overhead induced by the cheat detection

Refereeing is a costly mechanism in terms of CPU and network usage. In order to reduce these costs, C/S architectures introduce heuristics aimed towards skipping some refereeing requests. Our solution allows extending this idea by identifying situations where skipping requests is more logical, that is when the request comes from a node with a good *player reputation*.

We propose the following formula to decide how often refereeing requests can be skipped:

$$SkipRatio = \frac{(R-T)}{(V-T+1)}$$

With R the reputation value of the requesting node, T the threshold value and V the maximum reputation value in the system.

$SkipRatio$ will increase as the reputation value of the requesting node gets closer to the maximum value. We introduce the threshold to guarantee that requests from trustworthy players are the only ones that get skipped. We added the plus one to the denominator in order to ensure that even the most trustworthy node requests get tested once in a while. As reputation values grow very slowly, a node can only get to this point if it has acted honestly towards a significant number of other nodes for a long time.

E. Portability of our solution to other reputation systems

In terms of reputation assessment, the requirements of our refereeing architecture are very basic. We need every node to store subjective values about the behavior of the other nodes in their neighborhood. With this in mind, we designed our own reputation system because it was simpler to prototype it quickly and integrate it into our simulations. But really, any other reputation system with similar characteristics [21], [22] would do.

VII. Performance evaluation

The present Section details the results of the performance evaluation we conducted in order to check the scalability and efficiency of our solution. We base our evaluation on simulations in order be able to analyze the behavior of our system when thousands of nodes are involved.

Our distributed refereeing system aims at offering a stronger alternative to the C/S architecture in the MMOG context. To prove that our approach is worthwhile, the overhead generated by our solution must be kept low in order to allow scalability and to offer a user experience that is indistinguishable from classic C/S performances. Obviously, we also expect our solution to detect the highest possible number of cheating attempts.

A. Simulation setup and parameters

Our performance evaluation is based on the discrete event simulation engine of PeerSim [24].

The virtual world management fall out of scope from this paper as we will plug ourselves on a P2P gaming overlay later on. In peersim, we implemented nodes as if they were linked to other nodes in a similar manner as in real games, changing the links randomly. We respected a distribution of heavy density zones with a lot of neighbors, and lower density zones with only few nodes in neighborhoods.

Players, have states generated accordingly to what you could find in a normal MMORPG game. They are all different, and their levels differs, producing some quick and longer battles depending of the two players facing each other. They evolve for now in an infinite world, in which if they want to fight a probe request to the neighborhood will be sent to see which node is available. In our simulation players interact with other nodes from their neighborhood around every fight minutes for a fight of an average duration of around two and a half minutes.

Fights related messages are sent when a player emit an input. Inputs are considered to be polled 25 times per second, to respect a decent frame rate. Other messages such as testing messages and reputation messages are sporadic and are sent as soon as possible. We determine that reputation messages are sent when :

- an end of fight happened without incorrect actions
- a node tried to input too much incorrect actions

To avoid also cyclic behaviors that could have been created by producing fights every five minutes, we added some skewness in all the values of inactive state, fighting state, inputs, etc... making every node different from the others and desynchronized.

This world we described evolves in peersim, generated with random seeds given by peersim at each simulation that will produce the player random states. Every run simulates game interactions among 30,000 nodes over a 24-hour period. Every measure presented hereafter is a mean value computed with results from 40 different simulation runs. We performed our simulations on a Dual-CPU Intel Xeon X5690 running Debian wheezy v3.2.0-4 at 3.47Ghz, with 128GB of available memory. Every run generated an average of 24 CPU threads.

In order to find appropriate values for the parameters of our reputation system, we ran several simulations prior to the ones presented in this Section. We found the optimal parameter setting to be as follows: threshold $T = 300$, reset frequency $\rho = 50$, $\alpha = 0.2$, $\beta = 0.2$, and $\gamma = 0.8$. γ is much higher than α and β because we want the detection mechanism to focus on quick reactions to the strong punishments that incorrect nodes can receive.

The punishment δ and reward υ have specific behaviors in our reputation system. δ aims at making nodes to converge to 50 of reputation value when small errors are detected to warn the node.

We considered this value of 50 after studying in each of our multiple simulations what was the average reputation value an untrustworthy node can achieve in our system. An untrustworthy node for example can be a node that used to behave correctly since a while but suddenly decided to send some cheat attempts.

