VAST: 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§ and Jehn-Ruey Jiang*
*Department of CSIE, National Central University, Taiwan, R.O.C.
†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

***Abstract*—Peer-to-peer (P2P) architectures have recently become a popular design choice for building scalable networked virtual environments (VEs). The dynamic nature of VEs with a wide variety of characteristics, however, requires a flexible architecture to support different application needs. In this paper we present VAST, a generic P2P overlay that provides spatial publish / subscribe (SPS) services with layers to support different VE requirements. The overlay accommodates both regular and super-peers, and evolves based on the state of the application. VAST takes into account the physical topology (*i.e.*, network distance) between peers and the heterogeneity in peer resources so that requirements of different kinds of VEs are met in a practical, flexible, and efficient manner. Our analysis and simulations show that the adoption of super-peers provide a unique design space where both the required bandwidth at the server, and the average latencies among clients, can be effectively reduced, such that even a crowded virtual world region can be hosted under residential ADSL environment.**

I. Introduction

In recent years, massively multiuser virtual environments (MMVEs) such as the billion-dollar massively multiplayer online game (MMOG) industry, have demonstrated a growing interest and need for immersive interactions within virtual environments (VEs). A networked VE [1] allows users from across the globe to enter a virtual world and assume virtual representations called *avatars*, to interact with others for adventures, socialization, or training. Although state-of-the-art MMOG can host up to a million concurrent users (*e.g*, *World of Warcraft* hosts over 1-million peak concurrent users in China in 2008[1]), such scalability is achieved by heavy server-side investments — hosting many disjoint worlds using clusters of servers, each serving a few thousands of users maximally. To lower server costs and increase concurrent users to the range of millions *in a single virtual world*, many *peer-to-peer* (P2P) approaches have been proposed in recent years [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 be connected and how messages should exchange [2], [3].

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 [4], [5], [6], [7].

Content management The utilization of client resources to help distribute game content such as voice data [8], [9], or 3D models and textures [10], [11], [12], to players in need.

Fundamental to all these designs is a robust and efficient P2P overlay that must support basic primitive operations in a VE, while considering the reality of P2P networks (*e.g*, heterogenous peer resources, fluctuating network conditions, and dynamic join and leave of peers) and specific application requirements (*e.g*, latency constrains due to the interactive nature of MMVEs). In this paper, we propose a *generic overlay* design that supports the various application requirements and network considerations for large-scale networked virtual environments. We first identify the basic primitive in VEs as a *spatial publish / subscribe* (SPS) mechanism [13]. That is, the ability to publish or subscribe an arbitrary convex virtual area is a functionality generic enough to support various existing VE designs. We then propose *VAST*[2], a P2P overlay that is *practical*, where a super-peer-based architecture is adopted; *flexible*, where spatial publish / subscribe is realized through the use of a Voronoi-based Overlay Network (VON) [3]; and *efficient*, where each node constrains its use of resources, and *topology awareness*, or network latency, is considered during transmissions. The key features in our design include the maintenance of flexible publish / subscribe (pub/sub) relations among peers using a VON, super-peer assisted messaging, and topology-aware super-peer selection that minimizes messaging delays. Our analysis and simulations demonstrate VAST's scalability and efficiency in supporting basic VE requirements in practical VE scenarios.

The rest of the paper is organized as follows. We first present related work regarding P2P-based VE designs in Section II. In Section III, we present the rationales and main designs of VAST. Evaluations of VAST using simulations in terms of performance, correctness, and fault-tolerance is given in Section IV, and the paper is concluded in Section V.

[1]http://www.corp.the9.com/news/2008/news_080411.htm

[2](V)ON-based (A)pplication-layer (S)PS with (T)opology-awareness

II. Motivation and Related Work

A. P2P VE

The premise of P2P VEs is that even though there may be a total of millions of concurrent users in a VE, at any given time, a user is only interested to see or interact with a small number of other users within an *area of interest* (or AOI [1], often denoted as a visibility sphere or circle around the user). Existing P2P VE research can be classified mainly as follows:

Overlay management refers to constructing P2P overlays to address *neighbor discovery* (*i.e.*, finding the *AOI neighbors* of interest, given a position and an AOI-radius). It can be realized by a *spatial query* (as done by Colyseus [5] or GP3 [14]) or by a *spatial multicast* (as done by Solipsis [2] or VON [3]). However, spatial query introduces unwanted latencies, while spatial multicast may generate message overheads, and cannot support AOI of different interest ranges.

State management refers to the maintenance and distribution of game states onto certain chosen peers. SimMud [4] partitions the virtual world into many fixed-size regions, each of which managed by a coordinator peer. Coordinators periodically multicast updates to all region members. HYMS [15] partitions the virtual world into multiple dynamic cells managed by a central server. Qualified clients can then takeover server loads and communicate in P2P fashions among themselves. Colyseus [5] uses a randomized DHT or a range-queriable DHT to discover objects for peers. Each object has a primary node for its management, and any number of read-only object replicas may exist on the other nodes, enabling quick local access to the object. VSM [7] and VoroGame [16] propose state management algorithms by partitioning the world with Voronoi diagrams, so dynamic load balancing can be achieved.

Content management is another recent focus in P2P VEs, and deals with game content (*e.g*, 3D models and textures) that is becoming too large and dynamic. Recent works (*e.g*, FLoD [10], LODDT [11], and HyperVerse [12]) propose to use P2P networks to offload the content delivery from server to clients. The basic idea is that as VE users often have overlapped visibility, the content required for rendering can be obtained from not just the server, but nearby peers as well. Current proposals consist of a *discovery stage* and an *exchange stage*, where each peer first finds which objects are needed and which peers are available, before performing state and content exchange with the discovered peers.

