AN ANALYSIS OF CHEAT PREVENTION IN PEER-TO-PEER MASSIVELY
MULTIPLAYER ONLINE GAMES

A thesis presented to
the Honors Tutorial College

In Partial Fulfillment
of the Requirements for Graduation
from the Honors Tutorial College
with the degree of
Bachelor of Science

Ilya Kogan

June 2010
This thesis has been approved by

The Honors Tutorial College

and the

School of Electrical Engineering and Computer Science

Dr. Shawn Ostermann

Department Chair, School of Electrical Engineering and Computer Science

Thesis Advisor

Dr. David Chelberg

Honors Tutorial College, Director of Studies

Electrical Engineering and Computer Science

Jeremy Webster

Dean, Honors Tutorial College
Massively multiplayer online games have been dramatically increasing in popularity since their inception. Unfortunately, the commercial technologies to support these games have remained largely the same. Similarly, peer to peer technology has flourished dramatically. The marriage of massively multiplayer online worlds and peer to peer is the next logical step in the development of online worlds. Many such systems have been proposed but few take cheating into account making these systems infeasible for commercial deployment. New research in cheat detection and prevention attempts to mitigate some of these issues. These cheat prevention methodologies have advantages and disadvantages in their approaches and implementations and are not entirely complete. This thesis evaluates the various cheating scenarios and prevention techniques, compares the most prominent of these systems, and briefly introduces a new Hybrid topology for a Pseudo-P2P system for future consideration and development.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>4</td>
</tr>
<tr>
<td>List of Figures</td>
<td>6</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>1.1 Traditional Online Games</td>
<td>7</td>
</tr>
<tr>
<td>1.2 Massively Multiplayer Online Role Playing Games</td>
<td>8</td>
</tr>
<tr>
<td>2 Current Solutions</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Centralized Servers</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Traditional Approaches</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Peer-to-Peer</td>
<td>14</td>
</tr>
<tr>
<td>3 Cheating and Prevention</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Types of Cheating</td>
<td>20</td>
</tr>
<tr>
<td>3.2 Cheat Prevention</td>
<td>25</td>
</tr>
<tr>
<td>4 Pseudo-P2P</td>
<td>30</td>
</tr>
<tr>
<td>4.1 Complete Hosting</td>
<td>30</td>
</tr>
<tr>
<td>4.2 Player Hosted</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Hybrid Hosting</td>
<td>32</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>33</td>
</tr>
<tr>
<td>Bibliography</td>
<td>36</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Simple World Server Cluster ........................................ 14
2.2 The Spline Network Model ............................................. 17
3.1 Taxonomy of Different Types of Cheats ............................. 20
3.2 A Small Exchange Among Three Players in a P2P Setting ............ 24
3.3 Simplified Data Access Pattern in Game Software ................... 24
3.4 Consensus Validation Example ...................................... 25
1 Introduction

1.1 Traditional Online Games

Traditionally, computer gaming has been constrained by bandwidth and processing requirements, limiting the sizes and experiences to smaller, static fictional areas. The number of people interacting at one time has also been considerably limited. Even the latest incarnations of such traditional games as **CounterStrike** limit the number of players per instance to 64 [Cou, 2004].

Despite these relatively simple limitations, the “game mechanics” of most traditional games remain uncomplex. For example, the aforementioned **CounterStrike** is a “first person shooter” where the goal is simply to hit an opponent with enough shots to incapacitate him or her. **StarCraft** is considerably more complicated: essentially, each player has a wide selection of different kinds of virtual soldiers and factories that must be used strategically to overcome other players’ armies. This game, and most like it, allows a maximum of *eight* players [Sta, 1998], limiting the kinds of interactions that can occur between players.

While CounterStrike and StarCraft are fairly old games, many newer games are largely similar in their multiplayer aspects. Any game that is capable of hosting large numbers of players becomes a Massively Multiplayer game. For traditional games, the current technology works and has thus remained largely unchanged.
1.2 Massively Multiplayer Online Role Playing Games

Within the last decade, bandwidth and computing resources have reached a point where incredibly interactive online experiences have become possible. These experiences appeared in the form of Massively Multiplayer Online Role Playing Games, or MMORPGs. These games come in numerous different styles but are highly similar at their most basic level. Players are given an avatar that represents them in the game world. That avatar gains various attributes, items, and abilities as the player progresses through the game. These statistics remain persistent regardless of how many times the player may leave or rejoin the game. Additionally, the game world is always operating. There is no such thing as an “end” to the game nor a notion of “restarting”. The idea is to create a fictional virtual world that players may explore.

