A Peer Auditing Scheme for Cheat Detection in MMOGs

Josh Goodman
McGill University
School of Computer Science
Montréal, Canada
jgoodm7@cs.mcgill.ca

Clark Verbrugge
McGill University
School of Computer Science
Montréal, Canada
clump@cs.mcgill.ca

ABSTRACT

Although much of the research into massively multiplayer online games (MMOGs) focuses on scalability concerns, other issues such as the existence of cheating have an equally large practical impact on game success. Cheat prevention itself is usually addressed through the use of proprietary, ad-hoc or manual methods, combined with a strong centralized authority as found in a straightforward client/server network model. To improve scalability, however, the use of more extensible, yet less secure, peer-to-peer (P2P) models has become an attractive game design option. Here we present the *IRS* hybrid game model that efficiently incorporates a centralized authority into a P2P setting for purposes of controlling and eliminating game cheaters. Analysis of our design shows that with any reasonable parametrization malicious clients are purged extremely quickly and with minimal impact on non-cheating clients, while still ensuring continued benefit and scalability from distributed computations. Cheating has a serious impact on the viability of multiplayer games, and our results illustrate the possibility of a system in which scalability and security coexist.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; K.8.0 [**Personal Computing**]: General—*games*

Keywords

Computer Games, Cheating, Network Model, Trust Model, Peer to Peer, Peer Auditing.

1. INTRODUCTION

The presence of cheating players has a profoundly negative effect on a competitive multiplayer game—if some players are seen to have an unfair advantage the game quickly loses appeal to non-cheating players. The subsequent departure of a large part of a game's client base can be devastating to the viability of a subscription-based commercial game, particularly in the case of MMOGs, which rely on a large, sustained and loyal player population. With the exception of a few cheat-specific solutions, however, approaches to cheat prevention and elimination are primarily proprietary, often ad-hoc or manual solutions. In many cases and despite the advantages offered by more distributed P2P solutions, cheat reduction is simplified through the use of client/server models that maintain strong central authority. Limitations on scalability are an acceptable sacrifice for the ability to better control the game state and regulate player actions.

We develop the *IRS*, Invigilation Reputation Security, hybrid model which permits the use of distributed computation while at the same time providing strong guarantees with respect to cheat prevention and elimination. In our design the gamestate is still controlled by a centralized server, acting as both login point and arbiter of client behaviour. In order to reduce the computational load and thus increase the scalability of the central server, computations are distributed between clients. Importantly, these calculations are verified through a simple peer auditing scheme, ensuring clients are unable to effectively distort or take advantage of any implicit delegation of authority or extra information they may receive, and that cheating clients can be efficiently removed from gameplay.

With an unreliable client network faulty behaviour of various forms must be expected at both the network and software levels. Therefore, it is important to differentiate between frequent cheating and random computational or communication errors. To reduce the impact of false positives in auditing we therefore make use of a novel client trust metric. By weighting client trustworthiness according to the proportion of audit failures (of different forms), false positives can be essentially eliminated while still ensuring cheaters are rapidly purged. Through extensive experimental simulation, we are able to show that under reasonable parametrization our system eliminates a population of 15% active cheaters in 400 *seconds*, while non-cheating players have an expectation of 7.5 *months* of continuous gameplay before any spurious identification of cheating. Efficiency is considered in terms of both extra message cost and server CPU load: overhead results in approximately twice as many network messages, but server CPU load is only 10% of that of an equivalent client/server model. The added networks costs are seen as a necessary trade-off between security and efficiency.

Contributions of this work include:

- We present a design for a cheat-resistant, scalable hybrid network model for MMOGs. Our approach balances an authority-based solution to cheat elimination with scalability improvements offered by P2P solutions.

- To reduce false positives we make use of a novel client trust scoring method based on frequency and severity of bad behaviour. Our history-sensitive metric is easy to compute and could be applied in other P2P applications such as file-sharing where trust is a concern.
- Using detailed experimental simulation, we are able to give quantitative bounds on the expected lifetime of cheaters and non-cheaters in our system. This is easily tunable, and different auditing rates and trust parametrizations can be used to adapt the trade-off between false positives and cheat elimination to individual game needs.

