

AntiCheat: Cheat Detection and Prevention in P2P MOGs

McGill School of Computer Science Technical Report (SOCS-TR)

Kévin Huguenin, Amir Yahyavi, and Bettina Kemme
School of Computer Science, McGill University, Montréal, QC, Canada
{huguenin,yahyavi,kemme}@cs.mcgill.ca

August 2011

Abstract

In peer-to-peer games, cheaters can easily disrupt the game state computation and dissemination, perform illegal actions and unduly gain access to sensitive information. We propose AntiCheat – a cheat detection and prevention protocol following a mutual verification approach complemented with information exposure mitigation. It is based on a randomized dynamic proxy scheme for both the dissemination and verification of actions and further reduces the information exposed to players close to the minimum required to render the game.

We build a proof-of-concept prototype on top of Quake III. Experimentations with up to 48 players show that opportunities to cheat can be significantly reduced, even in the presence of colluding cheaters, while keeping good performance.

1 Introduction

Scalability and fairness are keys to the success of massively multi-player on-line games. The first is necessary for the technical viability of the game while the latter is required for the wide adoption by players. Distribution techniques such as the peer-to-peer paradigm, which delegates tasks to clients, increase scalability but increase the danger of cheating: peer-to-peer games are typically more prone to cheating as players have access to and can manipulate sensitive game data [5].

Cheating essentially consists in gaining an unfair advantage and comes mostly in the following three forms: disrupting the game state computation and dissemination, performing illegal actions, and gaining access to sensitive information [17, 18]. In centralized games, cheat detection and prevention are achieved by making the server verify the players' actions, ensure synchronization and reduce the information sent to players to the minimal amount required to render the game world [1]. In decentralized games however, detecting cheating is more difficult and natural trade-offs between responsiveness, scalability, verification and information disclosure appear together with the issues of trust and collusion.

Common decentralized approaches to deal with cheating include mutual verification and auditing, agreement protocols (e.g., lockstep [1]), and position-based information filtering. However, most of the proposed approaches have one of the following drawbacks: they (1) rely on a central server or trusted third parties [15]; (2) do not deal with collusion; (3) detect cheaters *a posteriori* [3]; (4) fail to provide responsiveness and scalability [1].

In order to develop an approach that avoids these pitfalls we analyzed the cheating opportunities created by mechanisms commonly used in several types of peer-to-peer multi-player games, with a

special focus on Donnybrook [2], a distributed version of Quake III. From there, we propose AntiCheat which uses the following techniques to prevent and detect cheating:

- vision-based filtering and indirect communication are used to reduce the information available at each player close to the minimum amount necessary to render the game world;
- players are assigned a proxy player in charge of update dissemination and action verification. Proxies are chosen at random and dynamically renewed to limit the impact of collusion while allowing on-the-fly mid-term verifications.
- verifications may lead to the emission of blames which can be used to directly take actions against cheaters.

We report on our experimentations in a 48-player Quake III game showing that opportunities to cheat are significantly reduced, even in the presence of collusion, while keeping good performance with respect to scalability and responsiveness.

Although the paper focuses on first person shooter games, we believe that the concepts and mechanisms involved in AntiCheat can be applied to a broad range of games including role playing games and real-time strategy games.

2 Background

This section gives some background on Multi-Player On-line Games (MOG) with a special focus on First-Person Shooter (FPS) and Role Playing Games (RPG), and lists the traditional techniques used to achieve scalability in MOGs.

2.1 Multi-player on-line games

In multi-player on-line games, players experience the action through a character they control, referred to as their *avatar*. Avatars evolve in a virtual space called the *game world* and can interact with objects and other avatars be they controlled by players or by artificial intelligence. In FPS games, players usually see the game world through the eyes of their avatars while in RPGs they do from above, centered on their avatar.

The design and implementation of MOGs rely on the notion of state which captures the characteristics of an avatar at a certain point of time. The state of an avatar typically includes its position, aim, objects it owns, health, *etc.* and is modified by the instructions of the player (e.g., move or shoot) and the interactions with other avatars or objects (e.g., collision).

