Latency and Scalability: A Survey of Issues and Techniques for Supporting Networked Games

Xinbo Jiang, Farzad Safaei, Paul Boustead
Smart Internet CRC,
Telecommunications and Information Technology Research Institute,
University of Wollongong, Australia
Email: {xinbo, farzad, paul}@titr.uow.edu.au

Abstract - **The popularity of on-line games that emphasize real-time interactivity is on the rise. A survey of issues and techniques for supporting on-line games is provided in this paper. Latency and scalability are two primary aspects on which this survey is based. Network latency heavily influences the design of on-line games with regards to the nature of interactivities involved, as well as, real-time flows such as voice and video if enhanced user immersive experiences are required. As a consequence, a trade-off between consistency and responsiveness, the two latency-related artifacts, is usually needed. This can be demonstrated through a number of latency compensation techniques presented in this paper, including the synchronization and optimistic approaches. To address the drawbacks of the current predominant Client-Server model, scalable architectures usually achieve scalability through the partition of either the physical or virtual world. In the case of distributed server design, this has resulted in systems such as the locale or proxy server architecture. In the peer-to-peer design, in addition to partition, hybrid system introduces the concept of 'super node' to allow the manageability in the overlay network. The latest peer-to-peer design emphasizes a structured approach such as the Pastry. Although such a design is not targeted for latency constraint applications, due to its unique features, experiments on hosting networked games on structured peer-to-peer network start to emerge.**

Index Terms - **Games, network support, latency, scalability.**

1. INTRODUCTION

The growth and popularity of on-line games has been phenomenal in the past decade. On-line games such as the Lineage, the World of Warcraft and Counter-Strike have attracted millions of players on a global scope and achieved huge commercial success [1, 2, 3]. The underlying network support, especially the Internet, contributes significantly to this phenomenon. With the recent expansion of the DSL into the common households[4], the next killer Internet application might well be on-line entertainment applications that emphasize a high level of real-time interactivity.

Currently, there are three primary genres of massively distributed on-line games: Role Playing Games (RPG), Real Time Strategy games (RTS), and First-Person Shooters (FPS). Of relevance to this paper is the difference in RTS, RPG and FPS with respect to the nature of interactivities. For example, fighting in a FPS game is real-time intensive and requires quick physical response from the players. On the other hand, fighting in RTS and RPG games is usually turn-based. In other words, the results of the fighting are predetermined by the game logic according to the level of strength of the avatar models or through somewhat random operations or factors. Statistics in [5] show that in 2004 these three classes of on-line games counted for more than 50% sales revenue of all computer games sold in the USA.

While network support is one of the cornerstones that have enabled this generation of entertainment, network resource limitations impose certain constraints on on-line game design and deployment. Among these latency is a particularly sensitive issue that, if not handled properly, will lead to poor game quality, sluggish responsiveness and inconsistency. As a result, on-line game design is heavily influenced by network latency. Furthermore, resource limitations, especially network bandwidth, computational power and latency, impose constraints on on-line game scalability with respects to the number and geographical spread of players that can be supported. As a consequence, a class of scalable architectural designs has been proposed as augmentation and/or alternatives to current predominant Client-Server model.

In this paper, we provide a survey of mechanisms and techniques for supporting on-line games. Latency and scalability are the two primary aspects on which this survey is based.

The rest of the paper is organized as follows. Section 2 describes the two latency-related artifacts, consistency and responsiveness, with focus on their concepts in on-line game design and influence over user experiences. Section 3 reviews several popular latency compensation techniques. After identifying the limitations of these compensation techniques, section 4 discusses a number of scalable architectures with more details on peer-to-peer design to reveal the novelty in this new generation of concepts. We conclude in Section 5.

2. LATENCY, CONSISTENCY AND RESPONSIVENESS

Online games are often based on a Collaborative Virtual Environment (CVE) [6] in that a shared virtual world is replicated at each participant's host computer to allow interactivity across geographic barriers [7, 8]. To a certain extent, the states of the replicas of the virtual world have to be consistent to allow game playability. The invocation of user actions changes the states of the world. These changes have to be reflected in other relevant players' replicas of the world as quickly as possible. However, network latency precludes instant dissemination of update information. As a consequence, a trade-off between consistency and responsiveness, the two latency-related artifacts, is usually required. The type of trade-off that is undertaken would be application-specific.