2. RELATED WORK

Much of the research into MMOG design focuses on scalability and consistency requirements as a core requirement of a feasible, large-scale multiplayer game design. Research into cheating concerns is less established. Below we summarize first approaches to scalable game design, followed by works aimed directly at MMOG cheating.

Pure client-server network design provides a game protection from abuse of authority cheats. Such architectures, however suffer from poor scalability due to the single server bottleneck, as well as the inconsistent workload caused by the periodicity of client play time [26, 11]. Clustering models and other designs extend this basic approach [14].

P2P architecture allows games to circumvent the scalability issues that result from the client-server model. With distributed computation, server bottlenecks are reduced or eliminated, even though this also gives clients greater authority over the gamestate [21, 10, 18]. Unfortunately, although increased scalability is attractive, measures to counteract abuse of authority are rarely covered in detail. [7].

Hybrid models are the natural extension, combining scalability with some amount of centralized authority. Mediator and FreeMMG, for instance, are hybrid models that allow peers to assume different authoritative roles, providing a centralized but dynamic locus of control where clients manage the gamestate in a server-like manner [15, 8]. Trust systems are essential for realistic management of cheating in such environments, and while these have been investigated in several contexts [20, 4] our design provides a simple, fault-tolerant solution. Shi *et al.* present a more complex, but structurally similar model to ours, in their work on fuzzy reputation in P2P systems [6]. Although it is applied to a different model, they also find it an effective means of lowering the false positive rate in identifying undesirable clients.

The magnitude of cheating and extent of industry interest is evident from various sources, including game surveys [17] and commercial game reports. Two popular games, World of Warcraft [1] and Final Fantasy XI [3], for example, give quasi-regular reports on elimination of cheaters from within their games. Many case studies on cheating have also been performed for individual games [22].

Understanding cheating is a first step to eventual control. Yan and Randell provide a classification system [28], and others have considered cheating in MMOGs. Kabus *et al.*, for instance, discuss 3 cheat elimination methods for P2P systems [19]. Of the three elimination schema presented, the idea of mutual verification is of key importance and closely related to the auditing scheme presented herein. Webb and Soh give a recent review of game cheating and provide in-depth discussions on current elimination strategies designed to address such problems [27].

Many individual cheats have been reported, with different solutions offered. A common form of cheating, the *input data attack,* also known as *aim-bots,* involves using external tools to improve response time and accuracy of player actions. Use of "captcha's" and/or the introduction of trusted hardware have been proposed as solutions [16, 25]. *Time cheating* is also a prevalent cheating behaviour, involving taking advantage of game network protocols designed to hide latency. Typical solutions include tight synchronization (lockstep) or making use of cryptographic promises to guarantee synchronous knowledge discovery among game players [5, 13, 24, 12]. Client knowledge in general must be limited; *maphack*s and *wallhack*s manipulate the game client environment to reveal more information than intended, such as location of distant or occluded enemies. Solutions in general are mainly based on centralizing and regulating the availability of inessential information [9]. Laurens *et al.* present a statistical approach that examines the difference in behaviour between cheating and honest clients [23].

In a distributed environment the prevention of authority based cheating is of top priority. The investigations into the IRS model in the following sections show that in our environment clients who abuse authority gain little success at effecting overall gameplay.

3. IRS MODEL DESIGN

The IRS model is designed to ensure the efficiency of a P2P design while maintaining the cheat resistance afforded by a client-server model. To reduce the computational burden on the centralized server and retain a stable gamestate, we use a communication model that allows clients to handle message resolution in place of the central server. The resolved messages computed by the clients must also be audited, monitored and tested to ensure the safety of the gamestate. Further refinement of cheat elimination is provided by a trust system which determines when disciplinary action is necessary. Below we describe the core features of our hybrid model design.

3.1 Communication Model

The basic communication model incorporates aspects of both P2P and client server communication architectures. The P2P design aspect allows certain client messages (computations) to be handled by other clients acting as proxies. The client-server design allows for a centralized server which handles login, matching and monitoring of all clients while maintaining the game state. The proposed communication model consists of 4 main phases: proxy assignment, message relaying, auditing and resolved message handling.

1. **Proxy assignment** is done by randomly assigning a proxy to each client, where each client acts as a proxy for exactly one other client. Proxy (re-)assignment occurs at regular intervals.

2. **Message relaying** is the process by which a client's message is relayed to its proxy by the server. The request message is then resolved by the proxy client and sent back to the server.

