Multimedia Systems
DOI 10.1007/s00530-014-0358-0

SPECIAL ISSUE PAPER

Towards a scalable refereeing system for online gaming

Maxime Véron · Olivier Marin · Sébastien Monnet · Zahia Guessoum

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

1 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]. Even emerging cloud-based solutions face limitations similar to that of C/S architectures. 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.

Client/server 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.

This paper presents a scalable game refereeing architecture that monitors the game both efficiently and securely. Our main contribution is a fairly trustworthy peer-to-peer (P2P) overlay that delegates game refereeing to the player nodes. It relies on a reputation system to assess node honesty and then discards corrupt referees and malicious players. Our secondary contribution is a speedup mechanism for reputation systems which accelerates the reputation-assessment by challenging nodes with fake game requests.

This paper is organized as follows. Section 2 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 3 outlines the characteristics of the reputation system our solution requires to achieve such performance, and details the

M. Véron (✉) · O. Marin · S. Monnet · Z. Guessoum
Laboratoire d'Informatique de Paris 6/CNRS, Université Pierre et Marie Curie, Paris, France
e-mail: maxime.veron@lip6.fr

Published online: 07 February 2014

simple reputation system we designed following this outline. Section 4 gives a preliminary performance evaluation obtained by simulating our solution. Section 5 discusses related work. Finally, Sect. 6 makes a case for decentralized architectures as building blocks for game softwares and discusses the complementarity of our approach with cloud gaming.

2 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 battles.

Our cheat detection mechanism relies on the integration of a large scale monitoring scheme. Every player node participates to the monitoring of game interactions among its neighbor nodes. As such, every node takes on the role of game *referee*. To limit collusions among players/referees 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 Sect. 2.4.

2.1 System model

We want our system to work on top of a P2P overlay designed for gaming, such as [4, 5]. This assumption on the infrastructure induces our base hypotheses.

Our model focuses on battles between two nodes. We argue that it is scalable in that it paves the way to handling very large numbers of one-on-one battles simultaneously. Although this model could be extended to small sets of nodes fighting cooperatively (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 is part of two overlays, it thus maintains two neighbor lists : a list of the player's immediate neighbors within the virtual world of the game, and a list of neighbor nodes within the reputation 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 [4–6] already address this issue.

After a battle, its assigned referees reach a consensus to update the status of each opponent in the virtual universe. We assume that in-game battles last long enough. Synchronization does not occur frequently, only at the end of a battle, which last at least several minutes. Furthermore, as it does not happen during a battle it will never block a game, it has only to be done before one of the players starts a new battle.

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.

2.2 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 in [7] 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 between nodes as long as there is at least a majority of correct referees per battle.

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 [8] to identify the most trustworthy nodes. Any such system [9, 10] would work, as long as it collects feedback about node behaviors and computes values that describe these behaviors.

2.3 A generic protocol for refereeing in-game battles

Our decentralized refereeing approach starts with an initialization phase that allows the reputation system to complete 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 skirmishes until a victory occurs. During a battle, every referee monitors the game commands sent by both opponents, and then decides the outcome of the skirmish based on the game data and on the correctness of the commands.

1. Initialisation: 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. Section 3.3 details this mechanism.
2. Fighting: 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: (1) one of the player health drops to 0, there is a winner; (2) one or both players are cheating, the battle is canceled; or (3) both players agrees to call it a draw. If a cheater attempts to declare itself a winner illegally, both the referees and the opponent will detect his attempt.

2.4 Referee selection

To optimize cheat detection, our system picks referees among the trustworthy nodes that belong to the neighborhood of the player nodes. As described in Sect. 3, 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 skirmishes 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 battle. 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.

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.

2.5 Cheat detection

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

Any player node can create events based on the game engine. 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 (Fig. 1); one verification per skirmish, 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 do not communicate to reach a consensus. This does not introduce a breach in our security, as we can detect incorrect nodes 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 disclaim its victory, both its opponent and the referees will eventually consider it as malicious.