B. VE Scalability

VE systems can be seen as state machines where various *game states* are updated via application-specific rules called *game logic* [7]. User behaviors are captured as *event* messages, which are sent to the manager of game states for processing. The manager may then send *update* messages to notify other users affected by the actions. For example, movement is often the most basic event, which is sent when a user moves and accounts for 70% of traffic in MMOGs. We refer to the user that performs behaviors and sends out event messages as an *agent*, and the game state manager as an *arbitrator*. In client-server architectures (*e.g*, first person shooter (FPS) games [17] or MMOGs [18]), each user client is an agent, and the server is the arbitrator. In a fully connected P2P architecture (*e.g*, real-time strategy (RTS) game), every user machine is both an agent and also arbitrator, where all hosts process all event messages [19]. We can thus see that this *event - process - update* cycle is a common design for VEs.

The key to building scalable VEs thus is mainly a task of dividing the processing and bandwidth workload of these cycles to many separate hosts (*e.g*, server clusters or super-peers in a P2P networks), while maintaining the basic *consistency* and *interactivity* requirements [20]. It is possible as the user's AOI is often limited, message exchanges thus can be localized. Typical AOI are circular in shape (as commonly used by MMOGs), but may also be rectangular (for RTS games) or even of arbitrary shape (as in FPS games, where visibility extends until encountering some obstacles).

C. Spatial Publish/Subscribe (SPS)

Recently, *spatial publish / subscribe* (SPS) [13] has been identified as a general mechanism to distribute state management. A SPS assumes that all message publications and subscriptions (pub/sub) occur within a Cartesian coordinate system. For simplicity, we will focus on a 2D coordinate system (*i.e.*, a plane). Each user node in a SPS may perform one of the following basic operations: 1) *point publication*: to send a message at a specific point; 2) *area publication*: to send a message to an area; 3) *point subscription*: to receive any messages at a point; and 4) *area subscription*: to receive any messages published within an area. The basic relations among them thus are: 1) a point publication is received by any area subscribers covering the point; 2) an area publication is received by any point or area subscribers whose subscriptions fall within the publication area; 3) a point subscription receives any messages sent by an area publication that covers the point; and 4) an area subscription receives any messages generated by a publication (point or area) falling within the subscription area.

With SPS as a primitive, VE state management can be supported with two layers of SPS, one for event and another for updates dissemination (*i.e.*, an *event layer* and an *update layer*). For example, if the entire world is divided into various regions (*e.g*, by grid [21], [4], hexagon, or Voronoi partitions [7]), each handled by an arbitrator of limited responsibility. The arbitrator can then perform an area subscription in the *event layer* over its managed region. Each user then sends any event as a point or area publication to the *event layer*, which will be received by the arbitrator whose regions are affected. After an arbitrator has processed and updated the relevant game states, updates can be sent as point publications on the *update layer* for each updated objects. Users who have area subscriptions for their AOIs may then receive the updates in view. Note that the subscription areas of arbitrators are generally fixed, and publications occur at the objet locations, while the pub/sub of the users move more dynamically.

III. VAST Architecture

The goal of VAST is to design an generic overlay architecture for a large-scale P2P VE, which supports SPS functions in a practical, flexible, and efficient manner. By "*practical*", we aim to address various real deployment issues, including the heterogeneity and churn of peers, the existence of NATs that often hinder P2P deployments, and the security vulnerability resulting from IP exposures in a P2P network. By "*flexible*" we mean to support the full SPS operations. By "*efficient*", the message overhead to implement our architecture should be small, while fast deliveries between any pairs of publishers and subscribers should be provided by considering the underlying network topology.

A. Basic Design

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 area(s) of interest (AOI). An illustration of the VE is shown in Fig. 1 (A). Each avatar in the VE represents a user of the application in the physical world.

Peers in a P2P network can be highly heterogeneous, with drastically different CPU capacity, bandwidth, and stability. Practical P2P systems thus often utilize *super-peers*, which are peers with higher CPU/bandwidth capacities and better stability, to help maintain the overlays, *e.g*, *ultrapeers* in Gnutella or super-peers in Skype. To address peer heterogeneity and utilize fully the peer resources, we divide peers in the P2P VE into super-peers (called *relays*) and regular peers (called *clients*), according to their capacity, stability, and trustworthiness. In VAST, 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 relay backbone or *relay mesh*. Each client performs SPS operations, while a relay manages overlay connectivity and message delivery with the clients' pub/sub information, *i.e.*, the pub/sub messages to/from a client are forwarded by the relay it connects to via the relay mesh. As a super-peer may also map to an avatar, it may also contain a client component that could perform SPS operations by attaching to itself. The P2P overlay with clients and relays is shown in Fig. 1 (B). An illustration of 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 (the relay it attaches to), 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.

In addition, if several clients subscribed to the same pub message are attached to the same relay, only one pub message is sent from the publisher's relay to the subscriber's relay. For example, in Fig. 1 (B), if all three clients attached to relay R_2 subscribe to client a''s publication, only one copy

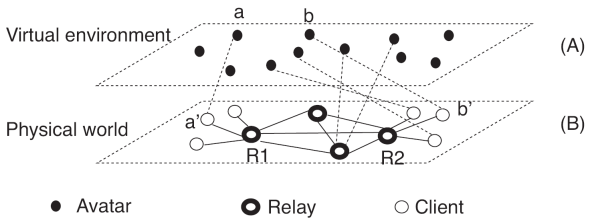


Fig. 1. VAST architecture.

of a''s pub message will be forwarded via R_1 to R_2, who then distributes to all three subscribing clients. This way, no redundant messages are forwarded over the relay mesh.