3. **Peer auditing** ensures that clients cannot abuse their authority by distorting proxy computation results. This involves comparing the proxy's resolved message to the result

of another client's response to the same request. More on this procedure is presented in section 3.2.

4. **Resolved message handling** is performed by the server upon reception of a proxy's resolved message. It consists of a quick test, which if successful results in the resolved message being relayed to the requesting client (and appropriate interest group). The quick test is an essential feature than ensures that no infeasible messages reach clients. It will be discussed in more detail in section 3.4.

After proxy assignment (1), communication operates as follows: a requesting client sends its request message to the server, which relays it to the requesting client's proxy (2). The proxy then computes the result of the request message and returns it to the server. In this step there is a random chance for an audit to occur as discussed in section 3.2 (3); all message are subjected to a quick-test as per section 3.4 (4). If the quick-test succeeds, the resolved message is sent to the requesting client and all interested clients. In the event of a timed-out proxy response or a failed quick-test the server will handle full message resolution.

3.2 Auditing Scheme

With clients performing computations for others, verification becomes an important aspect of maintaining the correctness of the game state. Of course, checking the veracity of a resolved message requires the full resolution of the request message, and so cannot be feasibly applied to all messages. We thus subject only a continuous sampling of message responses to *peer audits:* during the *message relaying* phase above (2), a second copy of the message is relayed to a random client (co-auditor) for resolution. When the server collects the proxy and co-auditor's resolved messages, it performs a comparison, which yields one of the following 4 possible results:

- *Identical* (IDENT): the two responses are the same
- *Equivalent* (EQUIV): the two responses are not identical but still represent acceptable, very similar solutions.
- *Inequivalent* (INEQ): the responses both provide allowable solutions, but are very different.
- *Infeasible* (INFEAS): at least one response is clearly against game rules.

As a concrete example, consider a system which uses peers to compute path-finding messages, sending start/end locations and expecting a full path as a response. Comparison of two message results might show two paths that pass through the exact same points (IDENT), two paths with the same start and end points and similar lengths (EQUIV), two paths with different end points and/or very different lengths (INEQ), or either of the paths may pass through an obstacle, move off the game world, or otherwise violate permitted movement behaviour (INFEAS).

It should be noted that rather than being the result of the comparison of two messages, INFEAS results are produced by the examination of individual messages—if either resolved message in an audit is infeasible the audit result will also be flagged as such. The result of a peer audit is considered a success if the message comparison yields an IDENT or EQUIV result. In the event of an INEQ or INFEAS result further examinations are needed in order to determine which, if not both, clients are responsible for the lack of accuracy. This is done by the monitoring process discussed in the next section.

3.3 Monitoring Scheme

Peer auditing provides a simple method for detecting malicious clients; however, in order to determine which, if not both, of the two clients involved in a failed audit is responsible for the suspicious behaviour, it becomes necessary to *monitor* failed audits. As a result, dedicated *monitor clients*, which are controlled by the game provider, are added.

Failed audits will be sent to an available monitor for processing along with the original request message. The monitor then computes the actual result from the request message and compares it to both resolved messages from the audit in question. The resulting comparison is then sent to the server.

It should be noted that in certain cases is might be possible for an audit containing two cheat messages to escape detection if both cheat message are similar enough to one another, or even identical. Therefore, audits deemed successful should be checked sporadically. The infrequent monitoring of successful audits also allows for legitimate clients to be commended and ensures that positive behaviour increases the client's trust rating (see Section 3.5).

3.4 Quick Testing

As mentioned above, *quick testing* is performed when a resolved message is received by the server. Since it is too costly to examine each resolved message's accuracy fully, cheaper tests are used to ensure that no INFEAS messages are sent to clients. *Quick tests* are designed to be much cheaper to perform than full message resolution, but are limited to coarse detection of infeasibility.

When the server successfully identifies an INFEAS resolved message through quick testing, the correct message result is instead recomputed by the server and relayed to the appropriate clients in order to prevent inaccurate information from being distributed. Infeasibility is of course a game-dependent, and even state-dependent property of a game calculation; in some cases INFEAS messages of certain types might not have a serious effect on the overall game state, or conversely some INFEAS messages may be especially undesirable. As a heuristic safety filter, quick testing can be easily disabled or extended at the game designer's discretion.