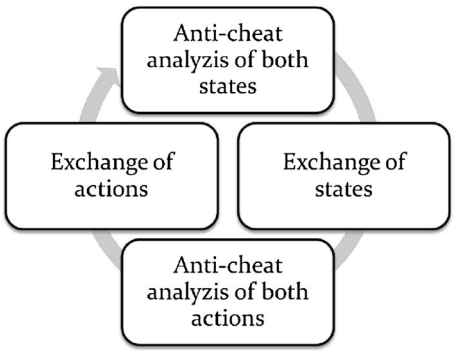

Fig. 1 Referee automaton used to analyze a full battle

2.6 Multiplying referees to improve cheat detection

One referee is not 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 (Fig. 2). 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 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

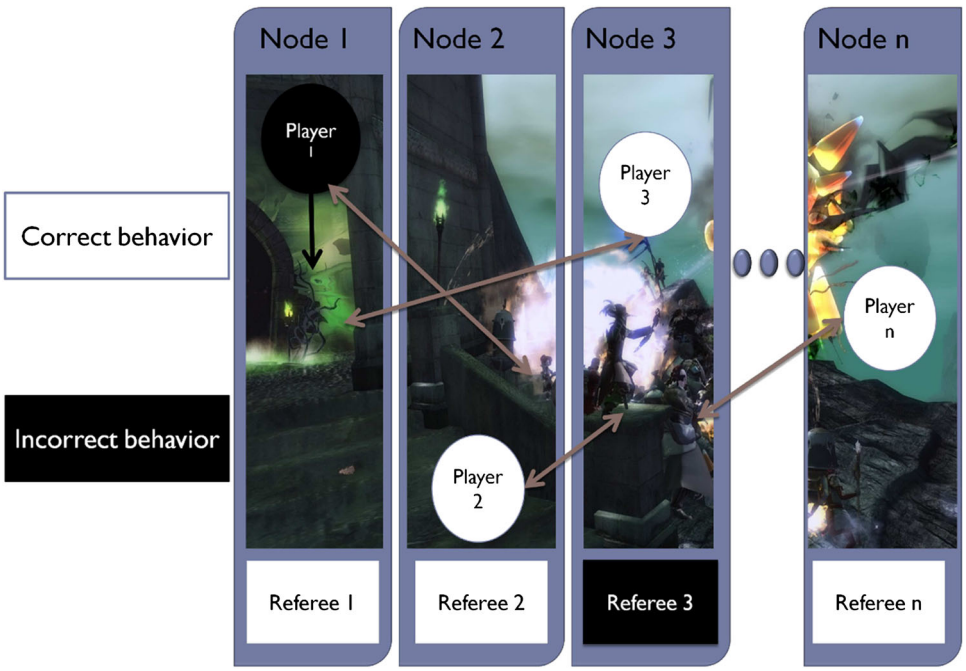

Fig. 2 N-referee configuration

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 Sect. 4.

Avoiding collusions for a given battle requires an assessment by a majority of correct referees (correct referees do not collude). Collusions among referees, or among players, or even between referees and players will result in a detection: (1) players will always receive a majority of correct referee decisions as there is a majority of correct referees by assumption, different decisions come from faulty referees; (2) correct referees detect players that do not respect their decisions when they receive further messages from both players; (3) the majority of correct referees will detect collusions between players and referees and updates of the virtual world data remain reliable anyway as they require a majority of referees. Thus, no possible configuration provides the ability to undergo an illegal action as long as, for each battle, a majority of referees are correct. Notice that if there is a majority of malicious referees for a given battle, they might behave badly, however this can be detected by correct players, leading to a reputation decrease for the referees.

3 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 [8] 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 Sect. 3.4), and the latter to establish a list of reliable referees before submitting requests for arbitration.

3.1 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, 1,000]. Value 0 represents a node which cannot be trusted, whereas the reputation value of a very trustworthy node approaches the unreachable 1,000. 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, 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 to escape banishment from the game if it gets detected by other nodes.

In our solution, nodes exchange their local reputation assessments sporadically to help build a common view of their neighborhood in terms of player/referee