A typical MMORPG will allow several thousand players some limited interaction, and several hundred players very complex and involved exchanges. In EVE Online, players participate in massive space battles involving hundreds of highly specialized and vastly different ships [EVE, 2003]. The outcomes of these battles could influence the entire future of the game and persist for the life of the game. EVE Online continually expands the kinds of interactions that players may have. Regions of space may be claimed and controlled by players. These claims can then be contested by others in the form of large confrontations involving hundreds of players simultaneously. All of this world state is guaranteed to persist for the foreseeable future\(^1\). It is unacceptable to login to the game only to find that assets have disappeared (except for in-game mechanics, such as theft or destruction). Countless

---

\(^1\)Massively multiplayer games have no projected end. Typically, most publishers will attempt to maintain their game for as long as it remains profitable. There is, however, no guarantee that the game will remain operational forever.
hours are spent by players in claiming, colonizing, defending, and battling over regions of space. A single cheat can have profound effects on the community. In 2004 an EVE exploit was discovered that allowed players to exploit a bug in game mechanics and generate large amounts of in-game currency for nothing. Due to EVE’s vast market system, this considerably changed the value of certain high-demand items. After the exploit was finally fixed in 2008 and thousands of player accounts were disabled, the market for those goods went into complete chaos. Player speculation about the sudden rise in value of the affected items caused several to attempt to buy up as much as possible at the reduced cost in order to resell at an expected higher value [Kokko, 2008].

While these games provide previously unparalleled experiences for players, they are continually limited by network bandwidth, processing power, and data storage, just as their predecessors. Several solutions have been proposed to attempt to alleviate the cost, with the most promising being a complete Peer-to-Peer solution that allows the game players to donate their own bandwidth and processing capability to assist in running the game. However, in a peer-to-peer situation, control of the game state is left up to players’ computers. Since players have complete control over their computing environment, they may be able to modify their game software to cheat and gain an unfair advantage over honest players [Kabus et al., 2005]. Fortunately, there may exist possible solutions to some or all cheating problems allowing truly massive online worlds to exist and thrive.
2 Current Solutions

2.1 Centralized Servers

Most traditional internet games require a central game host, or server, to run the game world and mediate between client requests. For example, in CounterStrike, a player chooses to host a game and thereby becomes the server. Any action other players perform is first sent to the server. The server calculates the new state of the world and sends it to the other players. Unfortunately, since the server must transmit new game states to each player whenever any player performs any action, the number of data pieces to transmit increases polynomially with the number of players. For example, if 10 players are racing around a track, each time any player changes direction, that triggers 1 message from that player to the server and 10 messages from the server back to all other players. If all 10 players begin turning simultaneously, each turn causes 1 message to be sent from each player (totaling 10 messages); each message then triggers 10 messages from the server, totaling 100 messages. Just 10 interacting players causes 100 messages to be sent from the server on each update. This is a minor problem when the number of players remains small but as it increases into the thousands or hundreds of thousands, as is desired for truly massive online worlds, the penalties incurred become unmanageable. While most MMORPGs deal with these issues by segmenting players into fairly distinct locales designed to limit their interaction, the goal is to allow more players complete interaction while maintaining low latencies. Higher latencies substantially lower the qualities of player experiences in multiplayer games.
Latency

Latency is the measure of delay between when a network application sends a request and receives a response. Many online services remain usable and even enjoyable at latencies nearing five or six hundred milliseconds. Sending a request for data from a website and having to wait almost one second to receive a response might be slightly irritating but is certainly tolerable. For a highly interactive game where any particular second there may be hundreds of events being transmitted, having to wait a second for each becomes unplayable. In 2005, Dick et al performed a study where players rated their experiences in various online games with respect to several factors, including latency. According to the study:

In the impaired case, for most of the games the players state that they can tolerate network latency up to 150ms, which also concurs with measurements from the literature. The average is 118.1ms. While one can see increased variance for the average delay, there is no clear differentiation between first person shooters like Counter Strike or Quake and the real time strategy game or sports games [Dick et al., 2005].

As the number of players increases, the stress placed on the server and the network infrastructure of the company or player hosting the game increases until the average latency becomes intolerable. In order to accommodate an ever increasing amount of players in ever more complicated game worlds, several approaches have been attempted and utilized with different degrees of success.

2.2 Traditional Approaches

Typical games, such as first person shooters and real time strategy games, are designed for small player numbers so these games can manage with a central server approach. MMORPGs become more enjoyable as more players are able to interact, giving the impres-
sion of a large persistent world. Most MMORPG developers must “cheat” by creating the illusion of large numbers or interactions. Zoning, Sharding, and Clustering are some of the most popular methods of mitigating latency issues in MMORPGs.

**Zoning**

The online world can be separated into distinct zones, each on separate servers. As players leave one zone and enter another, they hop from one server to a different one. These servers can be arranged in such a way that they optimize network utilization by, for example, installing them in various areas of the world, geographically close to a subset of the game userbase. *World of Warcraft* does this fairly heavily with its instanced dungeons and physically separated continents [WoW, 2004].