The processes of auditing and monitoring occur after resolved messages are relayed by the server to the appropriate clients. It is therefore possible for INEQ messages to reach clients. Quick-testing ensures no exceedingly harmful messages ever escape detection.

3.5 Trust

The above cheat detection mechanisms aid in determining the accuracy of a given message; however, with computation and message transfer instability as possible causes of errors, determining which clients are the actual cheaters becomes a challenge. In order to separate cheaters from honest clients a *trust metric* is implemented. *Trust* is calculated based on a client's history of *monitor comparisons* and *quick tests* and is a function of the number of IDENT, EQUIV, INEQ, and INFEAS results associated with that client. Section 4.1 presents our actual formula.

By design, IDENT and EQUIV results should cause the client's trust to rise; whereas, INEQ and INFEAS results should cause a drop in trust. An exponential decrease in trust associated with a rise in INEQ and INFEAS results ensures that if inaccurate results are few they will not result in a low score, whereas repeated offenses ramp down quickly.

A potential drawback to using such an exponential penalty is that over a long time even intermittent errors might affect legitimate clients negatively. Additionally, long time trustworthy clients might be able to abuse their built up trust scores to produce a burst of cheats. As a result, there is a need for the inclusion of a discount factor that disregards older behaviour. The analysis of such a factor is left as future work.

3.6 Disciplinary Action

The elimination of cheating clients is essential to the maintenance of an accurate gamestate. As a result *disciplinary action* will be taken on those who are caught submitting inaccurate resolved messages. Two forms of disciplinary action exist, *booting* and *banning*.

- **Booting** temporarily removes a client from the game; this is performed upon the reception of an inaccurate resolved message.
- **Banning** occurs when a client's trust score falls below the *ban threshold*, a threshold set by the game developers that defines the cut-off of between acceptable and unacceptable trust scores. Banning is the permanent removal of the client from the game.

The booting of clients is an essential measure to ensure few if any cheating clients are active in the system at any one time. It has the added effect of warning clients that their behaviour has crossed a boundary and will not be tolerated. Note that it is possible for legitimate clients to be booted due to spurious errors; however, given the rarity of such errors booting does not pose a large threat to the client's overall legitimate gameplay.

Banning is a more drastic action necessary to ensure the overall number of cheating clients is reduced over time. As with booting, this also functions as a deterrent, albeit more extreme.

It should be noted that when a client is booted or banned from the server, the client for which they performed proxy computations is then without a proxy. As a result, until the next proxy matching, the server will resolve messages for that client.

4. RESULTS

Our design process included validation through a simulation study, exploring the behaviour of the system under different cheating loads. Two main results are discussed here; the first examining the system given an existing, fixed population of cheating and legitimate clients, and the second showing how our design reacts to a dynamic population, where cheaters and legitimate clients are incrementally and continuously introduced.

The use of a hybrid P2P model greatly reduces CPU load. P2P models also typically distribute bandwidth costs. The continued use of a central authority for strong cheat detection and elimination, however, as well as auditing and proxy relaying naturally imposes some overhead in our design. Below we provide data on both improvement and extra messaging costs.

Data presented in this section represent the average of 10 runs for cheat elimination data and 5 runs for scalability investigations. Each run in a batch is performed under identical parametrization.

4.1 System Parametrization

Game populations were represented by different mixes of legitimate (non-cheating) clients and classes of cheaters, each group with varying cheating or failure rates. Below we describe the precise parametrization, including choice of trust metric and ban thresholds.

- **Legit Clients:** Legit clients can experience network or software errors, and so a failure rate of 4% is used, where 75% of failures produce EQUIV results and the remaining 25% are INEQ. This results in a 1% chance to return inaccurate information, showing the hybrid model in an extreme, worst-case setting.
- **Hackers:** Players actively working to manipulate and perhaps even destabilize game play will return cheat messages 50% of the time, half of which represent INEQ resolutions, and half INFEAS information.
- **Griefers:** In an attempt to cause players confusion, Griefers will transmit INEQ results 50% of the time.
- **Monitors:** Trusted clients owned and run by the game providers which are used to verify audits. They are assumed to compute 100% accurate results. In all our experiments monitor load is not insignificant; here 20 monitors are used.

