Short paper: Cheat Detection and Prevention in P2P MOGs

Kévin Huguenin, Amir Yahyavi, Bettina Kemme

Cheat Detection and Prevention in P2P MOGs

Kévin Huguenin, Amir Yahyavi, and Bettina Kemme
School of Computer Science, McGill University, Montréal, QC, Canada

I. 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. Although they achieve better scalability, peer-to-peer games are more prone to cheating as players have access to and manipulate sensitive game data [2].

While in centralized games cheat detection and prevention can be achieved by having the server verify the players' actions and reduce the information sent to players to the minimal amount required to render the game world, in decentralized games, it is more difficult as natural trade-offs between responsiveness, scalability, verification and information disclosure appear together with the issues of trust and collusion [3].

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

Based on Donnybrook [5], a distributed version of Quake III, we propose the following techniques to detect and prevent cheating while avoiding the aforementioned pitfalls:

- vision-based filtering and indirect communication to reduce the information available at each player close to the minimum amount necessary to render the game world;
- assign each player 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.

II. Background

Games usually run in a discrete event-loop, meaning that in each *frame* the states of the avatars and objects are updated and updates may be sent. In a centralized game, all objects in the game world are owned by the server and players send updates about their avatars to the server. The server verifies each update and sends it, given its global knowledge, to only those players who need it. In a P2P game, however, each client usually owns his avatar and some of the other objects. A simple P2P game lets each player send all changes directly to all other players. However, because of resource limitation, the following techniques are used:

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. An extended version is available as a Technical Report [1].

- Dead reckoning consists in predicting the state of an avatar, thus allowing to reduce the frequency of position updates while keeping the display smooth. In Donnybrook, *guidance* messages containing the avatar's expected next position and aim (computed locally) and its current position, aim, rate of fire, *etc.* are used by AI to simulate the avatar's action.
- Area of interest filtering consists in limiting the information a player receives about other avatars. Donnybrook distinguishes between the set of the top-5 avatars (with respect to an attention metric based on proximity, aim and interaction recency), called interest set (IS), and other avatars.

In Donnybrook, a player receives frequent updates only about avatars in his IS and guidance messages about *all* other avatars. This is implemented through a publish-subscribe scheme, in which each player manages the subscriptions to his avatar, with two subscriptions types (i.e., IS and others).

The cheating opportunities a player can exploit to help him achieve his goal (i.e, kill other players' avatars and avoid being killed in a typical *deathmatch* scenario) are to: (1) delay, drop or corrupt updates in order to confuse other players; (2) delay the updates he sends to base his actions on those he receives from others; (3) tamper with the game code to circumvent the game physical laws (e.g., limited velocity) and unduly change his state (e.g., increase his health); (4) increase his score by unduly claiming he has killed some other player; (5) exploit information available but not supposed to be disclosed (e.g., position of players not visible, expected position contained in guidance messages, and subscriptions from other players) to increase his chances to kill other players and foresee danger.

III. Cheat detection and Prevention

We propose a cheat detection and prevention approach based on mutual verification, indirect communication, and vision-based information filtering. These key mechanisms are facilitated by a dynamic randomized proxy architecture.

Proxy architecture In any frame, a player has a single designated proxy. To prevent collusion, proxies are chosen in a randomized but verifiable way. In order to limit in time the impact of collusion between a player and his proxy, the effects of a malicious proxy, and unnecessary information disclosed to proxies, proxies are periodically renewed. This is achieved by making each player maintain, for each player including himself, a pseudo-random number generator that he initializes with the player's id. At the beginning of each new period, a player determines both his own and the other players' proxies by picking a player id with the associated generator.

Subscriptions To prevent players from cheating the information received by players from updates is reduced to the

minimum amount required to render the game world, from the standpoint of his avatar. This is achieved by means of different subscription types based on the relative positions and aims of the avatars (Fig. 1). We used three subscriptions types:

- **Interest set (IS):** it contains the top-5 (with respect to an attention metric) avatars about who a player receives frequent state updates (just as in Donnybrooke).
- **Vision set (VS):** it contains the avatars the player can see, i.e., those within a spherical cone, defined by a given radius and angle (e.g., 50 meters and ± 45 degrees), centered on the avatar and not behind obstacles (e.g., walls). For VS subscription, a player receives infrequent guidance messages.

