A Spatial Publish Subscribe Overlay for Massively Multiuser Virtual Environments

Shun-Yun Hu[†], Chuan Wu[‡], Eliya Buyukkaya[§], Chien-Hao Chien[*], Tzu-Hao Lin[*],
Maha Abdallah[§], Jehn-Ruey Jiang[*], and Kuan-Ta Chen[†]

[†]Institute of Information Science, Academia Sinica, Taiwan, R.O.C.
[‡]Department of Computer Science, The University of Hong Kong, Hong Kong, P.R.C.
[§]Laboratoire d'Informatique de Paris 6 (LIP6), Universite' Paris 6, France
[*]Department of CSIE, National Central University, Taiwan, R.O.C.

***Abstract*—Peer-to-peer (P2P) architectures have become popular for designing scalable virtual environments (VEs) in recent years. However, one question that remains is whether a single overlay can be flexible enough to support different types of VEs. We present S-VON, a P2P overlay that attempts this goal by providing spatial publish / subscribe (SPS) services. Besides flexibility, S-VON also aims to be practical and efficient by utilizing super-peers and considering the physical topology (i.e., network distance) to reduce latencies. Our simulations show that super-peers provide a unique design space where both bandwidth usage and latencies can be effectively reduced, such that even a crowded Second Life region can be hosted with residential ADSL.**

I. Introduction

In recent years, massively multiuser virtual environments (MMVEs) such as massively multiplayer online games (MMOG), have demonstrated a growing interest and need for immersive interactions within virtual environments (VEs). A networked VE allows global users to assume virtual representations called *avatars* and enter a virtual world, to interact with others for adventure, socialization, or training. Although state-of-the-art MMOG can host up to a million concurrent users, such scalability is achieved by heavy content replications and server-side investments — hosting disjoint clusters of servers, each serving a few thousands of users maximally. To lower server costs and increase concurrent users to millions *in a single virtual world*, many peer-to-peer (P2P) approaches have been proposed in recent years [1], [2], [3], [4], [5], [6], [7]. These designs can be mainly categorized as follows:

Overlay management The construction of a P2P overlay that determines how nodes in a networked VE should connect and exchange messages [1], [2].

State management The maintenance and distribution of *game states* (*i.e.*, the various attributes used by game objects, such as a player's health points and equipments) onto many peers, while handling consistency, load balancing, and fault tolerance [3], [4], [5], [6], [7].

Content management The utilization of client resources to distribute game content such as voice [8], or 3D models and textures [9].

Fundamental to these designs is a robust and efficient P2P overlay that supports some primitive VE operations, while considering the reality of P2P networks (*e.g.*, heterogenous peer resources, fluctuating network conditions, and dynamic peer presence), alongside with application requirements (*e.g.*, latency constrains due to the interactive nature of VEs). In this paper, we propose a *generic* overlay design that supports the different requirements and network considerations for large-scale networked VE. We first identify the basic primitive in VEs as a *spatial publish / subscribe* (SPS) mechanism [10]. That is, the ability to publish or subscribe to an arbitrary convex virtual area is generic enough to support various existing VE systems. We then propose *S-VON*, a P2P overlay that is designed to be *practical*, where a super-peer architecture is used; *flexible*, where spatial publish / subscribe is supported via a Voronoi-based Overlay Network (VON) [1]; and *efficient*, where the overlay topology is considered during transmission to reduce latencies. Our analysis and simulations demonstrate S-VON's scalability and efficiency to support basic VE requirements, and show how a busy Second Life [7] region can be hosted under residential ADSL environment.

II. Background

VE systems can be seen as state machines where various *game states* are updated via application-specific rules called *game logic* [5]. User behaviors are captured as *event* messages, which are sent to the game state manager for processing. The manager may then send *update* messages to notify other users affected by the actions. An *event - process - update* cycle thus is a common design for VEs. The key to building scalable VEs is then a task of dividing the workload of these cycles to many separate hosts (*e.g.*, server clusters or peers in a P2P networks), while maintaining the basic *consistency* and *interactivity* requirements [11]. This division of workload is possible as even though a system may have a large number of total users, each user often interacts with only a small number of spatially nearby users within an *area of interest* (or AOI) [1], which is typically circular in shape. As the user's AOI is limited, message exchanges thus can be localized.