In the P2P overlay, 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 of a host may represent its latitude and longitude, or be derived using network positioning (*e.g*, GNP [22], Vivaldi [23]). We assume that such physical coordinates are relatively stable while the corresponding avatar moves arbitrarily in the VE. The Euclidean distances on this coordinate system approximate the latencies (or *network distance*) among the hosts. Each client in VAST 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 super-peers are analogous to "routers".

Compared with direct message exchanges among the clients (without going through relays), our design is motivated by the following considerations:

1) Heterogeneity in peer resources: Some clients are low in CPU power and bandwidth capacity and may not publish to many subscribers, as when the user density around an avatar is high; on the other hand, relays may have excess resources that should be maximally utilized to improve the quality of experience for all users.

2) IP hiding: As each client is only aware of the IP of its relay, which is selected from trustworthy hosts, the probability of IP-based attacks is lowered.

3) NAT-traversal: In our relay selection, we may only select hosts with public IPs; the difficult NAT traversal problem thus is bypassed by using relays for deliveries.

4) Inter-ISP traffic minimization: In VAST, only one copy of every pub message is delivered between the source relay (i.e., the pub client) and each destination relay, regardless of how many sub clients exist at the destination relay. If the pub/sub clients are located far apart physically, this effectively avoids redundant long-range messages. In case that clients reside in different ISPs, inter-ISP traffic will be significantly reduced.

A tracking server (called *gateway*) is included in the VAST architecture for bootstrapping, similar to many practical P2P systems such as BitTorrent, PPLive, or SopCast. VAST targets at maximal possible distributed designs and the gateway would only perform a minimum number of necessary tasks.

We describe below the key algorithms to implement VAST:

B. Pub/sub relationship discovery

Since each relay in VAST 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 could simply look up a mapping table to find and deliver message to the subscriber's relay. How to achieve this effectively becomes an important design challenge. We address this issue by utilizing a Voronoi-based Overlay Network (VON) [3] and noting that it is possible to support SPS by slightly extending VON's functions.

A VON is a fully-distributed overlay that allows neighbors to be discovered on a Voronoi-partitioned virtual space. Each node in VON has a coordinate point and specifies an AOI within which the node is constantly aware of all *AOI neighbors*. Nodes are allowed to move continuously in space and connect with new AOI neighbors. For simplicity, we assume a 2D space but note that VON is generalizable to 3D spaces. To discover new AOI neighbors, each node organizes the coordinates of itself and its AOI neighbors in a *Voronoi diagram* [24]. A Voronoi diagram partitions a space with n nodes into n *Voronoi regions* such that all points closest to a particular node are contained within the node's region. Certain *boundary neighbors* (*i.e.*, AOI neighbors whose Voronoi regions overlap with the AOI boundary) thus can be identified to check if the moving node should be notified of new neighbors outside the moving node's current AOI.

To join a VON, a join request is forwarded from any existing node (but often a *gateway* node), towards the direction of the joining position via *greedy forward* (*i.e.*, at each hop, the message is sent to the node whose coordinate is closest to the destination, also known as *compass routing*). Once the request has reached the *acceptor* node whose Voronoi region covers the joining spot, the acceptor can return a list of initial neighbors for the joining node to contact. Additional nodes within the joiner's AOI are discovered via notifications from known AOI neighbors. To maintain the overlay connectivity, each node should be aware of its closest *enclosing neighbors*, even if those neighbors are outside of its AOI (see Fig. 2 for neighbor definitions in a VON. The star is self, the squares are *enclosing neighbors*, the triangles are *boundary neighbors*, and the circles are both enclosing and boundary neighbors).

We note that it is trivial to discover and maintain pub/sub relations among nodes on a VON, by specifying the subscription area as the AOI of a node. Subscribers can thus learn of potential publishers by performing AOI neighbor discovery. As such knowledge is mutual, potential publishers are also aware of their subscribers continuously. Publications can thus be sent directly from publishers to subscribers. Although the original VON uses circular AOI, we note that AOI can in fact be of arbitrary convex shapes without affecting VON's correctness. In VAST, we thus also consider flexible AOIs, which can either be of arbitrary convex shape, or may constitute of multiple disjoint areas. In the latter case, an avatar's AOI maps to multiple nodes in the VON, each corresponding to one interest area for the avatar.

To add relays to the picture, we define a *VON peer* (or simply a *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 the VON peers (*e.g*, neighbor discovery, message publications), and actual clients only send pub/sub requests to their relays, while receiving publications from other clients. In other words, VAST is effectively a logical VON, where each VON node represents a client's interest (i.e., its subscription area), and many VON nodes are physically run at the same relay.

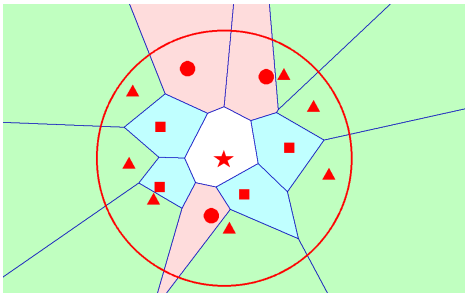


Fig. 2. A Voronoi-based Overlay Network.

C. Spatial Publish Subscribe

We now describe how subscriptions and publications can be supported on VON. To simplify our discussions, we will describe SPS support in purely logical terms (*i.e.*, we only discuss the functions performed by the VON peers and not how relays are used, as relays can be seen as a separate layer independent of the logical layer of VON).

