Addressing Cheating in Distributed MMOGs

Patric Kabus Wesley W. Terpstra [*] Mariano Cilia Alejandro P. Buchmann
{pkabus,terpstra,cilia,buchmann}@dvs1.informatik.tu-darmstadt.de
Databases and Distributed Systems Group
Darmstadt University of Technology, Germany

ABSTRACT

Massively Multiplayer Online Games (MMOGs) are a risky business: while they offer potential profits beyond those of conventional computer games, they also require costly investment in the necessary hardware infrastructure. In nearly every MMOG today, these costs come from the use of a Client/Server architecture where the load of possibly hundred thousands of players must be handled at the provider's backend. By using distributed Peer-to-Peer techniques, the load could be shifted completely or partially to the players' machines. But with the load, the control over the game may also fall into the hands of clients. While using a P2P architecture, this paper presents a spectrum of options which reduce running costs and simultaneously attempt to retain the provider's control over the game, in particular to control cheating.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Security

Keywords

Distributed Systems, Massively Multiplayer Online Gaming, Cheating

1. INTRODUCTION

With the success of the Internet, online games are becoming more and more popular. According to [14], almost five million players in the US are playing PC-based games like *Quake* or *Counter Strike* over the Internet. One of the most promising segments of the online games market is that of the *Massively Multiplayer Online Games (MMOGs)*. It has grown by the factor 8 during the last five years, having currently more than four million subscribers in the U.S. and Europe, and roughly the same number of players in Asia [21].

Traditionally, MMOGs are built relying on Client/Server architectures. From an abstract point of view, the client on the player's side acts as simple front-end which accepts the player's input and forwards it to the server. Processing the game state is completely done on the server-side and changes are sent back to clients, which show a graphical representation of the game world. Unlike traditional computer games, in MMOGs the development of the game is only the prelude to the real work: launching and running the game service at the backend. To be able to handle thousands of players simultaneously, large amounts of computing power and network bandwith are required on the server-side. The costs for supplying servers and connectivity can easily reach hundreds of thousands of dollars per month.

Another factor is that players are much more likely owners of powerful machines than the average computer user. Based on this fact, and considering that the amount of resources required in a MMOG is nearly proportional to the number of simultaneous players, the Peer-to-Peer architecture seems to be more appropiate than the traditional Client/Server aproach. Peer-to-Peer (P2P) architectures save costs due to utilization of resources on the client-side. This paper shows ways of applying P2P techniques to reduce costs, while at the same time maintaining the subscription-based business model and providing protection against cheating.

2. MOTIVATION

It is not simply by chance that almost every MMOG is based on a Client/Server (C/S) architecture: C/S has the striking advantage of giving game providers full control over their games. Since the business model of MMOGs is based on subscriptions, the provider wants to decide which players may join the game, namely those who are actually paying their fees. While subscriptions provide recurring profits, they also serve as an effective solution against piracy. C/S also is an important aid in cheat prevention: The server has the sole authority over the game's global state and validates every action request sent by clients before carrying them out. Finally, a C/S architecture is—at least compared to a P2P system—rather straightforward to implement.

On the downside of C/S systems are cost and scalability. For instance, the cost of acquiring servers for 30,000 simultaneous players are about US$ 800,000, the bandwidth costs

[*]GK Enabling Technologies for Electronic Commerce

NetGames'05, October 10–11, 2005, Hawthorne, New York, USA.
Copyright 2005 ACM 1-59593-157-0/05/0010 ...$5.00.

for the same number of players a hundred thousand dollars a month [19]. Considering block-busters like *World of Warcraft (WoW)* which sold hundreds of thousands of copies on their *first days*, one can imagine that the actual costs are a multiple of those mentioned above. For this reason, correctly pre-estimating the number of simultanous players becomes crucial. Underestimating will result in major customer dissatisfaction due to poor availability and/or performance of the game servers, while overestimating means wasted money due to idle resources. For successful titles like WoW, backed up by well-funded publishers, profits are high enough that lowering the costs may not be a primary issue. But success is barely predictable, and for less popular titles costs may endanger the feasibility of the whole project. Furthermore, lower upfront costs may remove entry barriers for less well-funded game publishers, allowing them to enter the market at all.

