A Hybrid Architecture for Massively Multiplayer Online Games

Jared Jardine
Computer Science Department
Brigham Young University
jaredlj@cs.byu.edu

Daniel Zappala
Computer Science Department
Brigham Young University
zappala@cs.byu.edu

ABSTRACT

Many massively multiplayer online games use client-server architectures that have enormous server-side bandwidth requirements. Peer-to-peer game architectures provide better scaling, but open the game to additional cheating, since players are responsible for distributing events and storing state. We have developed a hybrid game architecture that maintains centralized control of state, while significantly reducing server bandwidth. The architecture uses a combination of client-server and peer-to-peer event distribution, so that only critical events are processed by the server. In addition, the architecture uses measurements and monitoring to ensure that players are capable of handling event distribution and are indeed providing this service. By lowering the bandwidth needed to host a game, while also providing a simple way to prevent cheating, our hybrid architecture allows game companies to support more concurrent players while still providing a controlled game experience. We deploy a game using the hybrid architecture on PlanetLab and use a measurement study to demonstrate its advantages over a client-server architecture.

1. INTRODUCTION

Bandwidth requirements are a major concern for an organization that hosts a multiplayer online game. When a central server hosts a game, every player sends updates to the server, which computes the new game state and then sends a state update to the affected players. In the worst case, this means the combined upstream and downstream bandwidth requirement at the server scales quadratically with the number of players [14].

To address this problem, many researchers have developed peer-to-peer game architectures [1, 11, 12, 9, 2, 16, 3]. Because players distribute game updates among themselves, this eliminates the bottleneck of a central server, though raising the bandwidth requirement for clients. Other advantages include lower latency between clients (since messages are not relayed through a server) and increased fault tolerance. However, because they rely on peers, these architectures must also cope with state consistency, event ordering, and cheating. Preventing cheating is particularly difficult, because peers typically have control over game state and event distribution.

While client-server game architectures typically have a bandwidth bottleneck at the server, they do have some significant advantages. The main advantage is that the server controls the game state, which makes it easier to maintain state consistency and more difficult for players to cheat. Centralized control is an important part of making a game economically feasible. Players need to trust that the time and effort they invest in playing the game is rewarded, and that players either can't cheat or will be removed from the game if they are caught. In return for this service, the organization running the game server can charge players a subscription fee, offsetting the costs of game development. The client-server architecture is also simpler to implement, since it does not require peer-discovery, distributed event ordering and cheat prevention, and distributed storage and computation required by peer-to-peer architectures.

In this paper, we develop a hybrid architecture for multiplayer online games that combines the advantages of client-server and peer-to-peer games. We focus on role-playing games, for which we make a distinction between *positional* moves and *state-changing* moves. The majority of moves in a typical role-playing game are positional moves [11], which occur when a player moves in the game without affecting game state. In comparison, a state-changing moves changes game state, for example when a player attacks another player or creature, picks up an item, or crosses an important boundary. In our hybrid architecture, the server appoints players to act as regional servers, handling all of the positional moves for a geographical area of the game. The server then handles only the state-changing moves, retaining the centralized control that helps to maintain consistency and prevent cheating. This division of labor greatly reduces server bandwidth and allows it to scale to larger numbers of concurrent players.

To effectively use players as regional servers, the central server uses active measurements to determine their bandwidth capacity and latency. The bandwidth of a regional server is critical because this enables it to serve game updates to the players in its region. Low latency is also important because the central server sends state updates to the players through the regional servers. Finding a regional server that has low latency to the players in its region is hard, since this requires a measurement to each player and

NetGames '08, Worcester, MA, USA

because the set of players in a region changes over time. Choosing regional servers near the central server is a simple way of reducing overall latency, since the state updates have to be routed through the central server anyway.

To evaluate the hybrid game architecture, we create a multiplayer online game in which two teams of players search for hidden treasures. The game can be played either with a client-server architecture or with the hybrid architecture, allowing a direct comparison. We then run experiments with 50 players on PlanetLab, using an emulator to restrict the bandwidth of the hosts so that we can vary player capabilities. Our experiments show that the hybrid architecture scales better than the client-server architecture, requiring significantly less bandwidth at the central server while also lowering latency. The performance advantages of the architecture are largely dependent on having enough players who have the bandwidth necessary to act as regional servers.