In a second step, if nodes insist in cheating, the reputation value of the incorrect node will quickly reach the minimum value.

In the same manner , υ is used to make nodes converge slowly to the maximum value. A node inferior to 500 reputation value will be raised until it reaches it, then will aim for 1000.

We set the concurrent number of nodes in our system to the largest value our simulator could handle while running on our hardware: 30,000 nodes. It still is twenty times as big as the unofficial client/server concurrent limit we identified through various Internet sources[5], [4]. We also ran some short simulations with up to 60,000 nodes, and observed no difference in the behavior of our system. Actually, our approach seems to behave independently from the number of nodes in the network.

The metrics we used to assess the scalability and efficiency of our system are:

- the latency,
- the network bandwidth consumption,
- and the CPU overhead introduced by our architecture,
- as well as the percentage of undetected cheating occurrences.

B. Latency

We introduced a uniformly random latency for every exchange between two nodes. The lower and upper bounds are set to 10 and 40 ms respectively. We estimated these values by monitoring the traffic we generated while playing *Guild Wars 2*.

We also have a database containing pings from users of a famous nowadays online game, *League of Legends*. We found that the average ping of users was around 40ms. We decided that between two peers we simply double the value, reaching around 80ms ping, which makes us 40ms latency on messages. The justification behind this reasoning is that once a client can reach a high response server in 40ms ping, there is in average the same time then to go down on an other node. The slowest part of the two between the servers of League of Legends and the client is obviously the client's internet access which then accounts for a majority of the ping response time.

We also tried to see and verify that our assumption was correct and we sent pings requests between two different houses with standard ADSL2+ connections and managed to get the same latency results as we estimated.

In our solution, results show that we add a latency comparable to that of a single message exchange between two nodes in our network: 40ms. This comes from the two way communication we impose between referees and peers. Even if we consider a (very) slow network with a 100ms average latency, the 200ms latency introduced by our system remains affordable in the context of MMORPGs.

C. Bandwidth consumption

Similarly to latency, we evaluated the bandwidth consumption of our solution per message. We kept count of the number and the size of all the messages exchanged during every simulation run. With these logs, we correlated message contents and real memory usage of data such as integers or floats in classic games to compute realistic message sizes.

To compute the message size, we summed up everything we were sending in the network. There are two cases :

- state messages
- action messages

State messages in our scenario will send all the game related information it knows about himself. This contains 84 bytes when serialized before being sent on the network.

Actions messages are a bit smaller, they only send the difference of what the player will do compared to his state, all the non sent data is kept unchanged when receiving an action message. The actions messages contains only 16 bytes when serialized and sent to the network.

State and action messages are sent step by step in the network according to the automaton shown in Figure 2,giving an even amount of both. Knowing the size of the message we only had to count them to then deduce the bandwidth used on a single node over time.

As shown in Figure 5, we measured that each node in our solution consumes a mean of 4KB/s once the system has stabilized. Given that the maximum consumption never exceeds 8KB/s, we consider these results to be excellent.

We also performed a theoretical comparison between our approach and the client/server approach. For this purpose, we defined a virtual centralized server as a computer with unlimited resources, able of handling all concurrent requests without crashing. We then summed up the average workloads handled by every peer in our distributed solution during the simulation runs, divide it by the number of referees we run in the decentralized version,and fed them to the virtual centralized server. Figure 6 shows that the consumption peak in the client/server architecture reaches 67MB/s. With this load in a real game deployment, the server would crash and the players would be disconnected. Such situations occur pretty frequently with existing games nowadays. A cluster of server might be able to handle this direct overload, but then comes the need to put in place a complex coordination and consensus algorithm between the servers. Which for now we saw no reference of games companies being able to put this kind of technology at work.

