Towards Public Server MMOs

Chris Chambers Wu-chang Feng Wu-chi Feng
Portland State University
chambers@cs.pdx.edu wuchang@cs.pdx.edu wuchi@cs.pdx.edu

ABSTRACT

While massively multiplayer on-line games (MMOs) are enormously popular, their use of the client-server architecture causes them to suffer from scalability issues and high maintenance costs. In contrast, the public server architecture employed by most first-person shooter (FPS) games scales more easily by relying on user-supplied hosting and user-generated content, but lacks persistence between servers that is required in the MMO genre. This paper examines an architecture that leverages the resources of the public server approach to support a scalable, persistent MMO.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and Application-based Systems

General Terms

Design

Keywords

Online games, MMO

1. INTRODUCTION

On-line games of every sort are enormously popular with the most popular selling millions of copies and having more than 100,000 players concurrently active [1, 2, 3]. Hosting a popular interactive application with high concurrent usage and constant uptime is a challenging task. There are three main architectures used to host on-line games: peer-to-peer, client-server, and public server. In *peer-to-peer* (P2P) on-line games, the game publisher ships each client a complete copy of the game and players connect to each other directly in order to play against each other. In this architecture, there is no central host and no peer is given special authority. Instead, the game is simulated independently by each peer.

Netgames'06, October 30–31, 2006, Singapore.

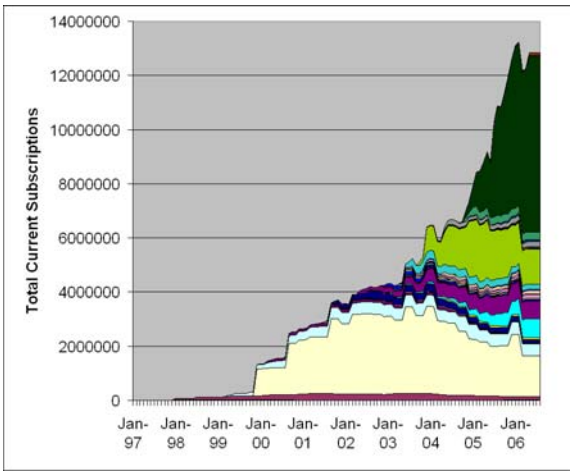

Figure 1: MMO subscriber growth over time by game

Such an architecture is typically used for head-to-head real-time strategy (RTS) games. In *public server* on-line games, the publisher sells the game client to players and gives away the server binary to anyone who wants to run a game server. Anyone can choose to put up servers or take them down as they see fit. Given this, the publisher's only involvement in the process is in tracking the servers that are up and helping to connect individual players to running servers. This is the most popular architecture for first-person shooter (FPS) games such as Half-life. In *client-server* on-line games, the publishers sell the game client to players, but host all of the game servers themselves. While the architecture gives the game publisher fine-grained control over content, it requires the game publisher to host and maintain a significant amount of computing infrastructure. This architecture is the most popular for subscription-based games with persistent content such as massively multi-player on-line (MMO) games.

Fueled by recent successes such as World of Warcraft and Lineage, MMO games have become particularly popular over the last several years. Figure 1 shows the steep growth in subscribers to MMOs since 1998 as compiled by Bruce

Game	User Content	Architecture	Persistent
Typical MMO		Client-server	X
Typical FPS	X	Public server	
Typical RTS		Peer-to-peer	
Second Life	X	Client-server	X
Neverwinter Nights	X	Public server	
PS MMO	X	Public server	X

Table 1: Game architectures

Woodcock [4]. As the figure shows, popular MMO games have several million subscribers. In an MMO, players pay a subscription fee to control a single avatar and adventure, socialize, or compete with thousands of other gamers in a persistent world. Although MMO games are some of the most lucrative games for game publishers, their use of the client-server architecture also makes them some of the most expensive to deploy and maintain.

One of the largest costs for a game publisher in running an MMO game is the hardware, software, and support costs associated with hosting the game in the client-server architecture. Because each server can typically handle several thousands of players concurrently, most popular MMO games require large server farms in order to support its large population of players. Such a cost does not exist in either the public server architecture where players host the game on their own dedicated servers.