To visualize the game world on the screen (from the perspective of its avatar), a player needs at least (partial) *replicas* of other avatars and game objects that are in its own avatar's vision field. If the state of an avatar changes, update messages must be sent to those players that need this information. Games usually run in a discrete event-loop, meaning that in each *frame* the states of the entities are updated and updates may be sent. State updates account for the largest part of the bandwidth needs of MOGs. The content of the partial replicas and the update frequency are crucial in the design of MOGs.

2.2 Peer-to-peer MOGs

In a centralized game, all entities in the game world are owned by the server and players send updates to the server. The server verifies each update and sends it, given its global knowledge, to only those players who need it.

In a peer-to-peer system, all entities are owned by clients: avatars are owned by players, while bots and game items are either distributed across the players or are controlled by the company orchestrating the game. Players are usually aware of all entities of the game. This is the case with many FPS (e.g., Quake III). In large-scale RPGs (e.g., World of Warcraft) it can be achieved by dividing the game world into zones (e.g., [19]).

A simple peer-to-peer game lets each player send all changes directly to all other players. However, since players have limited bandwidth, peer-to-peer MOGs use a collection of techniques to increase scalability that we describe in here.

Dead reckoning consists in predicting the state of an avatar based on previous observations thus allowing to reduce the frequency of position updates while keeping the display smooth. This often requires additional information: for instance the keys the player is pressing and the avatars it is chasing are useful to predict the future position and aim of its avatar. Donnybrook uses dead reckoning for avatars that are of less interest to a player. In this case, a player does not send frequent position updates but infrequent guidance messages containing the avatar's expected next position and aim (computed locally) and its current position, aim, rate of fire, *etc.*

Area of interest (AOI) filtering consists in limiting the updates a player receives (e.g., to those concerning entities inside a fixed-radius sphere around his avatar). Because using a fixed geographical area as AOI does not bound the actual number of updates a player receives, Donnybrook instead uses the set of the top-5 avatars with respect to an attention metric based on proximity, aim and interaction recency, called interest set (IS). A player receives frequent updates only about avatars in his IS and guidance messages about *all* other avatars.

Publish-subscribe is generally used to implement selective update mechanisms. Each player subscribes to the players whose avatars are of interest and receives updates from them. The subscription type (e.g., IS subscription) depends on game-related information such as relative positions and determines the frequency and the content of updates. Updates are disseminated either directly by the player to all his subscribers or by more sophisticated schemes such as multicast trees.

3 Cheating

A large amount of research has been devoted to classifying different types of cheating in peer-to-peer MOGs (e.g., [17, 18]). In this paper we focus on cheating that is driven by a game-related goal, that is those that give an unfair advantage to the cheater helping him win the game, but not those that aim to disrupt the game functions (e.g., denial of service). Common to most classifications are the following categories of cheating: disrupting the information dissemination, performing illegal actions, and exploiting the sensitive information disclosed.

To better understand how cheating can materialize under a typical P2P game architecture, we consider the typical *deathmatch* scenario in a FPS game such as Quake III. Killing other players increases a player's score, and the player with the highest score wins the game. The goal of a player is therefore easy to express: kill other players' avatars and avoid being killed. This being said, we can now look at the different cheats that help achieve this goal in the three categories listed above.

Information dissemination First, by delaying or dropping updates, or sending wrong or inaccurate information (in position update or for dead reckoning), a player can blind or confuse other players about his current state, thus decreasing the chances of his avatar being shot. A player can also implement a *look-ahead* cheat [1], i.e., deliberately delaying the updates he sends to base his actions on those he receives from others.

Illegal actions Second, a player can circumvent the game physical laws (e.g., limited velocity) and unduly change its state (e.g., increase its health and ammunitions) by tampering with the game code. A cheater can also unduly increase its score: in distributed FPS games, including Donnybrook, the players' scores are generally updated by having the players claim they have killed some other player, which constitutes a trivial opportunity to cheat, without proper verifications.

Information disclosure Finally, a player can exploit information available but not supposed to be disclosed (e.g., position of players behind walls) to increase his chances to kill other players and foresee danger, thus helping him evade. Since players may receive state updates about all players and manage subscriptions to their avatars, they are aware of avatars chasing them or aiming at them even for those behind their backs. In addition, the expected future position contained in dead reckoning messages can be exploited by means of aim aides to increase shooting accuracy. One has to be aware that cheaters have access to information their colluding partners hold.