In [9], consistency is classified into three dimensions: presentation, physics and interaction. Presentation consistency is the degree of matching between different users' views of the virtual world. The physical consistency is concerned with the laws of physics in the virtual world and whether their influence is consistent with our

1-4244-0000-7/05/$20.00 ©2005 IEEE.

expectations. Interaction consistency is the degree of matching between user action and virtual object reaction. Here, the concept of consistency is defined on the basis of participant's perception of the virtual world both in space and time dimensions.

The presentation and interaction consistency are directly influenced by network latency. This is demonstrated through a shooting action example [10]: In a FPS game, player A with 100ms latency is aiming at player B when player B is running at 500 units per second perpendicular to the player A. If an instant hit weapon is used and latency is not dealt with at the game logic, player A would have to aim 50 units ahead of the player B to be able to score a hit. This is because by the time the shooting message arrives at player B's computer, the player B's spatial location would have been updated as shown in Figure 1.

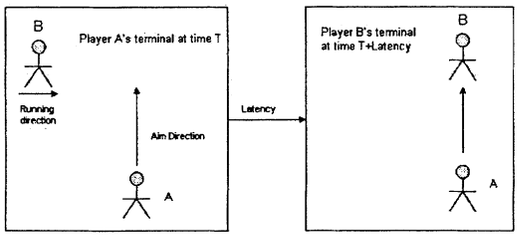

Figure 1: Aim Ahead to Hit

In this case, neither the presentation nor the interaction consistency is completely preserved - the shooting action occurs at different time instants on the two terminals (presentation inconsistency), and the player A has to aim ahead at B to score a hit that would be agreed by B (interaction inconsistency). Mauve M., et al. [11] closely associate the definition of consistency with the arrival time of an operation command issued on one client to other client sites: If a user initiated an operation O at time T1, which is expected to execute before T2, the replicated world at other client sites is ensured of state consistency if O is received by every other client before T2. The consistency in this sense only concerns the final state of the replicated world as the results of the operations. A more generic "short-term inconsistency" is defined in the paper as the consequence of a late arrival of an operation that leads to at least two sites having different states at a particular instant of time.

Regardless of how it is defined, consistency is obviously desirable for on-line games. If an inconsistency occurs, depending on the category of the application, the repair scheme provided by the application might produce unwanted artifacts to the users (see section 3.2). To avoid inconsistencies, it would be desirable to synchronize the game states (see section 3.1). However, this synchronization will lead to another unwanted latency-related artifact, that is, increased response time. As defined and stated in [11], response time or responsiveness is the time difference between when an operation is issued and when it is executed. In many situations, minimization of response time and avoiding inconsistencies are conflicting goals. For instance, in the example shown in Figure 1, the shooting action by player A may be delayed for execution until the action command has reached the other client sites to maintain the consistency. However, player A will now perceive an increased response time due to this synchronization which might not be satisfactory.

In [12, 13, 14, 15], the effects of latency on user experiences are investigated over a few popular on-line games with different characteristics. The general findings are that the level of latency sensitivity of the games depends on the nature of the interaction required by the applications. Sheldon N., et al. reported in [14] that the Warcraft III, a typical RTS game can have latencies from hundreds of milliseconds to a few seconds without significant impacts on the performance for a player (whether winning or losing). Nevertheless, the player might perceive some delayed executions of commands when the latency reaches around 500ms. This is primarily because the nature of RTS emphasizes strategies rather than interactive aspects. Most of the operations issued by a player take seconds or even minutes to carry out; hence typical network latencies have little to no impact on the game outcomes. In [12, 15], latency effects in the Unreal Tournament 2003, a popular FPS game, are investigated. Beigbeder T., et al. [12] reported that while latencies up to 300ms have no statistical significance to influence movement actions, the accuracy of shooting actions declines sharply beyond 100ms, dropping by 50% at 300ms. In [12], it is concluded that 100ms latency is acceptable and 200ms should be avoided based on objective statistical methodologies. However, it is concluded in [15] that 60ms latency starts to introduce disturbing experiences to users based on both objective and subjective statistical methodologies. FPS games have much stricter latency requirements than RTS and RPG due to the nature of interactivities involved. As a result, the number and geographical spread of players in a typical FPS game session is usually limited.