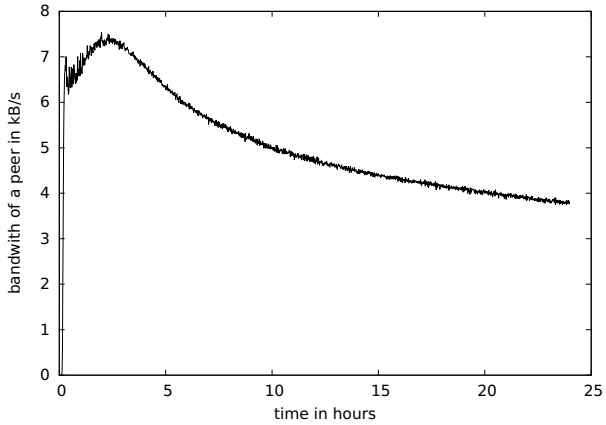

Fig. 5. P2P node bandwidth usage over time

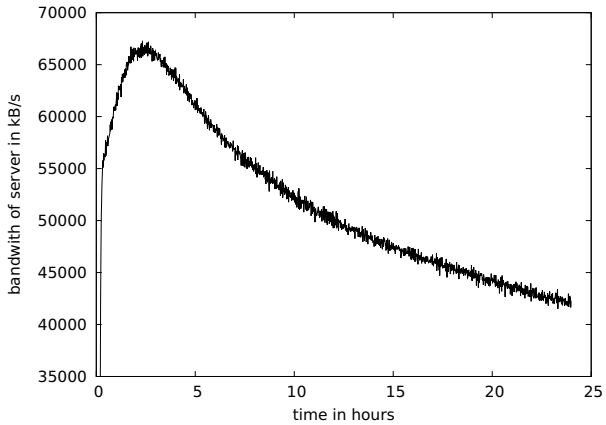

Fig. 6. Server relative bandwidth usage over time

Our approach is based on a reputation system: its ranking provides the ability to pick out trusted referees. We checked how much bandwidth our own in-built reputation mechanism uses in those 4KB/s. It appears that, compared to the real size of the game protocol messages, it uses only around 1.5% of all network data sent (as illustrated by Figure 7).

Reputation messages are sent only when needed, as we really took care about this part of the reputation system, and they also are small. They share roughly the same size as the action messages and weight 16 bytes, they contain a descriptor of a node, and the new reputation value we want to share for this node.

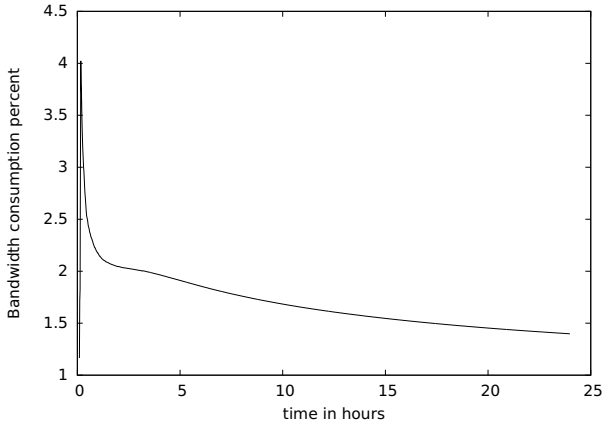

Fig. 7. Bandwidth consumed by the reputation system

D. CPU load

To estimate the CPU overhead introduced by our refereeing system, we assigned 1 *" cpu load point"* to every action/state creation (cpu usage associated with the game software) and to every action/state test (cpu usage induced by our approach). Please note that this method inevitably leads to an overestimation of the overhead induced by our approach. In a real implementation the CPU usage associated with tests is really small compared to the other CPU dependent operations a game can produce, such as Nvidia PhysX or artificial intelligence computations.

Number of referees	One	Three	Five
CPU overhead	27,85%	64,59%	210,56%

TABLE I
CPU OVERHEADS

We computed the values of the CPU overhead, and display the results in Table I. Obviously, our CPU overhead is closely related to the number of referees per battle. In spite of our test skipping optimization, the 5 referees per battle configuration induces a prohibitive overhead. Considering that most current games requires a dual CPU core, a 100% CPU overhead on a quad-core processor seems an acceptable maximum. Once again, those values are vastly overestimated.

We also aims at types of game that we could check on personal machines which are average in terms of gaming CPUs, whose showed that they mainly use roughly 50% of the quad-cores and nearly fully the dual-cores. Multiplayer online battle arenas such as League of Legends, Dota2, Bloodline champions all ran perfectly when bound to only one or two cores. MMORPGs like Tera online, Rift ran also without issues when placed on 2 cores only. Only the recent Guild Wars 2 showed some different frame rates when bound to 2 cores only as the game can produce a CPU bottleneck with some of the Extreme graphics options. As the graphics are extremely rich, CPUs are highly stressed by the quantity of information to send to the GPU.