2. RELATED WORK

Many peer-to-peer online game architectures have been introduced to provide scalability and bandwidth savings for online games. We broadly classify these based on how updates are delivered to players: by using either a Distributed Hash Table or by building a hierarchy of peers.

2.1 DHT Architectures

The core of several peer-to-peer game architectures is a publish-subscribe system built on top of a Distributed Hash Table. Players subscribe to receive updates from different game regions, which are mapped to locations in the DHT. As players modify the game state, they publish these changes to the DHT, which delivers the state to the subscribing players. A key part of the Mercury publish-subscribe system [1] is the support for efficient range queries, which enables players to precisely specify the updates they are interested in receiving.

The Colyseus architecture demonstrates how a game built on Mercury can support many more players than a client-server architecture, while supporting low latency game play [2]. In this architecture, all mutable objects (including players) are managed by a global storage architecture, which maps each object to a single node on the DHT. All updates are handled through the object owner, using the Mercury publish-subscribe system. To provide good performance, Colyseus makes local replicas of objects, then coordinates replica consistency.

SimMud [11] uses the Pastry DHT [15] to distribute game updates. The game world is divided into regions, and each region and its game state is mapped onto a DHT location and coordinated by the player running that DHT node. Each player regularly sends its current location to the region coordinator, which multicasts the position to every other player in the region. Interactions between players and objects are essentially state updates and are managed by the region coordinator.

The Zoned Federation model [9] likewise divides the game world into regions and stores the game state for each region in a DHT. In this case, however, the DHT is only used as a backup in case a zone owner fails or wishes to relinquish its role. Communication otherwise occurs directly between a player and the owner of the zone in which it resides.

2.2 Hierarchical Architectures

Another way to distribute updates among game players is to create a hierarchy. A common approach is to divide the game into regions and assign a player to manage the state of each region. Players then exchange updates with the responsible node for their region.

One of the key problems for these architectures is distributing the load when the number of players in a region becomes large. With load balanced trees [16], the responsible node builds a delivery tree among the players in the region, and uses the tree to deliver events for the region. With the subserver approach [12], the responsible node subdivides its region into smaller areas and assigns an area server for each of these pieces, resulting in a three-level hierarchy.

An alternative to this kind of top-down hierarchy is to instead build a bottom-up hierarchy based on event scoping [7]. In this approach, events may have a different scope, such as turning on the lights in a room, cutting the power to an entire building, or experiencing a snow storm in an entire city. Players join a tree that is organized by game scope, with the smallest scopes at the leaves of the tree. Players within a small scope exchange game updates directly, using a cryptographic protocol that orders local events and prevents cheating [8]. When events occur at larger scopes, they are delivered to all affected nodes using the scoped tree.

3. HYBRID GAME ARCHITECTURE

One of the problems with peer-to-peer game architectures is that they are often highly susceptible to cheating. Any architecture that allows players to store and manage game state is susceptible to subversion by that player. Likewise, any architecture that distributes updates through other peers – either through a DHT, application-layer multicast, or a hierarchy – allows any misbehaving or malicious player to disrupt moves for other players that happen to be routed through it, and it will be difficult for players to determine who is to blame.

These concerns can be overcome if an organization deploys the distributed architecture on machines it owns, or on a third-party service. In this case, players would connect to the distributed infrastructure at various access points, and would not themselves be responsible for game state or event distribution. However, in this case, the organization loses any cost savings that derive from using player bandwidth and computational resources.

Our hybrid game architecture provides security by using a central server to control access to the game state. This provides the server a great deal of power, because the ultimate goal of most games is to acquire or control state – whether that is treasure, power, or the lives of other players. Any time players must interact with game state, the server acts as the arbiter, ordering state-changing events and issuing state updates. The central server also registers players, monitors their actions, and can evict them from the game if it detects that they have misbehaved.

The hybrid game architecture achieves scalability by using peers to distribute game updates. The server divides the game into regions and assigns a player to distribute updates for that region. Whenever a player makes a positional move, it sends the move to its regional server, which distributes the move to the other players in the region (Figure 1). Whenever a player makes a state-changing move, it sends the move directly to the central server. If the state change is accepted, the central server sends the player the result and also issues an update to the appropriate regional server, which in turn