978-1-4244-7681-7/$26.00 © 2010 IEEE

To distribute state management, *spatial publish / subscribe* (SPS) [10] has been identified as a general mechanism. SPS assumes that all message publications and subscriptions (pub/sub) occur on a Cartesian coordinate system. For a simple 2D SPS, each user node may perform: 1) *point publication*: to send a message at a specific point, receivable by area subscribers whose areas cover the point; 2) *area publication*: to send a message to an area, receivable by any subscribers whose subscriptions overlap with the area; 3) *point subscription*: to receive any publication covering a point; and 4) *area subscription*: to receive any messages published within an area.

With SPS as a primitive, it is possible to support state management with two layers of SPS: one for events and another for updates. For example, if the entire world is divided into various regions, a state manager can perform area subscription over its assigned region at an *event layer*. When users send any events as point or area publications to the *event layer*, managers whose regions are affected can receive and process the events naturally. After a manager modifies the relevant game states, updates can be sent as point publications at an *update layer* for each updated object. Users who have performed area subscriptions over their AOI at the *update layer* may then receive the updates in view. Note that the subscribed areas of managers are often stationary, and publications occur at point locations, while the pub/sub of the users move more dynamically.

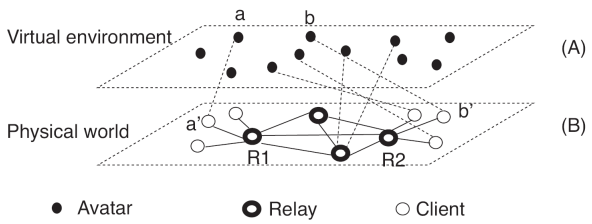


Fig. 1. S-VON architecture.

III. S-VON Architecture

We consider a large-scale P2P VE, where avatars (*i.e.*, clients) are arbitrarily distributed in the virtual world. Each avatar may be interested to receive updates from one or multiple arbitrary convex areas, which constitutes its AOI. An illustration of the VE is shown in Fig. 1(A), where each avatar represents a user in the physical world.

Peers in a P2P network can be highly heterogeneous, with drastically different CPU capacities, bandwidth, and stability. Practical P2P systems thus often maintain the overlay by utilizing *super-peers*, which are peers with higher CPU/bandwidth capacities and better stability, *e.g.*, *ultrapeers* in Gnutella or super-peers in Skype. To address peer heterogeneity and fully utilize peer resources, we also categorize peers into super-peers (called *relays*) and regular peers (called *clients*), according to their capacity, stability, and trustworthiness.

In this paper, both state management (i.e., how to maintain and update game objects given event messages and game logic) and security (i.e., how to ensure that clients and relays do not cheat) are beyond our scope. However, we note that state management is supportable by a basic SPS mechanism (please refer to [10]), and security issues can be reduced if more trustworthy hosts are selected as super-peers [12].

In S-VON, every client in the P2P overlay is attached to a relay. Relays are assumed to have public IP addresses and can reach each other directly, forming a backbone or a *relay mesh*. Clients perform SPS requests, while relays manage overlay connectivity and message delivery, *i.e.*, relays would help their clients to forward pub/sub messages via the relay mesh. As a super-peer may also host an avatar, it too can contain a client part that performs SPS operations by attaching to itself. The relation between clients and relays is shown in Fig. 1(B), where the mapping between the VE and the P2P overlay is as follows: Avatar a and b map to client a' and b' in the P2P overlay, respectively; if avatar a sends a pub message to avatar b, the message is passed from client a' to R_1 (its relay), then forwarded from R_1 to R_2 (the relay b' attaches to), and will be passed on by R_2 to b', eventually reaching avatar b. Note that the maximum number of hops is three for any client-to-client communications, and can be two if both clients connect to the same relay. Additionally, if several clients interested in a publication are attached to the same relay, only one pub message needs to be sent from the publisher's relay to the subscriber's, reducing potential inter-ISP traffic.

To be topology-aware, we assume that each host machine p (client or relay) can be assigned a physical coordinate, (x_p, y_p), that represents its relative location to each other on the physical Internet. The coordinates may represent its latitude and longitude, or be derived using network positioning (*e.g.*, Vivaldi [13]). We assume that such physical coordinates are relatively stable while the corresponding avatar moves in the VE. The Euclidean distances on this coordinate system approximate the latencies (or *network distance*) among the hosts. Upon joining the system, each peer first determines its physical coordinate, then connects to the closest available relay in terms of network distance. As an analogy, the P2P overlay is like a routing layer for end-to-end messaging between two avatars, where the relays are analogous to "routers".