In addition to hosting costs, persistent on-line games also require an enormous amount of content generation to keep subscribers playing. If the game is lacking in novel activities or progression, avid gamers will become bored and unsubscribe. In stark contrast to a linear single-player game that can be mastered in dozens of hours, publishers would like MMO players to be able to enjoy their game indefinitely, regardless of how much time they put into the game. Thus, continuing content development is critical to the longevity of a MMO. Unfortunately novel content is typically developed by the publisher at a slower rate than it can be played through, resulting in bored gamers. Some games, such as Second Life and public server games like Half-life are designed to allow user-generated content or mods that extend the lifetime of the game considerably.

Because the public server architecture can potentially minimize the hosting and content generation costs of an MMO, this paper examines an approach for running a persistent MMO game using the public server model. Table 1 summarizes the state of on-line gaming with respect to user-generated content, architecture and persistence. Only Second Life allows for user generated content along with a persistent world, but it takes place in a client-server architecture. Our solution to the hosting and content creation challenges posed by MMOs is to move to a public-server architecture and allow users to generate content. We call this architecture Public Server MMO (PSMMO). The intended goal of PSMMO is to inexpensively scale hosting resources and content generation with the number of users playing the game. This PSMMO architecture introduces some fundamental challenges that this paper addresses: (1) trust and authentication (2) content creation and (3) content distribution and exchange. We address these issues using a combination of incentives and public-key cryptography.

The rest of the paper is outlined as follows: In Section 2 we discuss work related to ours, in Section 3 we discuss user resources, in Section 4 we present our PSMMO design, and in Section 5 and Section 6 we share our conclusions.

2. RELATED WORK

A number of solutions have been proposed in recent years to address the problem of hosting MMOs. One solution is to dynamically host games in an on-demand fashion and take advantage of economies of scale and differences in gaming popularity in a centralized or grid-based fashion [5, 6]. We believe these efforts to be synergistic with our effort to harness user resources.

Another approach is to use clients to form a P2P network responsible for gameplay computation as well as storage [7, 8, 9]. Our solution is not P2P, but rather public server, which incurs certain trade-offs. While P2P networks are an attractive solution due to their scalability properties, they typically introduce increased latency for multi-hop tasks such as peer routing. Additionally some game players are unable or less able to participate in a P2P network due to firewalls and discrepancies between upload and download speeds in home networks.

3. HARNESSING USER RESOURCES

While there are inherent risks in using the public server architecture to host a persistent MMO game, the clear strength of the approach is in its ability to harness player resources. In particular, players have an abundance of hosting infrastructure resources they are willing to contribute and content generation resources they are willing to commit if given the proper tools and incentives. In this section, we quantify the extent of these resources and argue that they are significant enough to justify a public server MMO approach.

3.1 Quantity

The merits of our design rest on the willingness of gamers to contribute their resources and creative energy to the betterment of a compelling game. There is some empirical evidence that this willingness exists for popular public server games. Figure 2 shows the cumulative distribution function (CDF) of the percentage full of all Half-life 2 servers as polled every 10 minutes from 05/24/2006 to 05/29/2006. This figure shows that even though Half-life 2 is an enormously popular game with over 100,000 concurrent players at any moment, the user-contributed server resources are 70% idle. This represents over 18,000 idle Half-life servers.

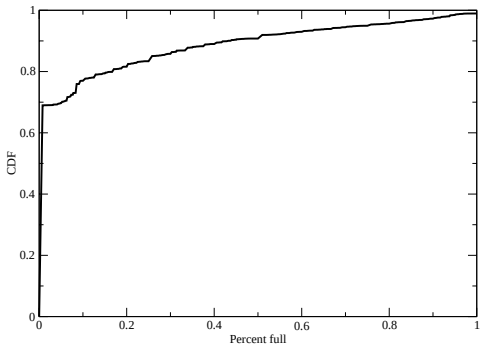

Figure 2: CDF of public server utilization for Half-life 2. 70% of all servers are empty.