A more advanced zoning concept is typically known as *locales* [Barrus et al., 1996] or *interest sets* [Steed and Abou-Haidar, 2003]. Physical areas are split up based on a player’s viewable region or objects a player is interested in\(^1\). These locales can either be split off on separate servers or, more commonly, define which events are sent to which players. This way, players who are unable to see each other will not get event updates for each other, despite being on the same physical server. The interest set concept allows a single server to accommodate many more players as the load no longer grows polynomially. However, since players may move freely about their space, there is a high chance that a large number of players will become interested in the same set of objects. The server can often only accommodate a small subset of it’s total player base sharing the same interest sets and latency quickly spikes to dramatic levels.

\(^1\)What a player is interested in is defined by the game and environment in which a player is interacting.
Sharding

Unfortunately, even separate zones must share a common persistent database to store world state and avatar information. In order to truly allow for “infinite” growth, most large games will implement a technique called sharding. The game world is essentially duplicated completely across multiple server clusters. Players are unable to cross between shards and each shard is physically and logically separate from other shards. These shards can be partitioned in even more flexible ways as they are completely separate from each other. While sharding does allow a particular game to accommodate an enormous number of users, it does not allow them to interact in one world [Waage, 2008]. Without the large, persistent virtual world, such a game is no longer massive and does not carry the immersion and intrigue that a single connected universe would.

Clustering

Clustering can be used to great effect to reduce computational load and storage capacity problems. The load of a particular game world, zone, or shard can be spread across several physical servers that have high throughput communication channels between themselves. However, while each server will handle a subset of the total players, they must communicate all player actions amongst each other.

In Figure 2.1, players are assigned a specific server to connect to. Each server shares any events received from players to all other connected servers over a high speed backplane. As the number of players increases, more servers can be added to handle the load; however, it is far more difficult to upgrade the backplane and the connection to the
The amount of players of massively multiplayer games is growing constantly. Game developers are beginning to look to other solutions that allow as many players as possible to interact while making the game experience both enjoyable to the players and cost less to maintain for the developers. **Peer-to-Peer** has proven to be a huge success in online file sharing and digital media streaming, so much so, that game developers and researchers have begun focusing their efforts on utilizing *P2P* as the game platform to finally realize the “Massive” in Massively Multipalyer Online Games.

### 2.3 Peer-to-Peer

Peer-to-Peer solutions allow game developers to move most of the game processing and bandwidth load to their players, not only reducing costs, but also allowing for truly
massive online worlds that are impossible with conventional load balancing technology. P2P
has seen immense growth in file sharing systems like BitTorrent [Cohen, 2008]. The idea
for BitTorrent arose from the problem of several thousand, or more, people attempting to
download one enormous file. If the file was hosted on one server, download rates would
slow down as more people began downloading the file. BitTorrent typically requires people
who already have parts of the file to send it to others who need those parts. As more parts
get spread around the “swarm”, the download speed increases as the number of “servers”
with desirable pieces increases. So, unlike with traditional methods, the download speed
actually “increases” as the number of downloaders increases.

Peer-to-peer networking technology is realized to great effect in the Skype com-
communications network. Skype is a messaging and calling application similar to AOL Instant
Messanger, MSN, or Yahoo IM. A user maintains a list of friends and is able to send text
messages instantly to any friend that is currently connected to the network. Skype enhances
this capability by allowing real-time voice and video communication with friends, as well as
the ability to dial ordinary telephones through the Skype network. Skype achieves very high
voice and video quality in large part due to it’s P2P infrastructure. Most data is exchanged
between peers without the mediation of a central server. Instead, certain Skype users\footnote{Any user with a publicly accessible IP address and sufficient computing resources may become a SuperNode [Baset and Schulzrinne, 2006].} are
elected to act as supernodes, correlating and transmitting data from many normal users
across the Skype network [Baset and Schulzrinne, 2006].

In a P2P MMORPG, similarly to Skype, players act as both the clients and the
servers. Rather than sending data to one or more servers that must send send that data
to all other players, players send data directly to each other. However, most players have residential speed internet connections and modest, consumer computers. They may have enough power and bandwidth to converse with 10 or 20 people simultaneously, but as that number increases, their capacity is very quickly exceeded. Games are highly interactive, requiring rapid and low latency response to user actions, making them considerably more complicated. Cheating aside, game players must be completely synchronized amongst themselves. If a player’s client becomes out of sync with others, it will become confused and could possibly crash. Fortunately, several approaches have been proposed by research communities as viable P2P Virtual World frameworks.