Subscriptions When a client subscribes, it specifies its subscription in the form of an area with a *reference point* inside. The reference point becomes the center position of a VON peer managed by the client's relay. Note that the area size is a small value for point subscriptions. The VON peer would then join a VON at the reference point with the subscription area as its AOI using VON's join procedure. After the join, the subscription area is learnt by all other VON peers whose Voronoi regions are covered by the subscription area. As the knowledge on VON neighbors is mutual, this effectively helps potential publishers to build up a *subscriber list* of remote VON peers who might be interested in future publications.

Publications Whenever a publication occurs, the publication request is first sent from the publishing client to its relay, and processed by the publisher's corresponding VON peer. The VON peer looks up its subscriber list and determines if any subscribers should receive the message. The publication is then sent directly to those subscribing VON peers (via their associated relays physically). If no subscribers are found, then the publication has no effect. If a publication area spans more than one Voronoi regions, it can be 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. To avoid redundant forwarding, a mechanism such as *VoroCast* [25] can be utilized.

D. Relay Selection and Load Balancing

When a new client joins the VAST network, it first contacts a *gateway* and attempts to locate its physical coordinate via some existing network position algorithms (e.g. Vivaldi). For example, the gateway could send back to the client a few initial relays with which the client can measure latencies and calculate its own physical coordinate. The client then asks some existing host in the relay mesh (*e.g.*, the gateway) to help forward its physical coordinate to the relay closest to it. The closest relay would then send back the joining client an initial list of its known relay neighbors. The client then could contact each relay and connects to the closest, available one. To prevent overload, each relay would set a *peer limit* as to the maximum number of clients it can accept. Clients refused by a relay to join may contact the next available relay until it finds one to join. After correct joining, the client may performs SPS operations using the methods described previously.

New relays in VAST are dynamically selected and added into the relay mesh, to balance the load on the existing relays and to bound the message passing delay between pub/sub clients. There are two possible types of load for a given relay: 1) the maintenance of subscriptions and subscription matching; and 2) the communication traffic to deliver messages to and from clients. Note that these two loads are somewhat independent (*e.g*, if subscription interests do not change, then the load would mainly be on communication).

Each existing relay in the relay mesh measures its own load overtime, which can consist of both CPU usage and bandwidth consumption. When the load on the relay becomes excessive, the relay will select a list of "migration candidate clients" from all the clients connected to itself, and recommend their migration to other relays. The migration candidate clients can be selected using different rules, *e.g.*, those with the largest distances to the relay, or the maximum numbers of messages delivered via the relay. The relay suggests to each migration candidate client another existing relay with the next closet network distance to the client and a light load, where the distance is computed using coordinates of the client and loading information of other relays. The migration candidate client will then signal the recommended relay for a migration. The relay contacted will evaluate whether to accept a migration or not, based on its current load and the distance between itself and the candidate client.

For those migration candidate clients whose migration requests are not accepted, they will be reported by the current relay to the gateway server. Each migration candidate client reported is associated with a *weight of migration*, which reflects the necessity level of its migration. This weight can be determined by several factors, such as the load on its current relay and the distance of the client to the relay.

The gateway server keeps a list of migration candidate clients reported from different relays and groups them according to their physical coordinates. The gateway server also maintains a dynamic pool of candidate relays, which are selected from all hosts in the overlay. The selection of candidate relays should typically be implemented as a continuous process: A host is treated as a client when it first joins, its quality with respect to bandwidth/computational capacity, stability and trustworthiness will be evaluated in an ongoing fashion, and it can be promoted to a candidate relay if it is found to satisfy the selection criteria. Detailed selection criteria are application-dependent, and are thus out of our current scope.

When a migration group in a physical region has become significant enough (with respect to client numbers and migration weights), the server will select one new relay for the migration candidate clients in the group from its pool of candidate relays. Let $\mathcal{C}$ denote the set of candidate relays and $\mathcal{P}$ be the set of migration candidate clients in the group. Let $\mathcal{R}$ be the set of all existing relays. A new relay is selected from the candidate pool $\mathcal{C}$ as the best one, via which the average messaging delay from a client in $\mathcal{P}$ to any relay in $\mathcal{R}$ is minimum.

Let $d(a, b)$ represent the Euclidean network distance between host a and b, computed using the physical coordinated of a and b. The new relay v is selected by solving the following optimization problem:

$$\min_{v \in \mathcal{C}} \frac{\sum_{p \in \mathcal{P}} d(v, p)}{|\mathcal{P}|} + \frac{\sum_{r \in \mathcal{R}} d(v, r)}{|\mathcal{R}| + 1} \quad (1)$$

Here, $|\mathcal{P}|$ and $|\mathcal{R}|$ denote the number of clients/relays in $\mathcal{P}$ and $\mathcal{R}$, respectively. The first operand in Eqn. 1 represents the average network distance between a client and the candidate new relay; the second denotes the average network distance between the relay and all the relays (including itself). The objective function represents the average network distance from a pub client to any destination relay via the new relay, which approximates the message passing latency from the client to any destination relay. When the best relay is found, the new relay would join the existing relay mesh and accept the migrated clients.

E. Fault Tolerance

Handling of the failure of a relay is closely related to the procedures for load balancing and new relay selection. If a relay fails due to the departure of the corresponding avatar, the clients originally connected to the relay will need to contact the remaining relays it knows. If the connection request from such a client is not accepted by the contacted relays, the client will be included in the migration candidate list at the tracking server with the most significant migration weights. New relays will be selected to serve such clients by the server.