4 The AntiCheat algorithm

AntiCheat prevents or detect all the aforementioned cheating categories. The goal is to minimize the potential for illegal actions, the dissemination of wrong or delayed information, and the dissemination of sensitive information to players who do not need it. AntiCheat achieves this through mutual verification and indirect communication (see Fig. 2), and vision-based information filtering (see Fig. 1). These key mechanisms are facilitated by an underlying dynamic architecture which periodically assigns to each player P a *proxy*, i.e., another player chosen using a verifiable pseudo-random number generator, in charge of managing P's subscriptions and verifying P's actions and the updates and subscriptions P sends. A crucial aspect of the architecture is that proxies are dynamically renewed to limit both the opportunities to collude and the impact of having a malicious proxy, while allowing mid-term follow-up. Because we do not fully guarantee that each cheat will be detected or that all cheat suspicions are actually correct, we use the concept of blames: verification procedures may lead to the emission of blames that are filtered out to prevent wrongful blames caused by cheaters and network conditions. Blames are eventually used to take action against cheaters or as input by a reputation system.

Our solutions assumes that the game runs in a discrete event-loop, dividing time in fixed-length frames, and clients are assumed to be synchronized (i.e., the time drift is negligible when compared to the frame length).

4.1 Proxy architecture

In a given frame, a player has a single designated proxy, responsible for managing subscriptions and forwarding message, and for the verification of the player's actions. As such, the proxy has tasks similar to a server.

The proxy selection process needs to be random and verifiable in order to prevent collusion from happening. To achieve this, AntiCheat makes use of pseudo-random number generators, initialized with the player's id plus a seed common to all players and fixed for the whole gaming session. Each

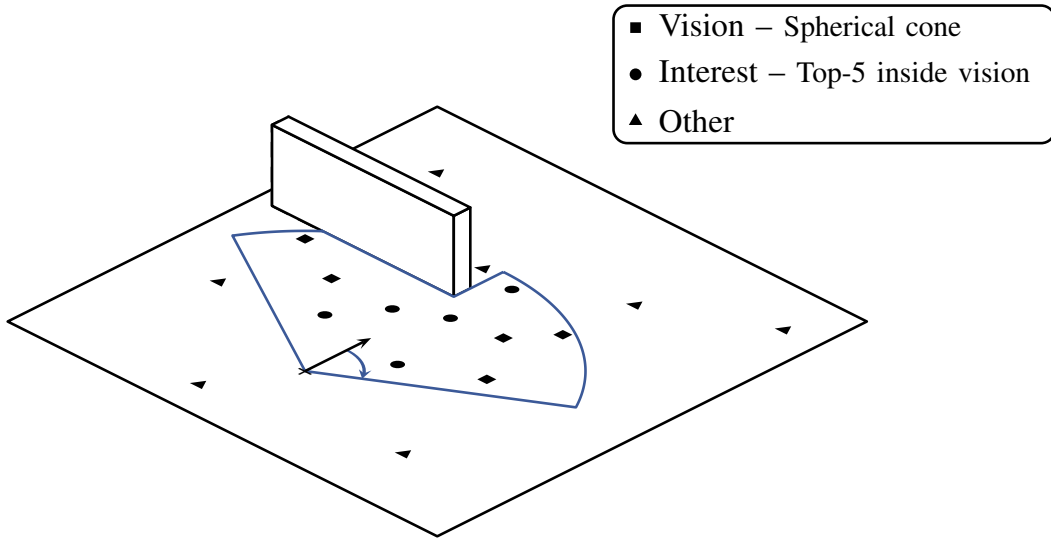


Figure 1: Subscription-types and corresponding areas: Vision set (VS) is composed of avatars inside a fixed-radius and angle spherical cone directed along the player’s aim; Interest set (IS) is composed of the 5 avatars inside VS which catch the player’s attention the most.

player initializes and maintains a pseudo-random number generator for each player, including himself, and therefore can determine both its own proxy and the other players’ proxies, in any frame, without the need for communication. Moreover, there is no need for explicit subscription by the players to their proxies as both parties are aware of their assignment to each other.