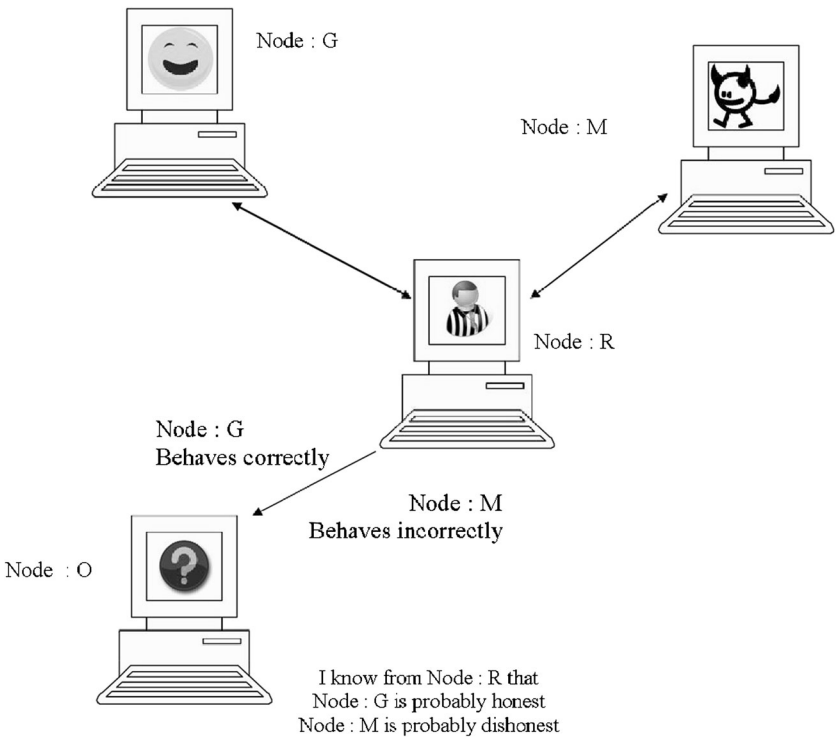

Fig. 3 Global reputation assessment in the network

trustworthiness. Reputation updates resulting from an incorrect interaction are sent immediately, while those that sanction honest behaviors are delayed until the end of the battle. This saves bandwidth and prevents reputation increases for nodes that exhibit an erratic behavior with respect to trustworthiness.

Figure 3 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 it intends to 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*.

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 [11] 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:

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

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 Sect. 3.2), $R(t)$ 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:

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

3.2 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 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 Sect. 2.4,
- and it reduces the CPU load by randomly skipping refereeing requests from reputable nodes, as described in Subsection 3.4.

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.

3.3 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 in sufficient number 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. Fake requests are identical to legitimate requests, and therefore recipient nodes cannot distinguish between both types. When two player nodes fail to book enough trustworthy referees for a real battle arbitration, they initiate a referee testing phase and pick a node among their list of currently untrustworthy nodes. The testing phase consists in sending requests associated with a normal battle, according to the following rules.

1. Requests contain both incorrect and correct actions/states.

2. Player nodes verify the referee answers to every action/ state request.
3. Player nodes maintain a ratio of correct/ incorrect event requests high enough to protect their own player reputation.

Since the tested referee cannot discern test requests, it may suspect the requesting nodes of being malicious and cause their player reputation value to be reset to 0. Rule 3 avoids this situation by balancing incorrect test requests with correct ones.

The testing phase lasts until the referee reputation value allows one of two possible decisions: either the launch of a real battle with the tested node as referee, or a broadcast to notify that the tested node is suspected unfit for refereeing. At that point, both player nodes notify the referee to cancel the battle. All games provide a forfeit/ surrender procedure: we use it as a means to end the testing phase in such a way that it remains undetectable for the referee.

Entrant 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 battle – the minimum number of referees required for a battle is discussed in Sect. 4. 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. Our system compensates for this by enforcing a full reset of the reputation value periodically, as described in Sect. 3.2

Entrant nodes also incur a high probability of getting tested when entering the system. Malicious entrant nodes may use this knowledge to their own advantage by acting honestly for a long period of time and then cheating. This strategy will be short-lived in our system since no node is ever fully trusted: a correct node will inflict a heavy punishment as soon as it detects a malicious interaction. 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? A variant strategy, reentrance to regain a clean slate, is not valid in our model as each player node is associated with a unique game identifier.

Fake testing remains effective for the whole game duration. Besides accelerating the detection of malicious nodes and the integration of honest nodes, it maintains a list of potential referees in the neighborhood of every node. This list comes handy when the most trustworthy neighbors are busy refereeing other battles. Maintaining this list also offers redemption to correct nodes that got their reputation spoiled by malicious nodes, or to nodes that exhibited an incorrect behavior because of a latency spike during a battle.

3.4 Reducing the overhead induced by the cheat detection

Refereeing is a costly mechanism in terms of CPU and network usage. To reduce these costs, C/S architectures introduce heuristics aimed towards skipping some refereeing requests. Our solution extends 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 use the following formula to decide how often refereeing requests can be skipped:

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

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 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.