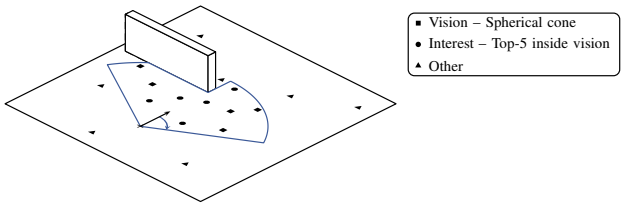

Fig. 1. Subscription-types and corresponding areas.

- **Others:** In order to determine the subscription type he needs, the player receives infrequent position updates about avatars neither in IS nor in VS. This is the default type and, thus it does not require explicit subscription.

To make the multi-type subscription management cheat-proof and hide from players the subscriptions 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 his 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 his proxy which then dispatches them accordingly.

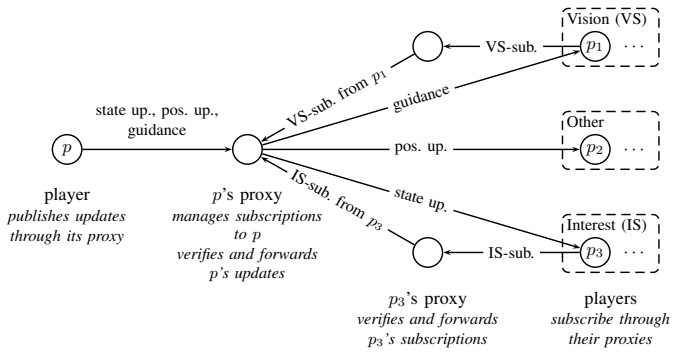

Fig. 2. Role of proxies: manage subscriptions and verify and forward updates.

Verifications In our verification scheme, all players verify each others' actions, with a special attention from proxies to the players they are in charge of:

- **Subscriptions** are done through proxies who assess the validity of their types (e.g., the target of a VS-subscription is indeed in the player's vision field). Proxies have sufficient information to perform such verifications (i.e, frequent position updates about the players they are in charge of and infrequent position update about the target).
- **State updates** are published through the proxies which verify that players' actions follow games physics (e.g., limited velocity) 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 guidance messages. All players can verify another player's actions but with relative accuracy depending on the information they have. Proxies also check that players do send timely updates.
- **Interactions** such as hit and kill-claims can be verified by proxies as well as by players acting as witnesses.

To prevent players from tampering with the messages they forward, updates and subscriptions are signed. Misbehaviors detected upon verifications are reported under the form of blames that eventually lead to sanctions. Ill-founded blames (e.g., purposeful, message loss) are dynamically filtered out.

IV. Evaluation

We built our solution in Quake III and performed our evaluation with 48 players in the `q3dm17` map. We compared our solution against Donnybrook along the following aspects:

- **Mutual verifications** We evaluated, 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 foresee a great potential for verifications: when the cheater colludes with 3 other cheaters, he is assigned an honest proxy in 93% of the cases and 10 players witness his actions (4 through frequent updates and 6 through guidance).
- **Information disclosure:** We observed that, despite the use of proxies, information disclosure is significantly reduced when compared to Donnybrook. For instance, a coalition of three cheaters has minimum information (i.e., infrequent position update alone) for 31% of the honest players while in Donnybrook, it has guidance message about 99% of them.
- **Responsiveness** The use of proxies increases the delays because of forwarding. Another source of delay is when an avatar enters in a player's IS or VS while only a possibly outdated position update is available. This phenomenon is however 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.

References

[1] K. Huguenin, A. Yahyavi, and B. Kemme, "Cheat Detection and Prevention in P2P MMOGs," McGill, Tech. Rep. SOCS-TR-2011.5, Aug 2011.

[2] P. Kabus, W. Terpstra, M. Cilia, and A. Buchmann, "Addressing Cheating in Distributed MMOGs," in *NETGAMES*, 2005.

[3] N. Baughman, M. Liberatore, and B. Levine, "Cheat-Proof Playout for Centralized and Peer-to-Peer Gaming," *IEEE/ACM ToN*, vol. 15, pp. 1–13, 2007.

[4] S. Webb, S. Soh, and J. Trahan, "Secure Referee Selection for Fair and Responsive Peer-to-Peer Gaming," *Simulation*, vol. 85, pp. 608–618, 2009.

[5] A. Bharambe, J. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games," in *SIGCOMM*, 2008.