E. Cheat detection ratio

With respect to cheat detection, since there are no available statistics on the number of cheating occurrences that go undetected in the gaming industry, we have nothing to compare ourselves with. Therefore we set the percentage and distribution of incorrect nodes arbitrarily, while trying to remain realistic with respect to both our own gaming experiences and the literature on byzantine behaviors in P2P networks.

For instance, we believe there are diverse malicious behaviors in the context of gaming. A person using the service may want to cheat as a player, but may not want to disturb other players. Conversely some attackers only focus on damaging the system and do not care to play at all. The distinction between *player reputation* and *referee reputation* reflects this analysis: it implies that the player component and the referee component on a single node can behave independently from one another.

In the literature about reputation systems[23], 5% is often a higher proportion of incorrect/malicious nodes compared to what is commonly considered. We chose to stretch this value even further: we set the proportion of potentially malicious players to 30% and the proportion of potentially malicious referees to 10%. The latter is lower than the former because a lack of reputable referees has a perverse effect on the simulation: it decreases substantially the number of battles that can proceed, and hence the overall number of cheating attempts.

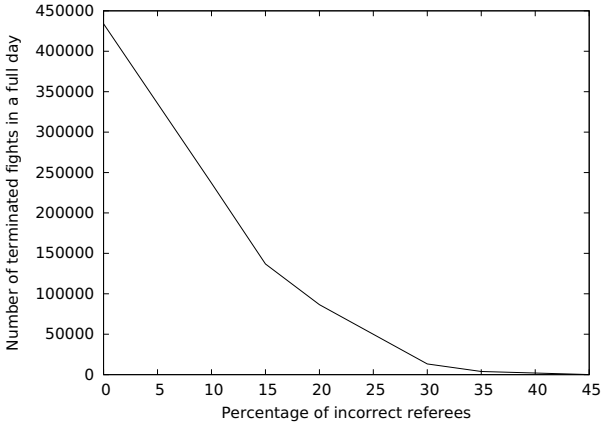

Fig. 8. Number of possible fights as the proportion of correct referees decreases

To study the impact of the proportion of incorrect referees on the quality of the cheat detection in our solution, we initially ran simulations with a proportion of incorrect referees varying between 0% and 45%. Figure 8 illustrates the rapid decrease in the number of battles per day that such a variation brings. We believe that 10% of incorrect referees remains a high value anyway, probably above the proportion any real game faces.

Despite this negative outcome our simulation results shown in table II demonstrate that, regardless of the number of battles that do proceed, *the proportion of undetected cheating occurences always remains stable under 0,013%* with three or

Number of referees	One	Three	Five
Cheat percentage	1,17%	0,0128%	0,0099%

TABLE II
UNDETECTED CHEAT RATIO

more referees per battle. In our opinion, this shows that our solution is effective for detecting and avoiding both incorrect referees and incorrect players.

In table III, we correlate the measures from tables II and I to obtain the CPU overhead relatively to the percentage of cheating attempts that went undetected. The results show that our "many-referees" configurations are CPU cost efficient compared to the single referee one, especially the 3 referee configuration.

Number of referees	One	Three	Five
CPU overhead for relative 1% cheat undetected	32,6625%	0,83%	2,10%

TABLE III
CPU OVERHEAD RELATIVE TO UNDETECTED CHEAT

VIII. Related work

There has been significant research on P2P overlays for online games. Several approaches, such as Colyseus [25], BlueBanana [26], and Solipsis [17], aim to adapt the positions of the nodes in the network to the needs of the applications. This makes them suitable for MMOGs. Colyseus is efficient enough to support FPS games. MMORPGs have lower update rates and have much smaller per-player bandwidth requirements than FPS games [27]. BlueBanana and Solipsis adapt the network overlay to the movements of the avatars. Unfortunately none of these approaches address the issue of cheating.

To deal with this open issue, [16], [28], [29], and [30] propose distributed cheat detection mechanisms. In [16] and [28] the authors use trusted entities to handle security, and in [30] super-peers are used as proxies for the overall security, while in [29] zones are created where every player asses other players actions. They gather all the information required to create a decentralized and secure approach for MMOGs, and introduce the concept of distributed refereeing into the P2P network so as to implant trusted entities. Since every action in a game can be discretized into small events, the analysis and arbitration of actions received from the game clients are delegated to those entities.