3.5 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 behaviors 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 [9, 10] would do.

4 Performance evaluation

The present section details the results of the performance evaluation we conducted 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 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.

4.1 Simulation setup and parameters

We use the discrete event simulation engine of PeerSim [12] to conduct simulations of our system. The present performance evaluation constitutes a first assessment of feasibility. We intend to validate our solution further by plugging it over a P2P gaming overlay with a *world* component later on, but *virtual world* management falls out of the scope of this paper.

Our simulation aims at reproducing the topology of real game overlays: random link changes and a node distribution with zones of varying densities. Player states, accordingly to what we expect to find in most MMORPG games, are all different. Player levels differ, producing both short and long battles. The world is infinite: players probe their neighborhood every five minutes to find opponents, and we set the average battle duration at two and a half minutes. Battle related messages are sent when a player emits an input. Inputs are 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. Reputation messages are sent either when a battle ended without incurring any incorrect action, or when a node tried to input too many incorrect actions. To prevent synchronizations and cyclic behaviors produced by the fixed periodicity of battles, we introduced skewness by ensuring that inactive state values, fighting state values, inputs, and so on, differ for every node.

Every run simulates game interactions among 30,000 nodes over a 24-h period, and uses random seeds provided by PeerSim to generate original player states. 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 3.47 Ghz, with 128 GB of available memory. Every run generated an average of 24 CPU threads.

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.

Punishment δ and reward υ produce specific behaviors in our reputation system. δ forces the reputation value to converge quickly towards 50 when small errors are detected. Multiple preliminary simulations yield 50 as the average reputation value an untrustworthy node achieves on the long run in our system. Conversely, υ enforces a slower convergence of the reputation value for honest nodes towards 500, the overall average for trustworthy nodes.

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 [13, 14]. 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.

4.2 Latency

We introduce a uniformly random latency for every exchange between two nodes and set the lower and upper bounds to 10 and 40 ms, respectively. We first estimated these values by monitoring the traffic we generated while playing *Guild Wars 2*, and later backed our estimations by interrogating a database we have put together with data acquired from a *League of Legends* server, and covering over 20 million games. We assume that doubling these values approximates the latency between two peers: the idea is to cumulate the time it takes for both clients to reach a high response server, and to consider that the client's internet access accounts for a majority of the ping response time. We verified our assumption summarily by exchanging ping requests between two different private locations in the Paris region with standard ADSL2+ connections: the latency results match our expectations.

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

4.3 Bandwidth consumption

Similarly to latency, we evaluated the bandwidth consumption of our solution per message. We kept count of the