Latency can be either compensated by smart application logic, and/or controlled through architectural design. These will be discussed through the rest of the paper.

3. LATENCY COMPENSATION TECHNIQUES

Latency compensation techniques for on-line games can be generally classified into two categories. The first is *synchronization* technique (also referred to as conservative algorithm [9, 16]), and the second is *optimistic* technique..

3.1 Synchronization Technique

Lockstep [17] is probably the simplest synchronization technique. In this scheme, if the player of the fastest host invokes an action that could change the state of the world, this change will not take effect until the slowest host has acknowledged this action. In [18], a Synchronized Messaging Service is proposed for on-line games that requires both state update fairness and player action fairness. An application independent service module is abstracted as a network support between the server and the clients to handle message delivery service. For instance, if an update message is sent from the server to clients, the service module at the server side chooses a delivery schedule that ensures all the clients receive the message at the same time. The obvious drawback of this kind of synchronization is the impact of the slowest link and host on performance of whole application. However, for a certain category of games such as chess, card and puzzle games, there is a strict requirement for the ordering of events and hence absolute synchronization is needed to ensure fairness.

As a compromise between delay and response time, a "bucket" synchronization method is proposed in [19] that delays the execution of simulation for a certain amount of time (e.g.100ms) in order to bring the response time and consistency to some balance. If any inconsistency happens (a lost event or a late event arrived), it is simply ignored and game progresses as if it never happened. In comparison with the lockstep, this bucket method can be viewed as a partial synchronization scheme that increases response time at the cost of data consistency.

3.2 Optimistic Technique

Optimistic technique intends to execute game optimistically and solve inconsistencies when they arise. In games that demand high level of interactivity and quick responsiveness such as FPS, the iteration of packaging and computation of updating commands (client side game logic) is executed on a per frame basis [10, 20]. For instance, a Counter-Strike game client needs to render 50-70 frames per second to achieve a state-of-the-art graphical presentation. This frequency defines the duration for client frame or iteration, resulting in 14 to 20ms frames. Ideally, the latency from the client to the server should be less than or equal to 20ms. In reality, this latency is hard to achieve for clients connecting to the server through the Internet. As a result, especially when the number of clients increases, the server might take more than a few client frames to respond with command updates. Dead reckoning [21] [10] is a typical optimistic technique that aims to use game logic at the client side to predict the states of the game to create the perception that the game is continuously progressive regardless of the underlying network latency. When the update message arrives, the local game state will have to make adjustments if the difference between the predicted states and the real ones is beyond a threshold. This technique compensates (masks) the latency at the cost of possible data consistency to achieve a high level of visual smoothness (high frame rate).

While dead reckoning is widely used in game design such as the Half-Life, QuakeWorld and Quake3, it can lead to unacceptable paradoxes. The reason is simple, if the client's prediction of a state is beyond a threshold, the state might not be fixable by the later updates. Two examples are given in [22], one is that a dead man still keeps shooting because the update message about his death is delayed due to network latency. The other is that a tank goes over a mine without being exploded because the update message moves the tank to a position that has bypassed the landmine. In [10, 23], TimeWarp is proposed to solve the possible state inconsistency as described above. It works by taking the snapshots of the past states. Whenever the state of an entity changes because of an unpredicted event, this event is also recorded. If a paradox due to data inconsistency occurs, the state of the world rolls back to the previous one (the current one is discarded) and then the current one is recomputed based on the actual events since then. The obvious drawback of this scheme is the extra memory needed to store the previous copies of the world and computational power required to do the copying and processing. Further improvement proposals are made in [22, 24], where either only a periodical snapshots are taken or an "event horizon" is defined to restrict the amount of executions that can be done optimistically.