**Diamond Park/Spline**

*Diamond Park* is a proof-of-concept P2P system which implements an interactive park that participants may tour using bicycles. Diamond Park is implemented on top of Spline, a P2P game framework created specifically for Diamond Park. Spline uses a *Distributed World Model* that behaves like an *Object Database*. All world objects are stored in the World Model and all applications and clients interact with the model by reading and writing data from and to it. Initially, this appears similar to a Central Server approach; however, Spline maintains P2P-like scalability by copying the entire model to all applications and synchronizing it periodically. While every application contains an exact copy of the initial World Model, each one only synchronizes the pieces of the model that their user is currently *interested in*. In order to combat user processing and bandwidth limitations, as mentioned in the previous section, Spline utilizes special servers in between some users [Waters et al., 1996]. It is important to notice some key terms used in most P2P implementa-
tations: interest sets and proxy servers/supernodes/relays. The special Spline servers are little more than SuperNodes and are typical to most P2P implementations.

![Figure 2.2: The Spline network model [Waters et al., 1996].](image)

**MASSIVE-3**

The designers of MASSIVE-3 chose to implement the entire system as objects in their environment. Even ownership change and object update events appear as “objects” in the system. Like Spline, a particular player “owns” some objects and thus has sole access to update them. However, another player may request ownership of a particular object and then begin to update that object. Unfortunately, since only one particular client may own an object at one time, if a large number of players are interested in that object, the owner must handle all of their update requests, causing this object to be incapable of load balancing and forcing all load onto the owner. MASSIVE-3 does not provide any mechanism to deal with slower links that may be easily congested by medium to large numbers of clients [Greenhalgh et al., 2000].
Pastry, PAST, and Scribe

Pastry, PAST, and Scribe are a set of frameworks that form a general purpose P2P network overlay that provides routing, data storage, and event delivery functionality [Hampel et al., 2006]. Pastry is a Distributed Hash Table (DHT) based routing overlay that guarantees efficient transmission of messages to any host on the network and that messages will arrive at their destination [Rowstron and Druschel, 2001a]. Like most DHT systems, Pastry simplifies the process of storing data in a distributed manner by presenting a programmer with simple routines to store and retrieve data. While the implementation of DHTs varies, they all provide the ability to put and get data and make varying guarantees regarding that data. Pastry, specifically, guarantees that data will be replicated to a configurable amount of nodes (for redundancy) and that any get operation can be achieved with the shortest path to the node that holds that data. PAST is a distributed file storage system developed on top of Pastry. Files are replicated across the network and the replicas stored numerically “close” to each other [Rowstron and Druschel, 2001b]. Scribe is an application level multicast infrastructure that allows events to be delivered to any member of a pastry network that has subscribed to those events3 [Castro et al., 2002]. These three systems are then married to achieve scalable region based load balancing in a P2P architecture [Hampel et al., 2006]. Using Scribe’s publish/subscribe model, it is possible to extend this architecture to also support interest sets and guarantee persistent storage using PAST.

Many P2P games share common principles, especially those of Interest Management, Logical Multicast, and SuperNodes [Knutsson et al., 2004]. Since so many of these

---

3For example, if a group of players are interested in a park, they can each “subscribe” to that park and get messages related to it without having to send those messages to all players in the system.
systems are highly similar, most of them are vulnerable to the same kinds of cheating methods and lend themselves to similar solutions. Unfortunately, cheating is seldom addressed. Therefore, there does not seem to exist a feasible option for commercial MMORPGs.

In order to satisfy commercial users, they must have some certainty that they have some control over the outcome of a game or an exchange between players. That certainty vanishes when it becomes possible for some players to cheat with no way for honest players to seek retribution. Each of these, and other, P2P systems is vulnerable to a myriad of cheating possibilities that are easily exploited due to the lack of a central authority. Typical games operating with a central server that is aware of the game state can easily check for and prevent cheating by relying on the central server to verify actions and remove players who attempt to cheat. Since P2P has no central authority, players can easily fool other players and the game system without any obvious method of detection or prevention.

3 Cheating and Prevention

Since so many peer to peer games are so similar and their designs are becoming more standard, the kinds of cheats and the approaches to achieving them also become highly similar. If these cheats can be classified and narrowed down to manageable categories, then more effective countermeasures can be developed. Fortunately, many cheats stem from a small amount of potential exploits in these game architectures.
### Classification of the various types of cheating

<table>
<thead>
<tr>
<th>Vulnerability ((-)es)</th>
<th>Possible Failure(s)</th>
<th>Exploiter(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inadequate In Underlying Systems</td>
<td>Person, Game Operator, Masquerade, Integrity Violation, Service Denial</td>
<td>Independent, Group Operator, Multiple Players, Operate and Player</td>
</tr>
</tbody>
</table>