These solutions introduce the basic mechanisms for fully decentralized P2P gaming. However they do not tolerate a high number of incorrect nodes. Note that our approach is quite similar to RACS [28]. However, RACS limits the arbitration to a single referee. It deals neither with cooperations amongst multiple referees, nor with the selection of trusted referees.

Another type of approach is to mix the benefit of P2P load balancing with a centralized server, as does [30]. Such hybrid approaches also need to identify trusted nodes in the network in order to delegate arbitration tasks. Although this helps the server when it comes to handle a lot of CPU heavy operations, the maximum number of concurrent players will remain bounded, and orders of magnitude below the requirements of MMOGs.

Our solution can make use of any fully decentralized reputation system to identify potential referees. It relies neither on region controllers nor on chosen nodes of the overlay. Moreover our approach delegates every arbitration to multiple referees. As our simulations show, this allows to enforce an efficient cheat detection without hindering scalability.

IX. Conclusion and future works

The scalability of massively multiplayer online games is a hot issue in the gaming industry. MMOGs built on top of fully decentralized P2P architectures could scale. However they would face a major problem: the impossibility of implementing a distributed refereeing authority that can be fully trusted.

In this paper, we propose a probabilistic solution based on a reputation system. It tests the behavior of nodes by submitting fake refereeing requests and then picks out those it identifies as trustworthy to referee real game actions. To improve the detection of dishonest arbitrations by malicious referees, every request can be submitted to multiple referees concurrently.

The first step we took towards validating our solution was to run simulations to test whether it is viable when thousands of nodes are involved. We included the simulation results in this paper and showed that it is possible to provide an efficient anti-cheat mechanism in large-scale peer to peer MMOGs without degrading their scalability. In a game involving 30,000 players, our solution manages to leave as little as 0,0128% of cheat undetected despite a cumulative proportion of 40% of incorrect nodes. In this context, its maximum bandwidth consumption never exceeds 7KB/s. This can be compared to the unrealistic 40MB/s that a central server would consume if it were to handle as many nodes as our solution does.

We are now working on the second step towards validating our solution: the implementation of a proper prototype and its deployment. We are integrating enhancements to its design that limit even further the number of tests when the system is stable, so as to improve CPU usage without degrading the accuracy of the detection. At the time of writing this paper, our implementation is still being tested.

We are also working on coupling our solution with a *matchmaking system*. Matchmaking systems connect players together for online play sessions. We have yet to encounter a matchmaking system that scales and responds in a timely manner, and we believe our approach is very well suited for such a functionality.

Finally, we plan to extend our approach to other kinds of cheating, for example when players perform legal actions but still behave maliciously within the game. For instance, $goldfarming$ consists in gathering virtual resources in order to sell them for profit in the real world; it is a source of intense frustration for MMORPG players. We are exploring the notion of using the multi-agent paradigm to solve this issue. Agent-based systems are good at representing and monitoring

complex behaviors and rich interactions between nodes. One such system could spot abnormal behaviors linked to stolen accounts and goldfarming. Additionally it could participate to the removal and replacement of bad referees after their detection.

REFERENCES

[1] J. Miller and J. Crowcroft, "The near-term feasibility of P2P MMOG's," in *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*, Nov. 2010, pp. 1 –6.

[2] "Blizzard Admits Diablo 3 Item Duping Is Why Asia's Server Was Shutdown," June 2012. [Online]. Available: http://www.cinemablend.com/games/Blizzard-Admits-Diablo-3-Item-Duping-Why-Asia-Server-Was-Shutdown-43472.html

[3] "Blizzard Seeking Hacker Behind WoW Insta-Kill Massacre," http://www.tomshardware.com/news/World-ofWarcraft-Aura-of-Gold-MMORPG-Insta-kill-exploit. [Online]. Available: http://www.tomshardware.com/news/World-ofWarcraft-Aura-of-Gold-MMORPG-Insta-kill-exploit,18219.html

[4] "World Of Warcraft Servers player capacity." [Online]. Available: http://www.warcraftrealms.com/realmstats.php?sort=Overall

[5] "World Of Warcraft Servers average concurrent players." [Online]. Available: http://www.warcraftrealms.com/activity.php?serverid=-1

[6] S. M. Specht and R. B. Lee, "Distributed denial of service: taxonomies of attacks, tools and countermeasures," in *Proc. of the Int. Workshop on Security in Parallel and Distributed Systems*, 2004, pp. 543–550.