Proxies are periodically renewed. This has several reasons. First, it limits the impact of collusion between a player and its proxy (recall that proxies perform verifications), and the effects of a malicious proxy (who might, e.g., delay forwarding). Second, one has to be aware that a proxy will receive frequent state updates from the player regardless of vision-based filtering. As far as the game rendering is concerned, this is a source of unnecessary information disclosure. Proxy renewal offers a time limit on such disclosure. Therefore, each fixed number of frames, the players determine their new proxy and the other players’ new proxies for the next period, making use of the pseudo-random number generators. The duration of the period is an important parameter of the system: higher values allow long-term follow-up and thus efficient verifications but may significantly disrupt the game in case of collusion or malicious proxies while lower values may be more subject to inconsistencies because of communication delay. The choice of the proxy renewal period is game dependent. For FPS games, renewing proxies every two seconds allows mid-term follow-up while tolerating typical communication delay.

Using pseudo-random number generators offers a simple and verifiable way to choose proxies. Such a selection process can be refined to take into account resource heterogeneity (i.e., some players may be able to act as a proxy for several players while others may be able to handle only a few because of CPU or bandwidth limitations), or to minimize the delay and the probability of a player being assigned a colluding proxy [16].

4.2 Mitigating unnecessary information exposure

We have identified the main sources of information one can exploit for cheating as: (1) information about players not in the avatar's vision field, (2) additional information sent for dead reckoning, and (3) the subscription information. To prevent players from cheating one must restrict the information received by players to the minimum amount required to render the game world, from the standpoint of its avatar, with an acceptable level of consistency and responsiveness.

In AntiCheat, the cheating opportunities coming from the first two sources of information are addressed by means of different subscription types. The subscription type is based on, among other things, the relative positions and aims of the avatars and determines both the information of the avatar's state included in updates and the update frequency. AntiCheat uses the notion of interest set, required to ensure both scalability and responsiveness, and introduces the notion of vision field inspired from (and therefore compatible with) the concept of area of interest filtering. There are three different subscription types:

- **Vision set (VS):** it contains the avatars the player can see. In FPS games, it corresponds to a spherical cone, defined by a given radius and angle (e.g., 50 meters and ± 60 degrees) and centered on the avatar. In 2D RPGs, seen from above, the vision field is a fixed-radius disc around the avatar. Vision set can be wider than the actual vision field to cope with rapid rotations and movements of the viewer. Beyond the avatar's vision field, the vision set takes into account the features of the game world, known by all players. For instance, the avatars that are in a player's vision range but behind a wall should not appear in its vision set. For VS subscription, players receive infrequent (i.e., typically every second) dead reckoning messages containing extra information to simulate the avatar's near-future actions.
- **Interest set (IS):** it is composed of avatars that catch the player's attention the most (i.e., a combination of the proximity, aim, and interaction recency). The size of the interest set is fixed (i.e., typically 5). Only avatars in a player's vision set are considered as candidates for its interest set, thus preventing the player to obtain frequent and accurate information about avatars he cannot see. For this type of subscription, players receive frequent (i.e., typically every 50 ms) state updates including the avatars positions, aims, ammunitions, weapons, health, *etc.* Avatars in a player's interest set are automatically removed from its vision set to prevent him from receiving information about future actions that he could exploit to cheat. This is of the utmost importance as avatars in IS are the ones that the player is most likely to interact with in a near-future.
- **Others:** While players do not actually need information about the players outside their vision sets for rendering the game world, they still need a minimum amount of information to determine whether a given player does or does not actually appear in their vision sets. To allow players to decide on the type of subscription they need, they receive infrequent (i.e., typically every second) partial state updates containing only the position of the avatars, sufficient to determine the subscription type. Note that proxies also need this information to assess the validity of the subscriptions of the players they are in charge of. This subscription type is the default one and thus does not require explicit subscription: unless a player explicitly does an IS or VS-subscription, he receives only infrequent position update.