Request Frequency: Clients send request messages every {0,3] seconds, chosen uniformly.
Verification: There is a 10% chance of audit creation and a 5% chance to monitor successful audits.
Trust Metric: $\mathcal{T} = \text{IDENT} + \text{EQUIV} - \text{INEQ}^{1.5} - \text{INFEAS}^{2}$, chosen to promote a slower growth for cheating clients.
Discipline: The ban threshold was set at -15 in all experiments. Booting lasts 30 seconds.

Other values and formulae for ban threshold and trust metrics were explored, but are beyond the scope of this paper. Settings here represent reasonably optimal choices within a broad range of acceptable choices.

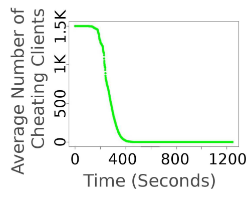
4.2 Collusion

With clients acting as proxies for each other collusion is a potential form of cheating. Clients may choose to use cheating to benefit friends for whom they are a proxy. If groupings are small colluders are ineffective due to the unlikelihood of finding themselves a proxy for one of their friends. With a group of four in a population of 10,000 clients, for example, a colluder would only have a 0.03% chance to be in a favourable condition for misbehaviour. Since in small groups, a colluder's rate of cheating is actually less than the failure rate of legit clients, colluders are not modelled separately in our experiments.

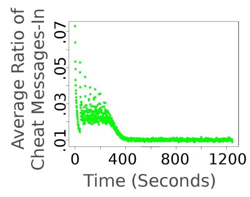
It should be noted that even in sufficiently large groups, colluders still remain vulnerable to cheat detection due to their elevated overall output of cheats.

4.3 Effects on a Static Setting

Experiments performed under static conditions begin from an existing client population and assume that no clients join or leave the system unless banned. Under these conditions the *hybrid model* is quickly able to eliminate cheaters from the system without creating an abundance of false positives. Data was collected and analyzed from many kinds of client populations, but results are very stable. Here the static set-

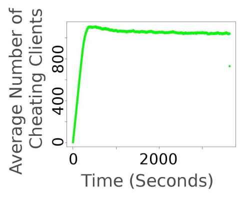


a) Number of cheating clients

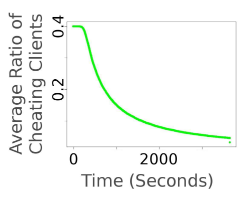


b) Ratio of cheating messages-in

Figure 1: Effects in a static setting

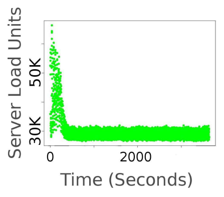


a) Number of cheating clients

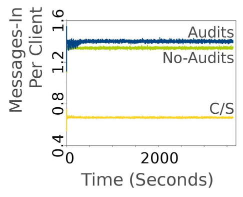


b) Ratio of cheating clients

Figure 2: Effects in a dynamic setting



a) Server load



b) Bandwidth comparison

Figure 3: Effects on load

ting contains 8,500 legit clients, 750 hackers and 750 griefers in experiments of 20 minutes in length.

Figure 1 shows how cheating is eliminated under these conditions. Graph a) displays the number of cheating clients in the system, showing a sharp decline ending at approximately 400 seconds. At this point of the original 1500 cheaters, 0 remain. The ratio of cheat messages in the system is shown in graph b). Again by approximately 400 seconds the number of incoming cheat messages has dropped to less than 1% of messages. Note that cheating messages do not drop to 0 due to the "noise" caused by the remaining legit clients' chance to return inaccurate messages.

In this static scenario the existence of false positives is very rare. Even with an extreme, worst-case simulation of legit client error (1% overall), ten 20-minute experiments with 85,000 legit clients, only reported 4 false positives.

4.4 Effects on a Dynamic Setting

Although cheat detection can be introduced to an existing, stable population, most games experience some amount of churn in their user base: new legit and cheating clients are continually introduced. Dynamic experiments thus examine behaviour under a continuous cheating load. Starting from an empty (no clients) game system, 10 clients are introduced per second. Of these 10 new clients, 2 are hackers, 2 are griefers and 6 are legit clients. To allow the system to grow and stabilize, experiments are extended to 1 hour in length.