In addition to being willing to contribute server resources, players are also keen to contribute game content. In a public server game the players typically have access to the art for the game in addition to the server binaries, and so publishers often allow players to easily modify and extend the art and gameplay. As examples of user interest in content creation, the developers for Half-life and Neverwinter Nights have released 6 and 7 official gameplay additions and variants, respectively. Their user bases however have created at least 492 and 4372 gameplay additions [10, 11]. As another example, Second Life is a MMO that has minimal developer content; most of the gameplay is emergent behavior generated by user behavior and user-driven content creation. Linden Labs estimates that of the 80,000 aggregate hours per day users spend in Second Life, 25% of user time is spent on content creation [12]. One concern is that users generate too much content and that it may be challenging to locate the high-quality additions. While a content rating system would help to solve this problem, we believe that the social nature of persistent on-line games will cause gamers to gravitate towards compelling content without any publisher-sponsored rating system.

3.2 Quality

User resources may be plentiful but their efficacy is less easily quantified. Regarding user hosting, the computational requirements for hosting MMOs are not well known due to the closed nature of successful industry games. While users are willing to contribute servers for games with dozens of clients such as Neverwinter Nights, it is unknown what sorts of gameplay sacrifices would be required to allow user machines to host compelling MMO gameplay. We do not address this issue in this paper and instead assume that any desired gameplay can be hosted in some way by user-contributed server resources. We also do not address public server reliability or response time fairness to clients, and instead assume that whatever service a given client requires is replicated in depth, as is the case for Half-life players searching for popular varieties of gameplay.

Regarding quality of content we note that user content can be extremely popular: the Counter-strike and Capture the Flag modifications for Half-life and Quake have been more successful than any publisher-generated content. In addition, user-generated content is the driving force behind the Second Life MMO.

As successful on-line games can be profitable and very important to users, there are certain legal and ethical challenges inherent in harnessing user resources for profit, such as intellectual property rights and server liabilities. We believe these issues are important, but we also believe that users enjoy contributing building blocks to their gaming world, and assume that some legal or monetary resolution for these issues can be achieved that enables the harnessing of user resources for scalable persistent worlds.

4. DESIGN

We preface the presentation of the nuts and bolts of our design discussion with a more in-depth description of the tasks and motivation involved in playing a MMO. Then we present our design goals and the details of the PSMMO architecture.

4.1 MMOs and Loot

The generic case of MMO gameplay involves controlling a single avatar with a set of abilities and performing tasks in world that advance the power, possessions and abilities of the character. These gameplay tasks can vary widely based on the genre of the game, from rescuing hostages to competitive fighting to killing monsters. Successful completion of the tasks generates rewards that slowly advance the state of the character. While a new character begins the game with only a few abilities or possessions, as a reward for hundreds or thousands of hours of playing the game the character typically has dozens of abilities and hundreds of possessions.

From the perspective of a gameplay host, these various aspects of persistence (abilities, possessions, and levels) are all alike in that they grant the player additional gameplay effects. Because of this, and in order to maintain generality, we refer to any persistent advancement a character can achieve as that player receiving *loot*. The substantial investment in time played and tasks completed typically means that players are very attached to their loot, and very interested in how to get better loot.

Because acquisition of loot is a primary motivator for persistent on-line games, we choose to focus our design on loot instead of gameplay. This is not to downplay the importance of gameplay, but rather to allow the publisher and community complete freedom to create whatever sort of game they would like and maintain a valid reward structure. As important as loot is to individual players, the assurance that a player's loot was earned fairly is of critical importance to the community's confidence in the virtual economy and the

lifetime of the game.

4.2 Goals and Pitfalls

Imagine taking an existing public server game such as Counter-strike and turning it into an MMO with persistent content. Instead of simply playing for the thrill of victory, players now compete to advance their persistent character and acquire loot such as better skills, weapons, outfits, and money, all tracked by each Counter-strike server but shared between servers. Furthermore, players host the servers and players design the loot. This system presents several challenges: How is loot shared? How can a server be assured that a given player should have a given piece of loot? Can a player run a server and issue their own avatar loot? Can players design extremely powerful loot only for their own use? How can users exchange loot?