Scalability becomes especially an issue if monolitihic server architectures are used; their possibilities to grow with the customer base are severely limited. Distributed architectures on the server-side like grids or object request brokers may be a solution to the scalability problem. There exist several commercial middleware solutions for MMOGs, like the *Internet Communications Engine (ICE)* [10] or *Butterfly.net* [11, 12], but todays MMOG developers seem to be reluctant in applying external solutions and rather rely on building their own [13].

Even with distributed server architectures, the provider still has to pay for all the bandwidth and CPU cycles. The only way of significantly reducing the costs is to shift at least part of the CPU and network load to the client, as it is done in P2P systems. This could also be a solution for the problem of predicting the amount of players, since every additional player that joins the game brings in additional resources into the system. Of course there are many other challenges that come along with P2P architectures: loss of authority over the game state, system consistency and availability, persistent data storage, and higher overall complexity, to name only the few most important. The primary focus of this paper is on the problems related to loss of authority, though consistency issues will be touched upon.

3. CHEATERS AND GRIEFERS

Before delving into the different approaches, this gives a short introduction into the issue of cheating. In a LAN-based multiplayer game (where players are in proximity and may know each other), cheating may be handled by the players themselves, but in the anonymity of the Internet it becomes a serious threat to the game provider's business. Cheaters try to gain unfair advantage over other players, for example by duplicating game items ("item dupe" cheat) or more generally short-cutting achievements which would take enormous time and/or effort for honest players. This can totally destroy the in-game economics of an online game: formerly valuable and rare items become widely available, powerful high-level characters which are usually a rare occurrence may now be seen everywhere. Honest players will soon notice that they cannot keep up with cheaters and either use cheats themselves (and thus accelerate the collapse of the economic system) or stop playing the game.

As already mentioned, in a C/S system, the server has the sole authority over the global game state. In theory, this makes it impossible for players to gain advantage by performing malicious alterations of the state. In practice, design flaws and implementation bugs may still allow successful cheating, but this is out of the scope of this paper. In a P2P environment, there is no such single authority, the global game state is distributed among the peer nodes. Consequently, each node is able to alter its local state arbitrarily.

Another kind of cheating is of interest in this context: players that acquire information that is not intended for them. A player that knows the position of an enemy that is hiding behind a wall is an example for this. Because he cannot see the enemy, he shouldn't know about his position. In a C/S system, the server can easily restrict the flow of information to the client. If a player can neither hear, see or otherwise notice an enemy he does not receive any information about his position. In a P2P system, where such information is distributed among peer nodes, it becomes very difficult to prevent leakage of this information. Every client that posseses this information is able to disclose it, wether this is against the rules or not.

Maybe even worse than cheaters are the so called "griefers". As the name implies, the sole intention of these people is to hurt other players' experience as much as possible. While griefing may actually be performed without breaking any game rules (e.g. insulting other players through the player chat), griefers may also exploit possible cheats to hurt other players, e.g. killing their avatars or stealing their items. Because of griefers, worst-case scenarios may become much more relevant than usual: While a worst-case usually may be a rare occurrence, griefers actively try to make it happen, thus raising the chance. "The rule of grief is: if it can be done, it will be done" [13].

The above points out that protection against cheaters and griefers is a mission-critical task. That is why special attention is turned on this issue in this paper.