To conclude for this section, it is noted that optimistic techniques intend to establish a trade-off between consistency and responsiveness through smart logic at the application level. It should be made clear that these compensation techniques have their limitations. The reasons are two folds. First, if latency goes beyond a certain upper bound that is application specific, it is likely that none of the compensation techniques could produce satisfactory results. This can be evidenced by the experiment [25] conducted on Quake 3 (a popular FPS) servers, which reported that users prefer servers less than 150-180 milliseconds away. Secondly, as reported in [7], real-time flows such as voice, video, gestures that might be needed for on-line game design to enhance user immersive experiences, have a strict requirements for latencies. In this case, it is less likely to achieve good results through compensation techniques. As a result, latency control remains an important issue to be addressed in a class of scalable architecture designs as discussed in the following sections.

4. SCALABLE ARCHITECTURES

At present time, Client-Server (CS) architecture is the predominant model for commercial on-line games. In CS architecture, a central server has the authority to execute the majority of the game logic and then send a list of objects to the client for rendering. The primary task of the client is to collect user commands and package them into data packets to send to the server for processing. The advantages of the CS model are well known and include simpler inconsistency management and billing model as well as simpler protection against fraud. However, besides being subject to a single point of failure, CS architecture may have scalability issues. In [20, 26], CPU resources are identified as the main bottleneck for game servers. Moreover, a Quake server is reported [27] that scales faster than linearly with the increase of the number of connected users , and possibly suffers bandwidth bottleneck even each user has a limited bandwidth requirement. In [7], it has been argued that latency control is of paramount importance for latency constraint applications such as on-line games, especially when real-time flows such as voice, video, gesture and haptics are required for enhanced user immersive experiences. In CS model, a client's update has to go through the central server before being disseminated to other clients. In addition to redundant traffic, this does not work to the benefit of latency control if the game is deployed over the Internet. We discuss scalable architecture designs through a number of augmentations and/or alternatives to CS model in this section, with more details on peer-to-peer design to reveal the novelty in this new generation of concepts.

4. 1 Distributed Server Architecture

It is natural to try to distribute the workload of a single server across a set of geographically dispersed servers close to the clients. In [28], it is proposed to have proxies located close to clients to function as an extension of the central server. The idea is that, instead of letting a central server be in charge of every game update, a local proxy can be given the task of handling some game state updates and traffic directing within the local area. Hence, the congestion at the server side is reduced and the latency within the local area is improved. In [29], a similar concept involving a central

arbiter is proposed where the traffic between clients is done in a peer-to-peer fashion (see section 4.2), and the role of a central arbiter is to solve inconsistencies when they arise. Through a simplified model, it has been reported that a central arbiter scheme is a compromise between the CS and the Peer-to-Peer model in terms of average latency and maximum duration of an inconsistency. Both local proxy and central arbiter can be regarded as a local server that takes part of the role of the central server. It should be noted that if the role of a local proxy or a central arbiter is assigned to a peer, the architecture is a Hybrid Peer-to-Peer as described in section 4.2.1.

Another distribution approach is closely associated with interest management. Although the virtual world in the game could be large in size, the limited sensing capability of an avatar suggests that a player usually has an "area of interest" and belongs to a locale [30]. Dividing the virtual world into "locales" and having one or more locales assigned to each of a set of servers is another way to design a distributed server model. There are two ways to do this. One is through server clustering [31], where locales are distributed over a set of co-located servers. This improves scalability of server computation but the bandwidth bottleneck at the server side remains, and all the traffic still has to go through the server. Another approach is to have locale servers distributed over geographical locations that are close to clients. As pointed out in [32], the major difficulty in this approach is to choose the best physical location for these locale servers in response to the players' movements in the virtual world. A correlation factor between a player's physical and virtual location is defined in [32] to investigate the performance of distributed locale servers through simulation. It's been reported that if the correlation factor is high, in other words, clients in the same virtual world hosted by the local locale server also tend to be in the same physical area, the average latency between clients can be reduced dramatically comparing to the CS model. A hardware called booster box is proposed in [33] to serve as a buffer between the distributed servers and the clients. It is argued that network support such as a booster box could potentially improve the performance of distributed locale servers if it combines the awareness of both the application and the network. For instance, the booster box knows both the physical and virtual location of an avatar, and hence network traffic regarding this avatar can be managed more efficiently.