We believe the following three design flaws must be avoided for the game to thrive. First, unauthorized loot creation must be prevented; players must earn their loot via the gameplay procedure, and if loot is to be persistent across servers, the servers must somehow trust that a player has not simply fabricated the loot. Second, the gameplay effects loot must be balanced; as users can design loot as well as gameplay, their incentive may be to create very powerful loot. However games are not enjoyable when poorly balanced. Third, automated play via computer programs must be prevented; as persistent games such as MMOs are supposed to reward time put into the game, any automatic way to earn loot without user interaction will disrupt the fair gameplay experience to the detriment of the game's popularity.

Given these pitfalls, we believe that a public server MMO should (1) authenticate loot in an unforgeable manner, (2) control the balance of loot and (3) verify users are playing their game instead of computer programs. The rest of this section presents our design decisions to meet these goals. The three key participants in our design are the clients, public servers, and publisher. Figure 3 shows each of the three participants and their general role in the architecture: the publisher handles player authentication, billing, global gameplay functions such as chat, and loot distribution to servers, who handle gameplay interactions with clients. In the rest of this section we present our architecture by focusing on the three challenges it strives to meet: authentication, persistent content, and the problem of trading persistent content.

4.3 Authentication

One central challenge in a public server MMO is authentication and trust. Since clients are paying a subscription, the loot server must be able to authenticate clients. All participants must be able to verify loot as authentic and trust that a given client is allowed to possess it. To meet these needs, we generate a number of keypairs. Generally, we need a keypair to authenticate the client to the authentication server, a keypair to sign loot, and a keypair to bind loot to a given

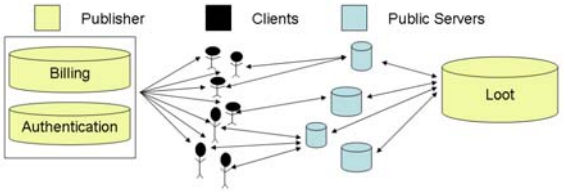

Figure 3: Participants in PS MMO

player. In terms of key distribution, each client i keeps a private key cl_priv_i, with public key cl_pub_i stored by the publisher. This is the client authentication keypair. The publisher advertises a master loot key $loot\_pub$ but keeps $loot\_priv$ secret, and uses $loot\_priv$ to sign loot. Finally, the publisher keeps a key pair ($bind\_pub_i$, $bind\_priv_i$) for each client i, advertising the public bind key, and uses this key to bind persistent items to players.

A different but related authentication problem arises from our incentive-based loot distribution model. Our model is to grant loot to public servers based on the number of player-minutes accumulated per server. The loot server needs to verify that player-minutes are not being granted to a server without a player actually present on the server. One could imagine, for instance, a player and public server colluding to accumulate player-minutes every minute of the day even when the player was away from the computer. We authenticate player-minutes with the use of periodic CAPTCHA tests that are known to be challenging for computers but easy for humans [13]. While CAPTCHA design is outside the scope of this work, we believe the goal of the tests should have an additional component aside from differentiating humans and computers: differentiating gamers from other humans. A game world has a unique environment and set of rules; it should be relatively easy to place some of this context into the CAPTCHA, for example with an image from the game. By binding the player to the domain of our game we can deter work-arounds such as CAPTCHA farms or CAPTCHA redirects.

4.4 Persistent Content

Persistent content is issued to the public servers according to accumulated authenticated player minutes logged at the server. This creates two incentives. The first is that players get loot for their invested time, as in a traditional MMO. The second is that servers are rewarded for hosting players with loot to distribute as they see fit. The public server receiving loot dispenses the loot according to its gameplay rules, which could vary widely. For example the gameplay could require players to compete against each other in a tournament, with all the combined authenticated player minutes going towards a prize for the winner. The server could distribute loot according to the defeat of computer-controlled scripted encounters, or randomly, or only to certain people. While the potential for abuse is clear in such a system, we believe that as users can vote with their attention

for different servers, they will tend to gravitate towards fair servers with compelling gameplay. This is the case for other public server communities such as Half-life where servers that allow cheating are eventually abandoned.