- A) Cheating by Exploiting Misplaced Trust
- B) Cheating by Collusion
- C) Cheating by Abusing the Game Procedures
- D) Cheating Related to Virtual Assets
- E) Cheating by Exploiting Machine Intelligence
- F) Cheating by Exploiting Client Infrastructure
- G) Cheating by Denying Service to Peer Players
- H) Cheating by Exploiting Timing
- I) Cheating by Compromising Passwords
- J) Cheating by Exploiting Lack of Secrecy
- K) Cheating by Exploiting Lack of Authentication
- L) Cheating by Exploiting a Bug or Design Loophole
- M) Cheating by Compromising Game Servers
- N) Cheating Related to Internal Misuse
- O) Cheating by Social Engineering

**Figure 3.1:** Taxonomy of different types of cheats, impacts, and causes. [Yan and Randell, 2005]

### 3.1 Types of Cheating

Many of the cheating problems in games stem from the fact that the game client is installed and operated on a player’s own machine, a device to which the player has complete, unrestricted access. The player is able to see and modify any data on their machine given enough knowledge. Once a single player manages to circumvent protections and devise cheating techniques, those techniques will usually be packaged into simple programs
that can be used by less knowledgeable players to cheat as well. If these cheats are not repaired quickly, they grow more widespread and honest players become disillusioned with the game, causing player fallout. Several types of cheats are summarized and classified below [Yan and Randell, 2005].

The cheating taxonomy in figure 3.1 is fairly broad and not specific to P2P games. Several of these cheating methods are solvable through general means. For example, collusion, exploitable game procedure, virtual asset management and exchange, and bugs are all game design issues. Internal misuse, social engineering, and compromised passwords are typically a policy or education issue and can be solved by an informed user base and do not require special consideration in P2P situations. Denial of Service is a fairly large and unsolved problem; the effects of which are not limited to cheating. Latency issues caused by external applications or other machines on the same networks can cause unintended denial of service. Coping with extreme network utilization is a vastly different topic and covered in most P2P architectures.

Many of the cheats presented above can be described as Cheating by Compromising the Client Software. Most multiplayer games deal with many of these cheats by simply not trusting the client software and making all validations on the server. Secrecy is guaranteed by only transmitting information to the client that they are allowed to report to the user. Authentication is performed by a centralized login server. Event ordering is validated on a central server and cheating attempts are either repaired or the guilty party is disconnected. These are the core problems of P2P architectures. They do not possess a central server that is capable of doing this kind of validation. Adding a system like this would require all
players to send all of their updates to the server and defeat the purpose of the architecture. Additionally, since, in a P2P architecture, the game clients are also game servers, *Compromising the Game Server* can be merged into compromising the client. Finally, modifiable client infrastructure is a different kind of misplaced trust, trust in the operating system or network equipment.

While it is undesirable to maintain a centralized server for event verification, many networks already maintain a centralized authentication system for a disparate set of machines. The key is that authentication occurs relatively rarely and only the notion that a client *has authenticated* must be preserved and validated. Kerberos is a fairly standard and widely deployed authentication and authorization system that uses a *cookie* to maintain authentication state. This cookie is a set of cryptographic information that can be used by servers to validate that a client has in fact authenticated and what kinds of credentials a client possesses [Neuman and Ts’o, 1994].

The core cheating methods, then, that must be considered and solved by a P2P architecture are:

- Event Timing and Disruption
- Lack of Secrecy
- Compromised Client or Infrastructure
As before, secrecy is commonly maintained by simply not sending information to clients who should not have access to that information. The position of a player who is invisible, for example, should not be known by other players\(^1\). In a traditional architecture, a central server “knows” all of the secrets that must be maintained. Therefore, in the event of a dispute where a cheating client claims an untrue secret (having an incredibly rare and powerful weapon, for example), the game server is able to step in and clear up the situation. Since there is no game server in a P2P scenario, no one is authoritative on “secret” state. In order for any form of validation to work, all clients must know exactly who might be holding the aforementioned sword. The game client is then trusted to hide that information from the player.

Once a game client is compromised, however, it can be reprogrammed to perform or not perform anything it was designed to originally do, causing event timing issues and disruption as well as secret information disclosure.

In Figure 3.2, the player has modified his game so that he can selectively ignore a player entirely, or only select messages. For example, if the two honest players are working together and a cheater wants to gain an advantage over one of them, the cheater can completely ignore one player, while ignoring any “disadvantageous”\(^2\) events from the other. This way, the cheater is able to have free reign over this exchange while the victims have no avenues available to defend themselves.

In Figure 3.3, the game software is modified such that the player is able to see information which would otherwise be unavailable. The *sensitive information* might be,

\(^1\) That is, unless game mechanics allow for that behavior. Exactly what is secret is defined by the particular game but things that are secret should not be known to those from whom the secret is being kept.