In distributed server architecture, the proxy and locale server design achieve scalability through the partition of physical and virtual world respectively. Booster box could be effective in that it addresses the gap between the application and the network.

4.2 Peer-to-Peer Architecture

It is appealing to utilize client's computing resources to achieve scalability. Unlike server farm or distributed servers that suffer deployment flexibility, peer-to-peer system could ideally dynamically scales up and down with the number of players. Current peer-to-peer applications that have achieved wider use are mainly about file storage and distribution such as, Napster [34], Gnutella [35] and KaZaA [36]. Napster uses a central server to store references to music contents that stored on participating clients. This is similar to CS model in on-line games. Gnutella 0.4 network works in a de-centralized manner, in which a flooding mechanism is used to search for contents. It was soon reported that Gnutella 0.4 generated too much traffic [37] [38] due to the uncontrolled number of messages exchanged. As a result, KaZaA introduced the concept of 'super node', an extra level of hierarchy that facilitates the manageability of the peer-to-peer overlay network, and has better control of the signaling messages. This extra level of hierarchy seems of particular interest to on-line games, and bears similarity to a class of hybrid peer-to-peer design in a number of studies. The latest development of peer-to-peer design uses hash functions to map the participating peers and the shared contents into an ID space. In addition, routing of a message at the application level is much efficient due to the structured approach. Although on-line game deployed on this structured peer-to-peer has not appeared to our best knowledge, researchers have started to investigate the possibility through experiments.

4.2.1 Hybrid Peer-to-Peer

Conventionally, the peer-to-peer concept involved in networked interactive applications emphasizes a de-centralized approach without a central sever [39]. Early work such as DIVE [40] has objects replicated on each client. This is clearly not scalable because a game state has to be broadcasted to all players. Hybrid peer-to-peer combines the advantages of both the centralized and distributed approaches. The scalability is usually achieved by partitioning either the virtual or physical world. The manageability is achieved by retaining the CS model through the assignment of a 'super node' to manage the collaboration between peers.

A federated peer-to-peer design is proposed in [41], which is a typical example of a hybrid peer-to-peer network. A Multicast-Reflector (MCR) is used to group members in the same Area of Interest (AOI) together (partition of virtual world). Members in the same AOI use the corresponding MCR to send and receive messages to and from other members within the AOI. When a packet arrives at the MCR, if the packet is originated from a host in the AOI that the MCR supports, it will be disseminated among the players in that group through the MCR. If the packet is from another AOI, the MCR will then use an algorithm to check the "level of affinity" of that AOI from which the packet is sent in determining whether to disseminate the packet. A MCR differs from a proxy (see section 4.1) in that it does not store a whole or part of the copy of the game states. It is only required to ensure the efficient forwarding of packets to the subscribed clients. As a result, a peer with sufficient bandwidth might qualify to become a MCR. When moving to another AOI, the player will be unregistered from the current MCR and registered with a new MCR. Apparently, this architecture improves scalability since the traffic is limited within the AOI with best effort. However, the allocation of MCR is only concerned with the virtual AOI regardless of the physical location of the players, which creates the same difficulties as the distributed locale servers do when determining the allocation of MCRs. In [42], a detailed hybrid peer-to-peer design is presented over Gnutella network through the MPEG-4MU - an extension to MPEG-4 standard that supports 3D multi-player virtual environment. In this study [42], an Ultrapeer with sufficient computing capacity and

bandwidth is assigned the role of session controller and/or bookkeeper, and functions as a server in a localized area (partition of physical world). The sense of scalability achieved in this context is the number of game sessions instead of the number of players in a session. Similar design concept can be found in [43], where the game virtual world is divided into zones, and each zone selects a zone owner that performs the tasks of a server for the peers in the zone.