Relays are *stateful* (*e.g*, they retain pub/sub relations), so it is important that proper mechanisms exist to restore their states in case of failures. Currently, when clients are disconnected from VAST due to a failed relay and re-connects back to an existing relay. They would re-perform subscriptions at the new relay, effectively building up the new pub/sub relations at the new relay and its associated neighbors. However, such procedure may take some time and would show as application pauses to the clients during the re-join.

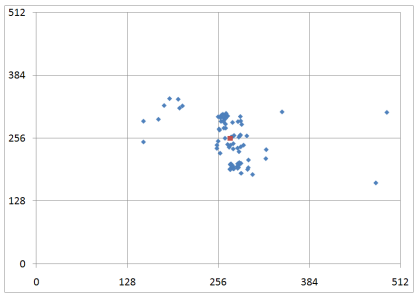


Fig. 3. Physical coordinates obtained from Vivaldi [23]

IV. Evaluation

The evaluation of VAST involves answering certain questions related to VAST's design goals. The main questions we would like to answer are: 1) How practical is VAST if deployed in a real world situation? 2) How does the size of relays affect the overall performance and correctness of VAST? 3) How does topology-awareness affect the general performance? and 4) How tolerant is VAST under the dynamics of P2P networks? Specifically, how well can VAST sustain node departure or massive failures?

We use simulations for our evaluation as it can be done within a more controlled environment. We choose to simulate a *Second Life* region, with a dimension of 256 x 256 meters, and maximum concurrent users of about 100 [26]. The AOI radius of a *Second Life* avatar is about 64 meters.

For the latencies between the nodes, we would like to use a full pair-wise latency dataset from the real world, and found a set of 90 nodes from PlanetLab's all-pair ping[3]. We do not set any bandwidth limitation on the nodes, so that full bandwidth requirement can be measured and observed with our scenarios.

As we are mainly interested in VAST's steady state behavior, we allow the full 90 nodes to join the system first, before allowing them to move with a given behavior pattern. Each node performs point publications of its current positions 10 times a second, and uses its AOI as the subscription area, to discover other nodes and receive their position updates. The subscription area also continuously moves with the node. We test both *random waypoint* and *cluster movement*, where the cluster size is set to 6 (*i.e.*, 15 nodes per cluster on average), and each node generally moves within its own cluster, with only a small probability to move towards other clusters. The movement speed is set to 5 meters per time-step, where each time-step is 100 ms (*i.e.*, 50 meters / second), this is a rather high speed given the AOI-radius of the avatar (*i.e.*, 2.5 seconds to cross a full AOI), as we would like to see how VAST performs under what might be currently considered as a crowded virtual world region.

We adjust relay size from 1 to 90 to simulate both a classical client-server (*i.e.*, 1 relay) and fully distributed P2P (*i.e.*, 90 relays) and any in-between range. Relays are chosen as hosts whose latencies are generally shorter from other host (*i.e.*, their physical coordinates are closer to the center of the Vivaldi coordinate system, see Fig. 3). Each simulation is run for 1000 time-steps (*i.e.*, roughly 100 seconds). One important parameter related to performance is the number of maximum clients at each relay. For this, we designate it as a little over the average number of clients each relay should accept if clients are uniformly assigned to relays. Specifically, the value is $peerlimit = (totalnodes - relaysize)/relaysize + 2$. In practice, the limit should be determined dynamically and individually according to the loading of the relay. Finally, each node is allowed to maintain the information of 10 closest relays, so that clients can find other relays to connect in case of relay failure.

To evaluate the effect of topology-awareness, we use two methods for a new node's join. Topology-aware join allows each node to join the system with its physical coordinate, as determined by the Vivaldi algorithm. Topology-unaware join lets each node uses its initial coordinate in the virtual world as its physical coordinate. This has the effect of clustering nodes that are close in the virtual world to connect to the same relay, at least initially.

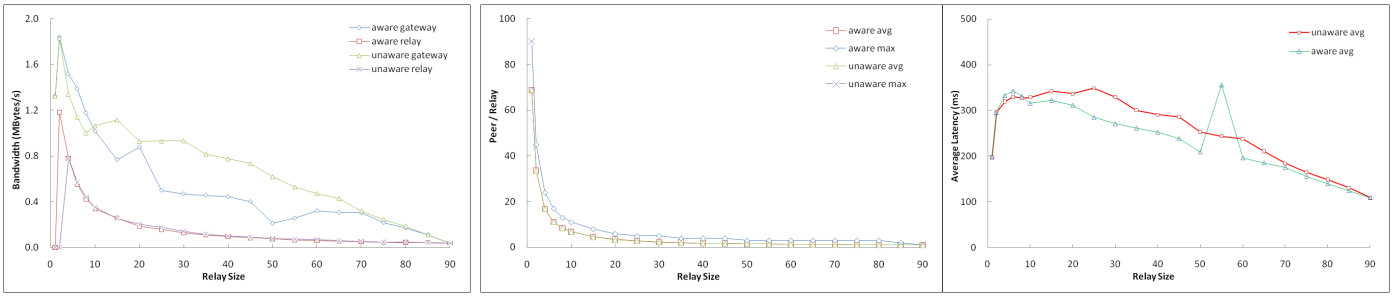
Below we discuss the simulation results according to VAST's performance, correctness, effect of movement models, and fault-tolerance. For brevity, we will use *peer* to refer to VON peers in our discussions below. As cluster movement is more realistic in virtual worlds, the discussions on performance and correctness are shown with cluster movement.