4. RELATED WORK

Baughman et. al. [1] propose a scheme that uses a per-frame lock-stepped commitment protocol on secret information to prevent *look-ahead* and *suppress-correct* cheats. Subsequently, they present an optimized version of their protocol, which requires synchronous lock-stepping only when two players come within range of interaction. The authors argue that "anything outside of a player's sphere of interest is immaterial to the player's upcoming decisions". However, this is not true for all multiplayer games. In common Massively Multiplayer Online Role-Playing Games (MMORPG), a player could fight against computer-controlled monsters while other players are not around. He could falsely claim that he killed lots of them, acquiring experience points and wealth. Furthermore, two players within interaction range could collaborate in cheating while other players are not within range. Falling back on a game-wide lock-step is infeasible in the context of thousands of players, as is typical in MMOGs. A simple logging scheme is also mentioned to detect forgery of hidden items. This has some similarities to the logging scheme we present in section 5.3, but does not encompass the entire game state. Their approach also focuses on verification rather than auditing via replay of the log history. The NEO protocol [7] was developed as an improvement to the one presented above. It addresses a broader range of cheats while at the same time reduces latency. However, to achieve good responsiveness, it prevents players with very slow connections to enter game areas where players with fast conections are located. Like the protocol described in [1], NEO does not solve the problem of players interacting with the environment while other players are not within range. This problem is mentioned, but its solution is deferred to the implementation of a special computational component which is left for future work.

Buro [3] presents a server-based architecture which addresses the *maphack* cheat popular in Real-Time Strategy Games (RTS). Chambers et al. [6] show that this kind of attack can also be addressed in a Peer-to-Peer architecture. While the maphack cheat is of the utmost significance to RTS games, it is far less relevant in MMORPGs. Besides that, the architecture presented in [6] relies on a distributed simulation where every player's client computes the complete game state individually. This is infeasible in a Massively Multiplayer context.

FreeMMG [5, 4] is a hybrid between Peer-to-Peer and Client/Server architecture. While a server part is responsible for managing subscriptions, authentication and storing backups of the virtual world, the game itself is running in a distributed fashion on the clients. The game world is split up into segments and every player is located in one of these segments. It is similar to the Region Controller approach presented in section 5.2, but it uses the Replicated Simulation architecture instead, where the whole state of a segment is replicated on the clients of the players within that region. Players tend to crowd in special locations like cities, trading points, and spawning areas. We expect that there will be too many players in these regions for the Replicated Simulation approach to scale, so long as the regions are static. This problem is compounded by the inability of players to interact between regions, encouraging clustering.

There exist more proposals for distributed gaming architectures like Mercury [2], SimMud[15] and MiMaze [17, 8, 9]. However, they do not explicitly address the problems originating from cheating.

5. APPROACHES

This section presents alternatives to the traditional C/S approach that rely on a P2P archicture which offers cost savings and better scalability, while at the same time retaining the business model and resistance against cheating. They should not be thought of as complete solutions but serve to identify a spectrum of possible directions for future research.

The first approach, *Distributed State Dissemination*, handles the processing of action requests like traditional C/S systems do, but relies for the disseminaton of state updates to the clients on a P2P delivery system in order to save on bandwidth. The computation of the global game state is still performed on the server-side, which is also responsible for cheat prevention. In the *Mutual Checking* scheme, the global game state is maintained in a distributed fashion on the clients. The global state is replicated on multiple clients which try to detect cheaters by comparing their local versions regularly. Few servers are still needed for managing the assignment of state to clients, subscription management and storing persistent data. The *Log Auditing* approach also distributes the global game state computation among the clients. But it does not try to *prevent* cheating by validating state updates before applying them. It rather tries to *detect* cheating later by analyzing signed log files of the state transitions caused by updates. The log auditing can be done during periods of low activity; this relieves load during peak times and synchronization costs. The last approach, *Trusted Computing*, acts on the assumption that players are not able to manipulate their client software in any way. This would make both prevention and detection of cheats caused by hacked clients unnecessary, providing the ideal environment for distributed online games.

5.1 Distributed State Dissemination

If the primary cost in running a game is the bandwidth, one approach is to push this cost towards the clients. If several clients are in the same area and must be informed about the same events, these messages can be sent to one client who then forwards the event notification onwards. This may impact latency for some clients.

While simple, this approach saves bandwidth where there are many players clustered in the same area. Since the more players in an area, the more notifications each receives, this approach has the benefit of saving server bandwidth in those situations where it is generally scarcer. Furthermore, since the server still maintains the global state, the clients have no possibility to incorrectly alter the game state.