4.2.2 Structured Peer-to-Peer

As mentioned in the beginning of this section, the first generation of peer-to-peer design such as Gnutella neither guarantees an answer to a query, nor the time it takes to find that answer. These drawbacks inspired the second generation of structured peer-to-peer design, whose main design objective is to guarantee an answer to a query with the minimum network hops, while retaining other inherent characteristics of a peer-to-peer network such as decentralized control, self-organization, adaptation and scalability. Typical examples of structured peer-to-peer design are Can [44], Chord [45] and Pastry [46]. While being different in design details, their common feature is that peers form an overlay network at the application level with each node holding a small amount of routing information to allow message routing.

Pastry's core task is to route a message with a key to a node with an ID numerically closest to that key. ID assignment for nodes in Pastry can be viewed in a ring space, whose size is determined by the number of digits assigned to that ring. A node's ID is assigned by hashing its IP address or Public Key. The hashing function (SHA-1) guarantees with high probability that the nodes will be evenly distributed around the ring. Each node in the Pastry ring space maintains a small routing table that facilitates the routing of a message. Given a message with a key, a node along the route will try to relay the message to another node whose ID shares one more prefix with the message key than itself as shown in Figure 2. The expected routing hops for a given message is $\log_{2^b} N$ where N is the total number of nodes in the overlay network and b (usually chosen to be 4) is a configuration parameter that determines the number base of the routing table. Another way to look at it is that if a message with a key "d35 is to be routed, if every hop can match one more digit of that key, then the maximum hop would be the number of digits of the key. The design of Chord is similar to Pastry. Instead of matching a digit per hop for a message key, Chord routes the message to the nodes whose IDs are numerically closer and closer to the key. Can maps a node into a virtual d-dimensional Cartesian coordinate space, where each node owns a distinct zone identified by a coordinate, and maintains a small routing table containing the IP addresses and zone coordinates of its immediate neighbours.

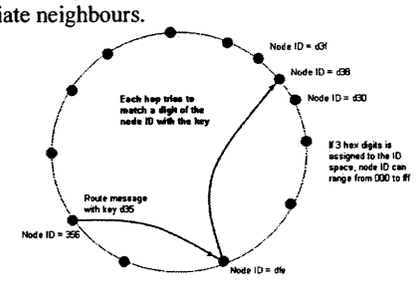

Figure 2: Pastry Routing Example

This category of peer-to-peer design is mainly concerned with load balancing (keys are uniformly distributed among nodes) and quick file retrieval (minimum number of hops). Clearly, it is not designed for latency constraint applications such as on-line games. For instance, Can, Pastry or Chord make no or little effort to achieve network locality. In other words, nodes close in the ID space could be far apart from each other geographically or in terms of network latency. Consequently, the routing algorithm might force a node to go through a neighbour node in the ID space but too far away geographically to search for a piece of information. However, there are a number of attractive features in this design such as decentralized control, self-organization, adaptation and scalability. For instance, the overlay network can grow to a large number of nodes at a small cost to individual nodes' resources. This is because the routing table in each node grows much slower than the number of nodes in the network. Moreover, the overlay network has a small maintenance cost to node joining or leaving. This is because only a small set of maintenance messages has to be exchanged among the neighbouring nodes of the joining or leaving node. As a result, experiments of hosting action games on this class of network are beginning to emerge. In [47], a simple RPG game is simulated leveraging a Pastry network. While the scalability is achieved by dividing the game into locales and appointing a "coordinator" for each locale (this is similar to the hybrid approach), the backup of game state replication is particularly interesting. In compliance with the Pastry routing algorithm, an object in the game with key K is stored in the node whose ID is numerically closest to K. This node then becomes the coordinator for this object. The next numerically closest node B will store a replica of the object. If the coordinator leaves the network, B will automatically be promoted to the coordinator for the object. The underlying Pastry network ensures that all peers interested in this object can find the new coordinator quickly. This is obviously leveraging the de-centralized and adaptation nature of the Pastry design.

5. CONCLUSIONS

In this paper, we have provided a survey of mechanisms and techniques for supporting on-line games. Latency and scalability are the two primary aspects on which this survey is based.