Figure 1 depicts the three different types of subscriptions and the corresponding areas in the game world: the vision set is the spherical cone centered on the avatar, directed along its aim and with angle

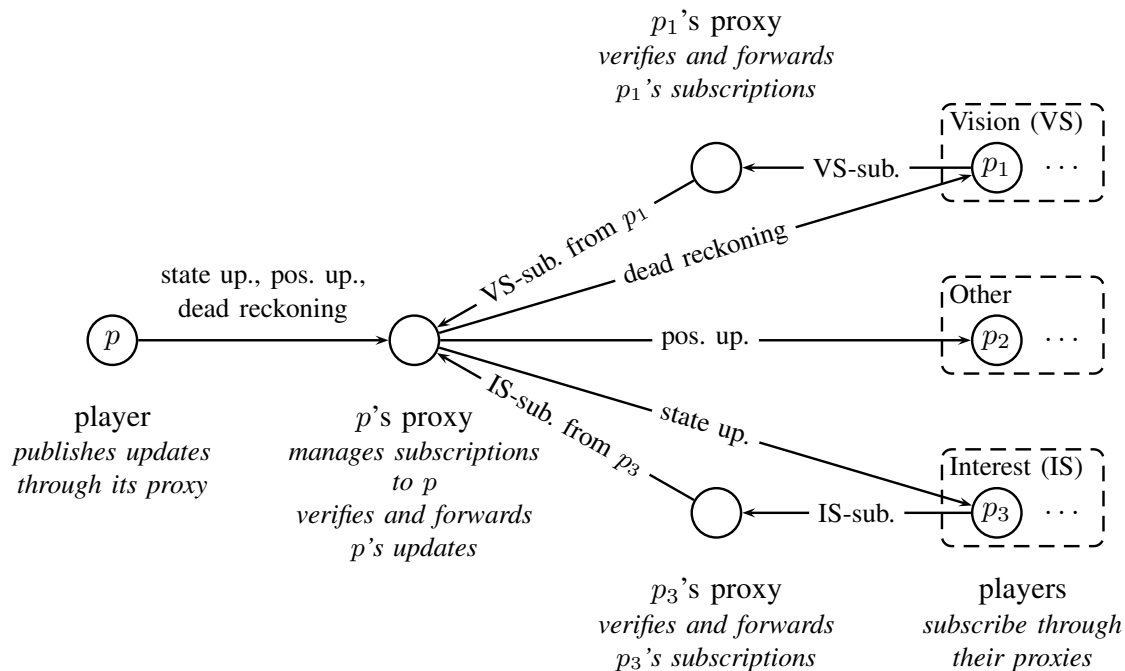


Figure 2: Proxies (1) manage incoming subscriptions and (2) verify and forward outgoing subscriptions and updates. Players p_1 and p_3 respectively IS and VS-subscribe to p by sending a subscription message to p 's proxy through their own proxy. By default, player p_2 receives infrequent position updates.

± 60 degrees, cut by a wall; amongst avatars in the vision cone, 5 are supposed to attract the player's attention the most and therefore constitutes its interest set.

To make the multi-type subscription management cheat-proof and hide from players the subscription from other players, subscriptions are handled *by* proxies and *through* proxies (see Fig. 2). A player's proxy relays (and verifies) the subscriptions the player does. Subscriptions to a player are sent to its proxy, which maintains the list of subscribers on the player's behalf. All updates are then published by players through their proxies: the player sends the three types of updates to its proxy which then dispatches them accordingly.

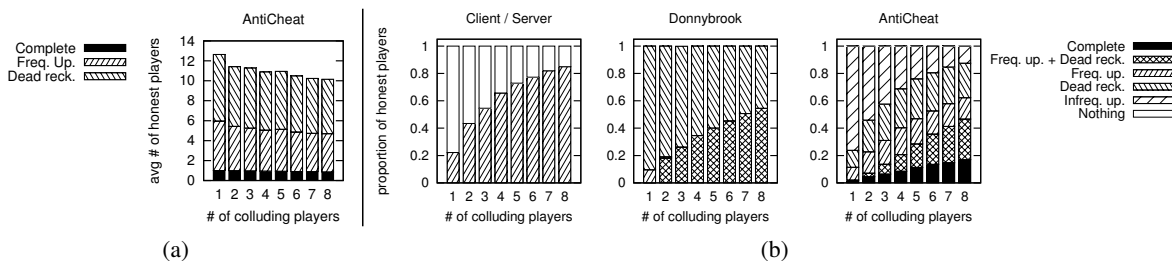


Figure 3: Experimental results using a 48-player trace from a Quake III game in the `q3dm17` map.