A. Performance

The first question we would like to answer is: how well can a P2P approach such as VAST distribute the server's load, and what would be the additional load at each relays? We find that given the movement patterns and user density in our scenario, an ordinary client roughly has 0.5 KB/s of upload and between 15 KB/s (for cluster movement) to 20 KB/s (for random movement) of download. Upload is fairly constant for clients as they only send periodic movement publications to their relays. The difference in client download reflects the different average AOI under the specific movement model, where it is about 15 neighbors under cluster movement (*i.e.*, 90 nodes / 6 clusters = 15 neighbors / cluster), and 20 neighbors under random way-point.

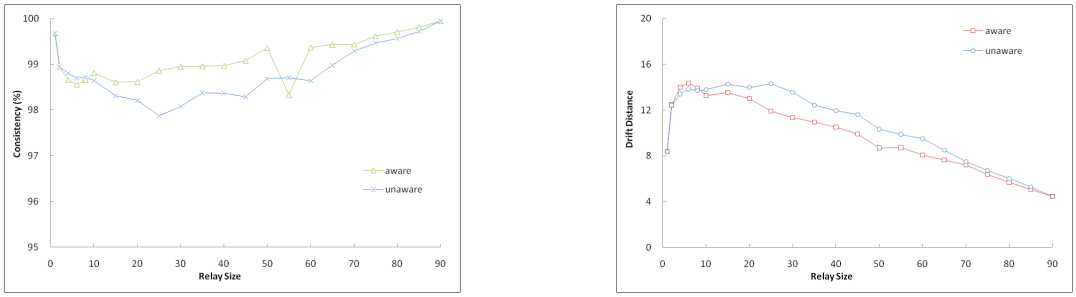
Fig. 4(a) shows the upload bandwidth for both the gateway and the relays under topology-aware and topology-unaware join mechanisms. We focus on upload as it is often the main bottleneck for both clients and servers. We can see that gateway upload is about 1.3 MB/s under a single relay (*i.e.*, client-server, or *C/S*-like), and gradually decreases to only about 40 KB / sec in 90 relays (or, *pure P2P*). We also note that gateway's upload increases first to 1.8 MB/s and 1.5 MB/s when the number of relays increases to 2 and 4, before falling down to 1.3 MB/s again at 6 relays, and further down to 500 KB/s at 25 relays. This shows that initially, due to the increased inter-relay communications, bandwidth usage

[3]http://pdos.csail.mit.edu/ŝtrib/pl_app/2003-02/2003-02-13/. Note that as it becomes more difficult to obtain full pair-wise ping as more nodes are in the system, we found that smaller total node size would give us more complete pair-wise ping.



(a) bandwidth usage (b) peer size per relay (c) transmission latency

Fig. 4. Performance Evaluation of VAST (cluster movement)

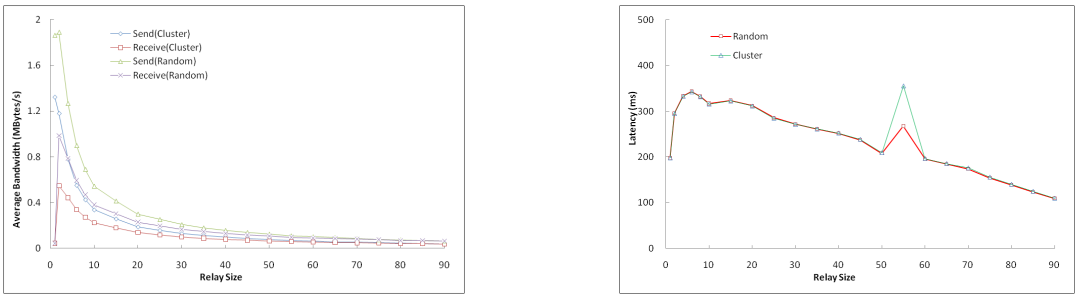


(a) discovery consistency (b) drift distance

Fig. 5. Correctness Evaluation of VAST (cluster movement)

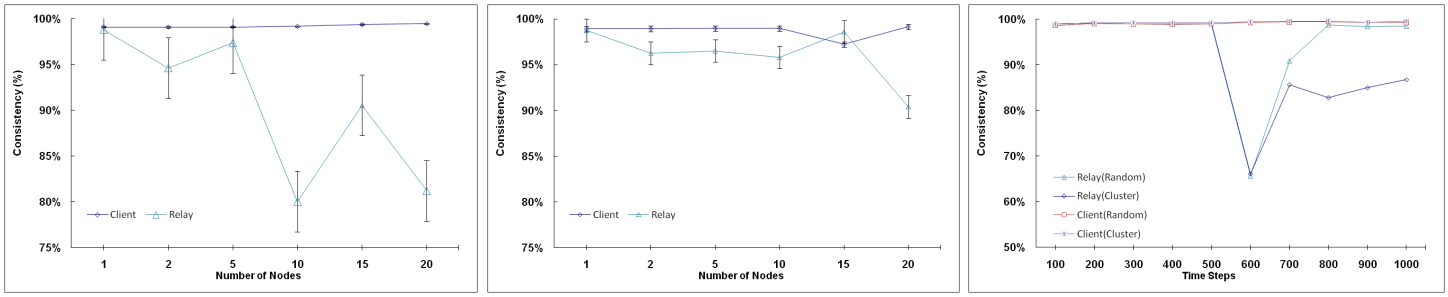
may in fact increase before falling down. For relays, it begins with the 1.1 MB/s upload for 2 relays, and gradually falls down to as low as 35 KB/s (when the relay hosts itself as the only peer). We also observe that, when there are more than sufficient relays for clients to connect (*e.g*, after 45 relays, more relays exist than clients), many relays thus handle its client as the only peer. The *effective relay size* thus is often much smaller than the designated relay size (*e.g*, for 60 relays, only 16 relays, or roughly 15% of all nodes, contain more than one peer). As shown in Fig. 4(b), the average hosted peers per relay is actually much smaller than the specified relay size. These results show that, under our scenario, a gateway server needs over 10 Mbps of upload in a pure C/S setup, but would require only one third as much bandwidth, when there is about 30 relays. Residential ADSL with less than 5 Mbps of upload would suffice to host such a gateway server. For relays, when the relay size is 1/3 of all nodes (*i.e.*, 30 relays), the upload is between 80 KB/s to 200 KB/s for each effective relay, which means that a 2 Mbps upload would suffice as relays. Thus, we can see that if 1/3 of all nodes are capable to perform relay tasks, our scenario of 90 quickly moving users can be supported with only a relatively light-weight server.