A tracking server (called *gateway*) is used for bootstrapping, similar to many practical P2P systems such as BitTorrent or PPLive. However, to support distribution maximally, the gateway only performs a minimum number of necessary bootstrapping tasks (e.g., helping a joining node to identify its physical coordinate and initial relay), and is not involved in normal operations. We describe S-VON's key designs below:

A. Pub/sub relationship discovery

Since each relay in S-VON forwards messages for the clients connected to itself, it needs to maintain the pub/sub relationship for all the attached clients. Ideally, when a publication request reaches the relay, the relay can simply look up a mapping table to find the subscriber's relay. How to achieve this effectively becomes an important challenge. We adopt a Voronoi-based Overlay Network (VON) [1] and note that it is possible to support SPS by slightly extending VON.

A VON is a fully-distributed overlay that supports the discovery of neighbors within an AOI (i.e., the *AOI neighbors*). Each VON node has a coordinate point and can move continuously on a 2D plane. A node joins the network by contacting a gateway to forward its join request to the neighbor closest to its join location, so that it can learn of some initial AOI neighbors. To discover new neighbors, each node organizes the coordinates of itself and its known neighbors in a *Voronoi diagram*, so that its *boundary neighbors* (*i.e.*, AOI neighbors whose Voronoi regions overlap with the AOI border, see the triangle and circle nodes in Fig. 2) can check if new neighbors will enter the AOI as it moves. To ensure overlay connectivity, each node needs to minimally connect with its closest *enclosing neighbors* (squares in Fig. 2). This way, even though only a few AOI neighbors are known initially, new neighbors can be discovered without a centralized server.

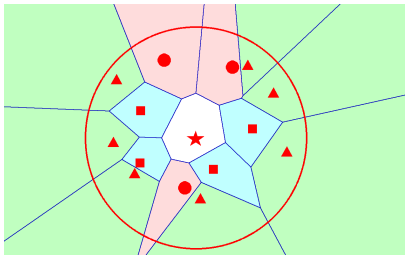


Fig. 2. Neighbors in VON. Big circle is the AOI of the center star node.

The original VON supports area subscriptions natively, but not area publications (i.e., VON allows a node to specify an AOI to receive point publications, but a message cannot be sent to a specified *area* unrelated to or beyond a node's AOI). We observe that by specifying the subscription area as the AOI, subscribers can first learn of potential publishers by performing AOI neighbor discovery. As such knowledge is mutual, the publishers are also made aware of their potential subscribers continuously. If each publisher maintains such a *subscriber list* of nodes with potential interests in its publications, a publisher can then send publications directly to subscribers. Our first extension to VON is thus to let each node, as a publisher, to maintain a subscriber list.

To add relays to the picture, we define a *VON peer* as a virtual client entity stored and managed by a relay. Relays thus act as proxies for the clients in forming a VON composed of the clients' corresponding VON peers. All VON-specific functions then are performed by VON peers (*e.g.*, neighbor discovery, message publications), and actual clients only send pub/sub requests to their relays, and receive publications. S-VON thus can be seen as a logical layer of VON, running on top of the relay mesh. Instead of representing a client machine, a VON peer represents a client's interest (*i.e.*, its subscription area), and many VON peers can physically be at the same relay. To support the proper discovery of other relays, each VON peer also keeps the contact info (e.g., IP and port) of its currently residing relay, and exchanges this info with other VON peers during neighbor discovery among the VON peers.

So the following differences exist between the original VON and S-VON: 1) VON assumes that any client can send messages directly to another client, while S-VON assumes only relays can do so; 2) VON is mainly a method for distributed neighbor discovery, while S-VON is designed to support spatial publish subscribe. VON thus only supports area subscriptions, but not area publications; 3) In VON, only one Voronoi diagram is maintained by a client host; however, for S-VON, several distinct ones are kept at a relay host, one for each VON peer, and none at the clients.

B. Spatial Publish Subscribe

We now describe how subscriptions and publications are supported. For simplicity, we will discuss SPS logically without mentioning the underlying relays.

Subscriptions When a client subscribes, it specifies its subscription as an area with a *reference point* inside. The reference point becomes the position of a VON peer managed by the client's relay. For point subscriptions, we simply use a small area. The VON peer joins the VON at the reference point with the subscription area as its AOI. After joining, the subscription is learnt by all other VON peers whose Voronoi regions are covered by the subscription area (i.e., the joiner's AOI neighbors). As the knowledge on neighbors is mutual, this effectively notifies these AOI neighbor to build up a unique *subscriber list* of the VON peers interested in its publications.