4.3 Verifications, blames, and punishment

In our verification scheme, all players verify each others' actions, with a special attention from proxies to the players they are in charge of. Detected misbehaviours lead to the emission of blames that may in turn lead to punishment (e.g., ban) or tarnish the concerned player's reputation. The verifications performed by the players are the following:

- **Subscriptions** are done through proxies which provide them with the opportunity to assess the validity of their types (i.e., IS or VS). Proxies verify whether the target of a VS-subscription is indeed in the player's vision spherical cone. They have sufficient information to perform such verifications, namely the subscriber's up-to-date position and aim (recall that the proxy receives frequent state updates from the players it is in charge of) and the target's position (recall that all players receive at least position updates from all other players). Note that since this latter information may be slightly outdated, verifications must be relaxed to prevent false positives. Regarding IS-subscriptions, proxies can compute interaction recency with sufficient accuracy because the proxy renewal period is set to 2 seconds and proxy receive handoff messages (see below) while in Donnybrook for instance recency decreases exponentially over time with a cut-off time of 3 seconds.
- **State updates** are published through the proxies which verify that players' actions follow games physics (e.g., gravity, limited velocity and angular speed) and game rules (e.g., decreasing health after a fall) by comparing successive state updates. Proxies also verify, *a posteriori*, that actual movements are consistent with predictions included in dead reckoning messages. Additionally to the proxy, all players can verify another player's actions but with relative accuracy depending on the information they have. For instance one can more accurately verify the actions of players in its IS. Proxies also check that players do send timely updates and other players verify that proxies forward them.
- **Interactions** such as hit and kill-claims can be verified by proxies as well as by players acting as witnesses.
- **Handoff** is performed between a player's successive proxies to allow longer-term follow-up: before a player's proxy is renewed, it sends a copy of the last state update it received from the player to the player's next proxy, i.e., its successor. To limit the impact of player-proxy collusion, a proxy also embeds the state update it received from its predecessor.

To prevent players from tampering with the messages they forward – namely updates, subscriptions and handoff messages – AntiCheat uses lightweight (i.e., ~ 100 bits while state update messages are 700 bits on average) digital signatures [4].

In AntiCheat, misbehaviors detected upon verifications are reported under the form of blames of different levels (typically coded by an integer) reflecting the gravity of the cheat. Blames can be collected and processed by a central entity (e.g., the servers of the company orchestrating the game which already take care of authentication and therefore constitute first class candidates for punishment), since the generated traffic is low, or achieved in a decentralized fashion using a collusion-resistant monitoring architecture a la AVMON [13].

An important aspect of blames, especially when they are emitted by untrusted players in an error-prone environment, is that they may be wrong. Wrongful blames occur for two reasons: cheaters blaming other players on purpose or due to message loss and communication delay. A wide range of

techniques [7, 9, 10] have been proposed to solve the first problem. These techniques perform specially well in dealing with collusion when extra information such as acquaintance and playing history are known, which is the case in MOGs [14]. The second problem can be addressed by pursuing an off-line statistical analysis of the distribution of wrongful blames and mitigate their impact accordingly: assuming independent and identically distributed message loss, one can compute the probability density function of wrongful blames (e.g., a Binomial distribution for i.i.d. Bernoulli loss) caused by them and calculate a tolerance threshold which balances the detection and false positive rates. Such an approach has been successfully used for free-rider detection in [8].

5 Evaluation

The implementation used for the evaluation consists of two parts: (1) a set management component, integrated into Quake III, which determines in each frame the key sets (i.e., interest, vision and other as defined in the previous section) of each avatar and writes them in a log file, and (2) a replay engine which manages the proxy infrastructure and implements signatures, verifications, subscriptions, and publications over a network, using as input the logs generated.

We performed our preliminary evaluation of AntiCheat with respect to the following aspects in the presence of collusion:

- **Information disclosure:** players can obtain several types of information about other players: complete information (i.e., proxies about the players they are in charge of), frequent state update (about avatars in the interest set), dead reckoning messages (about avatars in the vision set) and infrequent position update. The first type is the most informative, the second and third complement each other and are not directly comparable, and the fourth is the least informative. We measured the *joint* information obtained by a coalition of colluding cheaters about other players using a 48-player trace from a Quake III game in the `q3dm17` map – a worst case as it does not contain any walls. We considered three infrastructures: Client-Server where players receive frequent updates about avatars in their vision set and nothing about the rest; Donnybrook where players receive frequent updates for players in their interest set and dead reckoning message for others; AntiCheat as described in Section 4.