Our design for content creation is that the user community and publisher create content of two sorts: gameplay content such as the environment with which the player interacts and the available actions, and persistent content such as items or abilities that are intrinsic to the player and modify their appearance and gameplay. The community and publisher have complete freedom to design gameplay content, but persistent content must be balanced by the publisher before being admitted to the game. Two strategies for balancing content are either a human review and approval process or a published set of rules all loot must follow such as a set of equations describing a point system for the cost of various abilities.

The security of distributed content is ensured with the bind keys ($bind\_priv_i$, $bind\_pub_i$) and the master loot keys ($loot\_pub$, $loot\_priv$). A given piece of loot is signed with $loot\_priv$ and signed again with $bind\_priv_i$ for client i, and then given to the client. The client can present this loot at any public server or to any client for verification. No other client j can forge loot without knowledge of $loot\_priv$ or $bind\_priv_j$. This enables each client to store its own loot locally. One drawback of this design is that a player can never trade or lose an item. We will describe a way to relax this constraint later.

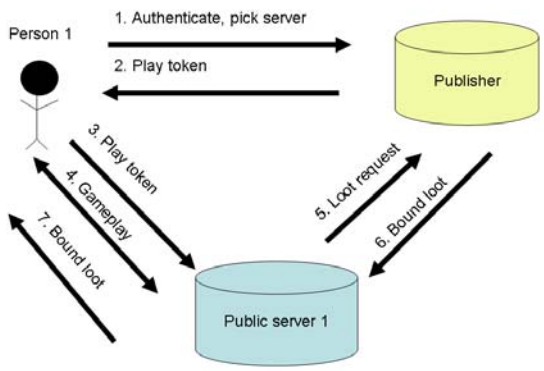

Figure 4: Player interactions with public server and publisher during normal gameplay

Figure 4 shows an overview of the player interactions with the publisher and public server during normal gameplay. The player is initially authenticated with the publisher as a subscribed gamer and receives a play token good for play on the desired public server for a certain period of time. That public server's credit is recorded by the publisher. Using the play token, the player authenticates itself to the public server and plays the game. When loot is distributed, the public server requests the item from the publisher, who signs it and debits the public server's account. The public server then gives the player the item.