Publications Whenever a publication occurs, the pub request is first sent from the publishing client to its relay, then processed by the client's corresponding VON peer. The VON peer looks up its subscriber list and determines if any subscribers should receive the message. Note that this function is not natively supported by VON. The publication is then sent directly to those subscribing VON peers (via their associated relays physically), and forwarded to the VON peers' associated clients. If no subscribers are found, the publication will have no effect. To support area publications, when a publication spans more than one Voronoi regions, it is first forwarded to each VON peer whose Voronoi regions are partially covered by the publication, and the above process is repeated at each affected VON peer. Note that the VON peers affected may or may not reside at the same relay physically, so the publisher's relay may need to forward the message. Contacting the affected relays is possible as the contact of the physical relay is stored with the VON peer. To avoid redundant forwarding, mechanisms such as *VoroCast* [14] can be utilized.

Movements When clients move continuously in the VE, their subscriptions are also updated. This is done by the clients first notifying their respective relays, who will then update the AOI of the clients' VON peer. VON's move procedure [1] is then followed so that all AOI neighbors can learn of the updated AOI, and update the contents of their subscriber lists to reflect the change in subscription interests. To ensure consistency, VON peers at the relays hold the authoritative versions of the client positions, all others are considered as replicas (e.g., at clients, or in the subscriber lists kept by the VON peers at other relays).

C. Fault Tolerance

Nodes on a P2P network tend to join and leave continuously. If a client fails, the associated relay simply removes the respective VON peer, and the change is handled by VON's leave procedure [1]. For a relay failure, the clients of the failed relay can simply find a new relay to join, and initiates the VON join procedure, similar to how it joins the system originally. As VON itself is designed to be fault tolerant, its fault tolerance is passed to relays, which are simply VON peer proxies.

IV. Evaluation

We try to answer the following questions: 1) How practical can S-VON be deployed? 2) How does topology-awareness affect the performance? and 3) How fault-tolerant is it?

We simulate a 256x256 meter *Second Life* region, where the maximum concurrent users is about 100 [7]. Although only one region is simulated, many regions jointly can support a large total user size, as commonly done by today's MMOGs. If the workload of a single region can be practically reduced, the entire system may also scale better. We set the AOI radius as 64 meters (same as a *Second Life* avatar). To study the steady state behaviors, nodes need to join the system first, before moving at a speed of 5 meters per step for 1000 time-steps, where each time-step is set to 100 ms. Note that this a relatively fast movement, akin to the *fly mode* in Second Life. Nodes move in a clustering pattern to mimic real world users with about 6 clusters. Each node publishes its position 10 times a second, and subscribes its AOI to discover other nodes or receive position updates. For latencies, we use a pair-wise ping data of 90 nodes from PlanetLab[1]. When node A sends a message to node B, the delay is looked up from the latency table so the receiver gets the message some time-steps later. No bandwidth limitation is imposed on the nodes, so that full bandwidth requirement can be evaluated.

Besides being a client, each node can also be a relay. We adjust the number of relays from 1 to 90 to mimic architectures from classical client-server (*i.e.*, 1 relay) to fully distributed P2P (*i.e.*, 90 relays). Relays are chosen as hosts with short latencies from other hosts. (*i.e.*, their physical coordinates are near the center of the Vivaldi coordinates). We set the maximum number of attached clients at a relay as a little over the average if all clients are uniformly assigned. Specifically, the value is $peerlimit = (totalnodes - relaysize)/relaysize + 2$. This is an important parameter that should adjust dynamically by the actual relay load. Finally, each node also keeps info on the 10 closest relays (learned during joining), so that substitute relays can be contacted in case of a relay failure.

To evaluate topology-awareness, *topology-aware join* allows each node to join the system with its physical coordinate, as determined by Vivaldi. *Topology-unaware join* lets each node uses its initial VE coordinate as the physical coordinate. This clusters the nodes close to each other in the VE to connect with the same relay, at least initially. Below we describe the results, and use *peers* to refer the VON peers for short.