We compiled the results under the forms of stacked histograms, shown in Figure 3b, as follows. Consider a coalition of two players for a game with eight players: player 1 with IS $\{4, 5\}$, VS $\{2, 6\}$ and other $\{3, 7, 8\}$, who acts as a proxy for player $\{3\}$; player 2 with IS $\{1, 6\}$, VS $\{7\}$ and other $\{3, 4, 5, 8\}$, who acts as a proxy for player $\{1\}$. The coalition therefore has: complete information about 1 honest player ($\{3\}$), frequent update **and** dead reckoning for 1 honest player ($\{6\}$), frequent update alone for 2 honest players ($\{4, 5\}$), dead reckoning alone for 1 honest player ($\{7\}$) and infrequent update alone for 1 honest player ($\{8\}$). It can be observed in Figure 3b that despite the information leakage caused by proxies, AntiCheat significantly reduces the information disclosed when compared to Donnybrook. In AntiCheat, a coalition of four cheaters has minimum information (i.e., infrequent position update alone) about 31% of the honest players and partial information (i.e., dead reckoning **or** frequent state update) about 48% of them. In Donnybrook, the same coalition has dead reckoning information alone about 65% of the honest players and precise information (i.e., dead reckoning **and** frequent state update) about the rest. Note that since in Donnybrook players receive dead reckoning messages about all the players not in their interest set, the proportion of honest players about whom a coalition has frequent state updates alone is very low ($<1\%$). The Client-Server case gives the minimum necessary information and thus serves as a baseline.

- **Effectiveness of verifications** In AntiCheat, verifications rely on proxies and witnesses (i.e., players having sufficient information to assess the validity of actions). To evaluate their potential effectiveness, we measure, for a given cheater, the average number of honest players that: act as proxy for him, have him in their IS, or have him in their VS. The results, shown in Figure 3a, foresee a great potential for verifications: even when he colludes with three other cheaters, a cheater is assigned an honest proxy in 93% of the cases and 10 players witness his actions (4 through frequent state updates and 6 through dead reckoning messages).
- **Responsiveness** The use of proxies increases delays, from 0.5 RTT to 1 RTT in theory, which was confirmed by our experimentations on a LAN and our preliminary tests on PlanetLab. Note however that other games also made use of relays anyway to cope with resource heterogeneity, e.g., forwarder pool in Donnybrook. Another source of delay is when an avatar, neither in IS nor in VS, enters IS. Indeed, the movement of the avatar should be displayed smoothly while only a possibly outdated position update is available. This phenomenon is however rather infrequent: in a frame, on average 88% of the players in IS were already in IS in the previous frame, 8.5% were in VS and only 3.5% would suffer from a slight delay. Widening the vision range (± 60 degrees in our experimentations) increases responsiveness and the effectiveness of verifications but also information disclosure.
- **Scalability** While the proxy architecture may incur some computational and bandwidth overhead, it does not hurt scalability since, due to its decentralized nature, a player is in charge of only one other player on average, regardless of the total number of players.

6 Related work

A range of cheat detection approaches for peer-to-peer games, including mutual verification and log auditing, are described in [9]. Following these guidelines, AntiCheat uses mutual verification but only short-term auditing for improved responsiveness. It also mitigates information disclosure.

RACS [15] relies on trusted referees who simulate the game world locally to assess the validity of the players' action and resolve conflicts while limiting information exposure. In [16], a referee selection algorithm optimizing both responsiveness and the probability of player-referee collusion is proposed for the case where (untrusted) players are used as referees.

In [3], information exposure in RTS games is mitigated as follows: players publish in clear-text their moves occurring in the other player's vision range; otherwise they send only its hash, that can be *a posteriori* verified. The decision is based on a low-resolution map of other players' vision range thus limiting the information on the position of their units. Other protocols [1, 12] made use of commitment schemes to prevent timing and lookahead attacks, at the price of responsiveness.

Based on the fact that knowing additional information impacts the way players behave (e.g., a player obtaining information about players behind walls is likely to stare at walls), statistical cheat detection approaches have been proposed [11]. Another singular approach to cheater detection [6] is to deliberately delay messages and analyze other players reaction.