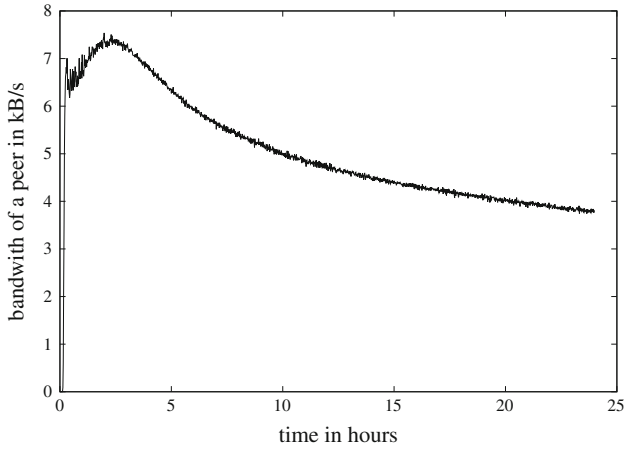

Fig. 4 P2P node bandwidth usage over time

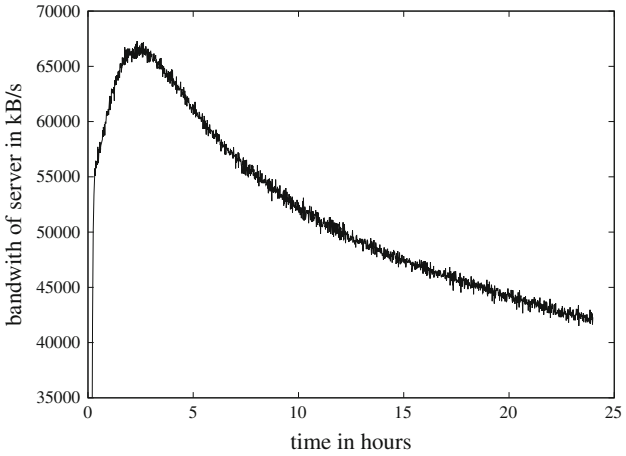

Fig. 5 Server relative bandwidth usage over time

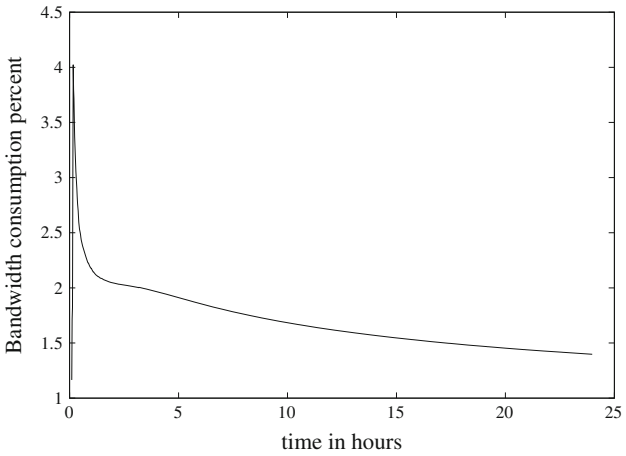

Fig. 6 Bandwidth consumed by the reputation system

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 and action messages. In our scenario, state messages contain all the game related information a node knows about itself. This amounts to 84 bytes when serialized before being sent on the network, and matches the value found in the literature [15]. Action messages are a bit smaller (16 bytes when serialized), as they only contain incremental information. By correlating these values with the message count of every node, we deduce the average bandwidth used on a single node over time.

As shown in Fig. 4, we measured that each node in our solution consumes a mean of 4 KB/s once the system has stabilized. Given that the maximum consumption never exceeds 8 KB/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 5 shows that the consumption peak in the client/server architecture reaches 67 MB/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.

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 4 KB/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 Fig. 6).

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.

4.4 CPU load

To estimate the CPU overhead introduced by our refereeing system, we assigned 1 "*cpu load point*" to every

Table 1 CPU overheads

Number of referees	One (%)	Three (%)	Five (%)
CPU overhead	27.85	64.59	210.56

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.

We computed the values of the CPU overhead, and display the results in Table 1. 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 require a dual CPU core, a 100 % CPU overhead on a quad-core processor seems an acceptable maximum. Once again, we wish to emphasize that our CPU load measurements are vastly overestimated.

We also conducted a quick CPU usage check on personal machines of several games representative of our solution targets. Experiments show that these games use roughly 50 % of the quad-cores and nearly fully the dual-cores. Multiplayer online battle arenas such as *League of Legends*, *Defense of the Ancients 2*, and *Bloodline Champions* all ran perfectly when bound to only one or two cores. MMORPGs like *Tera online* and *Rift* ran also without issues when placed on 2 cores only. The more recent *Guild Wars 2* showed some different frame rates when bound to 2 cores only, as Extreme graphics options can produce a bottleneck effect by increasing the quantity of information the CPU sends to the GPU.

4.5 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.

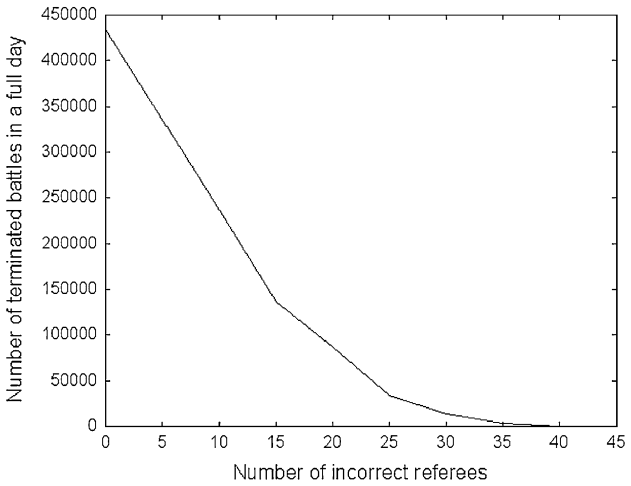

Fig. 7 Number of possible battles as the proportion of correct referees decreases

Table 2 Undetected cheat ratio

Number of referees	One (%)	Three (%)	Five (%)
Cheat percentage	1.17	0.0128	0.0099

In the literature about reputation systems [11], 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.