[1] http://pdos.csail.mit.edu/~strib/pl_app/2003-02/2003-02-13/

A. Performance

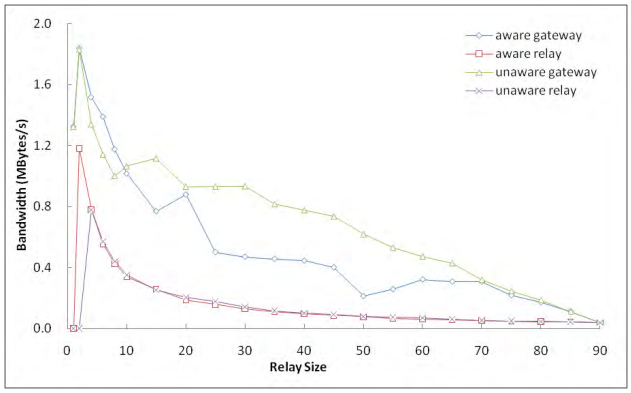
We first evaluate how well is the server load distributed, and at what costs to the relays? Simulations show that a client roughly has 0.5 KB/s of upload and 15 KB/s of download. Upload is fairly constant as only periodic movements exist. Fig. 3(a) shows the upload of both the gateway (i.e., the first relay) and other relays under topology-aware and topology-unaware join. We focus on upload as it is the main bottleneck to scalability. The gateway upload begins at about 1.3 MB/s and gradually decreases to about 40 KB/s in 90 relays (i.e., *pure P2P*). We note that the upload first increases before decreasing, possibly due to the increased inter-relay communications. For relays, the upload begins at 1.1 MB/s with 2 relays, and gradually falls to 35 KB/s (when the relay only hosts its own peer). This shows that a server with 10 Mbps upload requirement only needs 1/3 as much bandwidth with 30 relays. For relays, the upload needs are between 80 KB/s to 200 KB/s. Residential ADSL with less than 5 Mbps of upload thus would suffice to host a server, and 2 Mbps for relays.

Another important aspect to performance is the latencies incurred. Here we measure the most time-critical movement updates. Fig. 3(b) shows that under C/S, the average latency is about 200ms, while under pure P2P is 110ms (i.e., the average end-to-end latency in our dataset). With relays, the average latencies increase quickly to a high of 340ms, before decreasing continuously. This is because pure P2P has 1-hop latencies, C/S has round-trip latencies, while relays result in a mixture of 1 to 3 hops. One important observation is that the average latencies of using relays can be as good as C/S or even better, meaning that a tunable sweetspot exists in relay size (between 50 and 90 relays). Another expected result is that a visible difference exists between topology-aware and unaware joins, at roughly 20-30ms between 10 and 70 relays. However, when using relays is not economic (for very small number of relays), or not practical (for large number of relays), topology-awareness does not affect performance much.

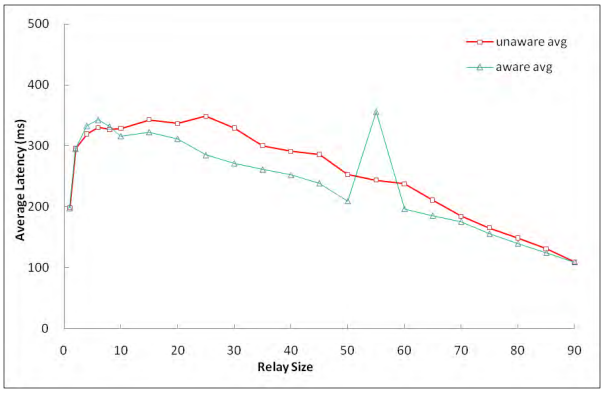
B. Correctness

While using some relays gives better performance than either pure C/S or P2P, we still need ensure that SPS performs correctly. Here we define *discovery consistency* (or consistency for short) as the number of AOI neighbors actually seen over the number of neighbors that should be seen [1]. It is a basic measure of how consistent the view of each node is from the actual view of the system. Fig. 4(a) shows the discovery consistency between topology aware and unaware simulations. We see that pure C/S or P2P achieves the best consistency, at over 99.8%. Consistency drops as latencies increase with the addition of relays, but then improves back as latencies reduce.

For our final evaluation, we consider the effect of failures and fail between 1 to 20 randomly selected nodes of a specific type (so between 1.11% to 20.22% of all 90 nodes). The relay size is fixed at 45 nodes (50% act a relays). Fig. 4(b) shows the effect on consistency after concurrent failures. The failures occur mid-way at 500 time-step in a 1000-step simulation. The average consistencies *after* the failures occur are shown.