[7] "Onlive - The leader in cloud gaming." [Online]. Available: http://www.onlive.com/

[8] "Nvidia Grid - cloud gaming." [Online]. Available: http://blogs.nvidia.com/blog/2013/08/30/best-in-show/

[9] "Nvidia shield device cloud streaming abilities." [Online]. Available: http://www.slashgear.com/nvidia-project-shield-cloud-streaming-gaming-abilities-revealed-07263550/

[10] "Weekly loot - discussing about future of the cloud gaming." [Online]. Available: http://mmoattack.com/mmo-videos/future-cloud-gaming-weekly-loot

[11] "New datacenter for league of legends." [Online]. Available: http://forums.euw.leagueoflegends.com/board/showthread.php?p=1544299#1544299

[12] "Steam traffic accounted for 2 percent of all net traffic during DOTA 2 update, claims Gabe." [Online]. Available: http://games.on.net/2013/03/steam-traffic-accounted-for-2-of-all-net-traffic-during-dota-2-update-claims-gabe/

[13] "Microsoft : Cloud will quadruple power of Xbox One." [Online]. Available: http://www.tomshardware.com/news/Xbox-One-Cloud-Jeff-Henshaw-Matt-Booty-Adam-Pollington,22775.html

[14] "Cloud services price related to bandwith usage." [Online]. Available: http://thomas.broxrost.com/2009/08/28/updated-awsgo-daddy-dedicated-server-cost-comparison/

[15] "Cloud status." [Online]. Available: http://cloudharmony.com/status

[16] T. Hampel, T. Bopp, and R. Hinn, "A peer-to-peer architecture for massive multiplayer online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, ser. NetGames '06. New York, NY, USA: ACM, 2006.

[17] D. Frey, J. Royan, R. Piegay, A.-M. Kermarrec, E. Anceaume, and F. Le Fessant, "Solipsis: A Decentralized Architecture for Virtual Environments," in *1st International Workshop on Massively Multiuser Virtual Environments*, Reno, NV, États-Unis, 2008.

[18] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 389–400.

[19] J. Yan and B. Randell, "A systematic classification of cheating in online games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, ser. NetGames '05. New York, NY, USA: ACM, 2005, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/1103599.1103606

[20] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, Mar. 2007.

[21] A. Rahbar and O. Yang, "PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 4, pp. 460 –473, Apr. 2007.

[22] E. Rosas, O. Marin, and X. Bonnaire, "CORPS: Building a Community Of Reputable PeerS in Distributed Hash Tables," *The Computer Journal*, vol. 54, no. 10, pp. 1721–1735, Sep. 2011.

[23] M. Srivatsa, L. Xiong, and L. Liu, "TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks," in *Proceedings of the 14th international conference on World Wide Web*, ser. WWW '05. New York, NY, USA: ACM, 2005, pp. 422–431.

[24] A. Montresor and M. Jelasity, "PeerSim: A Scalable P2P Simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, Seattle, WA, sep 2009, pp. 99–100.

[25] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 12–12.

[26] S. Legtchenko, S. Monnet, and G. Thomas, "Blue banana: resilience to avatar mobility in distributed MMOGs," in *The 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)*, Jul. 2010.

[27] K.-T. Chen, P. Huang, C.-Y. Huang, and C.-L. Lei, "Game traffic analysis: an MMORPG perspective," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '05. New York, NY, USA: ACM, 2005, pp. 19–24.

[28] S. D. Webb, S. Soh, and W. Lau, "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)*, 2007.

[29] T. Hampel, T. Bopp, and R. Hinn, "A peer-to-peer architecture for massive multiplayer online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, ser. NetGames '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1230040.1230058

[30] J. Goodman and C. Verbrugge, "A peer auditing scheme for cheat elimination in MMOGs," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, ser. NetGames '08. New York, NY, USA: ACM, 2008, pp. 9–14.