Unfortunately, this approach introduces a number of complications. The first problem is that griefers could cause major disruption in the game's quality of service by dropping or delaying notifications. Even worse, unless the server signs messages, griefers could confuse target clients, making them display incorrect information to their players. In a competitive PvP setting where players fight one another, this would be especially damaging.

Although signed messages could prevent forged packets in principle, in practice this scheme is not tenable. On a modern server, the cost of creating a 1024-bit RSA signature is certainly less than 0.005s. However, even if bettered by an order of magnitude or more, this is still too slow if that server must send out notifications for thousands of nodes.

5.2 Mutual Checking

The basic idea behind this approach is "you may not trust a single client, but you trust the consensus of multiple unaffiliated clients". The reason why a client should not participate in computing the global game state is that the client runs on the player's computer and might be tampered with. Reverse-engineering a client may be hard, but there is no way to completely prevent this. But one could exploit the fact that most of the players are honest and would report a cheater if they notice one, since he is disrupting their game experience. The more players that agree on the state of the game world, the less likely it is that anyone of them has cheated.

In this approach, every game client acts also as a server for a certain region of the game world, as a *Region Controller (RC)*. The game world is partitioned into different regions and every region is controlled by multiple controllers. Assigning RCs to regions is a task that is best performed on a server offered by the game provider. When a player logs into the system his client is assigned to a group of RCs which are currently responsible for the avatar's region. Furthermore, the server assigns a region where the player's computer will act as an RC. The login server also provides means to prevent non-subcribers from joining the game, thus enforcing the subscription business model.

Since players tend to gather in certain regions, the distribution of avatars over the game world may be uneven. This could mean that certain regions are overcrowded while others are rather empty. To prevent RCs from being overloaded by large numbers of players, the region size has to be dynamic. The server that keeps track of the regions tries to keep the number of players and the number of RCs per region within certain intervals by splitting and merging regions according to their size. After rearranging the regions it has to reassign RCs for the newly created regions.

The client of a player, whose avatar is located in a certain region, sends action requests issued by this player to all RCs which are responsible for that region. Every RC validates these requests and—if a request was legal—computes the next state of the region. Afterwards the RCs have to check wether they have arrived at the same state, e.g. by sending a hash of the current state to each other and finally sending necessary state updates back to the client. The client compares those answers received by the different RCs and updates its state accordingly.

This scheme addresses different kinds of threats. First of all, illegal action requests issued by a client are easily detected at the RCs. But even if a client colludes with one or more RCs, so that they will accept his requests, the other RCs will still detect a deviation of the game state when comparing it. Finally, colluding RCs that try to maliciously penalize a client will be detected, since the client will receive different updates from the malicious and the benign servers. Of course, malicious clients or RCs could hide their deviation from the correct game state for a certain amount of time by sending appropiate forged messages. However, at the latest, when the state is written to persistent storage, e.g. on a regular basis or when a client disconnects, any deviations will be either detected (since all RCs have to agree) or lost (and thus the cheating wouldn't have any effect).

As mentioned before, preventing leakage of information is difficult in a P2P system. But if a client is never an RC for its own region (i.e. the region where the player's avatar is currently located) he needs at least one colluding RC to get a glimpse on hidden information about this region. And if only the state of a foreign region is managed on his client, it makes it even less tempting to alter its state, since there is no direct benefit for the player.

To implement the approach as described above, some non-trivial problems have to be solved. Maybe the most difficult issue is consistency. First of all, inevitable deviations of the game state will occur due to network delays. Any disagreements on the current state of a region must be resolved through some kind of consensus, thus the consensus scheme must be loose enough to allow for certain deviations caused by latency but at the same time strict enough to prevent cheating with a sufficiently high probability. Again, any of the participants may be a cheater that tries to manipulate the result to his own advatage or, even worse, tries to make the participants come to different results, thus breaking consistency. This makes the consensus problem actually an instance of the *Byzantine Generals Problem* [16]. The solution to this problem generally introduces a significant messaging overhead. Unfortunately, cheat prevention and communication efficiency are conflicting issues. The more randomly selected RCs are responsible for a region, the less likely it is to coerce enough of them to perform a successful cheat. On the other hand, the more RCs have to agree, the more messaging overhead it will cause.