Latency heavily influences current on-line game design with regards to the nature of interactivities involved in the games. As a result, latency compensation techniques usually intend to establish a compromise between consistency and responsiveness, the two latency-related artifacts. Due to the limitations of these compensation techniques, latency remains an important issue to be addressed in scalable architecture designs.

To address the drawbacks of the current CS model, scalable architecture design distributes the computation over clusters of servers or peers. The scalability is usually achieved through the partition of the virtual or physical world. In the distributed server design, this has resulted in systems such as the locale or proxy server. In the peer-to-peer design, in addition to partition, hybrid design introduces the 'super node' to allow network manageability. The latest peer-to-peer design such as the Pastry emphasizes a structured approach. Although it is not designed for latency constraint applications, due to its unique features

such as decentralized control, self-organization, adaptation and scalability, experiments hosting networked games on structured peer-to-peer network start to emerge.

ACKNOWLEGEMENT

This work is supported by the Co-operative Research Centre for Smart Internet CRC (SITCRC) and the University of Wollongong (UOW), Australia.

REFERENCES

[1] Blizzard. www.blizzard.com.
[2] Counter-Strike. www.counter-strike.com.
[3] NCsoft. www.lineage.com.
[4] S. Wright and S. Tischer, "Architectural Consideration in Online Game Service over DSL Networks," presented at IEEE International Conference on Communications, 2004.
[5] Entertainment-Software-Association. www.theesa.com.
[6] C. Greenhalgh and S. Benford, "MASSIVE: a collaborative virtual environment for teleconferencing," *ACM Transaction on Computer-Human Interactions (TOCHI)*, vol. 2, pp. 239-261, 1995.
[7] F. Safaei, P. Boustead, C. D. Nguyen, J. Brun, and M. Dowlatshahi, "Latency-Driven Distribution: Infrastructure Needs of Participatory Entertainment Applications," in *IEEE Communication Magazine*, vol. 43, 2005, pp. 106-112.
[8] L. Pantel and L. C. Wolf, "On the Impact of Delay on Real-Time Multiplayer Games," presented at Proc. of the 12th int. workshop on Network and Operating systems Support for digital audio and video, Miami, Florida, USA, 2002.
[9] I. Vaghi, C. Greenhalgh, and S. Benford, "Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments," presented at Proc. of the ACM symposium on Virtual Reality Software and Technology, London, United Kindom, 1999.
[10] Y. W. Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization," presented at Proc. of Game Developers Conference'01, 2001.
[11] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications," *IEEE Transaction on Multimedia*, vol. 6, pp. 47-57, 2004.
[12] T. Beigbeder, R. Coughlan, C. Lusher, and J. Plunkett, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003," presented at SIGCOMM'04 Workshops, 2004.
[13] J. Nicholes and M. Claypool, "The Effects of Latency of Online Madden NFL Football," 2004.
[14] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The Effect of Latency on User Performance in Warcraft III," presented at NetGames, 2003.
[15] P. Quax and P. Monsieurs, "Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game," presented at SIGCOMM'04 Workshops, 2004.
[16] B. F. Eric Cornin, Anthony R. Kurc, Gugih Jamin, "An Efficient Synchronization Mechanism for Mirrored Game Architecture," in *Multimedia Tools and Applications*, vol. 23, 2002, pp. 7-30.
[17] T. A. Funkhouser, "RING: A Client-Server System for Multiuser Virtual Environments," presented at In Proc. 1995 Symposium on Interactive 3D Graphics, 1995.
[18] Y.-J. Lin, K. Guo, and S. Paul, "Snyc-Ms: Synchronized Messaging Service for Real-Time Multi-Player Distributed Games," presented at Proceedings of the 10th IEEE Conference on Network Protocols, 2002.