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 7 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 2 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 more referees per battle. In our opinion, this shows that our solution is effective for detecting and avoiding both incorrect referees and incorrect players.

Table 3 CPU overhead relative to undetected cheat

Number of referees	One (%)	Three (%)	Five (%)
CPU overhead for relative 1 % cheat undetected	32.6625	0.83	2.10

In Table 3, we correlate the measures from Tables 1 and 2 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.

5 Related work

There has been significant research on P2P overlays for online games. Several approaches, such as Colyseus [16], BlueBanana [17], and Solipsis [5], 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 [15]. 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 [4, 18, 19], and [20] propose distributed cheat detection mechanisms. In [4] and [18] the authors use trusted entities to handle security, and in [20] super-peers are used as proxies for the overall security, while in [19] zones are created where every player assesses other player 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 [18]. 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 [20]. Such hybrid approaches also need to identify trusted nodes in the network 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, 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.

6 The appeal of decentralized architectures for online games

Scalability is a crucial issue for online games; 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.

6.1 A critical assessment of C/S gaming architectures

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, the current total number of registered *World of Warcraft* (WoW) players reaches beyond 11,000,000 and unofficial sources estimate the maximum number of avatars per server to be around 60,000 [14]. The actual number of players logged simultaneously is nowhere near this limit: it averages around 1,500 per server [13]. This is tiny compared to the total number of WoW players logged concurrently worldwide, which a rough calculation using the above estimates evaluates around 275,000. 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.

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 [21]) 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.

The experimental results detailed in Sect. 4 show that our fully decentralized approach reaches decisions that are significantly trustworthy without impeding the scalability of the system.

6.2 Of clouds and games

Cloud deployment is a step towards decentralization and a growing trend within the digital games industry [22, 23]. The work presented in this paper focuses on a fully decentralized approach over P2P overlays, yet it also has potential as an extension to cloud deployment. The present subsection discusses the complementarity of our solution with *cloud gaming*.

Cloud gaming extends the traditional C/S cluster-based approach and allows dynamic provisioning of server resources. In addition, it can help alleviate the computing load on the client side by running all computations on cloud servers and streaming the resulting video feed [24]. This allows users to access large collections of games in exchange for a monthly fee, and to play higher quality games on small computing power devices such as smartphones and tablets.

We identify two main limitations to the *cloud gaming* model. Firstly, dynamic resource provisioning is extremely complex [25] and can be very costly. Secondly, even current high-end clouds such as Amazon's EC2 cannot provide support for cloud gaming to 25 % of their total end-user population because of latency issues [26]. In 2012 the combination of these issues almost lead to the bankruptcy of OnLive [27], one of the major cloud gaming pioneers.

Merging our P2P approach with cloud deployment might mitigate these limitations. Exploiting freely available resources on peers increases the cost-efficiency of a hybrid architecture and eliminates the negative impact of overprovisioning. The delegation of computation loads to CPU-potent peers on the same LAN as CPU-deficient devices contributes to solving the latency issues.

Conversely, adding cloud support to our solution would definitely improve its performance and increase its reliability further. Cloud servers can constitute a core set of efficient and trusted referees to turn to in adverse situations: when there are no reliable peer referees at hand, or when a consensus among peer referees takes too much time. Cloud servers are also most suited for handling persistent game data at the end of a battle. Cloud support remains a perspective in the context of this paper, as it would raise the cost of our solution and would still require complex consensus algorithms to ensure the consistency of the game data.

7 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 may scale, but prohibit the implementation of a refereeing authority that is totally trustworthy.

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.

We ran simulations to test whether our solution is viable when thousands of nodes are involved. The simulation results included in this paper show 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 7 KB/s. This can be compared to the unrealistic 40 MB/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, such as when players perform legal actions but still behave maliciously within the game. For instance, *goldfarming* consists in gathering virtual resources 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 log player actions and referee decisions to spot abnormal behaviors linked to stolen accounts and goldfarming. In addition this improvement could allow retroactive removal and replacement of bad referees.