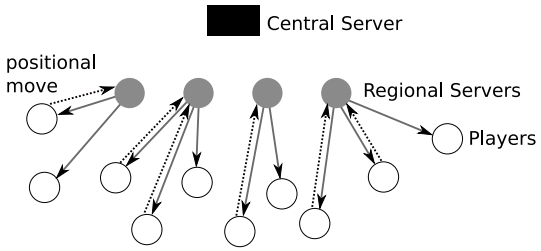

Figure 1: Distributing Positional Moves

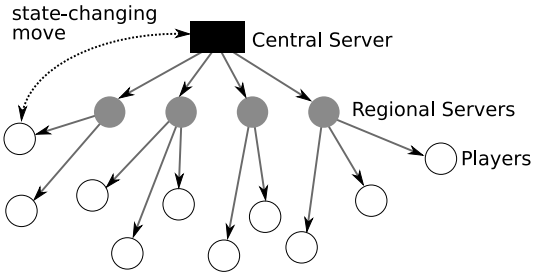

Figure 2: Distributing State-Changing Moves

distributes this update to the players in its region (Figure 2).

As an example, consider a player located in the foyer of an inn. When the player walks across the foyer, it makes a series of positional moves, which are sent to the regional server. The regional server periodically sends a global update to all other players in the foyer. When the same player picks up a coin left on the inn's floor, it is making a state-changing move. This is sent to the central game server, which updates the player's inventory and notifies the player of this change. The central server also sends the regional server an update that shows that the coin is no longer on the floor; the regional server includes this in its next update so that the players in the area will no longer be able to see the coin on the floor.

The bandwidth savings for the hybrid architecture depend on several factors. First, since positional moves are handled by regional servers, the central server uses no bandwidth to process them. The savings is proportional to the percentage of moves that are positional versus state-changing moves. Since positional moves are the most common moves in a game, this saves considerable bandwidth. Second, the central server sends state updates to a regional server, rather than to all players in the region. The savings is proportional to the average density of a region. In both cases, the savings depends on there being enough capable players to serve all the regions.

3.1 Measuring Capable Players

To act as a regional server, a player should have enough bandwidth capacity and should have low latency to the central server. To determine which players are capable, the central server performs several active measurements. First, we use pathrate [6] to estimate the capacity of the path between the central server and the player. Since the central server is located on a high-bandwidth node, and bottlenecks are likely to occur at the edges, this should be equivalent to the capacity of the player's Internet connection. Second, we measure the maximum latency between the central server and the player using ping.

The central server uses several criteria to determine which players will act as regional servers. First, the server uses some minimum requirements, for example a capacity of 1 Mbps and latency less than 200 ms. The server excludes from consideration any players that connect to the Internet from behind a NAT or similar mechanism, although there are some emerging methods that may make this a non-issue [13]. Finally, when there are more than enough capable players, the server groups them roughly into bins, with capacity preferred over latency. For example, players with capacities of 9 and 10 Mbps are roughly equivalent, and a player with a capacity of 10 Mbps and 75 ms latency is ranked higher than a player with a capacity of 1 Mbps and 25 ms latency.

Because regional servers donate bandwidth and computational resources to help improve game performance, special care should be given to ensure that they are not abused. If a regional server's player experience is degraded because they are overwhelmed, they will not want to participate in the game.

Experience with peer-to-peer software such as BitTorrent indicates that users do indeed donate bandwidth when they get a reward such as faster download time. With a hybrid architecture, regional servers will get state updates faster, providing smoother game play. In addition, as more residences obtain high speed Internet connections, users tend to use more peer-to-peer applications and their upload traffic approaches that of their download traffic [4]. Thus, users will likely act as regional servers when they have a high-speed connection; if an additional incentive is needed, organizations deploying a game may want to offer lower subscription fees.

3.2 Security Concerns

In most peer-to-peer architectures, the players have control over game state and deliver updates to players, which makes it easier for players to cheat. In our hybrid architecture, the ability to cheat is significantly limited because the central server controls all access to game state. Players deliver state updates, but only if they are a regional server. Thus the security concerns for the hybrid architecture are limited to the role of the regional servers.

One possible attack is for a regional server to drop or delay some or all of the state updates that it delivers to the players in its region. To cope with this possibility, players monitor regional server updates for latency and loss, then report bad performance to the central server. The central server may replace the regional server if enough players complain.

To provide additional protection against poor performance or failure, the central several requires each regional server to send a positional update periodically. If the server misses three consecutive updates, then it considers the server to have failed. To replace a regional server, the central server tells the region's players to use the central server temporarily. It then selects a new regional server and makes the transition. Because the central server collects periodic updates, it is able to substitute a new regional server without losing a significant amount of positional state.