[19] E. Lety, L. Gautier, and C. Diot, "MiMaze, a 3D Multi-Player Game on the Internet," presented at Proceeding of the 4th International Conference on Virtual System and MultiMedia, Gigu, Japen, Nov 1998.
[20] A. B. Ahmed Abdelkhalek, "Parallel and Performance of Interactive Multiplayer Game Servers," presented at Proceedings of the 18th Parallel and Distributed Processing Symposium, 2004.
[21] L. Pantel and L. C. Wolf, "On the Suitability of Dead Reckoning Schemes for Games," presented at NetGames, Germany, 2002.
[22] M. Mauve, "How to keep a dead man from shooting," presented at Proc. of the 7th International Workshop on Interactive Distributed Multimedia Systems, 2000.
[23] J. S. Steinman, "Interactive SPEEDS," presented at Proceedings of the 24th annual symposium on Simulation, New Orleans, Louisiana, United States, 1991.
[24] J. S. Steinman, "Breathing Time Warp," presented at Proceedings of the seventh workshop on Parallel and Distributed Simulation, May 1993.
[25] G. J. Armitage, "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3," presented at 11th International Conference on Networks, Sydney, 2003.
[26] A. Abdelkhalek, A. Bilas, and A. Moshovos, "Behavior and Performance of Interactive Multi-Player Game Servers," presented at 2001 IEEE International Symposium on Performance Analysis of System and Software, 2003.
[27] E.Cronin, B. Filstrup, and A. Kurc, "A Distributed MultiPlayer Game Server System." EECS589, Coruse Project Report, University of Michigan, May2001.
[28] M. Mauve, S. Fischer, and J. Widmer, "A Generic Proxy System for Networked Computer Games," presented at Netgames 2002, Braunschweig, Germany, 2002.
[29] J. D. Pellegrino and C. Dovrolis, "Bandwidth Requirement and state consistency in three multiplayer game architecture," presented at NetGames 2003, Redwood City, California, 2003.
[30] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales: Supporting Large Multiuser Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 16, pp. 50-57, 1996.
[31] Unknown-Author, "Overview of ButterFly Grid," http://www.emergentgametech.com.
[32] P. Boustead and F. Safaei, "Comparison of Delievery Architecture for Immersive Audio in Crowded Network Games," presented at NOSSDAV'04, Cork, Ireland, 2004.
[33] D. Bauer, S. Rooney, and P. Scotton, "Network Infrastructure for Massively Distributed Games," presented at NetGames 2002, Braunschweig, Germany, 2002.
[34] Napster. http://www.napster.com.
[35] Gnutella. www.gnutella.com.
[36] KaZaA. www.kazaa.com.
[37] J. Ritter, "Why Gnutella Can't Scale," http://www.darkridge.com/~jpr5/doc/gnutella.html, 2001.
[38] A. J. Howe, "Napster and Gnutella: a Comparison of Two Popular Peer-to-Peer Protocols," vol. 2005, http://members.tripod.com/ahowe_ca/pdf/napstergnutella.pdf, 2001.
[39] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of Networking in Multiplayer Computer Games," presented at Proc. of the International Conference on Application and Development of Computer Games in the 21st Century, 2001.
[40] O. Hagsand, "Interactive multiuser VEs in the DIVE system," *Multimedia*, vol. 3, pp. 30-39, 1996.
[41] S. Rooney, D. Bauer, and R. Deydier, "A Federated Peer-to-Peer Network Game Architecture," in *IEEE Communications Magzine*, vol. 42, 2004, pp. 114-122.
[42] A. Boukerche, R. B. Araujo, and M. Laffranchi, "A Hybrid Solution to Support Multiuser 3D Virtual Simulation Environments in Peer-to-Peer Networks," presented at Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2004.
[43] T. Limura, H. Hazeyama, and Y. Kadobayashi, "Zoned Federation of Game Servers: a Peer-to-Peer Approach to Scalable Multi-Player Online Games," presented at SIGCOMM'04 WORKSHOP, Portland, Oregon, USA, 2004.
[44] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," presented at Proceedings of SIGCOMM 2001, 2001.
[45] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," presented at Proceedings of SIGCOMM 2001, 2001.
[46] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," presented at Proc. IFIP/ACMMiddleware 2001, Heidelberg, Germany.
[47] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," presented at INFOCOM 2004, 2004.