4.5 Trading

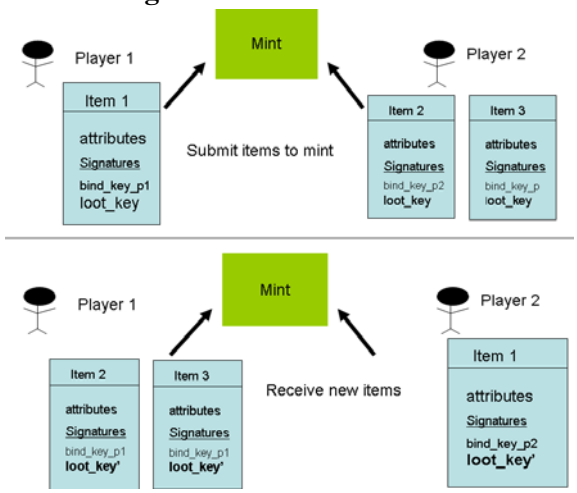

Figure 5: Example of how trading alters signatures on items

The trading of items is a backbone of many persistent worlds, but prohibited in our design so far. We would like to allow trading in a way that does not permit item duplication. The core of the problem is that once an item is traded between parties it should not be possessed by both parties. As PSMMO uses cryptographic signatures to indicate item authenticity and ownership, we need a way to invalidate ownership. In public key cryptography this problem is called certificate revocation and the typical solution is a certificate revocation list (CRL): a list kept by an authority listing invalid certificates. For PSMMO, this solution requires (1) frequently checking the CRL for freshness and (2) maintaining a CRL for all items for all players over the history of the game. We propose to avoid the scalability problems associated with such a list by establishing a globally synchronized trading session in which all items in the game are re-issued according to a new $loot\_priv, loot\_pub$ master loot key pair. The publisher would host a global market for persistent items, with players able to bid on items or establish trades in the time period preceding the trading session. No actual trading would occur however, until the trading session, during which each player would have his items re-issued according to whatever trades had been agreed upon. Figure 5 shows a pair of players with their items before and after the trading session, where they each possess the other's items. As the loot key has changed, the old items are no longer valid.

Our scheme requires the periodic re-signing of all items in the game to a new master loot key. What is the computational cost of that task? To scale trading with the number of subscribers, the publisher must possess resources capable of performing the signing task for all users in a given window. The free cryptography library *crypto++* reports

signature and signature verification times of 4.75ms and 0.18ms respectively for the RSA1024 public-key cryptography scheme on a 2.1GHz Pentium 4 [14]. Assuming each user has 100 items, and each user's items are processed (verified and signed) once, the Pentium 4 would then process 90,947 users in a 12 hour period. This gives a simplistic overview of the cost of the signing task, but it should be noted that re-issuing each item can be performed in parallel and can be performed lazily upon authentication. Computation time for signing can be further reduced with dedicated hardware support.

5. DISCUSSION

As our design can accommodate a variety of persistent games, it may be helpful to present an example game and discuss the user role in the game. With this basis for discussion, we detail limitations of our design.

5.1 Example Game

This imaginary game *CSX* is an adaptation of a FPS game such as Counter-strike to the persistent content model. As in Counter-strike, the main gameplay action involves teams of players acting as a squad to defeat enemies and defuse or protect bombs. Unlike Counter-strike, players can gain new abilities, outfits, and weapons by playing *CSX* and earning loot. The publisher has set up a loot server (issuing publisher and user-authored loot), authentication server, and possibly some other servers to address global functions such as chat, trade or movement between servers. The rest of the hosting infrastructure is provided by users.

Users have set up a large number of servers, a portion of which form a world players can move around and interact in without performing the main gameplay action. Another portion of public servers form small instances of the main gameplay action of a bomb threat. In order to advance in *CSX* players must convince a server to issue loot to the player, by completing gameplay tasks on the server, such as being on the winning team, or defeating another player. In *CSX* loot is issued in very small quantities as coins, which are redeemable once a month in the monthly trading session for actual player abilities. Servers may choose to guarantee that each defeated enemy yields loot, in which case the server will only allow enemies to spawn when the server has accumulated enough player minutes for the loot. Alternatively, servers may allow enemies to spawn more frequently and determine if they have loot upon their defeat. The restrictions imposed on user created *CSX* servers are that they can only issue approved loot, and they can only issue loot in accordance to cost in player-minutes.

5.2 Limitations

The focus of our design is on working persistent content into the public server model in a way that allows authentication of players and items, and the intended benefit is decreased hosting costs and content creation costs. A first limitation of our model is the use of clients to store persistent data. In this situation, the burden of backups, sharing and synchronizing data between different locations is on the client. We see two other primary sources of limitation: abuses of the unmonitored channel between clients and public servers, and scalability of the publisher's central authentication role.

Regarding hosting scalability, it should be noted that the publisher's hosting and content costs do increase with the number of users as the publisher must orchestrate whatever global gameplay tasks exist, such as the trading market or global chat functions. Similarly, a human-supervised content balance review process that controls loot admission into the world also becomes more laborious with the number of users generating content. Thus one limitation of our design is that we merely reduce the load on the publisher.