Another possible attack is for a player acting as a regional server to join its own region, then gain an advantage by seeing how other players move before it moves itself. To remove this possibility, the central server replaces the regional server whenever it moves into its own region.

The regional server may attempt collude with other players in the region, but the central server can check its actions by conducting an audit. For each state-changing move made, the central server checks whether the move was legitimate by using the server update logs to verify that a player had enough time to move into the area where the state-changing action occurred. These checks can occur either in real-time or after the fact, depending on available CPU time. In addition, the organization running the game can offer reduced subscription fees to regional servers to provide a financial motive to play fairly and treat players properly.

Players may also receive an unfair advantage by joining many regions at the same time. To prevent this from happening, the central server controls regional server assignments. When a player first joins the game, the server assigns it an initial position and tells the player the identity of the regional server it should contact. The central server also gives the regional server a list of all of the players in its region; the regional server refuses to give updates to anyone not on the list. Whenever a player need to move between regions, the player contacts the central server to get the identity of the new regional server, and the central server updates the membership lists for the affected regions.

4. EXPERIMENTAL RESULTS

To evaluate our hybrid architecture, we have implemented a simple multiplayer online game in which players search for treasure and bring it back to their base. The central server assigns players a team and a team base located at one edge of the game world. The server then matches pairs of players on opposing teams, and assigns each pair a treasure to find somewhere on the map. Each pair is given the region where the treasure is located, but not the exact location within that region, so they must search the area to find it. As players move within the game world, they are given regional state updates, telling them what they can see within the game region. The player who finds and returns the treasure to his base first wins a point for his team. The central server than randomly places a new treasure for that pair to find.

To provide some competition, players that meet each other can fight over treasures. The winner of a fight is determined by a combination of a random factor, a player's previous victories, and a player's team score. The winner keeps the treasure and the loser is sent back to his base. To increase the odds of a fight, a player moves at only 1/2 speed while carrying a treasure.

4.1 Experimental Setup

We run a series of experiments testing game performance on PlanetLab, using automated players programmed to move toward treasures as fast as they can. We have written the game so that it can be played using either a client-server architecture or the hybrid architecture. Because the messages in our game are small, we pad them with additional data so that they represent a more realistic traffic load. Based on a recent study of game traffic [10], we pad player updates to 20 bytes and state updates to 320 bytes.

We use a rate-limiter to model players with different Internet connection speeds. Player connections can be 10Mbps (a high-speed connection), 1Mbps (a cable modem connection), and 56 Kbps (a dial-up modem). Before assigning speeds to players, we measure the speed of each of the PlanetLab nodes we use in our experiments, and are careful to assign a bandwidth classification that is less than this measured speed.

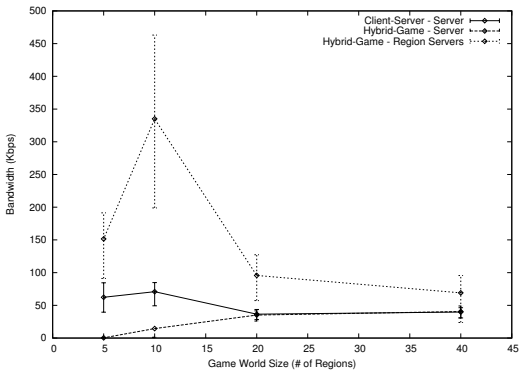

Figure 3: Varying Region Density: Outgoing Bandwidth

We run each experiment with 50 players. Players join the game using an exponential arrival rate that averages one player per second. Once the players have joined, the game lasts for 10 minutes. The results we report are for performance data after all the players have joined the game. For each experiment we first run the game using a client-server architecture and then run the game again with the hybrid architecture. To increase the statistical significance of our results we repeat each experiment five times.

We measure both incoming and outgoing bandwidth for the central server and for the players. We also measure the latency of each move, starting from the time the player sends the move to the time it receives an update from the regional server. Players try to make 5 moves per second, but may move slower than this since they do not make another move until they hear back from the regional server.

4.2 Varying Region Density

Our first experiment varies the number of players in a region. We vary density by keeping the number of players constant, but changing the size of the game world, and hence the number of regions. We use a game world that has 5, 10, 20, and 40 regions, resulting in an average region density of 10, 5, 2.5, and 1.25 players per region. Out of the 50 players, 12 are capable of acting as regional servers.