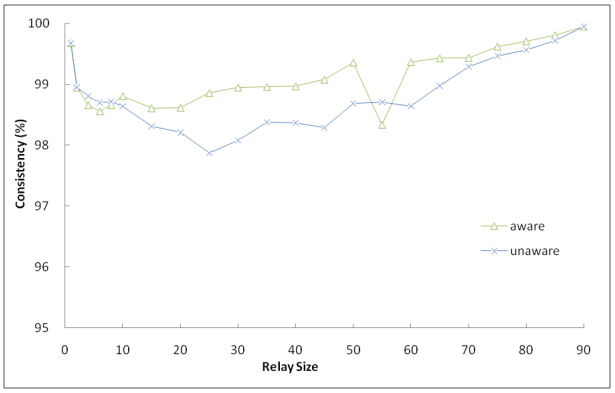


(a) bandwidth usage

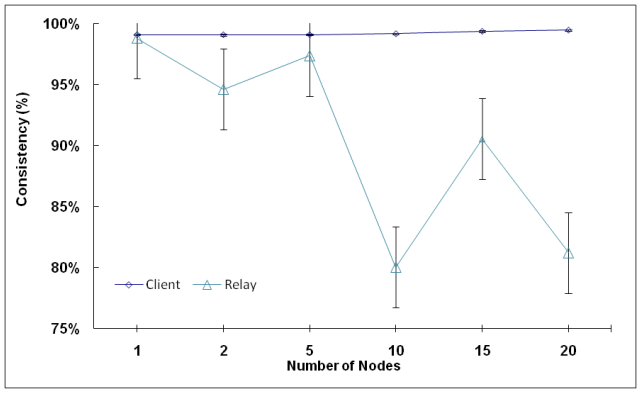


(b) transmission latency

Fig. 3. Performance Evaluation of S-VON



(a) discovery consistency



(b) effect of concurrent failures

Fig. 4. Correctness Evaluation of S-VON

We see that the failures of clients almost have no impact on consistency, as should be expected given a client's light role. However, failures of relays are more serious when the concurrent failures exceeds 5 (5.56% of all nodes, or 11.11% of all relays) under cluster movement. Discovery consistency drops to as low as 80% and do not seem to be restoring. However, we note that such relay-only failure may be rare.

V. Conclusion

In this paper we present S-VON, a generic P2P overlay that supports spatial publish / subscribe (SPS) operations for virtual environment applications. S-VON is designed to be practical, by utilizing a super-peer based design; flexible, by supporting SPS operations; and efficient, by considering network topology for quick message deliveries. By utilizing a Voronoi-based Overlay Network (VON), we design a method to support SPS in a distributed and low-latency manner. From our simulations, we show that S-VON effectively supports a crowded *Second Life* region, by lowering the bandwidth usage at the server, and the latencies between clients, all under current residential ADSL environment. As future work, we would like to explore how SPS can be used to support state management efficiently, and how to handle malicious client or relay behaviors.

References

[1] S.-Y. Hu *et al.*, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.
[2] E. Buyukkaya and M. Abdallah, "Data management in voronoi-based p2p gaming," in *Proc. IEEE CCNC*, 2008.
[3] B. Knutsson *et al.*, "Peer-to-peer support for massively multiplayer games," in *INFOCOM*, 2004, pp. 96–107.
[4] A. Bharambe *et al.*, "Colyseus: A distributed architecture for multiplayer games," in *Proc. NSDI*, 2006.
[5] S.-Y. Hu *et al.*, "Voronoi state management for peer-to-peer massively multiplayer online games," in *Proc. IEEE CCNC*, 2008, pp. 1134–1138.
[6] A. Bharambe *et al.*, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," in *Proc. SIGCOMM*, 2008.
[7] M. Varvello, C. Diot, and E. Biersack, "P2p second life: experimental validation using kad," in *Proc. INFOCOM*, 2009.
[8] J.-R. Jiang and H.-S. Chen, "Peer-to-peer aoi voice chatting for massively multiplayer online games," in *Proc. P2P-NVE*, 2007.
[9] S.-Y. Hu, J.-R. Jiang, and B.-Y. Chen, "Peer-to-peer 3d streaming," *IEEE Internet Computing*, vol. 14, no. 2, pp. 54–61, 2010.
[10] S.-Y. Hu, "Spatial publish subscribe," in *Proc. IEEE Virtual Reality (IEEE VR) Workshop MMVE*, 2009.
[11] G. Schiele *et al.*, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proc. GPC*, 2007.
[12] G.-Y. Huang *et al.*, "Scalable reputation management with trustworthy user selection for p2p mmogs," *IJAMC*, vol. 2, no. 4, pp. 380–401, 2008.
[13] F. Dabek *et al.*, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM*, 2004.
[14] J.-R. Jiang *et al.*, "Scalable aoi-cast for peer-to-peer networked virtual environments," in *Proc. ICDCS Workshops*, 2008.