IRS [7] considers a scenario where a server delegates computational tasks, including sporadic audit of other players, to one or several players named proxies. By cross-checking the results provided by different players and assessing trust based on players' recent computations, IRS manages to efficiently detect cheaters. Beside being partially centralized, IRS does not address unnecessary information exposure.

7 Acknowledgements

Kévin Huguenin was partially funded by a scholarship offered by University of Rennes I and an Explorateur grant offered by INRIA. Amir Yahyavi was funded by NSERC Strategic Grant STPGP/350626-2007.

8 Conclusion

This paper proposed AntiCheat, a decentralized protocol that detects and prevents cheating through mutual verifications and vision-based filtering facilitated by a dynamic proxy scheme. Preliminary experimentations on Quake III foresee a great potential for cheat detection and demonstrate a significant reduction of the unnecessary information disclosed while at the same time preserving responsiveness.

Short-term future work includes the implementation of AntiCheat under the form of a generic middleware for MOGs, that we plan to deploy and evaluate over PlanetLab.

References

- [1] Nathaniel Baughman, Marc Liberatore, and Brian Levine. Cheat-Proof Payout for Centralized and Peer-to-Peer Gaming. *IEEE/ACM ToN*, 15:1–13, 2007.
- [2] Ashwin Barambe, John Douceur, Jacob Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan, and Xinyu Zhuang. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In *SIGCOMM*, 2008.
- [3] Chris Chambers, Wu-Chang Feng, Wu-Chi Feng, and Debanjan Saha. Mitigating Information Exposure to Cheaters in Real-Time Strategy Games. In *NOSSDAV*, 2005.
- [4] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann, and Leif Uhsadel. A Survey of Lightweight-Cryptography Implementations. *IEEE Design & Test*, 24:522–533, 2007.
- [5] Lu Fan, Phil Trinder, and Hamish Taylor. Design Issues for Peer-to-Peer Massively Multiplayer Online Games. *IJAMC*, 4:108–125, 2010.
- [6] Stefano Ferretti and Marco Roccetti. AC/DC: An Algorithm for Cheating Detection by Cheating. In *NOSSDAV*, 2006.
- [7] J. Goodman and C. Verbrugge. A Peer Auditing Scheme for Cheat Elimination in MMOGs. In *NETGAMES*, 2008.
- [8] Rachid Guerraoui, Kvin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. LiFTinG: Lightweight Freerider-Tracking Protocol in Gossip. In *MIDDLEWARE*, 2010.
- [9] Patric Kabus, Wesley Terpstra, Mariano Cilia, and Alejandro Buchmann. Addressing Cheating in Distributed MMOGs. In *NETGAMES*, 2005.
- [10] Sepandar Kamvar, Mario Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.
- [11] Peter Laurens, Richard Paige, Phillip Brooke, and Howard Chivers. A Novel Approach to the Detection of Cheating in Multiplayer Online Games. In *ICECCS*, 2007.

- [12] Shunsuke Mogaki, Masaru Kamada, Tatsuhiro Yonekura, Shusuke Okamoto, Yasuhiro Ohtaki, and Mamun Reaz. Time-Stamp Service Makes Real-Time Gaming Cheat-Free. In *NETGAMES*, 2007.
- [13] Ramsés Morales and Indranil Gupta. AVMON: Optimal and Scalable Discovery of Consistent Availability Monitoring Overlays for Distributed Systems. *IEEE TPDS*, 20:446–459, 2009.
- [14] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient Online Content Voting. In *NSDI*, 2009.
- [15] Steven Webb, Sieteng Soh, and William Lau. RACS: A Referee Anti-Cheat Scheme for P2P Gaming. In *NOSSDAV*, 2007.
- [16] Steven Webb, Sieteng Soh, and Jerry Trahan. Secure Referee Selection for Fair and Responsive Peer-to-Peer Gaming. *Simulation*, 85:608–618, 2009.
- [17] Steven Daniel Webb and Sieteng Soh. Cheating in Networked Computer Games: A Review. In *DIMEA*, 2007.
- [18] Jeff Yan and Brian Randell. A Systematic Classification of Cheating in Online Games. In *NETGAMES*, 2005.
- [19] Anthony (Peiqun) Yu and Son T. Vuong. MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games. In *NOSSDAV*, 2005.