Figure 2 summarizes the effect on cheating; the addition of clients over time clearly does not hinder the elimination of cheaters. From Figure 2.a) it is evident that the cheating population does grow, but it quickly reaches a plateau. The number of legit clients continues to increase, however, and as can be seen in Figure 2.b) the relative number of cheaters in the system drops quickly and is asymptotically headed towards no cheating.

Further study shows that the depicted trends are proportional to the number of cheaters introduced per second, which indicates that the time required to eliminate cheaters is not strongly related to the rate at which they are introduced. With a longer study more false positives, 73, occur here. Under less extreme conditions, such as assuming legit clients suffers only a 0.1% failure rate, they would be expected to last for 7.5 months of continual play without fear of being falsely identified for approximately the first 3.

4.5 Effects on Overhead and Scalability

Cheat elimination is essential to practical game design, and usually considered well worth some amount of performance trade-off. Scalability, however, is also critical, and so it is important in the hybrid model that authority nodes are not over-loaded, and can function reasonably well in a large scale setting.

Proxy calculations in the hybrid model rely on two extra messages passed per incoming request, and so naturally the number of messages that the server is required to handle is approximately double that of the standard client-server model. CPU load, however, is dramatically less, with the hybrid server reaching only 10% of that of the client-server, assuming reasonable game computation requirements.

Through simulation it is possible to determine load and bandwidth data for a client/server and hybrid model simultaneously. Figure 3 shows server load and bandwidth data from a static setting, where each computation-request message is given a (random) assumed cost. In the hybrid model the server settles down to about 30,000 work units, whereas a client-server model's load (not shown) is concentrated around the 275,000 mark, approximately 10 times higher. The average cost incurred by clients due to audits and proxy resolution is also low; on average a client in this experiment would be required to handle only an extra 35ms worth of calculations per second.

Graph b) of Figure 3 compares server bandwidth requirements in the hybrid model without any auditing in place, in the hybrid model with full auditing, and in a standard client/server strategy. The addition of the auditing features of the hybrid model has minimal impact, whereas comparison with a client/server model confirms the above intuition that on a per-client basis, the volume of messages required by the server to support a client in the hybrid model is approximately doubled.

5. CONCLUSIONS AND FUTURE WORK

Eliminating cheaters is an unfortunate necessity for online computer games, and this is a problem that typically involves a trade-off between scalability and game security. Our hybrid model shows that cheating can be effectively controlled in a semi-P2P system with good scalability and acceptable overhead. Cheating clients are essentially limited to cheating rates below any reasonable threshold for utility in order to remain in the game for any significant length of time. In the equally important task of protecting honest clients, trust modelling ensures that even those who make occasional errors are not unfairly tagged as cheaters.

Server load in our model is approximately one tenth of a server operating on the same game data in a client-server model, at a cost of doubling network messages and thus latency. Although this represents an appropriate balance of efficiency and fairness, amelioration is possible, and we are currently evaluating improved distributed computation protocols that require fewer messages and maintain security. Adaptive auditing based on reputation is another potentially fruitful avenue for future work, trading back security for efficiency but only for historically reliable clients. Since clients may play naively for a long time before discovering and deciding to apply game cheats this kind of long-term trust would still need to react quickly if trust changes.

Our current work is in providing a concrete implementation of the design presented and analyzed here. We are integrating our complete network model into the Mammoth research game framework being developed at our institution [2] in order to alleviate the high cost of secure server-side path-finding. A full-game implementation will allow us to further investigate and tune the model properties, as well as provide analysis based on actual game data.

6. REFERENCES

[1] Blizzard Entertainment, World of Warcraft. `http://www.worldofwarcraft.com/index.xml`.

[2] Mcgill University, Mammoth. `http://mammoth.cs.mcgill.ca/`.

[3] SQUARE ENIX, Final Fantasy XI. `http://www.playonline.com/ff11us/index.shtml`.

[4] B. Ali, W. Villegas, and M. Maheswaran. A trust based approach for protecting user data in social networks. In *IBM CASCON 2007*, pages 288–293, Richmond Hill, Ontario, Canada, Jan. 2007.

[5] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *IEEE InfoCom*, pages 104–113, 2001.

[6] X. bin Shi, L. Fang, D. Ling, C. Xiao-hong, and X. Yuan-sheng. A cheating detection mechanism based on fuzzy reputation management of P2P MMOGs. In *SNPD 2007*, pages 75–80, Washington, DC, USA, 2007.