\(^2\) A disadvantageous event can be, for example, an event which would cause the cheater’s avatar to be incapacitated.
Figure 3.2: A small exchange among three players in a P2P setting. In 3.2(b), the red player is ignoring the messages from one, and selectively ignoring those of the other.

Figure 3.3: Simplified data access pattern in game software. In 3.3(b), the User Interface is made to access private data directly, rather than through the accessors.

for example, the location of another player that the cheater is not able to see. Utilizing this information, the cheater is able to gain an upper hand. If an honest player attempts to sneak up behind a cheater, that cheater already knows that player is there when the cheater ordinarily would not be have access to that information. Since the honest player’s game must send notice of their movements in advance\(^3\), the cheater is able to gain insight into a player’s intention well in advance of any offensive action by that player. These road

\(^3\)To generate quiet footsteps or shadows, for instance.
blocks are highly critical in preventing the commercial acceptance of large scale Peer-to-Peer MMORPGs. Fortunately, it may be possible to prevent some, or maybe all, of these cheating methods.

### 3.2 Cheat Prevention

The lack of any cheating defenses in P2P game frameworks makes them unusable for commercial applications. There exist several options to help prevent or mitigate the threat of cheating.

**Consensus**

Cheat prevention by Consensus allows for peers to mediate each other, deciding whether or not certain other peers are cheating. Kabus et al calls this *Mutual Checking* and summarizes the idea along with several problems [Kabus et al., 2005].

Figure 3.4 is a simple example of this method of cheat detection. Several players will be automatically elected as mediators for a certain locale, or group. Typically, players

![Consensus validation example](image)
are elected who are explicitly not interested in the region or objects they are mediating for. This helps prevent information disclosure by trusting the mediators with content that they themselves have no vested interest in.

Every message being sent between the Cheater and the Victim is also sent to each mediator. The mediators know the current world state of that locale or group and validate these messages. If, for example, the Victim attempts to incapacitate the Cheater who then ignores that message and continues the exchange, the mediators will detect this and invalidate any future messages from the Cheater [Kabus et al., 2005].

This method has become fairly popular and is utilized in Hampel et al., which introduces Backup Mediators to assist in preventing a cheating mediator from affecting the region, whether or not the architecture believes the mediator is interested in that region. Izaiku et al. introduce digital signing of events to help ensure that messages actually originated from the correct system and user. This helps to guarantee that the world state the events are based on are genuine. Kabus and Buchmann elaborate on the mutual checking proposed earlier and discusses responsible node choice by super-peers controlled by the game developers. While one of the main objectives of P2P architectures is to reduce load on the game provider by utilizing player resources, introducing super-peers strikes a compromise between load balancing and cheat resilience.

While this approach seems fairly straightforward, there are several problems that must be solved, chief of which is cheating through collusion [Yan and Randell, 2005]. It is possible that a cheater will collude with a mediator to validate his actions, as in 3.4 on the left. This can be mitigated by adding more mediators, but, as more mediators are added
the latency increases as messages have to be sent to more players before becoming valid. There may also be situations in which only a small number of players, possibly only one, are interested in a particular object. In that case, there are not enough players to participate in consensus. For example, a player may choose to play alone in an area out of interest range of other players. They may then return to the community claiming to have gained a large number of assets or amassed large amounts of in-game currency. The greater community would have no way of verifying this as no one was present in that player’s interest area that can validate that claim.
Many popular games introduce the concept of stealth\(^4\). In EVE Online, a *Cloaking Device* may be fitted to virtually any ship that prevents detection by others (with many penalties). In World of Warcraft, the Rogue class can stealth, with much the same effect as a Cloaking Device. Consensus requires that events be sent to others which can then modify their client to reveal invisible players. If invisible players are *not* required to send events to mediators, then they can freely move about the world at any speed and bypass virtually any barrier as they are only required to submit to cheat detection while visible.

The most comprehensive solution that can be extended to solve most of the above problems seems to be to use cryptographic digital signatures to create a *public key infrastructure* (PKI) [Wierzbicki, 2006]. The PKI can be used to maintain integrity of public and private game states by digitally signing each and ensuring that signature is maintained. The signature is validated on a kind of responsible peer, ensuring that the various states and their manipulations are valid and make sense. The PKI allows for the existence of a *web of trust* leading back to a controlled party. That is, a game developer can create their *own* peer that they have trusted and that peer can then digitally sign game states. In order for any state to be valid it must be signed either by this peer or another that this peer trusts. This, in essence, is a super-peer that is able to validate and secure game communications and prevent cheating. Like previous solutions, however, this comes at a trade-off as someone must run these trusted peers. Unfortunately, cheat detection is not guaranteed as there is no solid overarching network architecture that completely prevents cheating. Instead, the aim is to detect cheating and prevent the cheaters from participating.