Regarding client and public server collusion, we believe our system incentives work against widespread abuse. The incentives in our system are (1) people who contribute servers want them to be utilized by players and (2) players want to acquire loot and to have fun. As an example of abuse in *CSX*, a hacked public server could have special rules granting the server administrators very powerful non-authentic weapons. However a persistent social world such as a MMO comes implicit with a social reputation system, and in the long run we believe players will tend to avoid cheating servers. As a second abuse example, servers and clients could collude to receive loot without performing any meaningful gameplay tasks (i.e. clients in *CSX* log into an empty room with no bomb or enemies, answer periodic authentication queries, and eventually leave with loot), but we believe the player incentive of having fun will instead cause them to gravitate towards servers with compelling content. As a final example, servers could allow non-authenticated players to play on their server, although they would lose the incentive of gaining credit with the loot server. This leads us to one form of abuse that players could have incentives to gravitate towards: a free service that was not subscription based, but rather ran on user-contributed hardware and simply copied the approved content from the paid service game as it came out. As the bulk of the artistic content is available to clients and the server binaries are also available in our model, we do not believe there is a simple solution to this problem. However as these copycat publishers become popular they also become easier to locate and shut down legally. Furthermore, the cost of taking the publisher's role is not trivial even if the copycats do not have to perform the content balance.

6. CONCLUSION

Current MMOs are extremely popular but are costly to host and require an enormous amount of ongoing content creation to keep subscribers happy. The public server architecture offers an alternative that harnesses user resources to host and author content. We focus our design on the management of persistent content (loot) across public servers. The key challenges of the public server architecture are authentication, persistent content creation and distribution, and game balance.

Our design uses a central authority (the publisher) and is incentive-based. Players want better items, abilities, and other forms of persistent upgrades to their character, while servers want popularity and to distribute valuable upgrades. The key mechanism for both of these incentives is that loot is distributed from the publisher to user-run public servers based on accumulated player-minutes logged at that server. Once issued, loot is stored on the client, and cannot be forged as it is signed with cryptographic keys.

Our incentive-based design has certain limitations, such as requiring clients to store and backup their own persistent content and allowing for collusion between players and servers for short-term benefits. However we believe that the scalability benefits of the public server MMO model outweigh the limitations.

More broadly, our design ignores the substantial difference between the high level of performance and reliability of modern centralized MMOs and the more modest hosting characteristics of a single public server. We assume the overabundance of public servers can be used to form a similar high reliability and performance system for MMO gameplay. Future work would provide a design for this, in addition to meeting other challenges such as a system for exchanging players between servers according to game rules, a reputation system for public servers, and a more elegant solution for trading authentic content other than completely re-issuing all content.

7. REFERENCES

[1] Valve, Inc., "Steam," `http://www.steampowered.com/`, 2005.

[2] Microsoft, "MSN Games," `http://zone.msn.com/`, 2006.

[3] World of Warcraft, "World of Warcraft Community Site," `http://www.worldofwarcraft.com`.

[4] SirBruce, "Total MMOG Active Subscriptions," `http://www.mmogchart.com/Chart4.html`.

[5] IBM Corp., "On demand business," `http://www.ibm.com/ondemand`.

[6] A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, "Implementation of a Service Platform for Online Games," in *NetGames*, August 2004.

[7] T. Iimura, H. Hazeyama, and Y. Kadobayashi, "Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multiplayer Online Games," in *NetGames*, April 2004.

[8] S. Hu and G. Liao, "Scalable Peer-to-Peer Networked Virtual Environment," in *NetGames*, April 2004.

[9] Y. Kaneda, H. Takahashi, M. Saito, H. Aida, and H. Tokuda, "A Challenge for Reusing Multiplayer Online Games without Modifying Binaries," in *NetGames*, October 2005.

[10] Wikipedia, "List of Half-life Mods," `http://en.wikipedia.org/wiki/List_of_Half-Life_mods/`, 2006.

[11] IGN, "Neverwinter Nights Vault," `http://nwvault.ign.com/`, 2006.

[12] C. Ondrejka, "People Powered Places: Second Life, Saving Games, and 6 Missing Pieces," in *Microsoft Research Tech Talk*, August 2005.

[13] L. von Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Proceedings of Eurocrypt*, 2003.

[14] Open Source, "Crypto++: A Free C++ Class Library of Cryptographic Schemes," `http://cryptopp.com/`.