[7] F. R. Cecin, C. F. R. Geyer, S. Rabello, and J. L. V. Barbosa. A peer-to-peer simulation technique for instanced massively multiplayer games. In *DS-RT 2006*, pages 43–50, Washington, DC, USA, 2006.

[8] F. R. Cecin, R. Real, R. de Oliveira Jannone, C. F. R. Geyer, M. G. Martins, and J. L. V. Barbosa. FreeMMG: a scalable and cheat-resistant distribution model for internet games. In *DS-RT 2004*, pages 83–90, Washington, DC, USA, 2004.

[9] C. Chambers, W. chang Feng, W. chi Feng, and D. Saha. Mitigating information exposure to cheaters in real-time strategy games. In *NOSSDAV 2005*, pages 7–12, Washington, USA, June 2005.

[10] L. Chan, J. Yong, J. Bai, B. Leong, and R. Tan. Hydra: A massively-multiplayer peer-to-peer architecture for the game developer. In *Netgames 2007*, pages 37–42, Melbourne, Australia, Sept. 2007.

[11] W. chang Feng, D. Brandt, and D. Saha. A long-term study of a popular MMORPG. In *Netgames 2007*, pages 19–24, Melbourne, Australia, Sept. 2007.

[12] B. D. Chen and M. Maheswaran. A cheat controlled protocol for centralized online multiplayer games. In *NetGames 2004*, pages 139–143, Portland, OR, USA, Aug. 2004.

[13] E. Cronin, B. Filstrup, and S. Jamin. Cheat-proofing dead reckoning multiplayer games (extended abstract). In *Conf. on Appl. and Dev. of Comp. Games*, Jan. 2003.

[14] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin. An efficient synchronization mechanism for mirrored game architectures. In *NetGames 2002*, pages 67–73, Bruanschweig, Germany, 2002. IEEE.

[15] L. Fan, H. Taylor, and P. Trinder. Mediator: a design framework for P2P MMOGs. In *Netgames 2007*, pages 43–48, Melbourne, Australia, Sept. 2007.

[16] P. Golle and N. Ducheneaut. Preventing bots from playing online games. *Computers in Entertainment*, 3(3):3–3, 2005.

[17] R. Greenhill. Diablo and multiplayer game's future. `http://www.gamesdomain.com/gdreview/zones/shareware/may97.html`, May 1997.

[18] X. Jiang, F. Safaei, and P. Boustead. An approach to achieve scalability through a structured peer-to-peer network for massively multiplayer online role playing games. *Computer Communications*, 30(16):3075–3084, 2007.

[19] P. Kabus, W. W. Terpstra, M. Cilia, and A. P. Buchmann. Addressing cheating in distributed MMOGs. In *Netgames 2005*, pages 1–6, 2005.

[20] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *WWW 2003*, pages 640–651, 2003.

[21] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *IEEE InfoCom*, Mar. 2004.

[22] J. Kuecklich. Other playings: cheating in computer games. In *Other Players Conf.*, IT University of Copenhagen, Dec. 2004.

[23] P. Laurens, R. F. Paige, P. J. Brooke, and H. Chivers. A novel approach to the detection of cheating in multiplayer online games. In *ICECCS 2007*, pages 97–106, Washington, DC, USA, 2007.

[24] S. Mogaki, M. Kamada, T. Yonekura, S. Okamoto, Y. Ohtaki, and M. B. I. Reaz. Time-stamp service makes real-time gaming cheat-free. In *Netgames 2007*, pages 135–138, Melbourne, Australia, Sept. 2007.

[25] T. Schluessler, S. Goglin, and E. Johnson. Is a bot at the controls? detecting input data attacks. In *Netgames 2007*, pages 1–6, Melbourne, Australia, Sept. 2007.

[26] J. Smed, T. Kaukoranta, and H. Hakonen. A review on networking and multiplayer computer games. Technical Report Tech Report No. 454, University of Turku Centre for Computer Science, 2002.

[27] S. D. Webb and S. Soh. Cheating in networked computer games: a review. In *DIMEA 2007*, pages 105–112, 2007.

[28] J. Yan and B. Randell. A systematic classification of cheating in online games. In *Netgames 2005*, pages 1–9, Hawthorne, New York, USA, Oct. 2005.