\(^4\)In most games, stealth also comes with penalties. Cloaking Devices in EVE prevent ships from using most of their capabilities, even moving, in some cases. In World of Warcraft, stealthed players move very slowly and cannot interact with players or objects without exiting stealth.
in the game. Extending this solution to a full network architecture that supports P2P while also providing PKI-based cheat validation is future work and is briefly discussed later.

**Securing the Software**

Mönch et al. have developed a novel method to completely secure the game software to prevent the player from tampering with it and mitigating most attempts to cheat [Mönch et al., 2006]. Central to their idea is the introduction of a concept called *Mobile Guards*. Mobile Guards are essentially small algorithms that are used to “ensure the integrity of the protection mechanisms” [Mönch et al., 2006, pp 2]. A game client for an MMORPG must continually receive data from servers and other players. Players may require that each client validate itself prior to being sent such data. In order to validate itself, the client may, for example, be forced to calculate the result of some algorithm based on a request from the other player. If that calculation produces the expected result, then the program has not been tampered with.

To ensure that the player does not modify the program in such a way as to bypass these *Mobile Guards*, they are installed such that they are necessary in order to access certain vital methods. For example, only the Mobile Guard contains the memory address of the game code necessary to communicate with other players. While the player does indeed have complete control over their machine, key to this method is that new Mobile Guards are sent to the client too frequently to allow the player enough time to break them. Ideally, by the time a player manages to defeat one Mobile Guard, a new one will already be necessary for validation. The paper admits, though, that generating Mobile Guards is not trivial. It must be possible to generate and transmit new mobile guards to all players.
before they manage to break old mobile guards. In a P2P setting, all players must activate their new mobile guards and deprecate their old ones at the same time, or they risk false positives when one player has simply not gotten the new mobile guard yet.

While there do exist several solid cheat prevention methods, none of them fully protect P2P games. Each one has some deficiencies that must be solved before they become acceptable for commercial use.

4 Pseudo-P2P

A fully P2P, cheat resistant, massively multiplayer game is very difficult, if not impossible, to achieve without sacrificing game design. The idea of a hybrid solution has been proposed that allows some degree of balance between company hosted servers and P2P scalability [Wierzbicki, 2006]. While “coordinators”\(^1\) may be players or company provided servers, there is currently no specific definition of how a hybrid network could be built. Since coordinators do not require significant amounts of stored data, it is fairly trivial to create them on commodity servers, perhaps servers that are already installed at convenient locations. Several different topologies are feasible, each with their own uses, strengths, and weaknesses.

4.1 Complete Hosting

Commercial Content Delivery Networks like those hosted by Akamai are highly prevalent throughout the internet. These systems distribute primarily static content across

\(^1\)A coordinator is much like the trusted nodes or mediators introduced previously; however, they also may include the ability to perform other secure tasks as well.
the world using thousands of servers. Such a “CDN” would be ideal for distributing this load [Saroiu et al., 2002]. Since a coordinator need only carry it’s private key and have storage capacity for game state and player private state\(^2\), it can be installed on a CDN node or a similar server. Several CDN nodes can be distributed across the world to host such games. Since Pastry can use routing hops to determine closeness, it guarantees that the closest CDN server will be chosen for validation requests. A proper global distribution of servers will allow for optimum load sharing and provide validators that are close to large amounts of players. This requires the largest financial commitment by game developers as they must provide or pay for all hosting servers.

### 4.2 Player Hosted

As stated before, not all players are interested in cheating. Honest players could be used to host coordinators and provide a complete P2P solution that would work very similarly to Skype. Players would be selected due to both their computing power and trustworthiness to become coordinators and validate actions of other players in the game. How exactly to determine whether or not a player is trusted is a subject of current research. Several papers ([Gupta et al., 2003, Damiani et al., 2002, Mengshu et al., 2005] and others) have proposed many different approaches to trust management, establishment, and tracking. Both reputation based (in which a certain reputation threshold determines trustworthiness) and non-reputation based systems have been discussed to great extent. How exactly to determine a player’s trustworthiness is left to previous and further research, as needed. Player hosted systems face several drawbacks including lack of centralized authentication\(^2\) Art and other assets, typically the largest components of any game, are not required.
(necessary for subscription fees), lack of guaranteed persistent storage, and trouble handling low user volume.