As the number of regions decreases, the central server in the hybrid architecture uses less outgoing bandwidth as compared to the client-server architecture. Figure 3 shows the outgoing bandwidth used by the server for both architectures. As the number of regions decreases, the number of players per region increases, thus making it more worthwhile to distribute updates through regional servers.

This figure also shows that the aggregate regional server outgoing bandwidth is highest when there is a good balance between the number of regions and the number of players capable of acting as regional servers. In the case of the regional servers, higher outgoing bandwidth is good because it means the regional servers are able to handle more moves per second.

The reason the client-server architecture uses less bandwidth than the regional servers is because a single server cannot keep up with all of the updates the players need. Figure 4 shows the average latency for a player's move, as

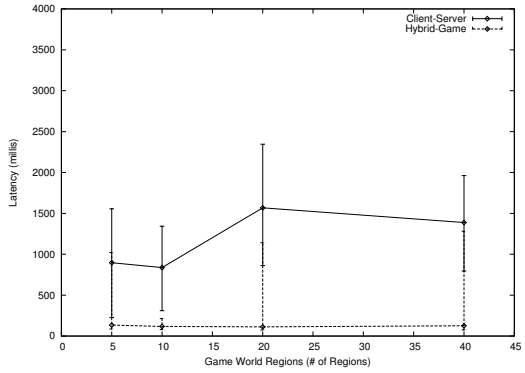

Figure 4: Varying Region Density: Latency

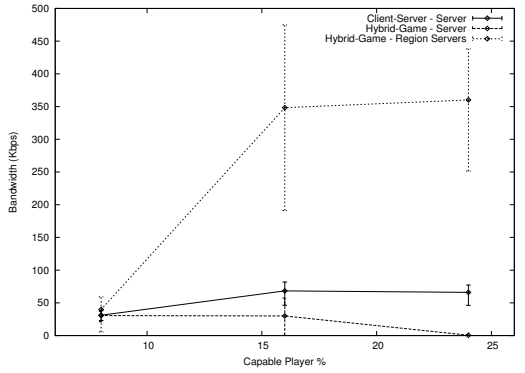

Figure 5: Varying Player Capability: Outgoing Bandwidth

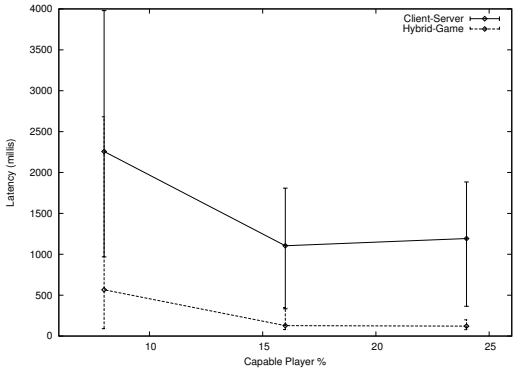

Figure 6: Varying Player Capability: Latency

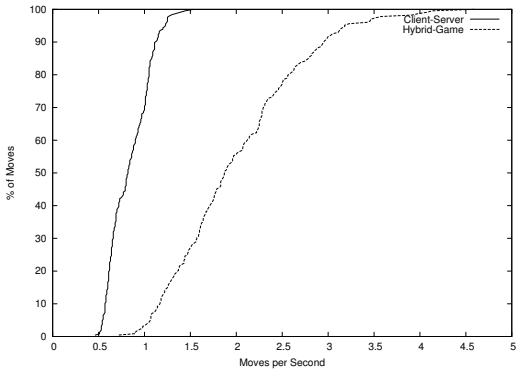

Figure 7: Moves Per Second: 12 Capable Players

a function of the number of regions, along with error bars for the 25th and 75th percentiles. The client-server architecture has trouble processing moves quickly, resulting in high latency for the players. Note that when there are 10 regions, the latency for the hybrid architecture is small and has little variation. This results in the game being able to handle about 2.5 moves per second for all players, compared to about 1 move per second for the client-server architecture.

4.3 Varying Player Capability

We next vary the number of players with the capacity to act as regional servers. We use a world with eight regions and a total of 50 players, then run experiments with an average of 4, 8, and 12 players capable of being regional servers. Studies have shown that as residential users increase their bandwidth capacities they are more likely to participate in resource sharing peer-to-peer architectures [5]. As broadband and fiber-optic Internet access becomes increasingly prevalent, capable player percentages should increase.