Another important aspect to VAST's performance is the latencies incurred for transmissions. Note that as we do not consider the effect of processing, the latencies in discussion here only refer to transmission latencies. We specifically measure the latencies for movement updates, as these are often the most time-critical updates for VE applications. From Fig. 4(c), we see that under C/S the average (and the maximum) latency is about 200ms, while the average latencies under pure P2P is 110ms, which is roughly the average end-to-end latency in our dataset. With the increase in relays, the average latencies increase quickly to a high of 340ms, before decreasing continuously. The main reason is that while pure P2P has an average 1-hop latency, and C/S has the average round-trip latencies. With relays, the message needs to travel three-hops maximally before reaching a subscriber. However, as the number of relays increases, the transmissions will compose of a mixture between 1 to 3 hops (*i.e.*, 1 hop for direct transmissions between clients who are also relays; 2 hops for clients hosted on the same relay; and 3 hops for clients communicating via 2 relays). The average latencies thus decreases until all communications occur with 1 hop (at 90 relays). One important observation is that while the average latencies initially increase with the usage of relays, they would decrease to a point where it is about as good as C/S (at roughly 50 relays), and even better. Thus, there is a sweetspot in relay size (between 50 and 90 relays), where the system has equal or better average latencies than C/S, and requires much lower upload bandwidth. Another expected observation is that a visible latency difference exists between topology-aware and unaware joins, at roughly 20-30ms between 10 and 70 relays. Depending on the application, this may or may not be a significant performance factor. It also means that topology-awareness is mostly relevant when using relays matters. When using relays is not economic (for very small number of relays), or not practical (for large number of relays), being topology-aware does not affect performance much.



(a) bandwidth (b) latency

Fig. 6. Effect of behavior models (random and cluster movement)



(a) failure under cluster movement (b) failure under random movement (c) time-series of 20 concurrent node failures

Fig. 7. Effect of node failures (random and cluster movement)

B. Correctness

While the basic performance of a relay-based architecture may be more desirable than pure C/S or P2P, we need to make sure that VAST still performs the basic publish / subscribe 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 (also known as *topology consistency* [3] or *awareness rate* [27]). It is a basic measure of how consistent is the view of each node from the actual view of the system. Another measure we use is *drift distance* [3], which measures the difference between the observed and the actual coordinate positions. Drift distance compliments discovery consistency in measuring correctness, as being aware of a neighbor does not equate to knowing its position correctly. Drift distance also increases with higher latencies. Fig. 5(a) shows the discovery consistency between topology aware and unaware simulations. We can see that pure C/S or P2P achieves the best consistency, at over 99.8%. Consistency drops as latencies increase with the introduction of relays, but then improves back as latencies become smaller again. Fig. 5(b) also shows the drift distance for both topology-aware and unaware simulations, where the shape of the curves closely resemble those of the latencies data. We can thus see that both the discovery consistency and the drift distance are a function of the average latency.

C. Movement Models

To verify that VAST may perform with similar characteristics under different movement patterns, Fig. 6(a) shows the bandwidth usage (both send and receive) for the relays under both random and cluster movements. Here we see that bandwidth usage at the relays decreases quickly as relay size increases, but cluster movement uses a smaller amount of bandwidth. This is likely due to the lower number of average AOI neighbors under cluster movement (about 15) than under random movement (about 20). However, Fig. 6(b) shows the average latencies for both movement models, and we can clearly see that no significant differences exist for latencies between the models.

D. Fault-tolerance

For our final evaluation, we consider the effect of node failure on system performance. Here we perform failures between 1 to 20 randomly selected nodes of a specific type. Fig. 7(a) shows failures under cluster movement and Fig. 7(b) shows failures under random movement. Each simulation is run with 1000 steps, and a failure scenario of the specified number of nodes occurs mid-way at 500 time-step. The average discovery consistencies *after* the failures occur are shown.

We can see that the failures of clients almost have no impact on the discovery consistency of the system, as should be expected given a client's light role in the system. However, failures of relays are more serious when the concurrent failures exceeds 5 under cluster movement. Discovery consistency would drop to as low as 80% and do not seem to be restoring. Failures of relays under random movements however recover better, even for 20 concurrent failures. Fig. 7(c) shows the time-series of the scenario of 20 concurrent node failures for

both relays and clients under both movement models. Even under such severe failures, the consistency recovers in 300 time-steps (*i.e.*, 3 seconds) for relays under random movement. The difference in recovery may be due to that in cluster movements, concurrent failures of relays are more likely to take down all the AOI neighbors of a given client, thus creating an *overlay partition*. However, this also shows that by keeping a few random neighbors outside the AOI, tolerance of against relay failures may greatly improve.