Another non-trivial challenge is the dynamic region assignment. On the one hand, regions of arbitrary shape are difficult to implement and may cause fragmentation of the game world, due to the constant splitting and merging of regions. On the other, regions made up of fixed size tiles (e.g. squares or hexagons) may not be flexible enough to handle a very uneven distribution of avatars, e.g. many people gathering in a single region.

5.3 Log Auditing

Rather than attempting to prevent cheating in real-time (and suffering synchronization costs), we could instead try to detect cheaters. The idea is, that if a cheater is detected, his account can be closed. Furthermore, the effects of the cheat, once detected, could be (at least partially) rolled back.

A primary concern with such a system is that hackers may possess multiple accounts stolen from honest players. By keeping an audit history, the actual criminal can be located by tracing the cheat to the beneficiary. Of course, this is very similar to real-world fraud. It might be expected that organized crime groups may form and attempt to "money" launder the benefits.

To implement such a detection scheme, we again propose a Region Controller. This time, however, there is only one RC responsible for a given area. When started, the RC receives the initial game state, a random-seed, and a unique log-file name from the central server farm. As clients join, they are provided with the log-file name.

The RC then proceeds to run a deterministic event-driven state-machine (EDSM). Clients send their commands as a signed sequence of packets, each referring to the last packet and including the log-file name. After a predetermined interval, or on RC log off, the current state and a digest of the signed player commands is returned to the central server farm. The whole procedure is illustrated in figure 1.

It is relevant to note that in a real system there may need to be several slave RCs which follow the current state. Their purpose would be to replace failing RCs in the P2P system. There need not be any synchronization overhead beyond forwarding the commands. Furthermore, due to the EDSM nature of the RC, it would also be possible to replay the server computation in case of a failure, relieving such backup RCs from needing to compute the state in real-time or in the non-failing case.

The clients in this system are identical to clients in a typical C/S system. Therefore, they have no further capability for cheating. Furthermore, the load of signing their player commands is not that high since even a low-end laptop can sign 1024-bit RSA 80 times a second. This should be significantly higher than the injection rate of player commands.

The RC, however, has several new cheating capabilities. Similar to the Mutual Checking (MC) RCs of section 5.2, the RC could collaborate with a client in order to provide that client with unfair additional information about the game state. Also, as in the MC scheme, the RC could introduce delays or disconnect clients when run by a griefer.

The new attacks of an RC include falsifying returned game state, falsifying player commands, and confusing clients. We believe that all of these threats are addressable. Each will be discussed in turn.

The first threat, falsifying game state, is where the RC reports a bogus game state, for example, too much experience awarded to a player. Since the entire RC is running

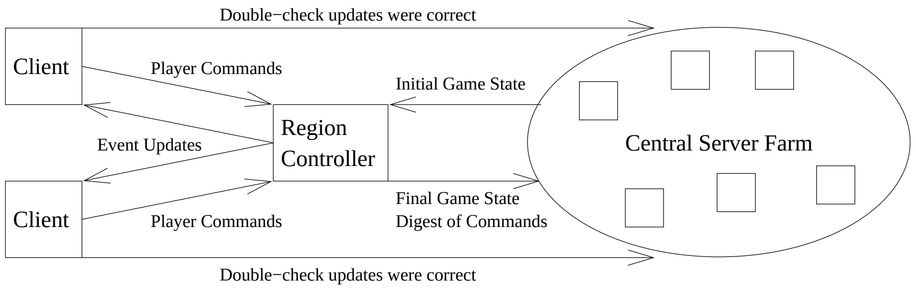

Figure 1: Message passing for audit checking

a event-driven state-machine (EDSM), the correct output can be reproduced by rerunning the EDSM on a trusted machine given the same initial state and player commands. Therefore, falsified game state can be detected.