References

1. Miller, J., Crowcroft, J.: 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, (2010)
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 (2012)
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. Hampel, T., Bopp, T., 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, ser. NetGames '06. ACM, New York (2006)
5. Frey, D., Royan, J., Piegay, R., Kermarrec, A.-M., Anceaume, E., Le Fessant, F.: Solipsis: A decentralized architecture for virtual environments. In: 1st International Workshop on Massively Multiuser Virtual Environments, Reno, NV, États-Unis (2008)
6. Bharambe, A., Douceur, J.R., Lorch, J.R., Moscibroda, T., Pang, J., Seshan, S., Zhuang, X.: Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In: Proceedings of the ACM SIGCOMM 2008 conference on Data communication, ser. SIGCOMM '08, pp. 389–400. ACM, New York (2008)
7. Yan, J., Randell, B.: 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, pp. 1–9. ACM, New York. [Online]. Available: http://doi.acm.org/10.1145/1103599.1103606 (2005)
8. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. **43**(2), 618–644 (2007)
9. Rahbar, A., Yang, O.: PowerTrust: a robust and scalable reputation system for trusted peer-to-peer computing. Parallel Distrib. Syst. IEEE Trans. **18**(4), 460–473 (2007)
10. Rosas, E., Marin, O., Bonnaire, X.: CORPS: building a community of reputable PeerS in distributed hash tables. Comput. J. **54**(10), 1721–1735 (2011)
11. Srivatsa, M., Xiong, L., Liu, L.: TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks. In: Proceedings of the 14th international conference on World Wide Web, ser. WWW '05, pp. 422–431. ACM, New York (2005)
12. Montresor A., Jelasity, M.: PeerSim: a scalable P2P simulator. In: Proceedings of the 9th International Conference on Peer-to-Peer (P2P'09), Seattle, WA, Sep 2009, pp. 99–100 (2009)
13. World Of Warcraft Servers average concurrent players. [Online]. Available: http://www.warcraftrealms.com/activity.php?serverid=-1 (2014)
14. World Of Warcraft Servers player capacity. [Online]. Available: http://www.warcraftrealms.com/realmstats.php?sort=Overall (2014)
15. Chen, K.-T., Huang, P., Huang, C.-Y., 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, ser. NOSSDAV '05, pp. 19–24. ACM, New York (2005)
16. Bharambe, A., Pang, J., Seshan, S.: Colyseus: a distributed architecture for online multiplayer games. In: Proceedings of the 3rd conference on Networked Systems Design and Implementation, Vol. 3, ser. NSDI'06, pp. 12–12. USENIX Association, Berkeley (2006)
17. Legtchenko, S., Monnet, S., Thomas, G.: 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)
18. Webb, S.D., Soh, S., Lau, W.: RACS: a referee anti-cheat scheme for P2P gaming. In: Proceedings of the International Conference on Network and Operating System Support for Digital Audio and Video (NOSSDAV'07) (2007)
19. Hampel, T., Bopp, T., 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, ser. NetGames '06. ACM, New York. [Online]. Available: http://doi.acm.org/10.1145/1230040.1230058 (2006)
20. Goodman, J., 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, ser. NetGames '08. ACM, New York (2008)
21. Specht, S.M., Lee, R.B.: Distributed denial of service: taxonomies of attacks, tools and countermeasures. In Proceedings of the International Workshop on Security in Parallel and Distributed Systems, pp. 543–550 (2004)
22. Onlive, The leader in cloud gaming. [Online]. Available: http://www.onlive.com/ (2014)
23. Nvidia Grid, cloud gaming. [Online]. Available: http://blogs.nvidia.com/blog/2013/08/30/best-in-show/ (2014)
24. Shea, R., Liu, J., Ngai, E.-H., Cui, Y.: Cloud gaming: architecture and performance, Netw. IEEE **27**(4), (2013)
25. Ricci, L., Carlini, E.: Distributed virtual environments: from client server to cloud and p2p architectures. In: High Performance Computing and Simulation (HPCS), 2012 International Conference on, pp. 8–17 (2012)
26. Choy, S., Wong, B., Simon, G., Rosenberg, C.: The brewing storm in cloud gaming: a measurement study on cloud to end-user latency. In: Proceedings of the 11th Annual Workshop on Network and Systems Support for Games, ser. NetGames '12, pp. 2:1–2:6. IEEE Press, Piscataway (2012)
27. Onlive goes bankrupt. [Online]. Available: http://www.theverge.com/2012/8/28/3274739/onlive-report (2014)