Many typical MMOs offset the immense maintenance, processing, and bandwidth costs by charging a monthly subscription fee for each player. Charging players in this manner requires being able to store individual information about each player and authenticating their access. Player-hosted topologies do not have central servers and, while it should be possible to create one solely for authentication purposes, the load on that server will increase with player numbers and still require maintenance. In addition, since player data will be stored on client machines, whether on the player’s own or on some trusted machine, those machines may either fail or simply not be available when a particular player chooses to play. This problem can be mitigated by storing player content on a player’s own machine with backups on several trusted servers, but, should a player’s machine fail, the recovery of their data is at the mercy of the players storing it. If a player ceases playing the game for a long period of time, it’s possible that all trusted players with backups of their data have also left in that time-frame. Such a player would have no way to recover their data. Finally, since players are the sole authority on valid player and game state, limited player numbers causes there to be limited or no coordinators. Without coordinators, the game could not function in a secure way. It doesn’t seem very feasible, then, to implement a player only hosted system.

4.3 Hybrid Hosting

In order to solve the issues with complete hosted and player hosted solutions, the strengths of both can be utilized in a hybrid system. Player and provider hosting could be
married to provide an approach where the game developer provides coordinators when necessary and players provide infrastructure when they are able. This allows the network to continue functioning regardless of the active player makeup. Depending on the ratio of active trusted players to untrusted players, there may be too few developer-provided peers to act as coordinators. This would cause the available coordinators to become overloaded and would likely increase game latency. While this is still a problem, it is of less significance than that of cheating as all players will experience this increase in latency maintaining a fair atmosphere. The problem itself can be mitigated by upgrading coordinators or adding new servers. The cost of upgrading distributed machines or adding new servers is considerably cheaper than maintaining large clusters as most large MMOs do today.

The coordinators being provided by the game company can never go offline (barring technical failure) and thus continue to offer their services to players at all times, regardless of how many untrusted players there may be. Private player state can always be stored and backed up in a deterministic way preventing the possibility of lost data. Authentication can be performed by the closest developer coordinator and there will always be some machine to sign state and validate actions. Again, a hybrid solution will certainly incur some cost to the developer but it is significantly less than for a complete hosting solution or for a traditional non-P2P game.

5 Conclusions

Traditional load balancing techniques are inadequate to support truly massive Massively Multiplayer Online Role Playing Games. Peer-to-Peer networks are the next step
in creating these vast, interactive virtual worlds, despite their current flaws. They allow for near-infinite expansion of the player base without putting a strain on the game developers and maintainers. Due to the decentralized nature of P2P networks, however, they are easily targeted by cheaters. Since it is fairly trivial for a player to ignore actions by others, or send illegal actions to others, some kind of cheat prevention mechanisms must be in place. This is a non-trivial problem because to prevent cheating in a P2P situation, the players’ insecure machines must usually be relied upon.

*Consensus* allows several players to validate the actions of another in order to, hopefully, detect and prevent cheating before it affects the game world. Due to its reliance on player participation, the lack of players, or collusion amongst players can cause this system to break down. *Securing the Software* allows the previously insecure game program running on the player’s computers to actually be secured. Any tampering is detected immediately and communication from broken software is ignored. Keeping all game programs synchronized with the latest protection mechanisms is a non-trivial task and an open problem, making even this cheat prevention method incomplete.

*Trust Enforcement* uses the proven public key infrastructure to secure communications and authenticate hosts. The addition of trusted peers dedicated to validation helps mitigate issues relating to a lack of validators or hidden data.

Several interesting topologies can be used to implement a trust enforcement system with their own strengths and weaknesses. Due to the flexibility of granting and revoking trust based on signed certificates, game developers can choose to deploy their own servers to validate and sign player communication or designated players as trusted by granting them
an appropriate key. Finally, a hybrid solution utilizing both developer resources and player resources to validate game state and communications utilizes the best of both topologies and gives the developer freedom to optimize their topology to best fit network load, their customer base, and the ever changing conditions of the game world.

Heavy research continues to be performed relating to cheat prevention in massively multiplayer online games with trust enforcement being the most workable solution. Under which conditions a player could earn trust and how that trust could be revoked with minimal game impact is a topic of intense, ongoing work. Additionally, the scalability and feasibility of the various topologies and possible use cases for trust enforcement must be tested and evaluated.

The prospects for immense, connected, and immersive virtual worlds are improving with every research effort. Further exploration of this problem space could allow commercial deployments to be a possibility. Connected, decentralized systems as accessible as to the World Wide Web could be a reality, allowing incredibly rich and interactive experiences when participating in games, collaborating on important tasks, socialising, or simply exploring our society’s vast network of information.
Bibliography


Addressing Cheating in Distributed MMOGs. In *NetGames '05: Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 1–6, New York, NY, USA. ACM.


[Saroiu et al., 2002] Saroiu, S., Gummadi, K. P., Dunn, R. J., Gribble, S. D., and Levy, H. M. (2002). An Analysis of Internet Content Delivery Systems. In *OSDI ’02: Proceedings of the 5th symposium on Operating systems design and implementation* Due to copyright restrictions we are not able to make the PDFs for this conference available for downloading, pages 315–327, New York, NY, USA. ACM.