This double-check can be pushed to several untrusted (but randomly chosen) clients as in the MC system. However, with this system there is no synchronization overhead during game-time. Furthermore, the double-check can also be performed in the trusted server-farm during periods of low activity or when a complaint is received.

The second threat, falsifying player commands, is where an RC alters the commands of one of the players, for example, forcing that player to trade-away valuable items. Since the player commands are signed, however, an audit will catch that the wrong command was provided in the digest. Finally, the chaining of player commands prevents intentional omission of some of the commands.

The final threat is where the RC sends confusing client information, similar to the distributed state dissemination scheme. However, unlike that scenario where the costs of signing state updates were too high; here they are not. The RC is only responsible for a few clients, and can therefore be required to sign his state updates. To save bandwidth on the server farm, these updates need only be stored by the receiving client. After a game session, a client can ask the trusted central server farm for a summary of the game state, and if it wishes to dispute the result, it can provide the signed logs of the cheating RC.

5.4 Trusted Computing

Trusted Computing (TC) is a highly controversial initiative of the *Trusted Computing Group* [20]. Discussing the social and political implications of TC is not subject of this paper. It is unlikely that the computer games industry will have a significant influence on whether TC will be deployed on a broad base or remain a niche market. Since the decision is up to hardware manufacturers and eventually the customer, game developers may some day find that TC is available on most computers and thus will want to make use of it.

Two things that TC could provide are of interest to providers of online games. First, the possibility that only software that is signed by the producer (and thus is trusted) may run on a TC enabled computer (or in a specially secured environment within the operating system). Second, the possibility that a TC enabled computer can prove its trustworthiness to other systems. The first thing would guarantee that the client software could not be manipulated, the second enables game providers to identify trusted game clients over the Internet.

Depending on which fraction of an online game's clients are trustworthy, different possibilities for game developers arise. The easiest case would be that all clients are trusted. A game could be completely distributed without worrying about possible attacks due to manipulated clients, which are the main problem with distributed computation on the client-side. Achieving consistency would be simplified compared to the mutual checking approach, since every game object needs only one trusted primary owner. Of course, multiple secondary copies are necessary to deal with the high fluctuation of nodes in a P2P sytem, but discussing availability issues is out of the scope of this paper.

Even if only a certain percentage of the clients are trustworthy, the game could significantly benefit from distribution. Only trusted clients would adopt the role of a Region Controller (RC) (see subsection 5.2). Players could be encouraged to enable TC features on their computer by lowering their subscription fees or offering them access to exclusive game content.

Trusted clients could even store (at least their own) player data locally, but since TC doesn't improve reliability this is not really an alternative to storing the data on a separate persistent storage. But maybe the frequency with which data is periodically written to persistent storage could be reduced. If a client crashes or loses its connection to the server, it could use the information stored locally when resuming the game. Only in case of severe failures where the local data is lost, need the backup state be restored from persistent storage.

Enforcing the subscription model with TC is straightforward: a single server run by the game provider may act as an entry point to the system. Since RCs always run on a trusted system, they will not allow players without an active subscription to join the game.

As the public discussion shows, Trusted Computing comes along with many dangers to the autonomy and privacy of the user. However, from an online game providers's point of view, it seems to be an ideal solution, provided that the security mechanims are functional and cannot be circumvented. An example for a platform that could be called "trusted" is the video game console Xbox from Microsoft [18]. It has built-in, hardware-based security mechanisms that should prevent untrusted client software from running. Nevertheless, sophisticated hackers were able to circumvent these mechanisms, since then a cat-and-mouse game has started between Microsoft, trying to ban modified Xboxes from their online service, Xbox LIVE, and those hackers, trying to protect their modifications from being detected. Fortunately, hacked Xboxes appear to be outnumbered, since applying the necessary modifications requires a certain degree of expertise which the average user does not possess. Regardless, this clearly shows one major pitfall of TC systems: if the security mechanisms are broken once, trust may never be fully restored. And with the fall of the security system, the success of an online game that relies on them may fall, too.