As shown in Figures 5 and 6, as the number of capable players increases, both the outgoing bandwidth for the central server and the latency for player moves decrease. Because the majority of moves are positional, and even global state updates are sent through the server hierarchy, the regional servers bear most of the cost of delivering state updates. The benefits of the regional servers are exhausted once there is at least one regional server available per region. The hybrid architecture has very low latency when there are enough regional servers, with very little variation. It is able to handle about 2 moves per second on average, while the client-server architecture handles less than one move per second.

Because our experiments are played over a live network, players will experience different quality of play over time. Figure 7 shows a CDF of the number of moves per second a player is able to make, combined over all the experiments run with 12 players capable of acting as regional servers (corresponding to 24% on Figures 5 and 6). For the hybrid architecture, players can make anywhere from 1 to 4.5 moves per second. An interesting comparison is that the 10th percentile for the hybrid architecture is equivalent to the 80th percentile for the client-server architecture.

5. CONCLUSION

Our experiments with the hybrid architecture indicate it can save considerable bandwidth for the central server. Peers acting as regional servers handle most of the bandwidth required by state updates. As long as there are enough players capable of acting as regional servers, latency can be kept low, allowing the game to handle more moves per second. Our results are particularly applicable to role-playing games, since the majority of moves in these games are typically positional.

One concern is the relatively poor performance of both architectures with respect to the number of moves per second they can make. Since both architectures use the same code base, they are equally affected by our design and our comparisons are fair. However, modern games can handle

thousands of simultaneous clients, so we plan to improve our implementation in order to make more realistic performance studies.

We also plan to further develop the hybrid architecture, with particular attention on further decreasing latency. PlanetLab is a useful testbed for games because it forces applications to cope with variability in connectivity and performance. We plan to run additional experiments to further examine the scalability of our architecture, to test failure scenarios, and to verify we are able to prevent cheating. Once we have these additional results, we plan to integrate the architecture into an established game for further testing.

6. REFERENCES

[1] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *ACM SIGCOMM Computer Communication Review*, 34(4):353–366, October 2004.

[2] A. Bharambe, J. Pang, and S. Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. Technical report, Carnegie Mellon University, September 2006.

[3] L. Chan, J. Yong, J. Bai, B. Leong, and R. Tan. Hydra: a massively-multiplayer peer-to-peer architecture for the game developer. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 37–42, New York, NY, USA, 2007. ACM.

[4] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The impact and implications of the growth in residential user-to-user traffic. *SIGCOMM Comput. Commun. Rev.*, 36(4):207–218, 2006.

[5] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The Impact and Implications of the Growth in Residential User-to-User Traffic. In *ACM SIGCOMM*, pages 207–218, New York, NY. USA, Aug. 2006. ACM Press.

[6] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Trans. Netw.*, 12(6):963–977, 2004.

[7] C. GauthierDickey, V. Lo, and D. Zappala. Using n-trees for scalable event ordering in peer-to-peer games. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 87–92, New York, NY, USA, 2005. ACM.

[8] C. GauthierDickey, D. Zappala, V. Lo, and J. 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, New York, NY, USA, 2004. ACM Press.

[9] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multi-player Online Games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and System Support for Games*, pages 116–120, New York, NY, USA, 2004. ACM Press.

[10] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk. Traffic Characteristics of a Massively Multi-Player Online Role Playing Game. In *Proceedings of 4th ACM SIGCOMMworkshop on Network and System Support for Games*, pages 1–8, New York, NY. USA, 2005. ACM Press.

[11] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. *INFOCOM 2004. Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 1:107, 2004.

[12] H.-H. Lee and C.-H. Sun. Load-balancing for peer-to-peer networked virtual environment. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 14, New York, NY, USA, 2006. ACM.

[13] J. Maassen and H. E. Bal. SmartSockets: solving the connectivity problems in grid computing. In *Proceedings of the 16th internation symposium on High Performance distributed computing*, pages 1–10, New York, NY, USA, 2007. ACM Press.

[14] J. D. Pellegrino and C. Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 52–59, New York, NY, USA, 2003. ACM.

[15] A. Rowstron and P. Drushel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms*, page 329, Heidelberg, Germany, November 2001. Springer Berlin Heidelberg.

[16] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito. A Distributed Event Delivery Method with Load Balancing for MMORPG. In *Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 1–8, New York, NY, USA, 2005. ACM Press.