V. Conclusion

In this paper we present VAST, a generic P2P overlay that supports spatial publish / subscribe (SPS) operations for virtual environment applications. VAST 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 VAST can effectively support a crowded *Second Life* region, by lowering the bandwidth usage at the server, and the communication latencies between clients, all under current residential ADSL environment.

While the basic performance, correctness, and fault tolerance of VAST has been investigated, we still have not yet evaluated its load balancing mechanisms, which will be our primary future work. Full area publications have also not been investigated, while whether publication message redundancy can be avoided should be investigated in-depth. VAST opens up many interesting design possibilities that are worthy of further investigations, for example:

- Improvement in fault tolerance, where maintaining a few random non-AOI neighbors may help a node to sustain better of relay failures.
- Minimal relay-to-relay forwarding, where due to the triangular inequality on the Internet, a message may achieve even shorter latencies via a third relay.
- The decision for the optimal relay size, given certain application-specific bandwidth and latency limits.
- The alleviation of load during initial relay join, so that gateway's loading is not increased with a few initial relays.
- More efficient handling of relay failure, such that clients experience minimal delay during relay fail.

References

[1] S. Singhal and M. Zyda, *Networked Virtual Environments*, 1999.

[2] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world," in *PDPTA*, 2003.

[3] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.

[4] B. Knutsson *et al.*, "Peer-to-peer support for massively multiplayer games," in *INFOCOM*, 2004, pp. 96–107.

[5] A. Bharambe *et al.*, "Colyseus: A distributed architecture for multiplayer games," in *NSDI*, 2006.

[6] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling large-scale, high-speed, peer-to-peer games," in *Proc. SIGCOMM*, 2008.

[7] S.-Y. Hu, S.-C. Chang, and J.-R. Jiang, "Voronoi state management for peer-to-peer massively multiplayer online games," in *Proc. 5th Annual IEEE Consumer Communications and Networking Conference (CCNC), 4th IEEE Intl. Workshop on Networking Issues in Multimedia Entertainment (NIME)*, 2008, pp. 1134–1138.

[8] L. S. Liu and R. Zimmermann, "Immersive peer-to-peer audio streaming platform for massive online games," in *Proc. 3rd IEEE Consumer Communications and Networking Conference (CCNC 2006)*, 2006, pp. 1229–1233.

[9] J.-R. Jiang and H.-S. Chen, "Peer-to-peer aoi voice chatting for massively multiplayer online games," in *Proc. International Workshop on Peer-to-Peer Network Virtual Environments*, 2007.

[10] S.-Y. Hu, T.-H. Huang, S.-C. Chang, W.-L. Sung, J.-R. Jiang, and B.-Y. Chen, "Flod: A framework for peer-to-peer 3d streaming," in *Proc. INFOCOM*, 2008.

[11] J. Royan, P. Gioia, R. Cavagna, and C. Bouville, "Network-based visualization of 3d landscapes and city models," *IEEE CG&A*, vol. 27, no. 6, pp. 70–79, 2007.

[12] J. Botev, A. Hohfeld, H. Schloss, I. Scholtes, P. Sturm, and M. Esch, "The hyperverse: concepts for a federated and torrent-based '3d web'," *International Journal of Advanced Media and Communication (IJAMC)*, vol. 2, no. 4, pp. 331–350, 2008.

[13] S.-Y. Hu, "Spatial publish subscribe," in *Proc. IEEE Virtual Reality (IEEE VR) Workshop MMVE*, 2009.

[14] M. Esch, J. Botev, H. Schloss, and I. Scholtes, "Gp3 - a distributed grid-based spatial index infrastructure for massive multiuser virtual environments," in *Proc. P2P-NVE*, 2008.

[15] K. Kim, I. Yeom, and J. Lee, "Hyms: A hybrid mmog server architecture," *IEICE Trans. Info. and Sys.*, vol. E87-D, no. 12, 2004.

[16] E. Buyukkaya, M. Abdallah, and R. Cavagna, "Vorogame: A hybrid p2p architecture for massively multiplayer games," in *Proc. IEEE CCNC*, 2009.

[17] T. Sweeney, "Unreal networking architecture," http://unreal.epicgames.com/network.htm, 1999.

[18] T. Alexander, *Massively Multiplayer Game Development*. Charles River Media, 2003.

[19] P. Bettner and M. Terrano, "1500 archers on a 28.8: Network programming in age of empires and beyond," Proc. GDC, 2001.

[20] G. Schiele *et al.*, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proc. GPC*, 2007.

[21] P. Rosedale and C. Ondrejka, "Enabling player-created online worlds with grid computing and streaming," Gamasutra Resource Guide, 2003.

[22] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM*, 2002.

[23] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. SIGCOMM*, 2004.

[24] F. Aurenhammer, "Voronoi diagrams-a survey of a fundamental geometric data structure," *ACM CSUR*, vol. 23, no. 3, pp. 345–405, 1991.

[25] J.-R. Jiang, Y.-L. Huang, and S.-Y. Hu, "Scalable aoi-cast for peer-to-peer networked virtual environments," in *ICDCS Workshops*, 2008.

[26] M. Varvello, F. Picconi, C. Diot, and E. Biersack, "Is there life in second life?" in *Proc. CoNEXT*, 2008.

[27] P. Morillo *et al.*, "Providing full awareness to distributed virtual environments based on peer-to-peer architectures," *LNCS*, vol. 4035, 2006.