6. CONCLUSION

Due to the high upfront and running costs, developing and deploying MMOGs is a risky business. Without sufficient funding and good game experience, a MMOG may easily become a failure. The approaches presented in this paper show possible ways to reduce resource consumption on the server-side (and thus costs) by applying techniques from Peer-to-Peer systems. Special attention was paid to retaining control over the global game state, since this is necessary to avoid cheating and to lock out players without valid subscriptions.

7. REFERENCES

[1] Nathaniel E. Baughman and Brian Neil Levine. Cheat-proof playout for centralized and distributed online games. In *Proceedings IEEE INFOCOM*, volume 2, pages 104–113, April 2001.

[2] Ashwin R. Bharambe, Sanjay Rao, and Srinivasan Seshan. Mercury: a scalable publish-subscribe system for internet games. In *Proceedings of the 1st workshop on Network and system support for games*, pages 3–9. ACM Press, 2002.

[3] Michael Buro. ORTS: A hack-free RTS game environment. In *Proceedings of the International Joint Conference on AI 2003*, 2003.

[4] Fábio Reis Cecin, Rafael de Oliveira Jannone, Cláudio Fernando Resin Geyer, Márcio Garcia Martins, and Jorge Luis Victoria Barbosa. FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games. In *Proceedings of ACM SIGCOMM 2004*, Workshops on NetGames '04, pages 172–172. ACM Press, 2004.

[5] Fábio Reis Cecin, Rodrigo Real, Mŕcio Garcia Martins, Rafael de Oliveira Jannone, Jorge Luis Victória Barbosa, and Cláudio Fernando Resin Geyer. Freemmg: A scalable and cheat-resistant distribution model for internet games. In *8th IEEE International Symposium on Distributed Simulation and Real Time Applications*, 2004.

[6] Chris Chambers, Wu chang Feng, Wu chi Feng, and Debanjan Saha. Mitigating information expose to cheaters in real-time strategy games. In *Proceedings of NOSSDAV 2005*, 2005.

[7] Chris GauthierDickey, Daniel Zappala, Virginia Lo, and James Marr. Low latency and cheat-proof event ordering for peer-to-peer games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 134–139. ACM Press, 2004.

[8] Laurent Gautier and Christophe Diot. Distributed synchronization for multiplayer interactive applications on the internet. Unpublished, 1998.

[9] Laurent Gautier and Chritophe Diot. Design and evaluation of MiMaze, a multi-player game on the internet. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, page 233. IEEE Computer Society, June 1998.

[10] Michi Henning. ICE - Massively Multiplayer Middleware. *ACM Queue Magazine*, 1(10), February 2004.

[11] IBM. Butterfly.net: Powering next-generation gaming with computing on-demand, 2002.

[12] Intel. Digital media: Massively multiplayer online gaming, 2003.

[13] Daniel James, Gordon Walton, Brian Robbins, Elonka Dunin, Greg Mills, John Welch, Jeferson Valadares, Jon Estanislao, and Steven DeBenedictis. IGDA 2004 Persistent Worlds Whitepaper, 2004.

[14] Alex Jarett, Jon Estanislao, Elonka Dunin, Jennifer MacLean, Brian Robbins, David Rohrl, John Welch, and Jeferson Valadares. IGDA Online Games White Paper 2003, 2003.

[15] Bjorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM 2004*, March 2004.

[16] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[17] Emmanuel Lety, Laurent Gautier, and Christophe Diot. MiMaze, a 3D multi-player game on the internet. In *Proceedings of the 4th International Conference on Virtual System and MultiMedia*, November 1998.

[18] Microsoft. XBOX. `www.xbox.com`, 2005.

[19] Jessica Mulligan, Bridgette Petrovsky, Bridgette Patrovsky, and Raph Koster. *Developing Online Games: An Insider's Guide*. Pearson Education, 2003.

[20] TCG. Trusted Computing Group. `www.trustedcomputinggroup.org`, 2005.

[21] Bruce Sterling Woodcock. An Analysis of MMOG Subscription Growth. `www.mmogchart.com`, 2005.